InvenSense Inc.
1197 Borregas Ave., Sunnyvale, CA 94089 U.S.A.
Tel: +1 (408) 988-7339 Fax: +1 (408) 988-8104
Website: www.invensense.com

Document Number :DOC-MPL-FS-V3.4.0
Release Date :04/06/2011

MPL Functional Specification Version 3.4.0

A printed copy of this document is
NOT UNDER REVISION CONTROL
unless it is dated and stamped in red ink as,
"REVISION CONTROLLED COPY."

Chapter 1

Purpose and Scope

This document is a guide to all of the functions available in the InvenSense Motion Processing Library (MPL), and corresponds with MPL Release v3.4.0. This release is designed to work with all MPU-3050 devices revision K or earlier.

MPL contains the code for controlling the InvenSense MPU-3050 series gyroscopes, including activating and managing built in motion processing features. All of the source code is in ANSI C and can be compiled in C or C++ environments.

All functions available in the MPL are described in this document, including all parameters involved in the function calls. The functions are divided into modules as follows:

Module	Name	Description
MLDMP	Motion Library DMP	Top level functions that define how to load the MPL.
ML	Motion Library	Controls basic operation of motion processing.
MLDL	ML Driver Layer	Used to configure hardware and low level ML functionality.
FIFO	Abstracted FIFO Driver Layer	Driver for the FIFO.
FIFOHW	Hardware FIFO Driver Layer	Driver for the HW FIFO.
COMPASSDL	Compass Driver Layer	Driver Layer for Compass support.
MLSL	ML System Layer	Hardware specific functions used by MLDL that must be written by the customer.
ML_CONTROL	ML Control	Processes gyroscopes and accelerometers to provide control signals that can be used in user interfaces to manipulate objects such as documents, images, cursors, menus, etc.
GESTURE	Gesture	Processes gyroscopes and accelerometers to provide recognition of a set of gestures.
ORIENTATION	Orientation	Determines the orientation of device in the space.
ML_PEDOMETER	Pedometer	Enables the step counting feature along with Gesture, Orientation & Tap.
ML_PEDOMETER_STAND_ALONE	Stand Alone Pedometer	Enables the step counting feature only.
ML_NAVWALK	Pedometer Navigation	Pedestrian navigation module where the computation is offloaded to the host processor for increased accuracy.
GLYPH	Glyphs	Character recognition engine.
MPU_SELF_TEST	MPU Self Test	API to manage and trigger the run of the Self Test for gyros and accelerometers.
ML_STORED_DATA	ML Stored Data	Load and Store calibration APIs.

For more information on how to use these functions in a specific application, refer to InvenSense Application Notes.

Chapter 2

About this document

This document is automatically generated from the source files using Doxygen's output format in the LaTeX. Heading, footer, and general document format are customized from the standard header template provided by Doxygen. This document is subdivided in the various sections, each describing the main source Modules composing the MPL and implementing specific features (e.g. Pedometer, Gesture Recognition Engine, etc...).

Every section starts with a brief description and an overview of the functions composing the module. Each of those functions is also fully documented in the analogous "Function Documentation" section. Clicking on the function prototype will lead to the portion of text full documentating it.

This **MPL Functional Specification** is best viewed in a PDF viewer, as it provides text hyperlinks and bookmarks on the left-hand side for ease of browsing. There is an Alphabatical Index of the modules and their functions available at the bottom of this document.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

2 Module Index

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 Class Index

Chapter 5

Module Documentation

5.1 MLDMP

These are the top level functions that define how to load the MPL.

Files

- file mldmp.c
  Shared functions between all the different DMP versions.
- file mldmp.h
  Top level entry functions to the MPL library with DMP support.

Functions

- tMLError MLDmpClose (void)
  Closes the motion sensor engine.
- tMLError MLDmpOpen (void)
  Open the default motion sensor engine.
- tMLError MLDmpStart (void)
  Start the DMP.
- tMLError MLDmpStop (void)
  Stops the DMP and puts it in low power.

5.1.1 Detailed Description

These are the top level functions that define how to load the MPL.

In order to use most of the features, the DMP must be loaded with some code. The loading procedure takes place when calling MLDmpOpen with a given DMP set function, after having open the serial communication with the device via MLSerialOpen(). The DMP set function will load the DMP memory and enable a certain set of features.

First select a DMP version from one of the released DMP sets. These could be:

- DMP default to load and use the default DMP code featuring pedometer, gestures, and orientation. Use MLDmpOpen().
- DMP pedometer stand-alone to load and use the standalone pedometer implementation. Use MLDmpPedometerStandAloneOpen().

After MLDmpOpenXXX any number of appropriate initialization and configuration routines can be called. Each one of these routines will return an error code and will check to make sure that it is compatible with the the DMP version selected during the call to MLDmpOpen.

Once the configuration is complete, make a call to MLDmpStart(). This will finally turn on the DMP and run the code previously loaded.

While the DMP is running, all data fetching, polling or other functions can be called and will return valid data. Some parameteres can be changed while the DMP is running, while others cannot. Therefore it is important to always check the return code of each function. Check the error code list in mltypes to know what each returned error corresponds to.

When no more motion processing is required, the library can be shut down and the DMP turned off. We can do that by calling MLDmpClose(). Note that MLDmpClose() will not close the serial communication automatically, which will remain open an active, in case another module needs to be loaded instead. If the intention is shutting down the MPL as well, an explicit call to MLSerialClose() following MLDmpClose() has to be made.

The MPL additionally implements a basic state machine, whose purpose is to give feedback to the user on whether he is following all the required initialization steps. If an anomalous transition is detected, the user will be warned by a terminal message with the format:

```
"Error :  illegal state transition from STATE_1 to
STATE_3"
```

5.1.2 Function Documentation

5.1.2.1 tMLError MLDmpClose (void)

Closes the motion sensor engine.

Does not close the serial communication. To do that, call MLSerialClose(). After calling MLDmpClose() another DMP module can be loaded in the MPL with the corresponding necessary intialization and configurations, via any of the MLDmpXXXOpen functions.

Precondition:

MLDmpOpen() must have been called.

```
result = MLDmpClose();
if (ML_SUCCESS != result) {
    // Handle the error case
}
```

Returns:

ML_SUCCESS, Non-zero error code otherwise.

5.1.2.2 tMLError MLDmpOpen (void)

Open the default motion sensor engine.

This function is used to open the default MPL engine, featuring, for example, sensor fusion (6 axes and 9 axes), sensor calibration, accelerometer data byte swapping, among others. Compare with the other provided engines.

Precondition:

MLSerialOpen() must have been called to instantiate the serial communication.

Example:

```
result = MLDmpOpen( );
if (ML_SUCCESS != result) {
    // Handle the error case
}
```

Returns:

Zero on success; Error Code on any failure.

5.1.2.3 tMLError MLDmpStart (void)

Start the DMP.

Precondition:

MLDmpOpen() must have been called.

```
result = MLDmpStart();
if (ML_SUCCESS != result) {
    // Handle the error case
}
```

Returns:

ML_SUCCESS if successful, or Non-zero error code otherwise.

5.1.2.4 tMLError MLDmpStop (void)

Stops the DMP and puts it in low power.

Precondition:

MLDmpStart() must have been called.

Returns:

ML_SUCCESS, Non-zero error code otherwise.

5.2 ML

Motion Library APIs.

Files

- file ml.c

  The Motion Library APIs.

Functions

- tMLError MLApplyAccelEndian (void)

  Setup the DMP to handle the accelerometer endianess.

- tMLError MLApplyCalibration (void)

  apply the choosen orientation and full scale range for gyroscopes, accelerometer, and compass.

- int MLCheckFlag (int flag)

  MLCheckFlag returns the value of a flag.

- tMLError MLDisableMotionDetect (void)

  Disables the ML_MOTION_DETECT engine.

- tMLError MLEnableMotionDetect (void)

  Enables the ML_MOTION_DETECT engine.

- tMLError MLGetArray (int dataSet, long *data)

  MLGetArray is used to get an array of processed motion sensor data.

- int MLGetEngines (void)

  MLGetEngines is used to determine which engines are currently enabled.

- tMLError MLGetFloatArray (int dataSet, float *data)

  MLGetFloatArray is used to get an array of processed motion sensor data.

- int MLGetGyroPresent (void)

  Check for the presence of the gyro sensor.

- int MLGetInterrupts (void)

  Get the current set of DMP interrupt sources.

- int MLGetMotionState (void)

  MLGetMotionState is used to determine if the device is in a 'motion' or 'no motion' state.

- tMLError MLSerialClose (void)

  Close the serial communication.

- void * MLSerialGetHandle (void)

  get the serial file handle to the device.

- tMLError MLSerialOpen (char const *port)

  Open the serial connection with the device.

- tMLError MLSetAccelCalibration (float range, signed char *orientation)

  Sets up the Accelerometer calibration and scale factor.

- tMLError MLSetArray (int dataSet, long *data)

  used to set an array of motion sensor data.

- tMLError MLSetBiasUpdateFunc (unsigned short function)

  MLSetBiasUpdateFunc is used to register which algorithms will be used to automatically reset the gyroscope bias.

- tMLError MLSetFifoInterrupt (unsigned char on)

  Enable generation of the DMP interrupt when a FIFO packet is ready.

- tMLError MLSetFloatArray (int dataSet, float *data)

  used to set an array of motion sensor data.

- tMLError MLSetGyroCalibration (float range, signed char *orientation)

  Sets up the Gyro calibration and scale factor.

- tMLError MLSetMagCalibration (float range, signed char *orientation)

  Sets up the Compass calibration and scale factor.

- tMLError MLSetMotionCallback (void(*func)(unsigned short motionState))

  MLSetMotionCallback is used to register a callback function that will trigger when a change of motion state is detected.

- tMLError MLSetMotionInterrupt (unsigned char on)

  Enable generation of the DMP interrupt when Motion or no-motion is detected.

- tMLError MLSetMPUSensors (unsigned long sensors)

  Controlls each sensor and each axis when the motion processing unit is running.

- tMLError MLSetNoMotionThresh (float thresh)

  MLSetNoMotionThresh is used to set the threshold for detecting ML_NO_MOTION.

- tMLError MLSetNoMotionThreshAccel (long thresh)

  MLSetNoMotionThreshAccel is used to set the threshold for detecting ML_NO_-MOTION with accelerometers when Gyros have been turned off.

- tMLError MLSetNoMotionTime (float time)

  MLSetNoMotionTime is used to set the time required for detecting ML_NO_MOTION.

- tMLError MLUpdateData (void)

  MLUpdateData updates all the realtime data from the motion algorithms.

- tMLError MLVersion (unsigned char ∗∗version)

  MLVersion is used to get the ML version.

5.2.1 Detailed Description

Motion Library APIs.

The Motion Library processes gyroscopes, accelerometers, and compasses to provide a physical model of the movement for the sensors. The results of this processing may be used to control objects within a user interface environment, detect gestures, track 3D movement for gaming applications, and analyze the blur created due to hand movement while taking a picture.

5.2.2 Function Documentation

5.2.2.1 tMLError MLApplyAccelEndian (void)

Setup the DMP to handle the accelerometer endianess.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.2.2.2 tMLError MLApplyCalibration (void)

apply the choosen orientation and full scale range for gyroscopes, accelerometer, and compass.

Returns:

ML_SUCCESS if successful, a non-zero code otherwise.

5.2.2.3 int MLCheckFlag (int *flag*)

MLCheckFlag returns the value of a flag.

MLCheckFlag can be used to check a number of flags, allowing users to poll flags rather than register callback functions. If a flag is set to True when MLCheckFlag is called, the flag is automatically reset. The flags are:

- ML_RAW_DATA_READY Indicates that new raw data is available.
- ML_PROCESSED_DATA_READY Indicates that new processed data is available.
- ML_GOT_GESTURE Indicates that a gesture has been detected by the gesture engine.
- ML_MOTION_STATE_CHANGE Indicates that a change has been made from motion to no motion, or vice versa.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must have been called.

Parameters:

flag The flag to check.

Returns:

TRUE or FALSE state of the flag

5.2.2.4 tMLError MLDisableMotionDetect (void)

Disables the ML_MOTION_DETECT engine.

Note:

This function replaces MLDisable(ML_MOTION_DETECT)

Precondition:

MLDmpOpen() or MLDmpPedometerStandAlone() must have been called.

Returns:

ML_SUCCESS or non-zero error code

5.2.2.5 tMLError MLEnableMotionDetect (void)

Enables the ML_MOTION_DETECT engine.

Note:

This function replaces MLEnable(ML_MOTION_DETECT)

Precondition:

MLDmpOpen() or MLDmpPedometerStandAlone() must have been called.

Returns:

ML_SUCCESS or non-zero error code

5.2.2.6 tMLError MLGetArray (int *dataSet*, long * *data*)

MLGetArray is used to get an array of processed motion sensor data.

MLGetArray can be used to retrieve various data sets. Certain data sets require functions to be enabled using MLEnable in order to be valid.

The available data sets are:

- ML_ROTATION_MATRIX
- ML_QUATERNION
- ML_EULER_ANGLES_X
- ML_EULER_ANGLES_Y
- ML_EULER_ANGLES_Z
- ML_EULER_ANGLES
- ML_LINEAR_ACCELERATION
- ML_LINEAR_ACCELERATION_WORLD

- ML_GRAVITY
- ML_ANGULAR_VELOCITY
- ML_RAW_DATA
- ML_GYROS
- ML_ACCELS
- ML_MAGNETOMETER
- ML_GYRO_BIAS
- ML_ACCEL_BIAS
- ML_MAG_BIAS
- ML_HEADING
- ML_MAG_BIAS_ERROR
- ML_PRESSURE

Please refer to the documentation of MLGetFloatArray() for a description of these data sets.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

dataSet A constant specifying an array of data processed by the motion processor.

data A pointer to an array to be passed back to the user. **Must be 9 cells long at least**.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.2.2.7 int MLGetEngines (void)

MLGetEngines is used to determine which engines are currently enabled.

MLGetEngines returns a bitwise OR of all enabled engines. The avalaible engines are:

- SENSOR FUSION
- MOTION DETECT

- CONTROL
- ORIENTATION
- GESTURE

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Returns:

a bit mask of the engines currently enabled.

5.2.2.8 tMLError MLGetFloatArray (int *dataSet*, float * *data*)

MLGetFloatArray is used to get an array of processed motion sensor data.

MLGetArray can be used to retrieve various data sets. Certain data sets require functions to be enabled using MLEnable in order to be valid.

The available data sets are:

- ML_ROTATION_MATRIX : Returns an array of nine data points representing the rotation matrix generated from all available sensors. This requires that ML_SENSOR_FUSION be enabled. The array format will be R11, R12, R13, R21, R22, R23, R31, R32, R33, representing the matrix:

  R11 R12 R13

  R21 R22 R23

  R31 R32 R33

  Please refer to the "9-Axis Sensor Fusion Application Note" document, section 7 "Sensor Fusion Output", for details regarding rotation matrix output.

- ML_QUATERNION : Returns an array of four data points representing the quaternion generated from all available sensors. This requires that ML_SENSOR_FUSION be enabled.

- ML_EULER_ANGLES_X : Returns an array of three data points representing roll, pitch, and yaw using the X axis of the gyroscope, accelerometer, and compass as reference axis. This is typically the convention used for mobile devices where the X axis is the width of the screen, Y axis is the height, and Z the depth.

In this case roll is defined as the rotation around the X axis of the device. The euler angles convention for this output is the following:

EULER ANGLE	ROTATION AROUND
roll	X axis
pitch	Y axis
yaw	Z axis

ML_EULER_ANGLES_X corresponds to the ML_EULER_ANGLES output and is therefore the default convention.

- ML_EULER_ANGLES_Y : Returns an array of three data points representing roll, pitch, and yaw using the Y axis of the gyroscope, accelerometer, and compass as reference axis. This convention is typically used in augmented reality applications, where roll is defined as the rotation around the axis along the height of the screen of a mobile device, namely the Y axis. The euler angles convention for this output is the following:

EULER ANGLE	ROTATION AROUND
roll	Y axis
pitch	X axis
yaw	Z axis

- ML_EULER_ANGLES_Z : Returns an array of three data points representing roll, pitch, and yaw using the Z axis of the gyroscope, accelerometer, and compass as reference axis. This convention is mostly used in application involving the use of a camera, typically placed on the back of a mobile device, that is along the Z axis. In this convention roll is defined as the rotation around the Z axis. The euler angles convention for this output is the following:

EULER ANGLE	ROTATION AROUND
roll	Z axis
pitch	X axis
yaw	Y axis

- ML_EULER_ANGLES : Returns an array of three data points representing roll, pitch, and yaw corresponding to the ML_EULER_ANGLES_X output and it is therefore the default convention for Euler angles. Please refer to the ML_-EULER_ANGLES_X for a detailed description.

- ML_LINEAR_ACCELERATION : Returns an array of three data points representing the linear acceleration as derived from both gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled.

- ML_LINEAR_ACCELERATION_WORLD : Returns an array of three data points representing the linear acceleration in world coordinates, as derived from both gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled.

- ML_GRAVITY : Returns an array of three data points representing the direction of gravity in body coordinates, as derived from both gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled.

- ML_ANGULAR_VELOCITY : Returns an array of three data points representing the angular velocity as derived from **both** gyroscopes and accelerometers. This requires that ML_SENSOR_FUSION be enabled, to fuse data from the gyroscope and accelerometer device, appropriately scaled and oriented according to the respective mounting matrices.

- ML_RAW_DATA : Returns an array of nine data points representing raw sensor data of the gyroscope X, Y, Z, accelerometer X, Y, Z, and compass X, Y, Z values. These values are not scaled and come out directly from the devices' sensor data output. In case of accelerometers with lower output resolution, e.g 8-bit, the sensor data is scaled up to match the 2^14 = 1 gee typical representation for a +/- 2 gee full scale range.

- ML_GYROS : Returns an array of three data points representing the X gyroscope, Y gyroscope, and Z gyroscope values. The values are not sensor fused with other sensor types data but reflect the orientation from the mounting matrices in use. The ML_GYROS values are scaled to ensure 1 dps corresponds to 2^16 codes.

- ML_ACCELS : Returns an array of three data points representing the X accelerometer, Y accelerometer, and Z accelerometer values. The values are not sensor fused with other sensor types data but reflect the orientation from the mounting matrices in use. The ML_ACCELS values are scaled to ensure 1 gee corresponds to 2^16 codes.

- ML_MAGNETOMETER : Returns an array of three data points representing the compass X, Y, and Z values. The values are not sensor fused with other sensor types data but reflect the orientation from the mounting matrices in use. The ML_MAGNETOMETER values are scaled to ensure 1 micro Tesla (uT) corresponds to 2^16 codes.

- ML_GYRO_BIAS : Returns an array of three data points representing the gyroscope biases.

- ML_ACCEL_BIAS : Returns an array of three data points representing the accelerometer biases.

- ML_MAG_BIAS : Returns an array of three data points representing the compass biases.

- ML_GYRO_CALIBRATION_MATRIX : Returns an array of nine data points representing the calibration matrix for the gyroscopes:

  C11 C12 C13

  C21 C22 C23

  C31 C32 C33

- ML_ACCEL_CALIBRATION_MATRIX : Returns an array of nine data points representing the calibration matrix for the accelerometers:

  C11 C12 C13

  C21 C22 C23

  C31 C32 C33

- ML_MAG_CALIBRATION_MATRIX : Returns an array of nine data points representing the calibration matrix for the compass:

  C11 C12 C13

  C21 C22 C23

  C31 C32 C33

- ML_PRESSURE : Returns a single value representing the pressure in Pascal

- ML_HEADING : Returns a single number representing the heading of the device relative to the Earth, in which 0 represents North, 90 degrees represents East, and so on. The heading is defined as the direction of the +Y axis if the Y axis is horizontal, and otherwise the direction of the -Z axis.

- ML_MAG_BIAS_ERROR : Returns an array of three numbers representing the current estimated error in the compass biases. These numbers are unitless and serve as rough estimates in which numbers less than 100 typically represent reasonably well calibrated compass axes.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

dataSet A constant specifying an array of data processed by the motion processor.

data A pointer to an array to be passed back to the user. **Must be 9 cells long at least**.

Returns:

ML_SUCCESS if the command is successful; an error code otherwise.

5.2.2.9 int MLGetGyroPresent (void)

Check for the presence of the gyro sensor.

This is not a physical check but a logical check and the value can change dynamically based on calls to MLSetMPUSensors().

Returns:

TRUE if the gyro is enabled FALSE otherwise.

5.2.2.10 int MLGetInterrupts (void)

Get the current set of DMP interrupt sources.

These interrupts are generated by the DMP and can be routed to the MPU interrupt line via internal settings.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Returns:

Currently enabled interrupt sources. The possible interrupts are:

- ML_INT_FIFO,
- ML_INT_MOTION,
- ML_INT_TAP

5.2.2.11 int MLGetMotionState (void)

MLGetMotionState is used to determine if the device is in a 'motion' or 'no motion' state.

MLGetMotionState returns ML_MOTION of the device is moving, or ML_NO_-MOTION if the device is not moving.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must have been called.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.12 tMLError MLSerialClose (void)

Close the serial communication.

This function needs to be called explicitly to shut down the communication with the device. Calling MLDmpClose() won't affect the exstablished serial communication.

Returns:

ML_SUCCESS; non-zero error code otherwise.

5.2.2.13 void* MLSerialGetHandle (void)

get the serial file handle to the device.

Returns:

the serial file handle.

5.2.2.14 tMLError MLSerialOpen (char const * *port*)

Open the serial connection with the device.

This is the entry point of the MPL and must be called prior to any other function call.

Parameters:

port port the device is connected to.

Returns:

ML_SUCCESS or error code.

5.2.2.15 tMLError MLSetAccelCalibration (float *range*, signed char * *orientation*)

Sets up the Accelerometer calibration and scale factor.

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided. Section 5, "Sensor Mounting Orientation" offers a good coverage on the mounting matrices and explains how to use them.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen().
MLDmpStart() must **NOT** have been called.

See also:

MLSetGyroCalibration().
MLSetMagCalibration().

Parameters:

range The range of the accelerometers in g's. An accelerometer that has a range of +2g's to -2g's should pass in 2.

orientation A 9 element matrix that represents how the accelerometers are oriented with respect to the device they are mounted in and the reference axis system. A typical set of values are {1, 0, 0, 0, 1, 0, 0, 0, 1}. This example corresponds to a 3 x 3 identity matrix.

Returns:

ML_SUCCESS if successful; a non-zero error code otherwise.

5.2.2.16 tMLError MLSetArray (int *dataSet*, long * *data*)

used to set an array of motion sensor data.

Handles the following data sets:

- ML_GYRO_BIAS
- ML_ACCEL_BIAS
- ML_MAG_BIAS
- ML_GYRO_TEMP_SLOPE

For more details about the use of the data sets please refer to the documentation of MLSetFloatArray().

Please also refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

dataSet A constant specifying an array of data.

data A pointer to an array to be copied from the user.

Returns:

ML_SUCCESS if successful; a non-zero error code otherwise.

5.2.2.17 tMLError MLSetBiasUpdateFunc (unsigned short *function*)

MLSetBiasUpdateFunc is used to register which algorithms will be used to automatically reset the gyroscope bias.

The engine ML_BIAS_UPDATE must be enabled for these algorithms to run.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must **NOT** have been called.

Parameters:

function A function or bitwise OR of functions that determine how the gyroscope bias will be automatically updated. Functions include:

- ML_NONE or 0,
- ML_BIAS_FROM_NO_MOTION,
- ML_BIAS_FROM_GRAVITY,
- ML_BIAS_FROM_TEMPERATURE,
- ML_BIAS_FROM_LPF,
- ML_MAG_BIAS_FROM_MOTION,
- ML_MAG_BIAS_FROM_GYRO,
- ML_ALL.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.18 tMLError MLSetFifoInterrupt (unsigned char *on*)

Enable generation of the DMP interrupt when a FIFO packet is ready.

Parameters:

on Boolean to turn the interrupt on or off

Returns:

ML_SUCCESS or non-zero error code

5.2.2.19 tMLError MLSetFloatArray (int *dataSet*, float * *data*)

used to set an array of motion sensor data.

Handles various data sets:

- ML_GYRO_BIAS
- ML_ACCEL_BIAS
- ML_MAG_BIAS
- ML_GYRO_TEMP_SLOPE

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen()
MLDmpStart() must **NOT** have been called.

Parameters:

dataSet A constant specifying an array of data.

data A pointer to an array to be copied from the user.

Returns:

ML_SUCCESS if successful; a non-zero error code otherwise.

5.2.2.20 tMLError MLSetGyroCalibration (float *range*, signed char * *orientation*)

Sets up the Gyro calibration and scale factor.

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided. Section 5, "Sensor Mounting Orientation" offers a good coverage on the mounting matrices and explains how to use them.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen().
MLDmpStart() must have **NOT** been called.

See also:

MLSetAccelCalibration().
MLSetMagCalibration().

Parameters:

range The range of the gyros in degrees per second. A gyro that has a range of +2000 dps to -2000 dps should pass in 2000.

orientation A 9 element matrix that represents how the gyro are oriented with respect to the device they are mounted in. A typical set of values are {1, 0, 0, 0, 1, 0, 0, 0, 1}. This example corresponds to a 3 x 3 identity matrix.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.21 tMLError MLSetMagCalibration (float *range*, signed char * *orientation*)

Sets up the Compass calibration and scale factor.

Please refer to the provided "9-Axis Sensor Fusion Application Note" document provided. Section 5, "Sensor Mounting Orientation" offers a good coverage on the mounting matrices and explains how to use them.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen().
MLDmpStart() must have **NOT** been called.

See also:

MLSetGyroCalibration().
MLSetAccelCalibration().

Parameters:

range The range of the compass.

orientation A 9 element matrix that represents how the compass is oriented with respect to the device they are mounted in. A typical set of values are {1, 0, 0, 0, 1, 0, 0, 0, 1}. This example corresponds to a 3 x 3 identity matrix. The matrix describes how to go from the chip mounting to the body of the device.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.22 tMLError MLSetMotionCallback (void(∗)(unsigned short motionState) *func*)

MLSetMotionCallback is used to register a callback function that will trigger when a change of motion state is detected.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must **NOT** have been called.

Parameters:

func A user defined callback function accepting a motionState parameter, the new motion state. May be one of ML_MOTION or ML_NO_MOTION.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.23 tMLError MLSetMotionInterrupt (unsigned char *on*)

Enable generation of the DMP interrupt when Motion or no-motion is detected.

Parameters:

on Boolean to turn the interrupt on or off.

Returns:

ML_SUCCESS or non-zero error code.

5.2.2.24 tMLError MLSetMPUSensors (unsigned long *sensors*)

Controlls each sensor and each axis when the motion processing unit is running.

When it is not running, simply records the state for later.

NOTE: In this version only full sensors controll is allowed. Independent axis control will return an error.

Parameters:

sensors Bit field of each axis desired to be turned on or off

Returns:

ML_SUCCESS or non-zero error code

5.2.2.25 tMLError MLSetNoMotionThresh (float *thresh*)

MLSetNoMotionThresh is used to set the threshold for detecting ML_NO_MOTION.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

thresh A threshold scaled in degrees per second.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.26 tMLError MLSetNoMotionThreshAccel (long *thresh*)

MLSetNoMotionThreshAccel is used to set the threshold for detecting ML_NO_-MOTION with accelerometers when Gyros have been turned off.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

thresh A threshold in g's scaled by 2^32

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.27 tMLError MLSetNoMotionTime (float *time*)

MLSetNoMotionTime is used to set the time required for detecting ML_NO_-MOTION.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() must have been called.

Parameters:

time A time in seconds.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.2.2.28 tMLError MLUpdateData (void)

MLUpdateData updates all the realtime data from the motion algorithms.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must have been called.

Note:

Functions like MLGetArray will return the same data if MLUpdateData is not called.

Returns:

- ML_SUCCESS
- Non-zero error code

5.2.2.29 tMLError MLVersion (unsigned char ** *version*)

MLVersion is used to get the ML version.

Precondition:

MLVersion can be called at any time.

Parameters:

version MLVersion writes the ML version string pointer to version.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.3 MLFIFO

Motion Library - FIFO Driver.

Files

- file mlFIFO.c

  FIFO Interface.

Functions

- tMLError FIFOClose (void)

  Close the FIFO usage.

- tMLError FIFOGetAccel (long *data)

  Returns 3-element vector of accelerometer data in body frame.

- tMLError FIFOGetAccelFloat (float *data)

  Returns 3-element vector of accelerometer data in body frame.

- tMLError FIFOGetControlData (long *data)

  Returns 4-element vector of control data.

- tMLError FIFOGetDecodedAccel (long *data)

  Get the Decoded Accel Data.

- tMLError FIFOGetEis (long *data)

  Returns 3-element vector of EIS shfit data.

- tMLError FIFOGetExternalSensorData (long *data)

  Returns 3-element vector of external sensor.

- tMLError FIFOGetGravBody (long *data)

  Get the 3-element gravity vector from the FIFO expressed in coordinates relative to the body frame.

- tMLError FIFOGetGyro (long *data)

  Returns 3-element vector of gyro data in body frame.

- tMLError FIFOGetLinearAccel (long *data)

  Returns 3-element vector of accelerometer data in body frame with gravity removed.

- tMLError FIFOGetLinearAccelWorld (long *data)

  Returns 3-element vector of accelerometer data in world frame with gravity removed.

- tMLError FIFOGetQuantAccel (long *data)

  Get the Quantized Accel data algorithm output from the FIFO.

- tMLError FIFOGetQuaternion (long *data)

  Returns 4-element quaternion vector derived from 6-axis or 9-axis if 9-axis was implemented.

- tMLError FIFOGetQuaternion6Axis (long *data)

  Returns 4-element quaternion vector derived from 6 axis sensors (gyros and accels).

- tMLError FIFOGetQuaternionFloat (float *data)

  Returns 4-element quaternion vector.

- tMLError FIFOGetSensorData (long *data)

  Returns 6-element vector of gyro and accel data.

- tMLError FIFOGetSensorGyroData (long *data)

  This gets raw gyro data.

- tMLError FIFOGetTemperature (long *data)

  Returns 1-element vector of temperature.

- tMLError FIFOParamInit (void)

  Initializes all the internal static variables for the FIFO module.

- tMLError FIFOSendAccel (uint_fast16_t elements, uint_fast16_t accuracy)

  Sends accelerometer data to the FIFO.

- tMLError FIFOSendControlData (uint_fast16_t elements, uint_fast16_t accuracy)

  Sends control data to the FIFO.

- tMLError FIFOSendDMPPacketNumber (uint_fast16_t accuracy)

  Adds a rolling counter to the fifo packet.

- tMLError FIFOSendGravity (uint_fast16_t elements, uint_fast16_t accuracy)

  Send the computed gravity vectors into the FIFO.

- tMLError FIFOSendGyro (uint_fast16_t elements, uint_fast16_t accuracy)

Sends gyro data to the FIFO.

- tMLError FIFOSendLinearAccel (uint_fast16_t elements, uint_fast16_t accuracy)

  Sends linear accelerometer data to the FIFO.

- tMLError FIFOSendLinearAccelWorld (uint_fast16_t elements, uint_fast16_t accuracy)

  Sends linear world accelerometer data to the FIFO.

- tMLError FIFOSendQuantAccel (uint_fast16_t elements, uint_fast16_t accuracy)

  Send the Quantized Accelerometer data into the FIFO.

- tMLError FIFOSendQuaternion (uint_fast16_t accuracy)

  Sends quaternion data to the FIFO.

- tMLError FIFOSendRaw (uint_fast16_t elements, uint_fast16_t accuracy)

  Sends raw data to the FIFO.

- tMLError FIFOSendRawExternal (uint_fast16_t elements, uint_fast16_t accuracy)

  Sends raw external data to the FIFO.

- tMLError FIFOSetGyroDataSource (uint_fast8_t source)

  Set the gyro source to output to the fifo.

- tMLError FIFOSetLinearAccelFilterCoef (float coef)

  Sets the filter coefficent used for computing the acceleration bias which is used to compute linear acceleration.

- unsigned long getAccMagSqrd (void)

  The gyro data magnitude squared: (1 g)^2 = 2^16 = 2^ACC_MAG_SQR_SHIFT.

- unsigned long getGyroMagSqrd (void)

  The gyro data magnitude squared : (1 degree per second)^2 = 2^6 = 2^GYRO_MAG_SQR_SHIFT.

- int_fast16_t GetSampleFrequencyHz (void)

  Returns the step size for quaternion type data.

- int_fast16_t GetSampleStepSizeMs (void)

  Returns the step size for quaternion type data.

- unsigned short MLGetFIFORate (void)

  Retrieve the current FIFO update divider - 1.

- tMLError MLSetFIFORate (unsigned short fifoRate)

  Command the MPU to put data in the FIFO at a particular rate.

- tMLError MLSetProcessedDataCallback (void(∗func)(void))

  MLSetProcessedDataCallback is used to set a processed data callback function.

- tMLError readAndProcessFIFO (int_fast8_t numPackets, int_fast8_t ∗processed)

  Reads and processes FIFO data.

- void SetSampleStepSizeMs (int_fast16_t ms)

  Sets the step size when the FIFO is not used Typically set when using the accelerometer without the DMP.

5.3.1 Detailed Description

Motion Library - FIFO Driver.

The FIFO API Interface.

5.3.2 Function Documentation

5.3.2.1 tMLError FIFOClose (void)

Close the FIFO usage.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.3.2.2 tMLError FIFOGetAccel (long ∗ *data*)

Returns 3-element vector of accelerometer data in body frame.

Parameters:

data 3-element vector of accelerometer data in body frame. One gee = 2^16.

Returns:

0 on success or an error code.

5.3.2.3 tMLError FIFOGetAccelFloat (float * *data*)

Returns 3-element vector of accelerometer data in body frame.

Parameters:

data 3-element vector of accelerometer data in body frame in g's.

Returns:

0 for success or an error code.

5.3.2.4 tMLError FIFOGetControlData (long * *data*)

Returns 4-element vector of control data.

Parameters:

data 4-element vector of control data.

Returns:

0 for succes or an error code.

5.3.2.5 tMLError FIFOGetDecodedAccel (long * *data*)

Get the Decoded Accel Data.

Parameters:

data a buffer to store the quantized data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.3.2.6 tMLError FIFOGetEis (long * *data*)

Returns 3-element vector of EIS shfit data.

Parameters:

data 3-element vector of EIS shift data.

Returns:

0 for succes or an error code.

5.3.2.7 tMLError FIFOGetExternalSensorData (long * *data*)

Returns 3-element vector of external sensor.

Parameters:

data 3-element vector of external sensor

Returns:

0 on success or an error code.

5.3.2.8 tMLError FIFOGetGravBody (long * *data*)

Get the 3-element gravity vector from the FIFO expressed in coordinates relative to the body frame.

Parameters:

data 3-element vector of gravity in body frame.

Returns:

0 on success or an error code.

5.3.2.9 tMLError FIFOGetGyro (long * *data*)

Returns 3-element vector of gyro data in body frame.

Parameters:

data 3-element vector of gyro data in body frame with gravity removed. One degree per second = 2^16.

Returns:

0 on success or an error code.

5.3.2.10 tMLError FIFOGetLinearAccel (long * *data*)

Returns 3-element vector of accelerometer data in body frame with gravity removed.

Parameters:

data 3-element vector of accelerometer data in body frame with gravity removed. One g = 2^16.

Returns:

0 on success or an error code. data unchanged on error.

5.3.2.11 tMLError FIFOGetLinearAccelWorld (long * *data*)

Returns 3-element vector of accelerometer data in world frame with gravity removed.

Parameters:

data 3-element vector of accelerometer data in world frame with gravity removed. One g = 2^16.

Returns:

0 on success or an error code.

5.3.2.12 tMLError FIFOGetQuantAccel (long * *data*)

Get the Quantized Accel data algorithm output from the FIFO.

Parameters:

data a buffer to store the quantized data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.3.2.13 tMLError FIFOGetQuaternion (long * *data*)

Returns 4-element quaternion vector derived from 6-axis or 9-axis if 9-axis was implemented.

6-axis is gyros and accels. 9-axis is gyros, accel and compass.

Parameters:

data 4-element quaternion vector. One is scaled to 2^30.

Returns:

0 on success or an error code.

5.3.2.14 tMLError FIFOGetQuaternion6Axis (long * *data*)

Returns 4-element quaternion vector derived from 6 axis sensors (gyros and accels).

Parameters:

data 4-element quaternion vector. One is scaled to 2^30.

Returns:

0 on success or an error code.

5.3.2.15 tMLError FIFOGetQuaternionFloat (float * *data*)

Returns 4-element quaternion vector.

Parameters:

data 4-element quaternion vector.

Returns:

0 on success, an error code otherwise.

5.3.2.16 tMLError FIFOGetSensorData (long * *data*)

Returns 6-element vector of gyro and accel data.

Parameters:

data 6-element vector of gyro and accel data

Returns:

0 on success or an error code.

5.3.2.17 tMLError FIFOGetSensorGyroData (long * *data*)

This gets raw gyro data.

The data is taken from the FIFO if it was put in the FIFO and it is read from the registers if it was not put into the FIFO. The data is cached till the next FIFO processing block time.

Parameters:

data Length 3, Gyro data

5.3.2.18 tMLError FIFOGetTemperature (long * *data*)

Returns 1-element vector of temperature.

It is read from the hardware if it doesn't exist in the FIFO.

Parameters:

data 1-element vector of temperature

Returns:

0 on success or an error code.

5.3.2.19 tMLError FIFOParamInit (void)

Initializes all the internal static variables for the FIFO module.

Note:

Should be called by the initialization routine such as MLDmpOpen().

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.3.2.20 tMLError FIFOSendAccel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends accelerometer data to the FIFO.

Parameters:

elements Which of the 3 elements to send. Use ML_ALL for 3 axis or ML_ELEMENT_1, ML_ELEMENT_2, ML_ELEMENT_3 or'd together for a subset.

accuracy Set to ML_32_BIT for 32-bit data, or ML_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.3.2.21 tMLError FIFOSendControlData (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends control data to the FIFO.

Control data is a 4 length vector of 32-bits.

Parameters:

elements Which of the 4 elements to send. Use ML_ALL for all or ML_ELEMENT_1, ML_ELEMENT_2, ML_ELEMENT_3, ML_ELEMENT_4 or'd together for a subset.

accuracy Set to ML_32_BIT for 32-bit data, or ML_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.3.2.22 tMLError FIFOSendDMPPacketNumber (uint_fast16_t *accuracy*)

Adds a rolling counter to the fifo packet.

When used with the footer the data comes out the first time:

```
<data0><data1>...<dataN><PacketNum0><PacketNum1>
```

for every other packet it is

```
<FifoFooter0><FifoFooter1><data0><data1>...<dataN><PacketNum0><PacketNum1>
```

This allows for scanning of the fifo for packets

Returns:

ML_SUCCESS or error code

5.3.2.23 tMLError FIFOSendGravity (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Send the computed gravity vectors into the FIFO.

The gravity vectors can be retrieved from the FIFO via FIFOGetGravBody(), to have the gravitation vector expressed in coordinates relative to the body.

Gravity is a derived vector derived from the quaternion.

Parameters:

elements the gravitation vectors components bitmask. To send all compoents use ML_ALL.

accuracy The number of bits the gravitation vector is expressed into. Set to ML_-32_BIT for 32-bit data, or ML_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.3.2.24 tMLError FIFOSendGyro (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends gyro data to the FIFO.

Gyro data is a 3 length vector of 32-bits. Should be called once after MLDmpOpen() and before MLDmpStart().

Parameters:

elements Which of the 3 elements to send. Use ML_ALL for all of them or ML_-ELEMENT_1, ML_ELEMENT_2, ML_ELEMENT_3 or'd together for a subset.

accuracy Set to ML_32_BIT for 32-bit data, or ML_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.3.2.25 tMLError FIFOSendLinearAccel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends linear accelerometer data to the FIFO.

Linear accelerometer data is a 3 length vector of 32-bits. It is the acceleration in the body frame with gravity removed.

Parameters:

elements Which of the 3 elements to send. Use ML_ALL for all of them or ML_-ELEMENT_1, ML_ELEMENT_2, ML_ELEMENT_3 or'd together for a subset.

NOTE: Elements is ignored if the fifo rate is < ML_MAX_NUM_ACCEL_-SAMPLES

Parameters:

accuracy Set to ML_32_BIT for 32-bit data, or ML_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

5.3.2.26 tMLError FIFOSendLinearAccelWorld (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends linear world accelerometer data to the FIFO.

Linear world accelerometer data is a 3 length vector of 32-bits. It is the acceleration in the world frame with gravity removed. Should be called once after MLDmpOpen() and before MLDmpStart().

Parameters:

elements Which of the 3 elements to send. Use ML_ALL for all of them or ML_ELEMENT_1, ML_ELEMENT_2, ML_ELEMENT_3 or'd together for a subset.

accuracy Set to ML_32_BIT for 32-bit data, or ML_16_BIT for 16 bit data.

5.3.2.27 tMLError FIFOSendQuantAccel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Send the Quantized Acceleromter data into the FIFO.

The data can be retrieved using FIFOGetQuantAccel() or FIFOGetDecodedAccel().

To be useful this should be set to mlFIFORate + 1 if less than ML_MAX_NUM_ACCEL_SAMPLES, otherwise it doesn't work.

Parameters:

elements the components bitmask. To send all compoents use ML_ALL.

accuracy Use ML_32_BIT for 32-bit data or ML_16_BIT for 16-bit data. Set to zero to remove it from the FIFO.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.3.2.28 tMLError FIFOSendQuaternion (uint_fast16_t *accuracy*)

Sends quaternion data to the FIFO.

Quaternion data is a 4 length vector of 32-bits. Should be called once after MLDmpOpen() and before MLDmpStart().

Parameters:

accuracy Set to ML_32_BIT for 32-bit data, or ML_16_BIT for 16 bit data.

5.3.2.29 tMLError FIFOSendRaw (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends raw data to the FIFO.

Should be called once after MLDmpOpen() and before MLDmpStart().

Parameters:

elements Which of the 7 elements to send. Use ML_ALL for all of them or ML_ELEMENT_1, ML_ELEMENT_2, ML_ELEMENT_3 ... ML_ELEMENT_7 or'd together for a subset. The first element is temperature, the next 3 are gyro data, and the last 3 accel data.

accuracy The element's accuracy, can be ML_16_BIT, ML_32_BIT, or 0 to turn off.

Returns:

0 if successful, a non-zero error code otherwise.

5.3.2.30 tMLError FIFOSendRawExternal (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends raw external data to the FIFO.

Should be called once after MLDmpOpen() and before MLDmpStart().

Parameters:

elements Which of the 3 elements to send. Use ML_ALL for all of them or ML_ELEMENT_1, ML_ELEMENT_2, ML_ELEMENT_3 or'd together for a subset.

accuracy ML_16_BIT to send data, 0 to stop sending this data. Sending and Stop sending are reference counted, so data actually stops when the reference reaches zero.

5.3.2.31 tMLError FIFOSetGyroDataSource (uint_fast8_t *source*)

Set the gyro source to output to the fifo.

Parameters:

source The source. One of
- ML_GYRO_FROM_RAW
- ML_GYRO_FROM_QUATERNION

Returns:

ML_SUCCESS or non-zero error code;

5.3.2.32 tMLError FIFOSetLinearAccelFilterCoef (float *coef*)

Sets the filter coefficent used for computing the acceleration bias which is used to compute linear acceleration.

Parameters:

coef Fitler coefficient. 0. means no filter, a small number means a small cut-off frequency with an increasing number meaning an increasing cutoff frequency.

5.3.2.33 unsigned long getAccMagSqrd (void)

The gyro data magnitude squared: (1 g)^2 = 2^16 = 2^ACC_MAG_SQR_SHIFT.

Returns:

the computed magnitude squared output of the accelerometer.

5.3.2.34 unsigned long getGyroMagSqrd (void)

The gyro data magnitude squared : (1 degree per second)^2 = 2^6 = 2^GYRO_MAG_-SQR_SHIFT.

Returns:

the computed magnitude squared output of the gyroscope.

5.3.2.35 int_fast16_t GetSampleFrequencyHz (void)

Returns the step size for quaternion type data.

Typically the data rate for each FIFO packet. When the gryos are sleeping this value will return the last value set by SetSampleStepSizeMs()

Returns:

step size for quaternion type data

5.3.2.36 int_fast16_t GetSampleStepSizeMs (void)

Returns the step size for quaternion type data.

Typically the data rate for each FIFO packet. When the gryos are sleeping this value will return the last value set by SetSampleStepSizeMs()

Returns:

step size for quaternion type data

5.3.2.37 unsigned short MLGetFIFORate (void)

Retrieve the current FIFO update divider - 1.

See MLSetFIFORate() for how this value is used.

The fifo rate when there is no fifo is the equivilent divider when derived from the value set by SetSampleSteSizeMs()

Returns:

The value of the fifo rate divider or ML_INVALID_FIFO_RATE on error.

5.3.2.38 tMLError MLSetFIFORate (unsigned short *fifoRate*)

Command the MPU to put data in the FIFO at a particular rate.

The DMP will add fifo entries every fifoRate + 1 MPU cycles. For example if the MPU is running at 200Hz the following values apply:

fifoRate	DMP Sample Rate	FIFO update frequency
0	200Hz	200Hz
1	200Hz	100Hz
2	200Hz	50Hz
4	200Hz	40Hz
9	200Hz	20Hz
19	200Hz	10Hz

Note: if the DMP is running, (state == ML_STATE_DMP_STARTED) then MLStateRunCallbacks() will be called to allow features that depend upon fundamental constants to be updated.

Precondition:

MLDmpOpen() with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must **NOT** have been called.

Parameters:

fifoRate Divider value - 1. Output rate is (DMP Sample Rate) / (fifoRate + 1).

Returns:

ML_SUCCESS if successful, ML error code on any failure.

5.3.2.39 tMLError MLSetProcessedDataCallback (void(∗)(void) *func*)

MLSetProcessedDataCallback is used to set a processed data callback function.

MLSetProcessedDataCallback sets a user defined callback function that triggers when all the decoding has been finished by the motion processing engines. It is called before other bigger processing engines to allow lower latency for the user.

Precondition:

MLDmpOpen() with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must **NOT** have been called.

Parameters:

func A user defined callback function.

Returns:

ML_SUCCESS if successful, or non-zero error code otherwise.

5.3.2.40 tMLError readAndProcessFIFO (int_fast8_t *numPackets*, int_fast8_t ∗ *processed*)

Reads and processes FIFO data.

Also handles callbacks when data is ready.

Parameters:

numPackets Number of FIFO packets to try to read. You should use a large number here, such as 100, if you want to read all the full packets in the FIFO, which is typical operation.

processed The number of FIFO packets processed. This may be incremented even if high rate processes later fail.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.3.2.41 void SetSampleStepSizeMs (int_fast16_t *ms*)

Sets the step size when the FIFO is not used Typically set when using the accelerometer without the DMP.

Returns:

step size for quaternion type data

5.4 MLFIFO_HW

Motion Library - FIFO HW Driver.

Files

- file mlFIFOHW.c
  The Motion Library Fifo Hardware Layer.

Functions

- void FIFOHWInit (void)
  Initializes the internal FIFO data structure.
- tMLError FIFOReset (void)
  Clears the FIFO status and its content.
- short MLDLGetFIFOCount (void)
  MLDLGetFIFOCount is used to get the number of bytes left in the FIFO.
- tMLError MLDLGetFIFOStatus (void)
  Used to query the status of the FIFO.

5.4.1 Detailed Description

Motion Library - FIFO HW Driver.

Provides facilities to interact with the FIFO.

5.4.2 Function Documentation

5.4.2.1 tMLError FIFOReset (void)

Clears the FIFO status and its content.

Note:

Halt the DMP writing into the FIFO for the time needed to reset the FIFO.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.4.2.2 short MLDLGetFIFOCount (void)

MLDLGetFIFOCount is used to get the number of bytes left in the FIFO.

This function returns the stored value and does not access the hardware. See MLDLGetFifoLength().

Returns:

the number of bytes left in the FIFO

5.4.2.3 tMLError MLDLGetFIFOStatus (void)

Used to query the status of the FIFO.

Returns:

ML_SUCCESS if the fifo is OK. An error code otherwise.

5.5 ML_SUPERVISOR

Basic sensor fusion supervisor functionalities.

Files

- file mlsupervisor.c

  Basic sensor fusion supervisor functionalities.

Functions

- tMLError MLAccelCompassSupervisor (void)

  Entry point for software sensor fusion operations.

- tMLError MLPressureSupervisor (void)

  Entry point for software sensor fusion operations.

- tMLError MLResetMagCalibration (void)

  Resets the magnetometer calibration algorithm.

- void MLSensorFusionSupervisorInit (void)

  This initializes all variables that should be reset on.

5.5.1 Detailed Description

Basic sensor fusion supervisor functionalities.

5.5.2 Function Documentation

5.5.2.1 tMLError MLAccelCompassSupervisor (void)

Entry point for software sensor fusion operations.

Manages hardware interaction, calls sensor fusion supervisor for bias calculation.

Returns:

error code. ML_SUCCESS if no error occurred.

5.5.2.2 tMLError MLPressureSupervisor (void)

Entry point for software sensor fusion operations.

Manages hardware interaction, calls sensor fusion supervisor for bias calculation.

Returns:

ML_SUCCESS or non-zero error code on error.

5.5.2.3 tMLError MLResetMagCalibration (void)

Resets the magnetometer calibration algorithm.

Returns:

ML_SUCCESS if successful, or non-zero error code otherwise.

5.6 MLDL

Motion Library - Driver Layer.

Files

- file mldl.c
  The Motion Library Driver Layer.
- file mldl_cfg.c
  The Motion Library Driver Layer.
- file mldl_cfg_mpu.c
  The Motion Library Driver Layer.

Functions

- tMLError MLDLCfgInt (unsigned char triggers)
  MLDLCfgInt configures the interrupt function on the specified pin.
- tMLError MLDLCfgSamplingMPU (unsigned char lpf, unsigned char divider)
  configures the output sampling rate on the MPU.
- void MLDLClearIntTrigger (unsigned char srcIndex)
  clear the 'triggered' status for an interrupt source.
- tMLError MLDLClockSource (unsigned char clkSource)
  MLDLClockSource function sets the clock source for the MPU gyro processing.
- tMLError MLDLClose (void)
  Closes/Cleans up the ML Driver Layer.
- tMLError MLDLDmpStart (unsigned long sensors)
  Starts the DMP running.
- tMLError MLDLDmpStop (unsigned long sensors)
  Stops the DMP running and puts it in low power as requested.
- struct mldl_cfg * MLDLGetCfg (void)
  Get a pointer to the internal data structure storing the configuration for the MPU, the accelerometer and the compass in use.

- tMLError MLDLGetIntStatus (unsigned char intPin, unsigned char *status)

  MLDLGetIntStatus returns the interrupt status from the specified interrupt pin.

- unsigned char MLDLGetIntTrigger (unsigned char srcIndex)

  query the current status of an interrupt source.

- unsigned char MLDLGetMPUSlaveAddr (void)

  Query the MPU slave address.

- tMLError MLDLIntHandler (unsigned char intSource)

  MLDLIntHandler function should be called when an interrupt is received.

- tMLError MLDLLoadDMP (const unsigned char *buffer, unsigned short length, unsigned short config)

  Load the DMP with the given code and configuration.

- tMLError MLDLOpen (void *mlslHandle)

  Open the driver layer and resets the internal gyroscope, accelerometer, and compass data structures.

- tMLError MLDLSetExternalSyncMPU (unsigned char extSync)

  This function sets the external sync for the MPU sampling.

- tMLError MLDLSetFullScaleMPU (float fullScale)

  set the full scale range for the gyros.

- tMLError MLDLSetGyroPower (unsigned long xOn, unsigned long yOn, unsigned long zOn)

  Sets the power state for the Gyro's.

- tMLError MLDLSetOffset (unsigned short const *offset)

  Set the gyro offset.

- tMLError MLDLSetOffsetTC (unsigned char const *tc)

  Set the Temperature Compensation offset.

- int mpu3050_close (struct mldl_cfg *mldl_cfg, void *mlsl_handle, void *accel_handle, void *compass_handle, void *pressure_handle)

  Close the mpu3050 interface.

- int mpu3050_config_accel (struct mldl_cfg *mldl_cfg, void *accel_handle, struct ext_slave_config *data)

Request slave to change configuration.

- int mpu3050_config_compass (struct mldl_cfg *mldl_cfg, void *compass_handle, struct ext_slave_config *data)

  Request slave to change configuration.

- int mpu3050_config_pressure (struct mldl_cfg *mldl_cfg, void *pressure_handle, struct ext_slave_config *data)

  Request slave to change configuration.

- int mpu3050_get_config_accel (struct mldl_cfg *mldl_cfg, void *accel_handle, struct ext_slave_config *data)

  Request slave configuration information.

- int mpu3050_get_config_compass (struct mldl_cfg *mldl_cfg, void *compass_handle, struct ext_slave_config *data)

  Request slave configuration information.

- int mpu3050_get_config_pressure (struct mldl_cfg *mldl_cfg, void *pressure_handle, struct ext_slave_config *data)

  Request slave configuration information.

- int mpu3050_open (struct mldl_cfg *mldl_cfg, void *mlsl_handle, void *accel_handle, void *compass_handle, void *pressure_handle)

  Initializes the pdata structure to defaults.

- int mpu3050_read_accel (struct mldl_cfg *mldl_cfg, void *accel_handle, unsigned char *data)

  read raw sensor data from the accelerometer device in use.

- int mpu3050_read_compass (struct mldl_cfg *mldl_cfg, void *compass_handle, unsigned char *data)

  read raw sensor data from the compass device in use.

- int mpu3050_read_pressure (struct mldl_cfg *mldl_cfg, void *pressure_handle, unsigned char *data)

  read raw sensor data from the pressure device in use.

- int mpu3050_resume (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *accel_handle, void *compass_handle, void *pressure_handle, bool resume_gyro, bool resume_accel, bool resume_compass, bool resume_pressure)

  resume the MPU3050 device and all the other sensor devices from their low power state.

- int mpu3050_suspend (struct mldl_cfg *mldl_cfg, void *gyro_handle, void *accel_handle, void *compass_handle, void *pressure_handle, bool suspend_gyro, bool suspend_accel, bool suspend_compass, bool suspend_pressure)

  suspend the MPU3050 device and all the other sensor devices into their low power state.

5.6.1 Detailed Description

Motion Library - Driver Layer.

The Motion Library Driver Layer provides the intrface to the system drivers that are used by the Motion Library.

5.6.2 Function Documentation

5.6.2.1 tMLError MLDLCfgInt (unsigned char *triggers*)

MLDLCfgInt configures the interrupt function on the specified pin.

The basic interrupt signal characteristics can be set (i.e. active high/low, open drain/-push pull, etc.) and the triggers can be set. Currently only INTPIN_MPU is supported.

Parameters:

triggers bitmask of triggers to enable for interrupt. The available triggers are:

- BIT_MPU_RDY_EN
- BIT_DMP_INT_EN
- BIT_RAW_RDY_EN

Returns:

Zero if the command is successful, an error code otherwise.

5.6.2.2 tMLError MLDLCfgSamplingMPU (unsigned char *lpf*, unsigned char *divider*)

configures the output sampling rate on the MPU.

Three parameters control the sampling:

1) Low pass filter bandwidth, and 2) output sampling divider.

The output sampling rate is determined by the divider and the low pass filter setting. If the low pass filter is set to 'MPUFILTER_256HZ_NOLPF2', then the sample rate

going into the divider is 8kHz; for all other settings it is 1kHz. The 8-bit divider will divide this frequency to get the resulting sample frequency. For example, if the filter setting is not 256Hz and the divider is set to 7, then the sample rate is as follows: sample rate = internal sample rate / div = 1kHz / 8 = 125Hz (or 8ms).

The low pass filter selection codes control both the cutoff frequency of the internal low pass filter and internal analog sampling rate. The latter, in turn, affects the final output sampling rate according to the sample rate divider settig. 0 -> 256 Hz cutoff BW, 8 kHz analog sample rate, 1 -> 188 Hz cutoff BW, 1 kHz analog sample rate, 2 -> 98 Hz cutoff BW, 1 kHz analog sample rate, 3 -> 42 Hz cutoff BW, 1 kHz analog sample rate, 4 -> 20 Hz cutoff BW, 1 kHz analog sample rate, 5 -> 10 Hz cutoff BW, 1 kHz analog sample rate, 6 -> 5 Hz cutoff BW, 1 kHz analog sample rate, 7 -> 2.1 kHz cutoff BW, 8 kHz analog sample rate.

Parameters:

lpf low pass filter, 0 to 7.

divider Output sampling rate divider, 0 to 255.

Returns:

ML_SUCESS if successful; a non-zero error code otherwise.

5.6.2.3 void MLDLClearIntTrigger (unsigned char *srcIndex*)

clear the 'triggered' status for an interrupt source.

Parameters:

srcIndex index of the interrupt source. Currently only INTPIN_MPU is supported.

5.6.2.4 tMLError MLDLClockSource (unsigned char *clkSource*)

MLDLClockSource function sets the clock source for the MPU gyro processing.

The source can be any of the following:

- Internal 8MHz oscillator,
- PLL with X gyro as reference,
- PLL with Y gyro as reference,
- PLL with Z gyro as reference,

- PLL with external 32.768Mhz reference, or
- PLL with external 19.2MHz reference

For best accuracy and timing, it is highly recommended to use one of the gyros as the clock source; however this gyro must be enabled to use its clock (see 'MLDLPowerMgmtMPU()').

Parameters:

clkSource Clock source selection. Can be one of:

- CLK_INTERNAL,
- CLK_PLLGYROX,
- CLK_PLLGYROY,
- CLK_PLLGYROZ,
- CLK_PLLEXT32K, or
- CLK_PLLEXT19M.

Returns:

Zero if the command is successful; an error code otherwise.

5.6.2.5 tMLError MLDLClose (void)

Closes/Cleans up the ML Driver Layer.

Put the device in sleep mode.

Returns:

ML_SUCCESS or non-zero error code.

5.6.2.6 tMLError MLDLDmpStart (unsigned long *sensors*)

Starts the DMP running.

Resumes the sensor if any of the sensor axis or components are requested

Parameters:

sensors Bitfield of the sensors to turn on. Combination of the following:

- ML_X_GYRO
- ML_Y_GYRO
- ML_Z_GYRO

- ML_DMP_PROCESSOR
- ML_X_ACCEL
- ML_Y_ACCEL
- ML_Z_ACCEL
- ML_X_COMPASS
- ML_Y_COMPASS
- ML_Z_COMPASS
- ML_X_PRESSURE
- ML_Y_PRESSURE
- ML_Z_PRESSURE
- ML_THREE_AXIS_GYRO
- ML_THREE_AXIS_ACCEL
- ML_THREE_AXIS_COMPASS
- ML_THREE_AXIS_PRESSURE

Returns:

ML_SUCCESS or non-zero error code

5.6.2.7 tMLError MLDLDmpStop (unsigned long *sensors*)

Stops the DMP running and puts it in low power as requested.

Suspends each sensor according to the bitfield, if all axis and components of the sensor is off.

Parameters:

sensors Bitfiled of the sensors to leave on. Combination of the following:

- ML_X_GYRO
- ML_Y_GYRO
- ML_Z_GYRO
- ML_X_ACCEL
- ML_Y_ACCEL
- ML_Z_ACCEL
- ML_X_COMPASS
- ML_Y_COMPASS
- ML_Z_COMPASS
- ML_X_PRESSURE
- ML_Y_PRESSURE
- ML_Z_PRESSURE
- ML_THREE_AXIS_GYRO

- ML_THREE_AXIS_ACCEL
- ML_THREE_AXIS_COMPASS
- ML_THREE_AXIS_PRESSURE

Returns:

ML_SUCCESS or non-zero error code

5.6.2.8 struct mldl_cfg* MLDLGetCfg (void) [read]

Get a pointer to the internal data structure storing the configuration for the MPU, the accelerometer and the compass in use.

Returns:

a pointer to the data structure of type 'struct mldl_cfg'.

5.6.2.9 tMLError MLDLGetIntStatus (unsigned char *intPin*, unsigned char * *status*)

MLDLGetIntStatus returns the interrupt status from the specified interrupt pin.

Parameters:

intPin Currently only the value INTPIN_MPU is supported.

status The available statuses are:

- BIT_MPU_RDY_EN
- BIT_DMP_INT_EN
- BIT_RAW_RDY_EN

Returns:

ML_SUCCESS or a non-zero error code.

5.6.2.10 unsigned char MLDLGetIntTrigger (unsigned char *srcIndex*)

query the current status of an interrupt source.

Parameters:

srcIndex index of the interrupt source. Currently the only source supported is INTPIN_MPU.

Returns:

1 if the interrupt has been triggered.

5.6.2.11 unsigned char MLDLGetMPUSlaveAddr (void)

Query the MPU slave address.

Returns:

The 7-bit mpu slave address.

5.6.2.12 tMLError MLDLIntHandler (unsigned char *intSource*)

MLDLIntHandler function should be called when an interrupt is received.

The source parameter identifies which interrupt source caused the interrupt. Note that this routine should not be called directly from the interrupt service routine.

Parameters:

intSource MPU, AUX1, AUX2, or timer. Can be one of: INTSRC_MPU, INTSRC_AUX1, INTSRC_AUX2, or INT_SRC_TIMER.

Returns:

Zero if the command is successful; an error code otherwise.

5.6.2.13 tMLError MLDLLoadDMP (const unsigned char * *buffer*, unsigned short *length*, unsigned short *config*)

Load the DMP with the given code and configuration.

Parameters:

buffer the DMP data.

length the length in bytes of the DMP data.

config the DMP configuration.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.6.2.14 tMLError MLDLOpen (void * *mlslHandle*)

Open the driver layer and resets the internal gyroscope, accelerometer, and compass data structures.

Parameters:

mlslHandle the serial handle.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.6.2.15 tMLError MLDLSetExternalSyncMPU (unsigned char *extSync*)

This function sets the external sync for the MPU sampling.

It can be synchronized on the LSB of any of the gyros, any of the external accels, or on the temp readings.

Parameters:

extSync External sync selection, 0 to 7.

Returns:

Zero if the command is successful; an error code otherwise.

5.6.2.16 tMLError MLDLSetFullScaleMPU (float *fullScale*)

set the full scale range for the gyros.

The full scale selection codes correspond to: 0 -> 250 dps, 1 -> 500 dps, 2 -> 1000 dps, 3 -> 2000 dps. Full scale range affect the MPU's measurement sensitivity.

Parameters:

fullscale the gyro full scale range in dps.

Returns:

ML_SUCCESS or non-zero error code.

5.6.2.17 tMLError MLDLSetGyroPower (unsigned long *xOn*, unsigned long *yOn*, unsigned long *zOn*)

Sets the power state for the Gyro's.

This sets the power state of each gyro. If the MPU is suspended then it applies the next time resume is called.

Parameters:

xOn Boolean to turn on the X gyro

yOn Boolean to turn on the Y gyro

zOn Boolean to turn on the Z gyro

Returns:

ML_SUCCESS or non-zero error code

5.6.2.18 tMLError MLDLSetOffset (unsigned short const * *offset*)

Set the gyro offset.

Parameters:

offset a pointer to the gyro offset for the 3 gyro axes.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.6.2.19 tMLError MLDLSetOffsetTC (unsigned char const * *tc*)

Set the Temperature Compensation offset.

Parameters:

tc a pointer to the temperature compensations offset for the 3 gyro axes.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.6.2.20 int mpu3050_close (struct mldl_cfg * *mldl_cfg*, void * *mlsl_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*)

Close the mpu3050 interface.

Stub for driver close.

Parameters:

mldl_cfg pointer to the configuration structure

mlsl_handle pointer to the serial layer handle

Returns:

ML_SUCCESS or non-zero error code

Just verify that the devices are suspended

Parameters:

mldl_cfg handle to the config structure

mlsl_handle handle to the mpu serial layer

accel_handle handle to the accel serial layer

compass_handle handle to the compass serial layer

pressure_handle handle to the compass serial layer

Returns:

ML_SUCCESS or non-zero error code

5.6.2.21 int mpu3050_config_accel (struct mldl_cfg * *mldl_cfg*, void * *accel_handle*, struct ext_slave_config * *data*)

Request slave to change configuration.

Parameters:

mldl_cfg pointer to the mldl configuration structure

accel_handle handle to the accel sensor

data the data being requested.

Returns:

0 or non-zero error code

5.6.2.22 int mpu3050_config_compass (struct mldl_cfg * *mldl_cfg*, void * *compass_handle*, struct ext_slave_config * *data*)

Request slave to change configuration.

Parameters:

mldl_cfg pointer to the mldl configuration structure

compass_handle handle to the compass sensor

data the data being requested.

Returns:

0 or non-zero error code

5.6.2.23 int mpu3050_config_pressure (struct mldl_cfg * *mldl_cfg*, void * *pressure_handle*, struct ext_slave_config * *data*)

Request slave to change configuration.

Parameters:

mldl_cfg pointer to the mldl configuration structure

pressure_handle handle to the pressure sensor

data the data being requested.

Returns:

0 or non-zero error code

5.6.2.24 int mpu3050_get_config_accel (struct mldl_cfg * *mldl_cfg*, void * *accel_handle*, struct ext_slave_config * *data*)

Request slave configuration information.

Use this specifically after requesting a slave configuration to see what the slave accually accepted.

Parameters:

mldl_cfg pointer to the mldl configuration structure

accel_handle handle to the accel sensor

data the data being requested.

Returns:

0 or non-zero error code

5.6.2.25 int mpu3050_get_config_compass (struct mldl_cfg * *mldl_cfg*, void * *compass_handle*, struct ext_slave_config * *data*)

Request slave configuration information.

Use this specifically after requesting a slave configuration to see what the slave accually accepted.

Parameters:

mldl_cfg pointer to the mldl configuration structure

compass_handle handle to the compass sensor

data the data being requested.

Returns:

0 or non-zero error code

5.6.2.26 int mpu3050_get_config_pressure (struct mldl_cfg * *mldl_cfg*, void * *pressure_handle*, struct ext_slave_config * *data*)

Request slave configuration information.

Use this specifically after requesting a slave configuration to see what the slave accually accepted.

Parameters:

mldl_cfg pointer to the mldl configuration structure

pressure_handle handle to the pressure sensor

data the data being requested.

Returns:

0 or non-zero error code

5.6.2.27 int mpu3050_open (struct mldl_cfg * *mldl_cfg*, void * *mlsl_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*)

Initializes the pdata structure to defaults.

Opens the device to read silicon revision, product id and whoami.

Parameters:

mldl_cfg The internal device configuration data structure.

mlsl_handle The serial communication handle.

Returns:

ML_SUCCESS if silicon revision, product id and woami are supported by this software.

Opens the device to read silicon revision, product id and whoami. Leaves the device in suspended state for low power.

Parameters:

mldl_cfg handle to the config structure

mlsl_handle handle to the mpu serial layer

accel_handle handle to the accel serial layer

compass_handle handle to the compass serial layer

pressure_handle handle to the pressure serial layer

Returns:

ML_SUCCESS if silicon revision, product id and woami are supported by this software.

5.6.2.28 int mpu3050_read_accel (struct mldl_cfg * *mldl_cfg*, void * *accel_handle*, unsigned char * *data*)

read raw sensor data from the accelerometer device in use.

Parameters:

mldl_cfg A pointer to the struct mldl_cfg data structure.

accel_handle The handle to the device the accelerometer is connected to.

data a buffer to store the raw sensor data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.6.2.29 int mpu3050_read_compass (struct mldl_cfg * *mldl_cfg*, void * *compass_handle*, unsigned char * *data*)

read raw sensor data from the compass device in use.

Parameters:

mldl_cfg A pointer to the struct mldl_cfg data structure.

compass_handle The handle to the device the compass is connected to.

data a buffer to store the raw sensor data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.6.2.30 int mpu3050_read_pressure (struct mldl_cfg * *mldl_cfg*, void * *pressure_handle*, unsigned char * *data*)

read raw sensor data from the pressure device in use.

Parameters:

mldl_cfg A pointer to the struct mldl_cfg data structure.

pressure_handle The handle to the device the pressure sensor is connected to.

data a buffer to store the raw sensor data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.6.2.31 int mpu3050_resume (struct mldl_cfg * *mldl_cfg*, void * *gyro_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*, bool *resume_gyro*, bool *resume_accel*, bool *resume_compass*, bool *resume_pressure*)

resume the MPU3050 device and all the other sensor devices from their low power state.

Parameters:

mldl_cfg pointer to the configuration structure

gyro_handle the main file handle to the MPU3050 device.

accel_handle an handle to the accelerometer device, if sitting onto a separate bus. Can match mlsl_handle if the accelerometer device operates on the same primary bus of MPU.

compass_handle an handle to the compass device, if sitting onto a separate bus. Can match mlsl_handle if the compass device operates on the same primary bus of MPU.

pressure_handle an handle to the pressure sensor device, if sitting onto a separate bus. Can match mlsl_handle if the pressure sensor device operates on the same primary bus of MPU.

resume_gyro whether resuming the gyroscope device is actually needed (if the device supports low power mode of some sort).

resume_accel whether resuming the accelerometer device is actually needed (if the device supports low power mode of some sort).

resume_compass whether resuming the compass device is actually needed (if the device supports low power mode of some sort).

resume_pressure whether resuming the pressure sensor device is actually needed (if the device supports low power mode of some sort).

Returns:

ML_SUCCESS or a non-zero error code.

5.6.2.32 int mpu3050_suspend (struct mldl_cfg * *mldl_cfg*, void * *gyro_handle*, void * *accel_handle*, void * *compass_handle*, void * *pressure_handle*, bool *suspend_gyro*, bool *suspend_accel*, bool *suspend_compass*, bool *suspend_pressure*)

suspend the MPU3050 device and all the other sensor devices into their low power state.

Parameters:

gyro_handle the main file handle to the MPU3050 device.

accel_handle an handle to the accelerometer device, if sitting onto a separate bus. Can match gyro_handle if the accelerometer device operates on the same primary bus of MPU.

compass_handle an handle to the compass device, if sitting onto a separate bus. Can match gyro_handle if the compass device operates on the same primary bus of MPU.

pressure_handle an handle to the pressure sensor device, if sitting onto a separate bus. Can match gyro_handle if the pressure sensor device operates on the same primary bus of MPU.

accel whether suspending the accelerometer device is actually needed (if the device supports low power mode of some sort).

compass whether suspending the compass device is actually needed (if the device supports low power mode of some sort).

pressure whether suspending the pressure sensor device is actually needed (if the device supports low power mode of some sort).

Returns:

ML_SUCCESS or a non-zero error code.

5.7 ML_CONTROL

Motion Library - Control Engine.

Files

- file mlcontrol.c

  The Control Library.

Typedefs

- typedef void(∗ fpGridCb)(unsigned short controlSignal, long ∗gridNum, long ∗gridChange)

  GridCallback function pointer type, to be passed as argument of MLSetGridCallback.

Functions

- tMLError MLDisableControl (void)

  Disables the ML_CONTROL engine.

- tMLError MLEnableControl (void)

  Enables the ML_CONTROL engine.

- tMLError MLGetControlData (long ∗controlSignal, long ∗gridNum, long ∗gridChange)

  MLGetControlData is used to get the current control data.

- tMLError MLGetControlSignal (unsigned short controlSignal, unsigned short reset, long ∗data)

  MLGetControlSignal is used to get the current control signal with high precision.

- tMLError MLGetGridNum (unsigned short controlSignal, unsigned short reset, long ∗data)

  MLGetGridNum is used to get the current grid location for a certain control signal.

- tMLError MLSetControlData (unsigned short controlSignal, unsigned short parameterArray, unsigned short parameterAxis)

  MLSetControlData is used to assign physical parameters to control signals.

- tMLError MLSetControlFunc (unsigned short function)

  MLSetControlFunc allows the user to choose how the sensor data will be processed in order to provide a control parameter.

- tMLError MLSetControlSensitivity (unsigned short controlSignal, long sensitivity)

  MLSetControlSensitivity is used to set the sensitivity for a control signal.

- tMLError MLSetGridCallback (fpGridCb func)

  MLSetGridCallback is used to register a callback function that will trigger when the grid location changes.

- tMLError MLSetGridMax (unsigned short controlSignal, long maximum)

  MLSetGridMax is used to set the maximum grid number for a control signal.

- tMLError MLSetGridThresh (unsigned short controlSignal, long threshold)

  MLSetGridThresh is used to set the grid size for a control signal.

5.7.1 Detailed Description

Motion Library - Control Engine.

The Control Library processes gyroscopes, accelerometers, and compasses to provide control signals that can be used in user interfaces. These signals can be used to manipulate objects such as documents, images, cursors, menus, etc.

5.7.2 Typedef Documentation

5.7.2.1 typedef void(∗ fpGridCb)(unsigned short controlSignal, long ∗gridNum, long ∗gridChange)

GridCallback function pointer type, to be passed as argument of MLSetGridCallback.

Parameters:

controlSignal Indicates which control signal crossed a grid threshold. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 and
- ML_CONTROL_4.

gridNumber An array of four numbers representing the grid number for each control signal.

gridChange An array of four numbers representing the change in grid number for each control signal.

5.7.3 Function Documentation

5.7.3.1 tMLError MLDisableControl (void)

Disables the ML_CONTROL engine.

Note:

This function replaces MLDisable(ML_CONTROL)

Precondition:

MLDmpOpen() with MLDmpDefaultOpen or MLDmpPedometerStandAlone() must have been called.

Returns:

ML_SUCCESS or non-zero error code

5.7.3.2 tMLError MLEnableControl (void)

Enables the ML_CONTROL engine.

Note:

This function replaces MLEnable(ML_CONTROL)

Precondition:

MLDmpOpen() with MLDmpDefaultOpen or MLDmpPedometerStandAlone() must have been called.

Returns:

ML_SUCCESS or non-zero error code

5.7.3.3 tMLError MLGetControlData (long * *controlSignal*, long * *gridNum*, long * *gridChange*)

MLGetControlData is used to get the current control data.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

controlSignal Indicates which control signal is being queried. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 or
- ML_CONTROL_4.

gridNum A pointer to pass gridNum info back to the user.

gridChange A pointer to pass gridChange info back to the user.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.7.3.4 tMLError MLGetControlSignal (unsigned short *controlSignal*, unsigned short *reset*, long * *data*)

MLGetControlSignal is used to get the current control signal with high precision.

MLGetControlSignal is used to acquire the current data of a control signal. If ML_GRID is being used, MLGetGridNumber will probably be preferrable.

Parameters:

controlSignal Indicates which control signal is being queried. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 or
- ML_CONTROL_4.

reset Indicates whether the control signal should be reset to zero. Options are ML_RESET or ML_NO_RESET

data A pointer to the current control signal data.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.7.3.5 tMLError MLGetGridNum (unsigned short *controlSignal*, unsigned short *reset*, long * *data*)

MLGetGridNum is used to get the current grid location for a certain control signal.

MLGetGridNum is used to acquire the current grid location.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

controlSignal Indicates which control signal is being queried. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 or
- ML_CONTROL_4.

reset Indicates whether the control signal should be reset to zero. Options are ML_RESET or ML_NO_RESET

data A pointer to the current grid number.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.7.3.6 tMLError MLSetControlData (unsigned short *controlSignal*, unsigned short *parameterArray*, unsigned short *parameterAxis*)

MLSetControlData is used to assign physical parameters to control signals.

MLSetControlData allows flexibility in assigning physical parameters to control signals. For example, the user is allowed to use raw gyroscope data as an input to the control algorithm. Alternatively, angular velocity can be used, which combines gyroscopes and accelerometers to provide a more robust physical parameter. Finally, angular velocity in world coordinates can be used, providing a control signal in which pitch and yaw are provided relative to gravity.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 or
- ML_CONTROL_4.

parameterArray Indicates which parameter array is being assigned to a control signal. Must be one of:

- ML_GYROS,
- ML_ANGULAR_VELOCITY, or

parameterAxis Indicates which axis of the parameter array will be used. Must be:

- ML_ROLL,
- ML_PITCH, or
- ML_YAW.

5.7.3.7 tMLError MLSetControlFunc (unsigned short *function*)

MLSetControlFunc allows the user to choose how the sensor data will be processed in order to provide a control parameter.

MLSetControlFunc allows the user to choose which control functions will be incorporated in the sensor data processing. The control functions are:

- ML_GRID Indicates that the user will be controlling a system that has discrete steps, such as icons, menu entries, pixels, etc.
- ML_SMOOTH Indicates that noise from unintentional motion should be filtered out.
- ML_DEAD_ZONE Indicates that a dead zone should be used, below which sensor data is set to zero.
- ML_HYSTERESIS Indicates that, when ML_GRID is selected, hysteresis should be used to prevent the control signal from switching rapidly across elements of the grid.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

function Indicates what functions will be used. Can be a bitwise OR of several values.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.7.3.8 tMLError MLSetControlSensitivity (unsigned short *controlSignal*, long *sensitivity*)

MLSetControlSensitivity is used to set the sensitivity for a control signal.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 or
- ML_CONTROL_4.

sensitivity The sensitivity of the control signal.

Returns:

error code

5.7.3.9 tMLError MLSetGridCallback (fpGridCb *func*)

MLSetGridCallback is used to register a callback function that will trigger when the grid location changes.

MLSetGridCallback allows a user to define a callback function that will run when a control signal crosses a grid threshold.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen(). MLDmpStart must **NOT** have been called.

Parameters:

func A user defined callback function

Returns:

Zero if the command is successful; an ML error code otherwise.

5.7.3.10 tMLError MLSetGridMax (unsigned short *controlSignal*, long *maximum*)

MLSetGridMax is used to set the maximum grid number for a control signal.

MLSetGridMax is used to adjust the maximum allowed grid number, above which the grid number will not be incremented. The minimum grid number is always zero.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 and
- ML_CONTROL_4.

maximum The maximum grid number for a control signal.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.7.3.11 tMLError MLSetGridThresh (unsigned short *controlSignal*, long *threshold*)

MLSetGridThresh is used to set the grid size for a control signal.

MLSetGridThresh is used to adjust the size of the grid being controlled.

Parameters:

controlSignal Indicates which control signal is being modified. Must be one of:

- ML_CONTROL_1,
- ML_CONTROL_2,
- ML_CONTROL_3 and
- ML_CONTROL_4.

threshold The threshold of the control signal at which the grid number will be incremented or decremented.

Returns:

Zero if the command is successful; an ML error code otherwise.

5.8 COMPASSDL

Motion Library - Compass Driver Layer.

Files

- file compass.c

  Compass setup and handling methods.

Functions

- tMLError CompassGetData (long *data)

  Get a sample of compass data from the device.

- unsigned short CompassGetId (void)

  Get the ID of the compass in use.

- unsigned char CompassGetPresent (void)

  Is a compass configured and used by MPL?

- unsigned char CompassGetSlaveAddr (void)

  Query the compass slave address.

- tMLError CompassSetBias (long *bias)

  Sets the compass bias.

5.8.1 Detailed Description

Motion Library - Compass Driver Layer.

Provides the interface to setup and handle an compass connected to either the primary or the seconday I2C interface of the gyroscope.

5.8.2 Function Documentation

5.8.2.1 tMLError CompassGetData (long * *data*)

Get a sample of compass data from the device.

Parameters:

data the buffer to store the compass raw data for X, Y, and Z axes.

Returns:

ML_SUCCESS or a non-zero error code.

5.8.2.2 unsigned short CompassGetId (void)

Get the ID of the compass in use.

Returns:

ID of the compass in use.

5.8.2.3 unsigned char CompassGetPresent (void)

Is a compass configured and used by MPL?

Returns:

ML_SUCCESS if the compass is present.

5.8.2.4 unsigned char CompassGetSlaveAddr (void)

Query the compass slave address.

Returns:

The 7-bit compass slave address.

5.8.2.5 tMLError CompassSetBias (long * *bias*)

Sets the compass bias.

Parameters:

bias Compass bias, length 3. Scale is micro Tesla's * 2^16. Frame is mount frame which may be different from body frame.

5.9 ML_STORED_DATA

Files

- file ml_stored_data.c

  functions for reading and writing stored data sets.

Functions

- int FindTempBin (float temp)

  Duplicate of the TempCompFindTempBin function in the libmpl advanced algorithms library.

- tMLError MLGetCalLength (unsigned int *length)

  Returns the length of the **MPL internal calibration data**.

- tMLError MLLoadCal (unsigned char *calData)

  Loads a set of calibration data.

- tMLError MLLoadCal_V0 (unsigned char *calData, unsigned short len)

  Loads a type 0 set of calibration data.

- tMLError MLLoadCal_V1 (unsigned char *calData, unsigned short len)

  Loads a type 1 set of calibration data.

- tMLError MLLoadCal_V2 (unsigned char *calData, unsigned short len)

  Loads a type 2 set of calibration data.

- tMLError MLLoadCal_V3 (unsigned char *calData, unsigned short len)

  Loads a type 3 set of calibration data.

- tMLError MLLoadCal_V4 (unsigned char *calData, unsigned short len)

  Loads a type 4 set of calibration data.

- tMLError MLLoadCal_V5 (unsigned char *calData, unsigned short len)

  Loads a type 5 set of calibration data.

- tMLError MLLoadCalibration (void)

  Load a calibration file.

- tMLError MLStoreCal (unsigned char *calData, int length)

Stores a set of calibration data.

- tMLError MLStoreCalibration (void)

  Store runtime calibration data to a file.

- float tempInDegC (float intTemp)

  Converts from internal sensor register implementation to degrees C.

5.9.1 Function Documentation

5.9.1.1 int FindTempBin (float *temp*)

Duplicate of the TempCompFindTempBin function in the libmpl advanced algorithms library.

To remove cross-dependency, for now, we reimplement the same function here.

Parameters:

temp the temperature (1 count == 1 degree C).

5.9.1.2 tMLError MLGetCalLength (unsigned int * *length*)

Returns the length of the **MPL internal calibration data**.

Should be called before allocating the memory required to store this data to a file. This function returns the total size required to store the cal data including the header (4 bytes) and the checksum (2 bytes).

Precondition:

Must be in ML_STATE_DMP_OPENED state. MLDmpOpen() or MLDmpStop() must have been called. MLDmpStart() and MLDmpClose() must have **NOT** been called.

Parameters:

length The length of the calibration data.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.9.1.3 tMLError MLLoadCal (unsigned char * *calData*)

Loads a set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

calData A pointer to an array of bytes to be parsed.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.4 tMLError MLLoadCal_V0 (unsigned char * *calData*, unsigned short *len*)

Loads a type 0 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature,
- gyro biases for X, Y, Z axes. This calibration data would normally be produced by the MPU Self Test and its size is 18 bytes (header and checksum included). Calibration format type 0 is currently **NOT** used and is substituted by type 5 : MLLoadCal_V5().

Note:

This calibration data format is obsoleted and no longer supported by the rest of the MPL.

Precondition:

One of the MLDmpOpen() must be called: MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.5 tMLError MLLoadCal_V1 (unsigned char * *calData*, unsigned short *len*)

Loads a type 1 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature,
- gyro biases for X, Y, Z axes,
- accel biases for X, Y, Z axes. This calibration data would normally be produced by the MPU Self Test and its size is 24 bytes (header and checksum included). Calibration format type 1 is currently **NOT** used and is substituted by type 5 : MLLoadCal_V5().

Note:

In order to successfully work, the gyro bias must be stored expressed in 250 dps full scale (131.072 sensitivity). Other full scale range will produce unpredictable results in the gyro biases.

Precondition:

One of the MLDmpOpen() must be called: MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.6 tMLError MLLoadCal_V2 (unsigned char * *calData*, unsigned short *len*)

Loads a type 2 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature compensation : temperature data points,
- temperature compensation : gyro biases data points for X, Y, and Z axes.
- accel biases for X, Y, Z axes. This calibration data is produced internally by the MPL and its size is 2222 bytes (header and checksum included). Calibration format type 2 is currently **NOT** used and is substituted by type 4 : MLLoadCal_-V4().

Precondition:

One of the MLDmpOpen() must be called: MLDmpDefaultOpen() or MLDmpPe-dometerStandAloneOpen().

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.7 tMLError MLLoadCal_V3 (unsigned char * *calData*, unsigned short *len*)

Loads a type 3 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature compensation : temperature data points,
- temperature compensation : gyro biases data points for X, Y, and Z axes.
- accel biases for X, Y, Z axes.

- compass biases for X, Y, Z axes and bias tracking algorithm mock-up. This calibration data is produced internally by the MPL and its size is 2429 bytes (header and checksum included). Calibration format type 3 is currently **NOT** used and is substituted by type 4 : MLLoadCal_V4().

Precondition:

One of the MLDmpOpen() must be called: MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.8 tMLError MLLoadCal_V4 (unsigned char * *calData*, unsigned short *len*)

Loads a type 4 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature compensation : temperature data points,
- temperature compensation : gyro biases data points for X, Y, and Z axes.
- accel biases for X, Y, Z axes.
- compass biases for X, Y, Z axes, compass scale, and bias tracking algorithm mock-up. This calibration data is produced internally by the MPL and its size is 2777 bytes (header and checksum included). Calibration format type 4 is currently used and substitutes type 2 (MLLoadCal_V2()) and 3 (MLLoadCal_V3()).

Precondition:

One of the MLDmpOpen() must be called: MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.9 tMLError MLLoadCal_V5 (unsigned char * *calData*, unsigned short *len*)

Loads a type 5 set of calibration data.

It parses a binary data set containing calibration data. The binary data set is intended to be loaded from a file. This calibrations data format stores values for (in order of appearance) :

- temperature,
- gyro biases for X, Y, Z axes,
- accel biases for X, Y, Z axes. This calibration data would normally be produced by the MPU Self Test and its size is 36 bytes (header and checksum included). Calibration format type 5 is produced by the MPU Self Test and substitutes the type 1 : MLLoadCal_V1().

Precondition:

One of the MLDmpOpen() must be called: MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

Parameters:

calData A pointer to an array of bytes to be parsed.

len the length of the calibration

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.10 tMLError MLLoadCalibration (void)

Load a calibration file.

Precondition:

Must be in ML_STATE_DMP_OPENED state. MLDmpOpen() or MLDmpStop() must have been called. MLDmpStart() and MLDmpClose() must have **NOT** been called.

Returns:

0 or error code.

5.9.1.11 tMLError MLStoreCal (unsigned char * *calData*, int *length*)

Stores a set of calibration data.

It generates a binary data set containing calibration data. The binary data set is intended to be stored into a file.

Precondition:

MLDmpOpen()

Parameters:

calData A pointer to an array of bytes to be stored.

length The amount of bytes available in the array.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.9.1.12 tMLError MLStoreCalibration (void)

Store runtime calibration data to a file.

Precondition:

Must be in ML_STATE_DMP_OPENED state. MLDmpOpen() or MLDmpStop() must have been called. MLDmpStart() and MLDmpClose() must have **NOT** been called.

Returns:

0 or error code.

5.9.1.13 float tempInDegC (float *intTemp*)

Converts from internal sensor register implementation to degrees C.

Parameters:

intTemp internal temperature sensor representation in LSBs.

Returns:

temperature in deg. C

5.10 MPU_SELF_TEST

C wrapper to integrate the MPU Self Test wrapper in MPL.

Files

- file ml_mputest.c

  C wrapper to integrate the MPU Self Test wrapper in MPL.

- file mputest.c

  MPU Self Test routines for assessing gyro sensor status after surface mount has happened on the target host platform.

Functions

- tMLError FactoryCalibrate (void *mlsl_handle)

  The main test API.

- tMLError MLSelfTestFactoryCalibrate (void *mlsl_handle)

  An MPL wrapper for the main MPU Self Test API FactoryCalibrate().

- tMLError MLSelfTestRun (void)

  Runs the MPU test at MPL runtime.

- tMLError MLSelfTestSetAccelZOrient (signed char zSign)

  Set the orientation of the acceleroemter Z axis as it will be expected when running the MPU Self Test.

- int MPUTest (void *mlsl_handle)

  The main entry point of the MPU Self Test, triggering the run of the single tests, for gyros and accelerometers.

- void SetTestParameters (unsigned int slaveAddr, float sensitivity, int pThresh, float totalTimeTol, int biasThresh, float rmsThresh, float SPShiftThresh)

  Modify the self test limits from their default values.

- int TestAccel (void *mlsl_handle, short *bias)

  If requested via TestSetupAccel(), test the accelerometer biases and calculate the necessary bias correction.

- int TestGyro (void *mlsl_handle, short gyro_biases[3], short *temp_avg)

Test the gyroscope sensor.

5.10.1 Detailed Description

C wrapper to integrate the MPU Self Test wrapper in MPL.

MPU Self Test functions.

Provides ML name compliant naming and an additional API that automates the suspension of normal MPL operations, runs the test, and resume.

These functions provide an in-site test of the MPU 3xxx chips. The main entry point is the MPUTest function. This runs the tests (as described in the accompanying documentation) and writes a configuration file containing initial calibration data. MPUTest returns ML_SUCCESS if the chip passes the tests. Otherwise, an error code is returned. The functions in this file rely on MLSL and MLOS: refer to the MPL documentation for more information regarding the system interface files.

5.10.2 Function Documentation

5.10.2.1 tMLError FactoryCalibrate (void * *mlsl_handle*)

The main test API.

Runs the MPU Self Test and, if successful, stores the encoded initial calibration data on the final storage medium of choice (cfr. MLSLWriteCal() and the MLCAL_FILE define in your mlsl implementation).

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

Returns:

0 on success or a non-zero error code from the callees on error.

5.10.2.2 tMLError MLSelfTestFactoryCalibrate (void * *mlsl_handle*)

An MPL wrapper for the main MPU Self Test API FactoryCalibrate().

See FactoryCalibrate() function for more details.

Precondition:

MLDmpOpen() **must** have been called to populate the mldl_cfg data structure.

On Windows, SetupPlatform() is also a madatory pre condition to ensure the accelerometer is properly configured before running the test.

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

Returns:

ML_SUCCESS on success or a bitmask error code from the Self Test.

5.10.2.3 tMLError MLSelfTestRun (void)

Runs the MPU test at MPL runtime.

If the DMP is operating, stops the DMP temporarely, runs the MPU Self Test, and re-starts the DMP.

Returns:

ML_SUCCESS or first non-zero error code otherwise.

5.10.2.4 tMLError MLSelfTestSetAccelZOrient (signed char *zSign*)

Set the orientation of the acceleroemter Z axis as it will be expected when running the MPU Self Test.

Specifies the orientation of the accelerometer Z axis : Z axis pointing upwards or downwards.

Parameters:

zSign The sign of the accelerometer Z axis; valid values are +1 and -1 for +Z and -Z respectively. Any other value will cause the setting to be ignored and an error code to be returned.

Returns:

ML_SUCCESS or a non-zero error code.

5.10.2.5 int MPUTest (void * *mlsl_handle*)

The main entry point of the MPU Self Test, triggering the run of the single tests, for gyros and accelerometers.

Prepares the MPU for the test, taking the device out of low power state if necessary, switching the MPU secondary I2C interface into bypass mode and restoring the original power state at the end of the test. This function is also responsible for encoding the output of each test in the correct format as it is stored on the file/medium of choice (according to MLSLWriteCal() function). The format needs to stay perfectly consistent with the one expected by the corresponding loader in ml_stored_data.c; currectly the loaded in use is MLLoadCal_V1 (record type 1 - initial calibration).

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

Returns:

0 on success. A non-zero error code on error. Propagates the errors from the tests up to the caller.

5.10.2.6 void SetTestParameters (unsigned int *slaveAddr*, float *sensitivity*, int *pThresh*, float *totalTimeTol*, int *biasThresh*, float *rmsThresh*, float *SPShiftThresh*)

Modify the self test limits from their default values.

Parameters:

slaveAddr the slave address the MPU device is setup to respond at. The default is DEF_MPU_ADDR = 0x68.

sensitivity the read sensitivity of the device in LSB/dps as it is trimmed. NOTE : if using the self test as part of the MPL, the sensitivity the different sensitivity trims are already taken care of.

pThresh number of packets expected to be received in a 600 ms period. Depends on the sampling frequency of choice (set by default to 125 Hz) and low pass filter cut-off frequency selection (set to 42 Hz). The default is DEF_PACKET_THRESH = 75 packets.

totalTimeTol time skew tolerance, taking into account imprecision in turning the FIFO on and off and the processor time imprecision (for 1 GHz processor). The default is DEF_TOTAL_TIMING_TOL = 3 %, about 2 packets.

biasThresh bias level threshold, the maximun acceptable no motion bias for a production quality part. The default is DEF_BIAS_THRESH = 40 dps.

rmsThresh the limit standard deviation (=∼ RMS) set to assess whether the noise level on the part is acceptable. The default is DEF_RMS_THRESH = 0.2 dps-rms.

SPShiftThresh the limit shift applicable to the Sense Path self test calculation.

5.10.2.7 int TestAccel (void * *mlsl_handle*, short * *bias*)

If requested via TestSetupAccel(), test the accelerometer biases and calculate the necessary bias correction.

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

bias output pointer to store the initial bias calculation provided by the MPU Self Test. Requires 3 elements to store accel X, Y, and Z axis bias.

Returns:

0 on success. A non-zero error code on error.

5.10.2.8 int TestGyro (void * *mlsl_handle*, short *gyro_biases*[3], short * *temp_avg*)

Test the gyroscope sensor.

Implements the core logic of the MPU Self Test. Produces the PASS/FAIL result. Loads the calculated gyro biases and temperature datum into the corresponding pointers.

Parameters:

mlsl_handle serial interface handle to allow serial communication with the device, both gyro and accelerometer.

gyro_biases output pointer to store the initial bias calculation provided by the MPU Self Test. Requires 3 elements for gyro X, Y, and Z.

temp_avg output pointer to store the initial average temperature as provided by the MPU Self Test.

Returns:

0 on success. On error, the return value is a bitmask representing: 0, 1, 2 Failures with PLLs on X, Y, Z gyros respectively (decimal values will be 1, 2, 4 respectively). 3, 4, 5 Excessive offset with X, Y, Z gyros respectively (decimal values will be 8, 16, 32 respectively). 6, 7, 8 Excessive noise with X, Y, Z gyros respectively (decimal values will be 64, 128, 256 respectively). 9 If any of the RMS noise values is zero, it could be due to a non-functional gyro or FIFO/register failure. (decimal value will be 512). 10, 11, 12 Excessive bias shift in Sense Path MPU Self Test for X, Y, Z gyros respectively (decimal values will be 1024, 2048, 4096 respectively).

5.11 MLSL

Motion Library - Serial Layer.

Functions

- tMLError MLSLDisableStreaming (void)
  called in order to disable all streaming activity.
- tMLError MLSLEnableIntStream (void)
  enables streaming of interrupt notifications.
- tMLError MLSLGetCalLength (unsigned int *len)
  Get the calibration length.
- unsigned char MLSLGetSerialOpen (void)
  Get the serial port status.
- tMLError MLSLReadCal (unsigned char *cal, unsigned int len)
  used to get the calibration data.
- tMLError MLSLReadCfg (unsigned char *cfg, unsigned int len)
  used to get the configuration data.
- tMLError MLSLSerialClose (void *sl_handle)
  used to close the serial port.
- tMLError MLSLSerialOpen (char const *port, void **sl_handle)
  used to open the serial port.
- tMLError MLSLSerialRead (void *mlsl_handle, unsigned char slaveAddr, unsigned char registerAddr, unsigned short length, unsigned char *data)
  used to read multiple bytes of data from registers.
- tMLError MLSLSerialReadFifo (void *sl_handle, unsigned char slaveAddr, unsigned short length, unsigned char *data)
  used to read multiple bytes of data from the fifo.
- tMLError MLSLSerialReadMem (void *sl_handle, unsigned char slaveAddr, unsigned short memAddr, unsigned short length, unsigned char *data)
  used to read multiple bytes of data from the memory.

- tMLError MLSLSerialReset (void *mlsl_handle)
  used to reset any buffering the driver may be doing
- tMLError MLSLSerialWrite (void *mlsl_handle, unsigned char slaveAddr, unsigned short length, unsigned char const *data)
  used to write multiple bytes of data from registers.
- tMLError MLSLSerialWriteFifo (void *sl_handle, unsigned char slaveAddr, unsigned short length, unsigned char const *data)
  used to write multiple bytes of data to the fifo.
- tMLError MLSLSerialWriteMem (void *sl_handle, unsigned char slaveAddr, unsigned short memAddr, unsigned short length, unsigned char const *data)
  used to write multiple bytes of data to the memory.
- tMLError MLSLSerialWriteSingle (void *sl_handle, unsigned char slaveAddr, unsigned char registerAddr, unsigned char data)
  used to write a single byte of data.
- tMLError MLSLSetYamahaCompassDataMode (unsigned char mode)
  set Yamaha compass I2C data mode.
- tMLError MLSLWriteCal (unsigned char *cal, unsigned int len)
  used to save the calibration data.
- tMLError MLSLWriteCfg (unsigned char *cfg, unsigned int len)
  used to save the configuration data.

5.11.1 Detailed Description

Motion Library - Serial Layer.

The Motion Library System Layer provides the Motion Library the interface to the system functions.

5.11.2 Function Documentation

5.11.2.1 tMLError MLSLDisableStreaming (void)

called in order to disable all streaming activity.

The streaming is done by another device/processor, and thus a command wil have to be sent to that device.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.2 tMLError MLSLEnableIntStream (void)

enables streaming of interrupt notifications.

The streaming is done by another device/processor, and thus a command will have to be sent to that device.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.3 tMLError MLSLGetCalLength (unsigned int * *len*)

Get the calibration length.

Parameters:

len lenght to be returned

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.4 unsigned char MLSLGetSerialOpen (void)

Get the serial port status.

Interrogates the underlying driver implementation for serial port status and returns it.

Returns:

1 if the serial port is open, 0 otherwise.

5.11.2.5 tMLError MLSLReadCal (unsigned char * *cal*, unsigned int *len*)

used to get the calibration data.

It is called by the MPL to get the calibration data used by the motion library. This data would typically be saved in non-volatile memory.

Parameters:

cfg Pointer to the calibration data.

len Length of the calibration data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.6 tMLError MLSLReadCfg (unsigned char * *cfg*, unsigned int *len*)

used to get the configuration data.

Is called by the MPL to get the configuration data used by the motion library. This data would typically be saved in non-volatile memory.

Parameters:

cfg Pointer to the configuration data.

len Length of the configuration data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.7 tMLError MLSLSerialClose (void * *sl_handle*)

used to close the serial port.

This port is used to send and receive data to the device.

Note:

This function is called by MLSerialClose(). Unlike previous MPL Software releases, explicitly calling MLSerialClose() is mandatory to properly shut-down the communication with the device.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.8 tMLError MLSLSerialOpen (char const * *port*, void ** *sl_handle*)

used to open the serial port.

This port is used to send and receive data to the device.

Note:

This function is called by MLSerialOpen(). Unlike previous MPL Software releases, explicitly calling MLSerialOpen() is mandatory to instantiate the communication with the device.

Parameters:

port The COM port specification associated with the device in use.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.9 tMLError MLSLSerialRead (void * *mlsl_handle*, unsigned char *slaveAddr*, unsigned char *registerAddr*, unsigned short *length*, unsigned char * *data*)

used to read multiple bytes of data from registers.

This should be sent by I2C or SPI.

Parameters:

slaveAddr I2C slave address of device.

registerAddr Register address to read.

length Length of burst of data.

data Pointer to block of data.

Returns:

Zero if successful; an error code otherwise

5.11.2.10 tMLError MLSLSerialReadFifo (void * *sl_handle*, unsigned char *slaveAddr*, unsigned short *length*, unsigned char * *data*)

used to read multiple bytes of data from the fifo.

This should be sent by I2C or SPI.

Parameters:

slaveAddr I2C slave address of device.

length Length of burst of data.

data Pointer to block of data.

Returns:

Zero if successful; an error code otherwise

5.11.2.11 tMLError MLSLSerialReadMem (void * *sl_handle*, unsigned char *slaveAddr*, unsigned short *memAddr*, unsigned short *length*, unsigned char * *data*)

used to read multiple bytes of data from the memory.

This should be sent by I2C or SPI.

Parameters:

slaveAddr I2C slave address of device.

memAddr The location in the memory to read from.

length Length of burst data.

data Pointer to block of data.

Returns:

Zero if successful; an error code otherwise

5.11.2.12 tMLError MLSLSerialReset (void * *mlsl_handle*)

used to reset any buffering the driver may be doing

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.13 tMLError MLSLSerialWrite (void * *mlsl_handle*, unsigned char *slaveAddr*, unsigned short *length*, unsigned char const * *data*)

used to write multiple bytes of data from registers.

This should be sent by I2C or SPI.

Parameters:

slaveAddr I2C slave address of device.

registerAddr Register address to write.

length Length of burst of data.

data Pointer to block of data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.14 tMLError MLSLSerialWriteFifo (void * *sl_handle*, unsigned char *slaveAddr*, unsigned short *length*, unsigned char const * *data*)

used to write multiple bytes of data to the fifo.

This should be sent by I2C or SPI.

Parameters:

slaveAddr I2C slave address of device.

length Length of burst of data.

data Pointer to block of data.

Returns:

Zero if successful; an error code otherwise

5.11.2.15 tMLError MLSLSerialWriteMem (void * *sl_handle*, unsigned char *slaveAddr*, unsigned short *memAddr*, unsigned short *length*, unsigned char const * *data*)

used to write multiple bytes of data to the memory.

This should be sent by I2C or SPI.

Parameters:

slaveAddr I2C slave address of device.

memAddr The location in the memory to write to.

length Length of burst data.

data Pointer to block of data.

Returns:

Zero if successful; an error code otherwise

5.11.2.16 tMLError MLSLSerialWriteSingle (void * *sl_handle*, unsigned char *slaveAddr*, unsigned char *registerAddr*, unsigned char *data*)

used to write a single byte of data.

It is called by the MPL to write a single byte of data to the MPU. This should be sent by I2C or SPI.

Parameters:

slaveAddr I2C slave address of device.

registerAddr Register address to write.

data Single byte of data to write.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.17 tMLError MLSLSetYamahaCompassDataMode (unsigned char *mode*)

set Yamaha compass I2C data mode.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.18 tMLError MLSLWriteCal (unsigned char * *cal*, unsigned int *len*)

used to save the calibration data.

It is called by the MPL to save the calibration data used by the motion library. This data would typically be saved in non-volatile memory.

Parameters:

cfg Pointer to the calibration data.

len Length of the calibration data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.11.2.19 tMLError MLSLWriteCfg (unsigned char * *cfg*, unsigned int *len*)

used to save the configuration data.

Is called by the MPL to save the configuration data used by the motion library. This data would typically be saved in non-volatile memory.

Parameters:

cfg Pointer to the configuration data.

len Length of the configuration data.

Returns:

ML_SUCCESS if successful, a non-zero error code otherwise.

5.12 MLERROR

Motion Library - Error definitions.

Motion Library - Error definitions.

Definition of the error codes used within the MPL and returned to the user. Every function tries to return a meaningful error code basing on the occuring error condition. The error code is numeric.

The available error codes and their associated values are:

- (0) ML_SUCCESS
- (1) ML_ERROR
- (2) ML_ERROR_INVALID_PARAMETER
- (3) ML_ERROR_FEATURE_NOT_ENABLED
- (4) ML_ERROR_FEATURE_NOT_IMPLEMENTED
- (6) ML_ERROR_DMP_NOT_STARTED
- (7) ML_ERROR_DMP_STARTED
- (8) ML_ERROR_NOT_OPENED
- (9) ML_ERROR_OPENED
- (10) ML_ERROR_INVALID_MODULE
- (11) ML_ERROR_MEMORY_EXAUSTED
- (12) ML_ERROR_DIVIDE_BY_ZERO
- (13) ML_ERROR_ASSERTION_FAILURE
- (14) ML_ERROR_FILE_OPEN
- (15) ML_ERROR_FILE_READ
- (16) ML_ERROR_FILE_WRITE
- (20) ML_ERROR_SERIAL_CLOSED
- (21) ML_ERROR_SERIAL_OPEN_ERROR
- (22) ML_ERROR_SERIAL_READ
- (23) ML_ERROR_SERIAL_WRITE
- (24) ML_ERROR_SERIAL_DEVICE_NOT_RECOGNIZED

- (25) ML_ERROR_SM_TRANSITION
- (26) ML_ERROR_SM_IMPROPER_STATE
- (30) ML_ERROR_FIFO_OVERFLOW
- (31) ML_ERROR_FIFO_FOOTER
- (32) ML_ERROR_FIFO_READ_COUNT
- (33) ML_ERROR_FIFO_READ_DATA
- (40) ML_ERROR_MEMORY_SET
- (50) ML_ERROR_LOG_MEMORY_ERROR
- (51) ML_ERROR_LOG_OUTPUT_ERROR
- (60) ML_ERROR_OS_BAD_PTR
- (61) ML_ERROR_OS_BAD_HANDLE
- (62) ML_ERROR_OS_CREATE_FAILED
- (63) ML_ERROR_OS_LOCK_FAILED
- (70) ML_ERROR_COMPASS_DATA_OVERFLOW
- (71) ML_ERROR_COMPASS_DATA_UNDERFLOW
- (72) ML_ERROR_COMPASS_DATA_NOT_READY
- (73) ML_ERROR_COMPASS_DATA_ERROR
- (75) ML_ERROR_CALIBRATION_LOAD
- (76) ML_ERROR_CALIBRATION_STORE
- (77) ML_ERROR_CALIBRATION_LEN
- (78) ML_ERROR_CALIBRATION_CHECKSUM

5.13 ML_NAVWALK

Motion Library - Navigation Engine.

Files

- file navWalk.c

  navWalk source file.

Functions

- int NavWalkClearNumOfSteps ()

  Used to clear the navWalk's step counter.

- int NavWalkDisable ()

  Disable the navWalk.

- int NavWalkEnable ()

  Enables the navWalk.

- int NavWalkGetNumOfSteps (unsigned int *steps)

  Used retrieve the number of steps taken by the user.

- float NavWalkHeading ()

  Returns the angle in radians of the heading.

- float NavWalkHeadingToVelocity ()

  Returns the angle in radians of the angle from the heading to the velocity vector.

- int NavWalkSetDataRate (int dataRate)
- int NavWalkSetStepCallback (void(*func)(unsigned short stepNum))

  Used to register a callback function that will trigger when the user takes a step.

- int NavWalkSetStrideLength (unsigned short strideLength)

  Used to set the user's stride length.

5.13.1 Detailed Description

Motion Library - Navigation Engine.

The NavWalk application counts steps by the user. The navWalk application requires computation on the host. This navWalk generally has better accuracy than than PedometerStandAlone which runs all on the DMP.

5.13.2 Function Documentation

5.13.2.1 int NavWalkClearNumOfSteps ()

Used to clear the navWalk's step counter.

The step counter will be cleared.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen().

Returns:

ML_SUCCESS if successful, non-zero error code otherwise

5.13.2.2 int NavWalkDisable ()

Disable the navWalk.

Should be called after MLOpen(MLNavWalkOpen);

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.13.2.3 int NavWalkEnable ()

Enables the navWalk.

Should be called after MLOpen(MLNavWalkOpen);

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.13.2.4 int NavWalkGetNumOfSteps (unsigned int * *steps*)

Used retrieve the number of steps taken by the user.

Typically it is used for polling mode.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen().

Parameters:

steps The number of steps taken by the user.

Returns:

ML_SUCCESS if successful, non-zero error code otherwise

5.13.2.5 float NavWalkHeading ()

Returns the angle in radians of the heading.

The angle from the world frame to the body when rotating only about Z

5.13.2.6 int NavWalkSetDataRate (int *dataRate*)

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Parameters:

dataRate Should be same number as passed with MLSetFIFORate()

5.13.2.7 int NavWalkSetStepCallback (void(∗)(unsigned short stepNum) *func*)

Used to register a callback function that will trigger when the user takes a step.

MLNavWalkSetInterrupt() must also be called to tie this to an interrupt or it must be setup to generate a FIFO interrupt.

Precondition:

MLDmpOpen() must be called with MLDmpDefaultOpen().
MLDmpStart() must **NOT** have been called.

Parameters:

func An user defined callback function with an argument of the step number (stepNum). NULL will turn off callback.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.13.2.8 int NavWalkSetStrideLength (unsigned short *strideLength*)

Used to set the user's stride length.

Value will be used in the calorie calculation.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen().

Parameters:

strideLength User stride length in centimeters

Returns:

ML_SUCCESS if successful, non-zero error code otherwise

5.14 GESTURE

Motion Library - Gesture Engine.

Classes

- struct tGesture

  Gesture data structure.

Modules

- TAP

  Motion Library - Gesture Engine - Tap Detection Algorithm.

- SHAKE

  Motion Library - Gesture Engine - Shake Detection Algorithm.

- YAW_ROTATE

  Motion Library - Gesture Engine - Yaw Rotation Detection Algorithm.

Files

- file gesture.c

  Gesture Library Implemenation.

Functions

- int MLDisableGesture (void)

  Used to disable gesture features.

- int MLEnableGesture (void)

  Used to enable the gesture features.

- int MLGetGesture (tGesture *gesture)

  Used to retrieve the most recent gesture information.

- int MLGetGestureState (int *state)

  Used to retrieve the state of the gesture engine.

- int MLSetGestureCallback (void(∗callback)(tGesture ∗gesture))

  Used to register a callback function that will trigger when a gesture is detected.

- int MLSetGestures (unsigned short gestures)

  Used to register which gestures will trigger the user defined callback function.

5.14.1 Detailed Description

Motion Library - Gesture Engine.

The Gesture Library processes gyroscopes and accelerometers to provide recognition of a set of gestures. These include tapping, shaking along various axes, and rotation about a horizontal axis.

5.14.2 Function Documentation

5.14.2.1 int MLDisableGesture (void)

Used to disable gesture features.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Returns:

- ML_SUCCESS if the gestures were successfully disabled.
- Non-zero error code otherwise.

5.14.2.2 int MLEnableGesture (void)

Used to enable the gesture features.

Will setup default values for control parameters.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Returns:

- ML_SUCCESS if successful

- Non-zero error code otherwise.

5.14.2.3 int MLGetGesture (tGesture * *gesture*)

Used to retrieve the most recent gesture information.

The flag GOT_GESTURE should be checked prior to executing this funtion.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() and then MLDmpStart().

Parameters:

gesture A pointer to a tGesture data structure. Do not try to free this pointer.

Returns:

- ML_SUCCESS if able to retrieve the gesture.
- Non-zero error code otherwise

5.14.2.4 int MLGetGestureState (int * *state*)

Used to retrieve the state of the gesture engine.

When the gesture engine detects a high probability gesture event this function will return ML_STATE_RUNNING, indicating that it is actively processing incoming data to determine if a programmed gesture has occured.

Otherwise this fucntion will return ML_STATE_IDLE.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() and then MLDmpStart().

Parameters:

state Sets this value to:
- ML_STATE_IDLE
- ML_STATE_RUNNING if a recent event occured.

Returns:

- ML_SUCCESS correctly called
- Non-zero error code otherwise

5.14.2.5 int MLSetGestureCallback (void(∗)(tGesture ∗gesture) *callback*)

Used to register a callback function that will trigger when a gesture is detected.

The structure returned will be of type tGesture. The first two bytes of this structure specify the type of gesture being returned. For more information please see the following documentaiton for the following gestures

- ML_PITCH_SHAKE tGestureShake
- ML_ROLL_SHAKE tGestureShake
- ML_YAW_SHAKE tGestureShake
- ML_TAP tGestureTap
- ML_YAW_IMAGE_ROTATE tGestureYawImageRotate

Note:

Do not free pointer in callback. The internals of these functions own the data.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Parameters:

callback A function that will be called when a gesture is received.

Returns:

- ML_SUCCESS if successful
- Non-zero error code otherwise.

5.14.2.6 int MLSetGestures (unsigned short *gestures*)

Used to register which gestures will trigger the user defined callback function.

Parameters:

gestures A gesture or bitwise OR of gestures to be detected.

- ML_TAP Detects when the device containing the sensors is tapped.
- ML_PITCH_SHAKE Detects when the device containing the sensors is rotated quickly around the roll axis.
- ML_ROLL_SHAKE Detects when the device containing the sensors is shaken along the pitch axis.

- ML_YAW_SHAKE Detects when the device containing the sensors is shaken along the yaw axis.
- ML_YAW_IMAGE_ROTATE Detects when the device containing the sensors is rotated horizontally through a given angle in a given amount of time.
- ML_SHAKE_ALL Equivalent to ML_PITCH_SHAKE | ML_ROLL_SHAKE | ML_YAW_SHAKE.
- ML_GESTURE_ALL Equivalent to ML_TAP | ML_PITCH_SHAKE | ML_ROLL_SHAKE | ML_YAW_SHAKE | ML_YAW_IMAGE_ROTATE.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Returns:

- ML_SUCCESS if successful
- Non-zero error code otherwise.

5.15 TAP

Motion Library - Gesture Engine - Tap Detection Algorithm.

Classes

- struct tGestureTap

  Tap gesture data structure.

Functions

- int MLResetTap (void)

  Resets the number of taps detected to 0 and all timers associated with detecting multi taps.

- int MLSetMaxTaps (unsigned short max)

  Sets the maximum nuber of taps to detect before reporting and resetting the count.

- int MLSetNextTapTime (unsigned short time)

  Used to set the time interval below which the number of taps detected will increase.

- tMLError MLSetTapInterrupt (unsigned char on)

  Enable generation of the DMP interrupt when a tap occurs.

- int MLSetTapThreshByAxis (unsigned int axis, unsigned short threshold)

  Used to set the threshold for detecting a tap.

- int MLSetTapTime (unsigned short time)

  Used to set the time interval above which the tap number will increase.

5.15.1 Detailed Description

Motion Library - Gesture Engine - Tap Detection Algorithm.

Tap allows detection of one or more sequential taps. Call MLEnableGesture() and then MLSetGestures() using ML_TAP to enable tap detection.

See also:

GESTURE

5.15.2 Function Documentation

5.15.2.1 int MLResetTap (void)

Resets the number of taps detected to 0 and all timers associated with detecting multi taps.

Note:

This will not change the tap parameters.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Returns:

- ML_SUCCESS if able to reset tap.
- Non-zero error code otherwise.

5.15.2.2 int MLSetMaxTaps (unsigned short *max*)

Sets the maximum nuber of taps to detect before reporting and resetting the count.

The count will also be reset if the time specified to MLSetNextTapTime() expires before a new tap is detected. Use 0 to reset Max Taps to its default.

Note:

Taps are reported immediately once the max number of taps have been detected. Setting this value properly can help reduce the lag time between taping a device and having that tap reported.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

max Maximum number of taps to report. 0 sets the default of 2.
- Range: 0-65536 Taps
- Reccomended Range: 1-3 Taps

Returns:

- ML_SUCCESS if able to set the maximum taps.
- Non-zero error code otherwise.

5.15.2.3 int MLSetNextTapTime (unsigned short *time*)

Used to set the time interval below which the number of taps detected will increase.

Allows a user to adjust the timing required for detecting double tap, triple tap, and so on. If an additional tap is detected before this time expires, the tap number will increment; when this time expires, the tap number will be set to zero.

Note:

This value should be greater than the value set by MLSetTapTime(). If it is not, each tap will be reported independently. This is similar to setting MLSetMaxTaps() to 1.
Taps will not be reported until after this time has elapsed since the tap. The larger this value, the longer the lag time between detecting a tap and reporting it. Each new tap resets the timer.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

time The time interval required for the tap number to increase in milliseconds.

- Range: 0-65536ms
- Reccomended Range: 300-500ms

Returns:

- ML_SUCCESS if able to reset tap.
- Non-zero error code otherwise.

5.15.2.4 tMLError MLSetTapInterrupt (unsigned char *on*)

Enable generation of the DMP interrupt when a tap occurs.

Parameters:

on Boolean to turn the interrupt generation on or off

Returns:

ML_SUCCESS or non-zero error code

5.15.2.5 int MLSetTapThreshByAxis (unsigned int *axis*, unsigned short *threshold*)

Used to set the threshold for detecting a tap.

Allows a user to adjust the sensitivity of the tap detection algorithm for each axis independently.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

axis The axis to change. One or or more of

- ML_TAP_AXIS_X
- ML_TAP_AXIS_Y
- ML_TAP_AXIS_Z

threshold The threshold for detecting a tap. The larger the number the stronger the tap must be.

- Range: 0-32768
- 50: light tap
- 100: Average Tap
- 200: Heavy tap

Returns:

- ML_SUCCESS if able to set the tap threshold.
- Non-zero error code otherwise.

5.15.2.6 int MLSetTapTime (unsigned short *time*)

Used to set the time interval above which the tap number will increase.

Allows a user to adjust the timing required for detecting double tap, triple tap, and so on. If an additional tap is detected before this time expires, the tap number will not increment; when this time expires another tap number can increment up until the time set by MLSetNextTapTime().

Note:

For best performance this value should be set between 100ms and 500ms. When streaming data using the fifo, if this value is set to a value less than the time between fifo updates only one tap will be reported per fifo update, and hence some taps can be missed.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

time Time in milliseconds before tap will be registered.

- Range: 0-65536ms
- Reccomended Range: 100-500ms

Returns:

- ML_SUCCESS if able to set the tap time.
- Non-zero error code otherwise.

5.16 SHAKE

Motion Library - Gesture Engine - Shake Detection Algorithm.

Classes

- struct tGestureShake
  Shake gesture data structure.

Functions

- int MLResetShake (int axis)
  Reset the shake state of the specified axis or axes.
- int MLSetHardShakeThresh (unsigned short axis, unsigned short threshold)
  Used to set the threshold for detecting a shake.
- int MLSetMaxShakes (int axis, int max)
  Used to set the time maximum number of shakes to detect before resetting.
- int MLSetNextShakeTime (unsigned short time)
  Used to set the time interval below which the shake number will increase.
- int MLSetShakeFunc (unsigned short function)
  Used to set the types of shakes to detect.
- tMLError MLSetShakePitchInterrupt (unsigned char on)
  Enable generation of the DMP interrupt when a pitch shake occurs.
- tMLError MLSetShakeRollInterrupt (unsigned char on)
  Enable generation of the DMP interrupt when a roll shake occurs.
- int MLSetShakeThresh (unsigned short axis, unsigned short threshold)
  Used to set the threshold for detecting a shake.
- int MLSetShakeTime (unsigned short time)
  Used to set the delay before a shake will be registered.
- tMLError MLSetShakeYawInterrupt (unsigned char on)
  Enable generation of the DMP interrupt when a yaw shake occurs.

5.16.1 Detailed Description

Motion Library - Gesture Engine - Shake Detection Algorithm.

Shake allows detection of one or more sequential shakes. Call MLEnableGesture() and then MLSetGestures() using ML_PITCH_SHAKE, ML_ROLL_SHAKE, ML_YAW_-SHAKE, and/or ML_SHAKE_ALL to enable shake detection.

See also:

GESTURE

5.16.2 Function Documentation

5.16.2.1 int MLResetShake (int *axis*)

Reset the shake state of the specified axis or axes.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

axis The Axis or Axes to reset Can be one or more (bitwise-or) of the following:

- ML_PITCH_SHAKE
- ML_ROLL_SHAKE
- ML_YAW_SHAKE
- ML_SHAKE_ALL == ML_PITCH_SHAKE | ML_ROLL_SHAKE | ML_YAW_SHAKE

Returns:

- ML_SUCCESS if able to reset shake.
- Non-zero error code otherwise.

5.16.2.2 int MLSetHardShakeThresh (unsigned short *axis*, unsigned short *threshold*)

Used to set the threshold for detecting a shake.

Parameters:

axis The shake axis being modified. Must be one of:

- ML_PITCH_SHAKE,

- ML_ROLL_SHAKE, or
- ML_YAW_SHAKE.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

threshold The threshold for detecting a snap. The smaller the number the more sensitive the detection. Most snaps will not be detected with a threshold above 300.

- Range: 0-65536
- Reccomended Range: 50-300.

Returns:

- ML_SUCCESS if able to set the snap threshold.
- Non-zero error code otherwise.

5.16.2.3 int MLSetMaxShakes (int *axis*, int *max*)

Used to set the time maximum number of shakes to detect before resetting.

After the maximum number of shakes have been detected the shake state machine is reset so that the next shake detected with be shake number 1

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

axis The axis or axes to set. Can be one or more (bitwise-or) of the following:

- ML_PITCH_SHAKE
- ML_ROLL_SHAKE
- ML_YAW_SHAKE
- ML_SHAKE_ALL == ML_PITCH_SHAKE | ML_ROLL_SHAKE | ML_YAW_SHAKE

max The maximum number of shakes to detect before resetting on these axes.

Returns:

- ML_SUCCESS if able to set maximum shakes.
- Non-zero error code otherwise.

5.16.2.4 int MLSetNextShakeTime (unsigned short *time*)

Used to set the time interval below which the shake number will increase.

Time allows a user to adjust the timing required for detecting double shake, triple shake, and so on. If an additional shake is detected before this time expires, the shake number will increment; when this time expires, the shake number will be set to zero.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

time The time interval in milliseconds required for the shake number to increase.

- Range: 0-65536ms
- Reccomended Range: 200-800ms

Returns:

- ML_SUCCESS if able to set the next shake time.
- Non-zero error code otherwise.

5.16.2.5 int MLSetShakeFunc (unsigned short *function*)

Used to set the types of shakes to detect.

Parameters:

function The shake detection functions. Must be one of mutually exclusive **ML_SOFT_SHAKE** and **ML_HARD_SHAKE**, and one of mutully exclusive **ML_NO_RETRACTION** and **ML_RETRACTION**. For example, **ML_HARD_SHAKE** | **ML_RETRACTION** is a valid combination, providing the most robust (and least sensitive) shake algorithm. The possible function choices are:

- ML_SOFT_SHAKE will maximize the sensitivity of the shake detection function.
- ML_HARD_SHAKE will maximize the robustness of the shake detection function.
- ML_NO_RETRACTION allows single direction pulses to register as shakes.
- ML_RETRACTION requires, for single shake,that the user move in one direction and than return.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Returns:

- ML_SUCCESS if able to set the shake functioin.
- Non-zero error code otherwise.

5.16.2.6 tMLError MLSetShakePitchInterrupt (unsigned char *on*)

Enable generation of the DMP interrupt when a pitch shake occurs.

Parameters:

on Boolean to turn the interrupt generation on or off

Returns:

ML_SUCCESS or non-zero error code

5.16.2.7 tMLError MLSetShakeRollInterrupt (unsigned char *on*)

Enable generation of the DMP interrupt when a roll shake occurs.

Parameters:

on Boolean to turn the interrupt generation on or off

Returns:

ML_SUCCESS or non-zero error code

5.16.2.8 int MLSetShakeThresh (unsigned short *axis*, unsigned short *threshold*)

Used to set the threshold for detecting a shake.

Parameters:

axis The shake axis being modified. Must be one of:

- ML_PITCH_SHAKE,
- ML_ROLL_SHAKE, or
- ML_YAW_SHAKE.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

threshold The threshold for detecting a shake. The smaller the number the more sensitive the detection. Most shakes will not be detected with a threshold above 300.

- Range: 0-65536
- Reccomended Range: 50-300.

Returns:

- ML_SUCCESS if able to set the shake threshold.
- Non-zero error code otherwise.

5.16.2.9 int MLSetShakeTime (unsigned short *time*)

Used to set the delay before a shake will be registered.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

time The delay before a shake will be registered in milliseconds.

- Range: 0-65536ms
- Reccomended Range: 100-500ms

Returns:

- ML_SUCCESS if able to set the shake time.
- Non-zero error code otherwise.

5.16.2.10 tMLError MLSetShakeYawInterrupt (unsigned char *on*)

Enable generation of the DMP interrupt when a yaw shake occurs.

Parameters:

on Boolean to turn the interrupt generation on or off

Returns:

ML_SUCCESS or non-zero error code

5.17 YAW_ROTATE

Motion Library - Gesture Engine - Yaw Rotation Detection Algorithm.

Classes

- struct tGestureYawImageRotate

  Yaw image rotate gesture data structure.

Functions

- int MLGetYawRotation ()

  Used to get the image rotation integral.

- int MLSetYawRotateThresh (unsigned short threshold)

  Used to set the threshold for detecting a yaw image rotation.

- int MLSetYawRotateTime (unsigned short time)

  Used to set the time threshold for detecting a new yaw image rotation.

5.17.1 Detailed Description

Motion Library - Gesture Engine - Yaw Rotation Detection Algorithm.

Yaw allows detection of one ore more sequential yaw roations. Call MLEnableGesture() and then MLSetGestures() using ML_YAW_IMAGE_ROTATE to enable yaw rotation detection.

See also:

GESTURE

5.17.2 Function Documentation

5.17.2.1 int MLGetYawRotation ()

Used to get the image rotation integral.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Returns:

Image rotation integral in degrees. If an error occurs 0 will be returned.

5.17.2.2 int MLSetYawRotateThresh (unsigned short *threshold*)

Used to set the threshold for detecting a yaw image rotation.

Allows a user to adjust the rotation angle required for detecting a yaw image rotation.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

threshold The threshold for detecting a yaw image rotation in degrees.

- Range: 0-65536 degrees
- Reccomended Range 45-90 degrees

Returns:

- ML_SUCCESS if able to set the rotate threshold.
- Non-zero error code otherwise.

5.17.2.3 int MLSetYawRotateTime (unsigned short *time*)

Used to set the time threshold for detecting a new yaw image rotation.

Allows a user to adjust the time between detecting multiple yaw image rotations. Once a rotation is detected, it will be reported after this time has elapsed. If a new yaw rotation is detected before this time has elapsed the timer will be reset and the new yaw rotation will be reported after the time has elapsed since the new yaw rotation.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen()

Parameters:

time The time threshold in milliseconds before reporting.

- Range: 0-1000ms
- Reccomended range 50-500ms

Returns:

- ML_SUCCESS if able to set the rotate time.
- Non-zero error code otherwise.

5.18 ORIENTATION

Motion Library - Orientation Engine.

Files

- file orientation.c

  Determines the orientation of the device.

Functions

- tMLError MLDisableOrientation (void)

  Turns off the orientation feature.

- tMLError MLEnableOrientation (void)

  Turns on the orientation feature.

- tMLError MLGetOrientation (int ∗orientation)

  Gets the last reported orientation.

- tMLError MLGetOrientationState (int ∗state)

  Used to retrieve the state of the orientation engine.

- tMLError MLSetOrientationCallback (void(∗callback)(unsigned short))

  Sets the callback for the orientation changes.

- tMLError MLSetOrientationInterrupt (unsigned char on)

  turns on interrupt generation when there is an orientation change

- tMLError MLSetOrientations (int orientation)

  Used to register which orientations will trigger the user defined callback function.

- tMLError MLSetOrientationThreshold (float angle, float hysteresis, unsigned long time, unsigned int axis)

  Sets the threshold for the orientation angle, hysteresis and time.

5.18.1 Detailed Description

Motion Library - Orientation Engine.

Report when the orientation of the device changes.

Orientation functions are available after calling MLDmpOpen() with either MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen().

- `MLDmpOpen( MLDmpDefaultOpen );`
- `MLDmpOpen( MLDmpPedometerStandAloneOpen );`

These functions allow for changes in orientation to be detected. To use, first open using MLEnableOrientation() then set the list of orientations to be detected using MLSetOrientations(), then set the callback with which to be notified when the orientation changes using MLSetOrientationCallback(). See example below:

```
void OrientationCallback(unsigned short newOrientation)
{
   // Do something with newOrientation
}

// ...

// Set up orientation
result = MLEnableOrientation();
if (ML_SUCCESS != result)
{
    // Handle error condition
}

// Enable detection of all orientations
result = MLSetOrientations(ML_ORIENTATION_ALL);
if (ML_SUCCESS != result)
{
    // Handle error condition
}

result = MLSetOrientationCallback(OrientationCallback);
if (ML_SUCCESS != result)
{
    // Handle error condition
}
```

5.18.2 Function Documentation

5.18.2.1 tMLError MLDisableOrientation (void)

Turns off the orientation feature.

Returns:

- ML_SUCCESS if successful
- Non-zero error code on failure.

5.18.2.2 tMLError MLEnableOrientation (void)

Turns on the orientation feature.

This will also reset any other orientation function calls. Call before any other orientation function.

Precondition:

MLDmpOpen() Must be called with either MLDmpDefaultOpen() or MLPedometerStandAloneOpen() before calling this function. MLDmpStart() must **NOT** have been called.

Returns:

- ML_SUCCESS if successful
- Non-zero error code on failure.

5.18.2.3 tMLError MLGetOrientation (int * *orientation*)

Gets the last reported orientation.

Can also be used to get the inital orientation in case the device starts in the initial orientation.

Parameters:

orientation One of

- ML_X_UP,
- ML_X_DOWN,
- ML_Y_UP,
- ML_Y_DOWN,
- ML_Z_UP,
- ML_Z_DOWN

Returns:

- ML_SUCCESS if successful
- Non-zero error code on failure.

5.18.2.4 tMLError MLGetOrientationState (int * *state*)

Used to retrieve the state of the orientation engine.

When the orientation engine detects a high probability orientation event, This function will return ML_STATE_RUNNING, indicating that it is actively processing incoming data to determine if a programmed orientation has occured.

Otherwise this fucntion will return ML_STATE_IDLE.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() and then MLDmpStart().

Parameters:

state Sets this value to:

- ML_STATE_IDLE
- ML_STATE_RUNNING if a recent event occured.

Returns:

- ML_SUCCESS correctly called
- Non-zero error code otherwise

5.18.2.5 tMLError MLSetOrientationCallback (void(*)(unsigned short) *callback*)

Sets the callback for the orientation changes.

Must be called after MLEnableOrientation(). The callback function will have passed as its arguement the new orientation as one of:

- ML_X_UP,
- ML_X_DOWN,
- ML_Y_UP,
- ML_Y_DOWN,
- ML_Z_UP,
- ML_Z_DOWN

Parameters:

callback The callback you want to use for orientation callbacks. Only one callback can be used.

Returns:

- ML_SUCCESS if successful
- Non-zero error code on failure.

5.18.2.6 tMLError MLSetOrientationInterrupt (unsigned char *on*)

turns on interrupt generation when there is an orientation change

Parameters:

on Boolean to turn the error code on or off

Returns:

ML_SUCCESS or non-zero error code

5.18.2.7 tMLError MLSetOrientations (int *orientation*)

Used to register which orientations will trigger the user defined callback function.

Allows a user to register which orientations will trigger the user defined callback function. Zero is returned if the command is successful; otherwise, an ML error code is returned.

Precondition:

MLDmpOpen() Must be called with either MLDmpDefaultOpen() or MLPedometerStandAloneOpen() before calling this function.

Parameters:

orientation An orientation or bitwise OR of orientations to be detected. The orientations are:

- ML_X_UP,
- ML_X_DOWN,
- ML_Y_UP,
- ML_Y_DOWN,
- ML_Z_UP,
- ML_Z_DOWN, and
- ML_ORIENTATION_ALL: Bitwise or of all previous orientations.

Returns:

ML_SUCCESS if successful, a non-zero error code on failure.

5.18.2.8 tMLError MLSetOrientationThreshold (float *angle*, float *hysteresis*, unsigned long *time*, unsigned int *axis*)

Sets the threshold for the orientation angle, hysteresis and time.

The angle is defined to be the angle that the gravity vector makes with the normal vector of the body plane in plane of the axis. I.E. if the body plane is (0, 0.5, 0.866) the angles in the respective planes are (0, 30, 60).

Hysteresis is the minimum amount of angle by which the old orientation difference has to exceed the new orientation to cause a change. This value is used by first converting to a vector and then performing the threshold on the gravity vector. Thus the hysteresis value is the minimum angle and is is non-linear with the angle. If y is up, Z is 0, and a hysteresis of 5 degrees (vector sin(5) == 0.087g), x will be up when the gravity vector is (0.749, 0.662, 0) corresponding to (48.5 41.5, 0), with the actual hysteresis being 7 degrees.

The hysteresis is used to determine which axis is up in areas of ambiguity. It is also possible to create an area where no axis is up. This is a dead zone. In this case the orientation will trigger an interrupt if configured and try to determine the axis that is most up. These interrupts and callbacks will continue to fire until one of the axes crosses a threshold.

Parameters:

angle in degrees

hysteresis minimum value in degrees

time time in ms that the device must remain in the new orientation to register an orientation change

axis Which axis to set this threshold for

- ML_X_AXIS
- ML_Y_AXIS
- ML_Z_AXIS

Returns:

ML_SUCCESS or non-zero error code

5.19 ML_PEDESTRIAN_NAVIGATION

Motion Library - Pedestrian Navigation Engine.

Files

- file pedestrian_navigation.c

  Pedestrian Naivigation source file.

Functions

- int MLEnablePedestrianNavigation ()

  Enables the pedestrian navigation engine.

- void MLPedestrianNavigationGetUserLocation (float *x, float *y, float *heading)

  polls a user position

- int MLPedestrianNavigationSetCallback (void(*func)(float x, float y, float heading))

  Sets a callback to be called when the user position is detected as to have changed.

- int MLPedestrianNavigationSetHeading ()

  Resets the heading to current heading location.

- int MLPedestrianNavigationSetPosition (float x, float y)

  Sets the current location in meters.

- int MLPedestrianNavigationSetStepSize (float stepsize)

  Sets the user's step size in centimeters.

5.19.1 Detailed Description

Motion Library - Pedestrian Navigation Engine.

The Pedestrian navigation enginer keeps track of steps, direction, heading, and distance.

This is module is mutually exclusive with ML_PEDOMETER. It sits on top of ML_PEDOMETER and computes the user location. Other functions can be used in ML_PEDOMETER, but not MLEnablePedometer() and MLDisablePedometer().

5.19.2 Function Documentation

5.19.2.1 int MLEnablePedestrianNavigation ()

Enables the pedestrian navigation engine.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.19.2.2 void MLPedestrianNavigationGetUserLocation (float * *x*, float * *y*, float * *heading*)

polls a user position

Parameters:

x user's x position in meters.

y user's y position in meters.

heading user's heading in radians.

5.19.2.3 int MLPedestrianNavigationSetCallback (void(*)(float x, float y, float heading) *func*)

Sets a callback to be called when the user position is detected as to have changed.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Parameters:

func Callback function to call when event occurs. x,y in meters heading in radians.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.19.2.4 int MLPedestrianNavigationSetHeading ()

Resets the heading to current heading location.

Useful for relative navigation.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.19.2.5 int MLPedestrianNavigationSetPosition (float *x*, float *y*)

Sets the current location in meters.

Parameters:

x Current x position in meters

y Current y position in meters

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.19.2.6 int MLPedestrianNavigationSetStepSize (float *stepsize*)

Sets the user's step size in centimeters.

Parameters:

stepsize Step Size in centimeters

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.20 ML_PEDOMETER

Motion Library - Pedometer Engine.

Files

- file pedometer.c

  pedometer source file.

Functions

- tMLError MLClearNumOfSteps ()

  Used to clear the pedometer's step counter.

- tMLError MLDisablePedometer ()

  Disable the pedometer.

- tMLError MLEnablePedometer ()

  Enables the pedometer.

- tMLError MLGetNumOfSteps (unsigned int *steps)

  Used retrieve the number of steps taken by the user.

- tMLError MLPedometerSetDataRate (int dataRate)
- tMLError MLPedometerSetDirection (int dir)

  Sets direction that the user is walking.

- tMLError MLSetStepCallback (void(*func)(unsigned short stepNum))

  Used to register a callback function that will trigger when the user takes a step.

- tMLError MLSetStrideLength (unsigned short strideLength)

  Used to set the user's stride length.

5.20.1 Detailed Description

Motion Library - Pedometer Engine.

The Pedometer application counts steps by the user.

The pedometer application requires computation on the host. This pedometer generally has better accuracy than than PedometerStandAlone which runs all on the DMP.

5.20.2 Function Documentation

5.20.2.1 tMLError MLClearNumOfSteps ()

Used to clear the pedometer's step counter.

The step counter will be cleared.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen().

Returns:

ML_SUCCESS if successful, non-zero error code otherwise

5.20.2.2 tMLError MLDisablePedometer ()

Disable the pedometer.

Should be called after MLOpen(MLPedometerOpen);

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must have been called.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.20.2.3 tMLError MLEnablePedometer ()

Enables the pedometer.

Should be called after MLOpen(MLPedometerOpen);

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.20.2.4 tMLError MLGetNumOfSteps (unsigned int * *steps*)

Used retrieve the number of steps taken by the user.

Typically it is used for polling mode.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen().

Parameters:

steps The number of steps taken by the user.

Returns:

ML_SUCCESS if successful, non-zero error code otherwise

5.20.2.5 tMLError MLPedometerSetDataRate (int *dataRate*)

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Parameters:

dataRate Should be same number as passed with MLSetFIFORate()

5.20.2.6 tMLError MLPedometerSetDirection (int *dir*)

Sets direction that the user is walking.

Parameters:

dir dir=0 Means the user is walking along the device's positive y (default) dir=1 Means the user is walking along the device's positve x

5.20.2.7 tMLError MLSetStepCallback (void(*)(unsigned short stepNum) *func*)

Used to register a callback function that will trigger when the user takes a step.

MLPedometerSetInterrupt() must also be called to tie this to an interrupt or it must be setup to generate a FIFO interrupt.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen(). MLDmpStart() must **NOT** have been called.

Parameters:

func A user defined callback function with an argument of the step number (stepNum). NULL will turn off callback.

Returns:

ML_SUCCESS if succesful, or non-zero error code.

5.20.2.8 tMLError MLSetStrideLength (unsigned short *strideLength*)

Used to set the user's stride length.

Value will be used in the calorie calculation.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen().

Parameters:

strideLength User stride length in centimeters

Returns:

ML_SUCCESS if successful, non-zero error code otherwise

5.21 GLYPH

Motion Library's Characters Recognition Engine.

Classes

- struct tMLGlyphData

  Describes the data to be used by the character recognition algorithm.

Files

- file mlglyph.c

  The Control Library.

Functions

- tMLError MLAddGlyph (unsigned short glyphID)

  Adds a new glyph.

- tMLError MLBestGlyph (unsigned short *finalGlyph)

  Finds the best match for a glyph.

- tMLError MLClearGlyph (void)

  Clears the glyph.

- tMLError MLDisableGlyph (void)

  Disables the glyph engine.

- tMLError MLEnableGlyph (void)

  Enables the glyph engine.

- tMLError MLGetGlyph (int index, int *x, int *y)

  Returns a trajectory data point.

- tMLError MLGetGlyphLength (unsigned short *length)

  Returns the glyph length.

- tMLError MLGetLibraryLength (unsigned short *length)

  Returns the length of the library of glyphs.

- tMLError MLLoadGlyphs (unsigned char ∗libraryData)

  Loads a library of glyphs.

- tMLError MLResetGlyphLibrary ()

  Resets glyph library.

- tMLError MLSetGlyphProbThresh (unsigned short prob)

  Sets the minimum recognition probability.

- tMLError MLSetGlyphSpeedThresh (unsigned short speed)

  Sets the glyph speed threshold.

- tMLError MLStartGlyph (void)

  Turns on motion tracking.

- tMLError MLStopGlyph (void)

  Turns off motion tracking.

- tMLError MLStoreGlyphs (unsigned char ∗libraryData, unsigned short ∗length)

  Stores a library of glyphs.

5.21.1 Detailed Description

Motion Library's Characters Recognition Engine.

The glyph library process a series of user trajectories and stores them for later reconition or compares them against previously stored glyphs.

The library has two main purpose: training the glyph patterns, and recognizing the stored pattens.

To run the functions in ML_GLYPH library, user first has to call MLEnableGlyph(). Then depends on the mode the code could look something like:

```
[...]
MLEnableGlyph();
[...]
if (TRAINING_MODE) {
    MLStartGlyph();
    [...]
    MLAddGlyph(GlyphID);
    MLClearGlyph();
    MLStopGlyph();
```

```
    MLStoreGlyphs(LIBRARY_DATA);
} else {
    MLLoadGlyphs(LIBRARY_DATA);
    [...]
    MLStartGlyph();
    [...]
    MLBestGlyph(RECOGNIZED_CHARACTER);
    MLClearGlyph();
    MLStopGlyph();
}
[...]
```

5.21.2 Function Documentation

5.21.2.1 tMLError MLAddGlyph (unsigned short *glyphID*)

Adds a new glyph.

MLAddGlyph extends the library by adding a newly created glyph.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

glyphID The glyph ID to be associated with this glyph.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.2 tMLError MLBestGlyph (unsigned short * *finalGlyph*)

Finds the best match for a glyph.

Processes all glyphs stored in the library and returns the best match for the user's trajectory.

If a best match cannot be found or the minimum probability for the trajectory was not met (if a minimum probability threshold for the glyph was provided via MLSetGlyphProbThresh()), finalGlyph will point to an invalid glyph instance and the function signal it by returning a non-zero error code.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

finalGlyph The glyph ID associated with the matched glyph.

Returns:

ML_SUCCESS is returned if the command is successful; otherwise, a non-zero error code is returned.

5.21.2.3 tMLError MLClearGlyph (void)

Clears the glyph.

MLClearGlyph resets the glyph trajectory.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.4 tMLError MLDisableGlyph (void)

Disables the glyph engine.

MLDisableGlyph disables the glyph recognition engine.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen(). MLDmpStart() must **NOT** have been called yet.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.5 tMLError MLEnableGlyph (void)

Enables the glyph engine.

MLEnableGlyph enables the glyph recognition engine.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen(). MLDmpStart() must **NOT** have been called yet.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.6 tMLError MLGetGlyph (int *index*, int * *x*, int * *y*)

Returns a trajectory data point.

MLGetGlyph returns the value of the glyph trajectory at a given data point. It can be used for displaying the trajectory in real time for user feedback.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

index The index of the trajectory data point.

x The x value of the trajectory data point.

y The y value of the trajectory data point.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.7 tMLError MLGetGlyphLength (unsigned short * *length*)

Returns the glyph length.

MLGetGlyphLength returns the number of data points in the glyph trajectory.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

length The length of the glyph trajectory.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.8 tMLError MLGetLibraryLength (unsigned short * *length*)

Returns the length of the library of glyphs.

Should be called before allocating the memory required to store this data to a file.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

length The length of the library.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.9 tMLError MLLoadGlyphs (unsigned char * *libraryData*)

Loads a library of glyphs.

MLLoadGlyph parses a binary data set containing a library of glyphs. The binary data set is intended to be loaded from a file.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

libraryData A pointer to an array of bytes to be parsed.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.10 tMLError MLResetGlyphLibrary ()

Resets glyph library.

MLResetGlyphLibrary clears the glyph library of all data.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.11 tMLError MLSetGlyphProbThresh (unsigned short *prob*)

Sets the minimum recognition probability.

MLSetGlyphProbThresh sets the minimum probability required for returning a successful glyph recognition.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

prob The minimum recognition probability

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.12 tMLError MLSetGlyphSpeedThresh (unsigned short *speed*)

Sets the glyph speed threshold.

MLSetGlyphSpeedThresh determines the minimum speed at which the user must move in order to influence the glyph trajectory.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

speed The minimum speed threshold

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.13 tMLError MLStartGlyph (void)

Turns on motion tracking.

MLStartGlyph enables tracking of the user's trajectory, and is intended to be associated with a button press.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.14 tMLError MLStopGlyph (void)

Turns off motion tracking.

MLStopGlyph disables tracking of the user's trajectory, and is intended to be associated with a button release.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.21.2.15 tMLError MLStoreGlyphs (unsigned char * *libraryData*, unsigned short * *length*)

Stores a library of glyphs.

MLStoreGlyph generates a binary data set containing a library of glyphs. The binary data set is intended to be stored to a file.

Precondition:

MLDmpOpen() Must be called with MLDmpDefaultOpen() or MLDmpPedometerStandAloneOpen()

Parameters:

libraryData A pointer to an array of bytes to be stored.

length The length of the array of bytes.

Returns:

Zero is returned if the command is successful; otherwise, an error code is returned.

5.22 ML_PEDOMETER_STAND_ALONE

Motion Library - Pedometer Stand-Alone Engine.

Files

- file pedometerStandAlone.c

  pedometer source file.

5.22.1 Detailed Description

Motion Library - Pedometer Stand-Alone Engine.

The Pedometer application counts steps by the user. The pedometer application can not be used with most of the gesture and tap features or the cross axis support. The use of gyros are optional and may be turned off for power savings.

5.23 PEDOMETER

Motion Library - Full Power Pedometer Engine.

Files

- file mlpedometer_fullpower.c

  Motion Library - Full Power Pedometer Engine.

- file mlpedometer_lowpower.c

  Motion Library - Low Power Pedometer Engine.

Functions

- tMLError MLDisablePedometerFullPower (void)

  Disable the pedometer engine.

- tMLError MLDmpPedometerStandAloneClose (void)

  Closes the Pedometer engine.

- tMLError MLDmpPedometerStandAloneOpen (void)

  Open up the Stand Alone Pedometer.

- tMLError MLDmpPedometerStandAloneStart (void)

  Start the DMP.

- tMLError MLDmpPedometerStandAloneStop (void)

  Stops the DMP and puts it in low power.

- tMLError MLEnablePedometerFullPower (void)

  Registers the Full power pedometer to be initialized just before starting.

- tMLError MLGetPedometerFullPowerStepCount (unsigned long *steps)

  Get the current step count.

- tMLError MLGetPedometerFullPowerWalkTime (unsigned long *timeMs)

  Get the current walk time.

- tMLError MLPedometerSetNoMotionThresh (float thresh)

  MLSetNoMotionThresh is used to set the threshold for detecting ML_NO_MOTION.

- tMLError MLPedometerSetNoMotionTime (float time)

  MLSetNoMotionTime is used to set the time required for detecting ML_NO_MOTION.

- tMLError MLPedometerStandAloneClearNumOfCalories ()

  Used to clear the pedometer's calorie counter.

- tMLError MLPedometerStandAloneGetNumOfCalories (unsigned short *calories)

  Used to calculate the number of calories burned by the user.

- tMLError MLPedometerStandAloneGetNumOfSteps (unsigned long *steps)

  Used retrieve the number of steps taken by the user.

- tMLError MLPedometerStandAloneGetWalkTime (unsigned long *time)

  Used retrieve the number of steps taken by the user.

- tMLError MLPedometerStandAloneSetNumOfSteps (unsigned long steps)

  Used to set the initial step count.

- tMLError MLPedometerStandAloneSetStepBuffer (unsigned short minSteps)

  Set the number of steps to buffer before reporting new steps.

- tMLError MLPedometerStandAloneSetStepBufferResetTime (unsigned int timeMs)

  Set the maximum time to wait for a next step to increment the buffer.

- tMLError MLPedometerStandAloneSetStrideLength (unsigned short stride-Length)

  Used to set the user's stride length.

- tMLError MLPedometerStandAloneSetWalkTime (unsigned long time)

  Used to set the initial step count.

- tMLError MLPedometerStandAloneSetWeight (unsigned short weight)

  Used to set the user's weight.

- tMLError MLSetPedometerFullPowerParams (const struct stepParams *params)

  Set the parameters to use for the full power pedometer.

- tMLError MLSetPedometerFullPowerStepBuffer (unsigned short minSteps)

  Set the number of steps to buffer before reporting new steps.

- tMLError MLSetPedometerFullPowerStepBufferResetTime (unsigned int timeMs)

  Set the maximum time to wait for a next step to increment the buffer.

- tMLError MLSetPedometerFullPowerStepCallback (void(∗func)(unsigned long stepNum, unsigned long walkTimeMs))

  Set a function to be called every time a step is detected.

- tMLError MLSetPedometerFullPowerStepCount (unsigned long steps)

  Set the number of steps taken so far.

- tMLError MLSetPedometerFullPowerWalkTime (unsigned long timeMs)

  Set the amount of time walked so far.

5.23.1 Detailed Description

Motion Library - Full Power Pedometer Engine.

Motion Library - Low Power Pedometer Engine.

The Pedometer full power engine counts steps by the user using the computation power from the application processor. The MPU's DMP processor is still used to execute some complex computation but the use of the host processor will allow to use some augmented functionalities such as the gesture engine while still countin the steps.

The Pedometer low power engine counts steps by the user offloading the computation power from the application processor to the MPU's processor. For this reason the feature supported in this mode are limited to the step counting.

5.23.2 Function Documentation

5.23.2.1 tMLError MLDisablePedometerFullPower (void)

Disable the pedometer engine.

Returns:

ML_SUCCESS or non-zero error code

5.23.2.2 tMLError MLDmpPedometerStandAloneClose (void)

Closes the Pedometer engine.

Does not close the serial communication. To do that, call MLSerialClose(). After calling MLDmpPedometerStandAloneClose() another DMP module can be loaded in the MPL with the corresponding necessary intialization and configurations, via any of the MLDmpOpenXXX functions.

Precondition:

MLDmpPedometerStandAloneOpen() must have been called.

Returns:

ML_SUCCESS, Non-zero error code otherwise.

5.23.2.3 tMLError MLDmpPedometerStandAloneOpen (void)

Open up the Stand Alone Pedometer.

You may only have one motion sensor engine open. This function is mutually exclusive with MLDmpOpen(). MLSerialOpen() should be called before calling this to open up the communication layer.

Returns:

ML_SUCCESS or non-zero error code

5.23.2.4 tMLError MLDmpPedometerStandAloneStart (void)

Start the DMP.

Precondition:

MLDmpOpen() must have been called.

Returns:

ML_SUCCESS if successful, or Non-zero error code otherwise.

5.23.2.5 tMLError MLDmpPedometerStandAloneStop (void)

Stops the DMP and puts it in low power.

Precondition:

MLDmpStart() must have been called.

Returns:

ML_SUCCESS, Non-zero error code otherwise.

5.23.2.6 tMLError MLEnablePedometerFullPower (void)

Registers the Full power pedometer to be initialized just before starting.

Note:

MLDmpStart will return ML_ERROR_INVALID_CONFIGURATION if MLSetFIFORate was called with a value 10 or greater

The full power pedometer needs to know the data rate, so initialization is delayed until the MLDmpStart is called at which time initialization is performed.

Precondition:

MLSetFIFORate must be set to a value [0..9] before MLDmpStart is called Must be called before MLDmpStart

Returns:

ML_SUCCES or non-zero error code

5.23.2.7 tMLError MLGetPedometerFullPowerStepCount (unsigned long * *steps*)

Get the current step count.

Parameters:

steps Curren step count

Returns:

ML_SUCCESS or non-zero error code

5.23.2.8 tMLError MLGetPedometerFullPowerWalkTime (unsigned long * *timeMs*)

Get the current walk time.

Parameters:

timeMs current time walking

Returns:

ML_SUCCESS or non-zero error code

5.23.2.9 tMLError MLPedometerSetNoMotionThresh (float *thresh*)

MLSetNoMotionThresh is used to set the threshold for detecting ML_NO_MOTION.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must **NOT** have been called.

Parameters:

thresh A threshold scaled in g

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.23.2.10 tMLError MLPedometerSetNoMotionTime (float *time*)

MLSetNoMotionTime is used to set the time required for detecting ML_NO_-MOTION.

Precondition:

MLDmpOpen() or MLDmpPedometerStandAloneOpen() and MLDmpStart() must **NOT** have been called.

Parameters:

time A time in seconds.

Returns:

ML_SUCCESS if successful or Non-zero error code otherwise.

5.23.2.11 tMLError MLPedometerStandAloneClearNumOfCalories ()

Used to clear the pedometer's calorie counter.

Clears the counter

Precondition:

MLDmpOpen() with MLDmpPedometerStandAlone() must have been called.

Returns:

ML_SUCCESS or non-zero error code

5.23.2.12 tMLError MLPedometerStandAloneGetNumOfCalories (unsigned short * *calories*)

Used to calculate the number of calories burned by the user.

It depends on the number of steps taken, the weight of the user, and the stride length of the user.

Precondition:

MLDmpOpen() with MLDmpPedometerStandAlone() must have been called.

Parameters:

calories number of calories buffer

Returns:

ML_SUCCESS or non-zero error code

5.23.2.13 tMLError MLPedometerStandAloneGetNumOfSteps (unsigned long * *steps*)

Used retrieve the number of steps taken by the user.

Typically it is used for polling mode.

Precondition:

MLDmpOpen() with MLDmpPedometerStandAlone() must have been called.

Parameters:

steps The number of steps taken by the user.

Returns:

ML_SUCCESS or non-zero error code

5.23.2.14 tMLError MLPedometerStandAloneGetWalkTime (unsigned long * *time*)

Used retrieve the number of steps taken by the user.

Typically it is used for polling mode.

Precondition:

MLDmpOpen() with MLDmpPedometerStandAlone() must have been called.

Parameters:

steps The number of steps taken by the user.

Returns:

ML_SUCCESS or non-zero error code

5.23.2.15 tMLError MLPedometerStandAloneSetNumOfSteps (unsigned long *steps*)

Used to set the initial step count.

If the pedometer has been off for a while, use this to set the initial step count.

Parameters:

steps The number of steps to start from.

Returns:

ML_SUCCESS or non-zero error code otherwise.

5.23.2.16 tMLError MLPedometerStandAloneSetStepBuffer (unsigned short *minSteps*)

Set the number of steps to buffer before reporting new steps.

NOTE: This is the number of steps to buffer BEFORE reporting new steps. Thus if the value is set to 5, the 6th step will be reported

Parameters:

minSteps The number of steps to buffer before reporing new steps

Returns:

ML_SUCCESS or non-zero error code

5.23.2.17 tMLError MLPedometerStandAloneSetStepBufferResetTime (unsigned int *timeMs*)

Set the maximum time to wait for a next step to increment the buffer.

Parameters:

timeMs How lon in ms

Returns:

ML_SUCCESS or non-zero error code

5.23.2.18 tMLError MLPedometerStandAloneSetStrideLength (unsigned short *strideLength*)

Used to set the user's stride length.

Value will be used in the calorie calculation.

Precondition:

MLDmpOpen() with MLDmpPedometerStandAlone() must have been called.

Parameters:

strideLength User stride length in centimeters

Returns:

ML_SUCCESS or non-zero error code

5.23.2.19 tMLError MLPedometerStandAloneSetWalkTime (unsigned long *time*)

Used to set the initial step count.

If the pedometer has been off for a while, use this to set the initial step count.

Parameters:

steps The number of steps to start from.

Returns:

ML_SUCCESS or non-zero error code otherwise.

5.23.2.20 tMLError MLPedometerStandAloneSetWeight (unsigned short *weight*)

Used to set the user's weight.

Value will be used in the calorie calculation.

Precondition:

MLDmpOpen() with MLDmpPedometerStandAlone() must have been called.

Parameters:

weight User's weight to use in kilograms

Returns:

ML_SUCCESS or non-zero error code

5.23.2.21 tMLError MLSetPedometerFullPowerParams (const struct stepParams * *params*)

Set the parameters to use for the full power pedometer.

Parameters:

params the parameters

Returns:

ML_SUCCESS or non-zero error code.

5.23.2.22 tMLError MLSetPedometerFullPowerStepBuffer (unsigned short *minSteps*)

Set the number of steps to buffer before reporting new steps.

NOTE: This is the number of steps to buffer BEFORE reporting new steps. Thus if the value is set to 5, the 6th step will be reported

Parameters:

minSteps The number of steps to buffer before reporing new steps

Returns:

ML_SUCCESS or non-zero error code

5.23.2.23 tMLError MLSetPedometerFullPowerStepBufferResetTime (unsigned int *timeMs*)

Set the maximum time to wait for a next step to increment the buffer.

Parameters:

timeMs How lon in ms

Returns:

ML_SUCCESS or non-zero error code

5.23.2.24 tMLError MLSetPedometerFullPowerStepCallback (void(∗)(unsigned long stepNum, unsigned long walkTimeMs) *func*)

Set a function to be called every time a step is detected.

Parameters:

func a pointer to a function taking an unsigned long to be called

Returns:

ML_SUCCESS or non-zero error code.

5.23.2.25 tMLError MLSetPedometerFullPowerStepCount (unsigned long *steps*)

Set the number of steps taken so far.

Parameters:

steps

Returns:

5.23.2.26 tMLError MLSetPedometerFullPowerWalkTime (unsigned long *timeMs*)

Set the amount of time walked so far.

Parameters:

timeMs time walked in ms

Returns:

ML_SUCCESS or non-zero error code

5.24 ML_SUPERVISOR_9AXIS

Advanced sensor fusion functionalities, including 9 axis compass sensor fusion and temperature bias compensation.

Files

- file mlsupervisor_9axis.c

  Advanced sensor fusion functionalities, including 9 axis compass sensor fusion and temperature bias compensation.

Functions

- void examineLargeMagField ()

  Simple check if we are in a large magnetic field and sets the mlxData.mlLargeField apropriately.

- void MLAccelCompassFusion (double magFB)

  Apply 9 axis sensor fusion.

- tMLError MLDisable9axisFusion (void)

  Disable the compass interaction with sensor fusion.

- tMLError MLEnable9axisFusion (void)

  Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

5.24.1 Detailed Description

Advanced sensor fusion functionalities, including 9 axis compass sensor fusion and temperature bias compensation.

5.24.2 Function Documentation

5.24.2.1 tMLError MLDisable9axisFusion (void)

Disable the compass interaction with sensor fusion.

Note:

The temperature compensations manager is part of the set of advanced features and is enabled as soon as any of MLEnable9AxisFusion()/MLDisable9AxisFusion() is called, regardless of whether a compass is setup in the system or not.

Returns:

ML_SUCCESS if the fusion was successfully disabled. An error code otherwise.

5.24.2.2 tMLError MLEnable9axisFusion (void)

Enable the compass interaction with sensor fusion and all other advanced proprietary motion processing features.

Note:

The temperature compensations manager is part of the set of advanced features and is enabled as soon as any of MLEnable9AxisFusion() / MLDisable9AxisFusion() is called, regardless of whether a compass is setup in the system or not.
If a compass is not setup in the system or is not detected, some funtionalities will not be available.

Returns:

ML_SUCCESS if the advanced sensor fusion functionalities were successfully enabled. ML_ERROR_INVALID_MODULE or other non-zero error code otherwise.

Chapter 6

Class Documentation

6.1 ext_slave_descr Struct Reference

Description of the slave device for programming.

`#include <mpu.h>`

6.1.1 Detailed Description

Description of the slave device for programming.

Defines the functions and information about the slave the mpu3050 needs to use the slave device.

Parameters:

init function used to preallocate memory used by the driver.

exit function used to free memory allocated for the driver.

suspend function pointer to put the device in suspended state.

resume function pointer to put the device in running state.

read function that reads the device data.

config function used to configure the device.

get_config function used to get the device's configuration.

name text name of the device.

type device type. enum ext_slave_type

id enum ext_slave_id.

reg starting register address to retrieve data.

len length in bytes of the sensor data. Should be 6.

endian byte order of the data. enum ext_slave_endian

range full scale range of the slave ouput: struct tFixPntRange.

```
struct ext_slave_descr {
  int (* init) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* exit) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* suspend) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* resume) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata);
  int (* read) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata,unsigned char *data);
  int (* config) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata,struct ext_slave_config *config);
  int (* get_config) (void *mlsl_handle,struct ext_slave_descr *slave,struct
  ext_slave_platform_data *pdata,struct ext_slave_config *config);
  char * name;
  unsigned char type;
  unsigned char id;
  unsigned char reg;
  unsigned int len;
  unsigned char endian;
  struct tFixPntRange range;
};
```

6.2 ext_slave_platform_data Struct Reference

Platform data for mpu3050 slave devices.

`#include <mpu.h>`

6.2.1 Detailed Description

Platform data for mpu3050 slave devices.

The orientation matricies are 3x3 rotation matricies that are applied to the data to rotate from the mounting orientation to the platform orientation. The values must be one of 0, 1, or -1 and each row and column should have exactly 1 non-zero value.

Parameters:

get_slave_descr Function pointer to retrieve the struct ext_slave_descr for this slave.

irq the irq number attached to the slave if any.

adapt_num the I2C adapter number.

bus the bus the slave is attached to: enum ext_slave_bus.

address the I2C slave address of the slave device.

orientation[9] the mounting matrix of the device relative to MPU.

irq_data private data for the slave irq handler.

private_data additional data, user customizable. Not touched by the MPU driver.

```
struct ext_slave_platform_data {
  struct ext_slave_descr *(* get_slave_descr) (void);
  int irq;
  int adapt_num;
  int bus;
  unsigned char address;
  signed char orientation[9];
  void * irq_data;
  void * private_data;
};
```

6.3 mpu3050_platform_data Struct Reference

Platform data for the mpu3050 driver.

`#include <mpu.h>`

6.3.1 Detailed Description

Platform data for the mpu3050 driver.

Contains platform specific information on how to configure the MPU3050 to work on this platform. The orientation matricies are 3x3 rotation matricies that are applied to the data to rotate from the mounting orientation to the platform orientation. The values must be one of 0, 1, or -1 and each row and column should have exactly 1 non-zero value.

Parameters:

int_config Bits [7:3] of the int config register.

orientation[MPU_NUM_AXES ∗MPU_NUM_AXES] Orientation matrix of the gyroscope.

level_shifter 0: VLogic, 1: VDD.

accel Accel platform data.

compass Compass platform data.

pressure Pressure platform data.

```
struct mpu3050_platform_data {
  unsigned char int_config;
  signed char orientation[MPU_NUM_AXES  *MPU_NUM_AXES];
  unsigned char level_shifter;
  struct ext_slave_platform_data accel;
  struct ext_slave_platform_data compass;
  struct ext_slave_platform_data pressure;
};
```


6.4 tGesture Struct Reference

Gesture data structure.

6.4.1 Detailed Description

Gesture data structure.

When a gesture is detected a structure of this type is returned to MLGetGesture() or the callback specified by MLSetGestureCallback().

Parameters:

type Type of gesture. One of:

- ML_PITCH_SHAKE use tGestureShake
- ML_ROLL_SHAKE use tGestureShake
- ML_YAW_SHAKE use tGestureShake
- ML_TAP use tGestureTap
- ML_YAW_IMAGE_ROTATE use tGestureYawImageRotate

strength See tGestureShake, tGestureTap or tGestureYawImageRotate

speed See tGestureShake, tGestureTap or tGestureYawImageRotate

num See tGestureShake, tGestureTap or tGestureYawImageRotate

meta See tGestureShake, tGestureTap or tGestureYawImageRotate

reserved See tGestureShake, tGestureTap or tGestureYawImageRotate

```
typedef struct  {

    unsigned short type;
    short          strength;
    short          speed;
    unsigned short num;
    short          meta;
    short          reserved;

}   tGesture;
```

6.5 tGestureShake Struct Reference

Shake gesture data structure.

6.5.1 Detailed Description

Shake gesture data structure.

When a shake is detected a structure of this type is returned to MLGetGesture() or the callback specified by MLSetGestureCallback().

This structure contains the axis of the shake, and the number of shakes detected so far and the strength and speed of the shake.

Parameters:

type Type of gesture, set to one of

- ML_PITCH_SHAKE
- ML_ROLL_SHAKE
- ML_YAW_SHAKE

strength Type of shake. One of

- ML_SOFT_SHAKE
- ML_HARD_SHAKE

speed Maximum angular velocity of the shake or peak shake when multiple shakes have been detected. Units in degrees per second. For more precision use the reserved parameter.

num Number of Shakes detected so far. Use MLSetMaxShakdes() to set the maximum before this will be reset to 0.

meta Direction of the shake in the frame of reference of the device. Signed value with:

- 0: Positive
- Non-Zero: Negetive

reserved Fraction part of the maximum angular velocity of the shake or peak shake when multiple shakes have been detected. Units are 65536 lsb's per degree. The formula for creating a floating point representation of the speed is:

```
float speed = (float)gesture->speed +
              (float)(gesture->reserved/(float)(1 << 16));
```

6.6 tGestureTap Struct Reference

Tap gesture data structure.

6.6.1 Detailed Description

Tap gesture data structure.

When a tap is detected a structure of this type is returned to MLGetGesture() or the callback specified by MLSetGestureCallback().

If type field is ML_TAP then this structure contains the tap information including number of taps detected so far and the direction of the tap.

In addtion to the meta data telling the direction. The relative magnitude of the tap impulse on each axis is also specified in the following fields:

- strength: X
- speed: Y
- reserved: Z

Parameters:

type Type of gesture, set to ML_TAP for a tGestureTap type

strength Magnitude of the tap impulse on the X axis.

speed Magnitude of the tap impulse on the Y axis.

num Number of Taps detected so far. Use MLSetMaxTaps() to set the maximum before this will be reset to 0.

meta Direction of the tap in the frame of reference of the device. Signed value with:

- 1: X
- 2: Y
- 3: Z

reserved Magnitude of the tap impulse on the Z axis.

6.7 tGestureYawImageRotate Struct Reference

Yaw image rotate gesture data structure.

6.7.1 Detailed Description

Yaw image rotate gesture data structure.

When a yaw image rotation is detected a structure of this type is returned to MLGetGesture() or the callback specified by MLSetGestureCallback().

This structure ccontains the direction of the Yaw Image Rotation.

Parameters:

type Type of gesture, set to ML_YAW_IMAGE_ROTATE

strength Unused.

speed Unused.

num Unused

meta Direction of the rotation in the frame of reference of the device. :

- 0: Positive
- Non-Zero: Negetive

reserved Unused

6.8 tMLError Struct Reference

The MPL Error Code return type.

`#include "mltypes"`

6.8.1 Detailed Description

The MPL Error Code return type.

```
typedef unsigned char tMLError;
```

6.9 tMLGlyphData Struct Reference

Describes the data to be used by the character recognition algorithm.

`#include <mlglyph.h>`

6.9.1 Detailed Description

Describes the data to be used by the character recognition algorithm.

When training and recognizing characters, data is stored and read from this data container.

Parameters:

yGlyph

xGlyph

GlyphLen

features

gestures

segments

library

libraryLength

probs

finalGesture

updatingGlyph

speedThresh

probFinal

minProb

Index