**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-V2.0
Release Date :07/22/2011

# Embedded MotionApps™ Platform Functional Specification Version 2.0

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 the functions available in the InvenSense Embedded MotionApps Platform (Embedded MPL), and corresponds with Embedded MotionApps Platform Release v2.0. This release is designed to work with all IMU-3000 devices revision K or earlier and MPU-6050 devices revision C or earlier.

Embedded MPL contains the code for controlling the InvenSense IMU-3000 and MPU-6050 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 library are described in this document, including all parameters involved in the function calls.

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 Embedded MPL.

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 **Embedded 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

# Module Documentation

## 4.1 APIs

Motion Library APIs.

### Files

- file ml.c

  *The Motion Library APIs.*

- file mlarray_lite.c

  *APIs to read different data sets from FIFO.*

### Functions

- inv_error_t inv_apply_calibration (void)

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

- inv_error_t inv_apply_endian_accel (void)

  *Setup the DMP to handle the accelerometer endianess.*

- int inv_check_flag (int flag)

  *inv_check_flag returns the value of a flag.*

- inv_error_t inv_get_accel_bias (long *data)

*inv_get_accel_bias is used to get the accelerometer baises.*

- inv_error_t inv_get_accel_bias_float (float *data)

  *inv_get_accel_bias_float is used to get the accelerometer baises.*

- inv_error_t inv_get_accel_cal_matrix (long *data)

  *inv_get_accel_cal_matrix is used to get the accelerometer calibration matrix.*

- inv_error_t inv_get_accel_cal_matrix_float (float *data)

  *inv_get_accel_cal_matrix_float is used to get the accelerometer calibration matrix.*

- inv_error_t inv_get_euler_angles (long *data)

  *inv_get_euler_angles is used to get the Euler angle representation of the current sensor fusion solution.*

- inv_error_t inv_get_euler_angles_float (float *data)

  *inv_get_euler_angles_float is used to get an array of three data points three data points representing roll, pitch, and yaw corresponding to the INV_EULER_ANGLES_X output and it is therefore the default convention for Euler angles.*

- inv_error_t inv_get_euler_angles_x (long *data)

  *inv_get_euler_angles_x is used to get the Euler angle representation of the current sensor fusion solution.*

- inv_error_t inv_get_euler_angles_x_float (float *data)

  *inv_get_euler_angles_x is used to get the Euler angle representation of the current sensor fusion solution.*

- inv_error_t inv_get_euler_angles_y (long *data)

  *inv_get_euler_angles_y is used to get the Euler angle representation of the current sensor fusion solution.*

- inv_error_t inv_get_euler_angles_y_float (float *data)

  *inv_get_euler_angles_y_float is used to get the Euler angle representation of the current sensor fusion solution.*

- inv_error_t inv_get_euler_angles_z (long *data)

  *inv_get_euler_angles_z is used to get the Euler angle representation of the current sensor fusion solution.*

- inv_error_t inv_get_euler_angles_z_float (float *data)

  *inv_get_euler_angles_z_float is used to get the Euler angle representation of the current sensor fusion solution.*

- inv_error_t inv_get_gravity_float (float *data)

  *inv_get_gravity_float is used to get an estimate of the body frame gravity vector, based on the most recent sensor fusion solution.*

- inv_error_t inv_get_gyro_and_accel_sensor_float (float *data)

  *inv_get_gyro_and_accel_sensor_float is used to get the most recent set of all sensor data.*

- inv_error_t inv_get_gyro_bias (long *data)

  *inv_get_gyro_bias is used to get the gyroscope biases.*

- inv_error_t inv_get_gyro_bias_float (float *data)

  *inv_get_gyro_bias_float is used to get the gyroscope biases.*

- inv_error_t inv_get_gyro_cal_matrix (long *data)

  *inv_get_gyro_cal_matrix is used to get the gyroscope calibration matrix.*

- inv_error_t inv_get_gyro_cal_matrix_float (float *data)

  *inv_get_gyro_cal_matrix_float is used to get the gyroscope calibration matrix.*

- inv_error_t inv_get_gyro_float (float *data)

  *inv_get_gyro_float is used to get the most recent gyroscope measurement.*

- int inv_get_gyro_present (void)

  *Check for the presence of the gyro sensor.*

- inv_error_t inv_get_heading (long *data)

  *inv_get_heading is used to get heading from Rotation Matrix.*

- inv_error_t inv_get_heading_float (float *data)

  *inv_get_heading_float is used to get single number representing the heading of the device relative to the Earth, in which 0 represents North, 90 degrees represents East, and so on.*

- int inv_get_interrupts (void)

  *Get the current set of DMP interrupt sources.*

- inv_error_t inv_get_linear_accel_float (float *data)

  *inv_get_quaternion_float is used to get the quaternion representation of the current sensor fusion solution.*

- inv_error_t inv_get_linear_accel_in_world_float (float *data)

*inv_get_linear_accel_in_world_float is used to get an estimate of linear acceleration, in the world frame, based on the most recent accelerometer measurement and sensor fusion solution.*

- inv_error_t inv_get_mag_bias (long *data)

  *inv_get_mag_bias is used to get Magnetometer Bias*

- inv_error_t inv_get_mag_bias_float (float *data)

  *inv_get_mag_bias_float is used to get an array of three data points representing the compass biases.*

- inv_error_t inv_get_mag_cal_matrix (long *data)

  *inv_get_mag_cal_matrix is used to get magnetometer calibration matrix.*

- inv_error_t inv_get_mag_cal_matrix_float (float *data)

  *inv_get_mag_cal_matrix_float is used to get an array of nine data points representing the calibration matrix for the compass:*

  *C11 C12 C13*
  *C21 C22 C23*
  *C31 C32 C33*

- inv_error_t inv_get_mag_raw_data (long *data)

  *inv_get_mag_raw_data is used to get Raw magnetometer data.*

- inv_error_t inv_get_mag_raw_data_float (float *data)

  *inv_get_mag_raw_data_float is used to get Raw magnetometer data*

- inv_error_t inv_get_magnetometer (long *data)

  *inv_get_magnetometer is used to get magnetometer data.*

- inv_error_t inv_get_magnetometer_float (float *data)

  *inv_get_magnetometer_float is used to get magnetometer data*

- int inv_get_motion_state (void)

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

- inv_error_t inv_get_rot_mat (long *data)

  *inv_get_rot_mat is used to get the rotation matrix representation of the current sensor fusion solution.*

- inv_error_t inv_get_rot_mat_float (float *data)

  *inv_get_rot_mat_float is used to get an array of nine data points representing the rotation matrix generated from all available sensors.*

- void * inv_get_serial_handle (void)

  *Get the serial file handle to the device.*

- inv_error_t inv_get_temperature_float (float *data)

  *inv_get_temperature_float is used to get the most recent temperature measurement.*

- inv_error_t inv_get_version (unsigned char **version)

  *inv_get_version is used to get the ML version.*

- inv_error_t inv_serial_start (char const *port)

  *Open serial connection with the MPU device.*

- inv_error_t inv_serial_stop (void)

  *Close the serial communication.*

- inv_error_t inv_set_accel_bias (long *data)

  *inv_set_accel_bias is used to set the accelerometer bias.*

- inv_error_t inv_set_accel_bias_float (float *data)

  *inv_set_accel_bias_float is used to set the accelerometer bias.*

- inv_error_t inv_set_accel_calibration (float range, signed char *orientation)

  *Sets up the Accelerometer calibration and scale factor.*

- inv_error_t inv_set_bias_update (unsigned short function)

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

- inv_error_t inv_set_compass_calibration (float range, signed char *orientation)

  *Sets up the Compass calibration and scale factor.*

- inv_error_t inv_set_fifo_interrupt (unsigned char on)

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

- inv_error_t inv_set_gyro_bias (long *data)

  *inv_set_gyro_bias is used to set the gyroscope bias.*

- inv_error_t inv_set_gyro_bias_float (float *data)

  *inv_set_gyro_bias_float is used to set the gyroscope bias.*

- inv_error_t inv_set_gyro_calibration (float range, signed char *orientation)

*Sets up the Gyro calibration and scale factor.*

- inv_error_t inv_set_mag_bias (long *data)

  *inv_set_mag_bias is used to set Compass Bias*

- inv_error_t inv_set_mag_bias_float (float *data)

  *inv_set_mag_bias_float is used to set compass bias*

- inv_error_t inv_set_motion_interrupt (unsigned char on)

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

- inv_error_t inv_set_mpu_sensors (unsigned long sensors)

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

- inv_error_t inv_set_no_motion_thresh (float thresh)

  *inv_set_no_motion_thresh is used to set the threshold for detecting INV_NO_MOTION*

- inv_error_t inv_set_no_motion_threshAccel (long thresh)

  *inv_set_no_motion_threshAccel is used to set the threshold for detecting INV_NO_MOTION with accelerometers when Gyros have been turned off*

- inv_error_t inv_set_no_motion_time (float time)

  *inv_set_no_motion_time is used to set the time required for detecting INV_NO_MOTION*

- inv_error_t inv_update_data (void)

  *inv_update_data fetches data from the fifo and updates the motion algorithms.*

### 4.1.1 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.

### 4.1.2 Function Documentation

#### 4.1.2.1 inv_error_t inv_apply_calibration (void)

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

**Returns:**

INV_SUCCESS if successful, a non-zero code otherwise.

#### 4.1.2.2 inv_error_t inv_apply_endian_accel (void)

Setup the DMP to handle the accelerometer endianess.

**Returns:**

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

#### 4.1.2.3 int inv_check_flag (int *flag*)

inv_check_flag returns the value of a flag.

inv_check_flag 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 inv_check_flag is called, the flag is automatically reset. The flags are:

- INV_RAW_DATA_READY Indicates that new raw data is available.
- INV_PROCESSED_DATA_READY Indicates that new processed data is available.
- INV_GOT_GESTURE Indicates that a gesture has been detected by the gesture engine.
- INV_MOTION_STATE_CHANGE Indicates that a change has been made from motion to no motion, or vice versa.

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must have been called.

**Parameters:**

***flag*** The flag to check.

**Returns:**

true or false state of the flag

#### 4.1.2.4 inv_error_t inv_get_accel_bias (long * *data*)

inv_get_accel_bias is used to get the accelerometer baises.

The argument array elements are ordered X,Y,Z. The values are scaled such that 1 g (gravity) = 2^16 LSBs.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long** .

**Returns:**

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

#### 4.1.2.5 inv_error_t inv_get_accel_bias_float (float * *data*)

inv_get_accel_bias_float is used to get the accelerometer baises.

The argument array elements are ordered X,Y,Z. The values are in units of g (gravity).

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.6 inv_error_t inv_get_accel_cal_matrix (long * *data*)

inv_get_accel_cal_matrix is used to get the accelerometer calibration matrix.

Calibration matrix data members will have a value of 1, 0, or -1. The matrix has members

C11 C12 C13

C21 C22 C23

C31 C32 C33

The argument array elements are ordered C11, C12, C13, C21, C22, C23, C31, C32, C33.

**Precondition:**

MLDmpOpen()
MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.7 inv_error_t inv_get_accel_cal_matrix_float (float * *data*)

inv_get_accel_cal_matrix_float is used to get the accelerometer calibration matrix.

Calibration matrix data members will have a value of 1.0, 0, or -1.0. The matrix has members

C11 C12 C13

C21 C22 C23

C31 C32 C33

The argument array elements are ordered C11, C12, C13, C21, C22, C23, C31, C32, C33.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.8 inv_error_t inv_get_euler_angles (long * *data*)

inv_get_euler_angles is used to get the Euler angle representation of the current sensor fusion solution.

Euler angles may follow various conventions. This function is equivelant to inv_get_-euler_angles_x, refer to inv_get_euler_angles_x for more information.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long** .

**Returns:**

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

#### 4.1.2.9 inv_error_t inv_get_euler_angles_float (float * *data*)

inv_get_euler_angles_float is used to get an array of three data points three data points representing roll, pitch, and yaw corresponding to the INV_EULER_ANGLES_X output and it is therefore the default convention for Euler angles.

Please refer to the INV_EULER_ANGLES_X for a detailed description.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.10 inv_error_t inv_get_euler_angles_x (long * *data*)

inv_get_euler_angles_x is used to get the Euler angle representation of the current sensor fusion solution.

Euler angles are returned according to the X convention. 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.

| Element | **Euler angle** | **Rotation about** |
|---|---|---|
| 0 | Roll | X axis |
| 1 | Pitch | Y axis |
| 2 | Yaw | Z axis |

Values are scaled where 1.0 = 2^16.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long** .

**Returns:**

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

#### 4.1.2.11 inv_error_t inv_get_euler_angles_x_float (float * *data*)

inv_get_euler_angles_x is used to get the Euler angle representation of the current sensor fusion solution.

Euler angles are returned according to the X convention. 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.

| Element | **Euler angle** | **Rotation about** |
|---|---|---|
| 0 | Roll | X axis |
| 1 | Pitch | Y axis |
| 2 | Yaw | Z axis |

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.12 inv_error_t inv_get_euler_angles_y (long * *data*)

inv_get_euler_angles_y is used to get the Euler angle representation of the current sensor fusion solution.

Euler angles are returned according to the Y convention. 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.

| Element | **Euler angle** | **Rotation about** |
|---|---|---|
| 0 | Roll | Y axis |
| 1 | Pitch | X axis |
| 2 | Yaw | Z axis |

Values are scaled where 1.0 = 2^16.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.13 inv_error_t inv_get_euler_angles_y_float (float * *data*)

inv_get_euler_angles_y_float is used to get the Euler angle representation of the current sensor fusion solution.

Euler angles are returned according to the Y convention. 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.

| Element | **Euler angle** | **Rotation about** |
|---|---|---|
| 0 | Roll | Y axis |
| 1 | Pitch | X axis |
| 2 | Yaw | Z axis |

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.14 inv_error_t inv_get_euler_angles_z (long * *data*)

inv_get_euler_angles_z is used to get the Euler angle representation of the current sensor fusion solution.

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. Euler angles are returned according to the Y convention.

| Element | **Euler angle** | **Rotation about** |
|---|---|---|
| 0 | Roll | Z axis |
| 1 | Pitch | X axis |
| 2 | Yaw | Y axis |

Values are scaled where 1.0 = 2^16.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.15 inv_error_t inv_get_euler_angles_z_float (float * *data*)

inv_get_euler_angles_z_float is used to get the Euler angle representation of the current sensor fusion solution.

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. Euler angles are returned according to the Y convention.

| Element | **Euler angle** | **Rotation about** |
|---|---|---|
| 0 | Roll | Z axis |
| 1 | Pitch | X axis |
| 2 | Yaw | Y axis |

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.16 inv_error_t inv_get_gravity_float (float * *data*)

inv_get_gravity_float is used to get an estimate of the body frame gravity vector, based on the most recent sensor fusion solution.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long at least**.

**Returns:**

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

#### 4.1.2.17 inv_error_t inv_get_gyro_and_accel_sensor_float (float * *data*)

inv_get_gyro_and_accel_sensor_float is used to get the most recent set of all sensor data.

The argument array elements are ordered gyroscope X,Y, and Z, accelerometer X, Y, and Z, and magnetometer X,Y, and Z. The magnetometer elements are not populated in UMPL. The gyroscope and accelerometer data is not scaled or offset, it is copied directly from the sensor registers, and cast as a float. In the case of accelerometers with 8-bit output resolution, the data is scaled up to match the 2^14 = 1 g typical represntation of +/- 2 g full scale range

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.18 inv_error_t inv_get_gyro_bias (long * *data*)

inv_get_gyro_bias is used to get the gyroscope biases.

The argument array elements are ordered X,Y,Z. The values are scaled such that 1 dps = 2^16 LSBs.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.19 inv_error_t inv_get_gyro_bias_float (float * *data*)

inv_get_gyro_bias_float is used to get the gyroscope biases.

The argument array elements are ordered X,Y,Z. The values are in units of dps (degrees per second).

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.20 inv_error_t inv_get_gyro_cal_matrix (long * *data*)

inv_get_gyro_cal_matrix is used to get the gyroscope calibration matrix.

The gyroscope calibration matrix defines the relationship between the gyroscope sensor axes and the sensor fusion solution axes. Calibration matrix data members will have a value of 1, 0, or -1. The matrix has members

C11 C12 C13

C21 C22 C23

C31 C32 C33

The argument array elements are ordered C11, C12, C13, C21, C22, C23, C31, C32, C33.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.21 inv_error_t inv_get_gyro_cal_matrix_float (float * *data*)

inv_get_gyro_cal_matrix_float is used to get the gyroscope calibration matrix.

The gyroscope calibration matrix defines the relationship between the gyroscope sensor axes and the sensor fusion solution axes. Calibration matrix data members will have a value of 1.0, 0, or -1.0. The matrix has members

C11 C12 C13

C21 C22 C23

C31 C32 C33

The argument array elements are ordered C11, C12, C13, C21, C22, C23, C31, C32, C33.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.22 inv_error_t inv_get_gyro_float (float * *data*)

inv_get_gyro_float is used to get the most recent gyroscope measurement.

The argument array elements are ordered X,Y,Z. The values are in units of dps (degrees per second).

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.23 int inv_get_gyro_present (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 inv_set_mpu_sensors().

**Returns:**

true if the gyro is enabled false otherwise.

#### 4.1.2.24 inv_error_t inv_get_heading (long * *data*)

inv_get_heading is used to get heading from Rotation Matrix.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to data to be passed back to the user.

**Returns:**

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

#### 4.1.2.25 inv_error_t inv_get_heading_float (float * *data*)

inv_get_heading_float is used to get 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.

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to the data to be passed back to the user.

**Returns:**

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

#### 4.1.2.26 int inv_get_interrupts (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:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

**Returns:**

Currently enabled interrupt sources. The possible interrupts are:

- INV_INT_FIFO,
- INV_INT_MOTION,
- INV_INT_TAP

#### 4.1.2.27 inv_error_t inv_get_linear_accel_float (float * *data*)

inv_get_quaternion_float is used to get the quaternion representation of the current sensor fusion solution.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 4 cells long**.

**Returns:**

INV_SUCCESS if the command is successful; an ML error code otherwise. inv_-get_linear_accel_float is used to get an estimate of linear acceleration, based on the most recent accelerometer measurement and sensor fusion solution. The values are in units of g (gravity).

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.28 inv_error_t inv_get_linear_accel_in_world_float (float * *data*)

inv_get_linear_accel_in_world_float is used to get an estimate of linear acceleration, in the world frame, based on the most recent accelerometer measurement and sensor fusion solution.

The values are in units of g (gravity).

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.29 inv_error_t inv_get_mag_bias (long * *data*)

inv_get_mag_bias is used to get Magnetometer Bias

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long** .

**Returns:**

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

#### 4.1.2.30 inv_error_t inv_get_mag_bias_float (float * *data*)

inv_get_mag_bias_float is used to get an array of three data points representing the compass biases.

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.31 inv_error_t inv_get_mag_cal_matrix (long * *data*)

inv_get_mag_cal_matrix is used to get magnetometer calibration matrix.

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***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.

#### 4.1.2.32 inv_error_t inv_get_mag_cal_matrix_float (float * *data*)

inv_get_mag_cal_matrix_float is used to get an array of nine data points representing the calibration matrix for the compass:

C11 C12 C13

C21 C22 C23

C31 C32 C33

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.33 inv_error_t inv_get_mag_raw_data (long * *data*)

inv_get_mag_raw_data is used to get Raw magnetometer data.

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long** .

**Returns:**

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

#### 4.1.2.34 inv_error_t inv_get_mag_raw_data_float (float * *data*)

inv_get_mag_raw_data_float is used to get Raw magnetometer data

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.35 inv_error_t inv_get_magnetometer (long * *data*)

inv_get_magnetometer is used to get magnetometer data.

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.36 inv_error_t inv_get_magnetometer_float (float * *data*)

inv_get_magnetometer_float is used to get magnetometer data

UMPL_NINE_AXIS is required.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to an array to be passed back to the user. **Must be 3 cells long**.

**Returns:**

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

#### 4.1.2.37 int inv_get_motion_state (void)

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

inv_get_motion_state returns INV_MOTION of the device is moving, or INV_NO_-MOTION if the device is not moving.

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must have been called.

**Returns:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.38 inv_error_t inv_get_rot_mat (long * *data*)

inv_get_rot_mat is used to get the rotation matrix representation of the current sensor fusion solution.

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

Values are scaled, where 1.0 = 2^30 LSBs.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.39 inv_error_t inv_get_rot_mat_float (float * *data*)

inv_get_rot_mat_float is used to get an array of nine data points representing the rotation matrix generated from all available sensors.

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**.

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

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

**Returns:**

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

#### 4.1.2.40 void∗ inv_get_serial_handle (void)

Get the serial file handle to the device.

**Returns:**

The serial file handle.

#### 4.1.2.41 inv_error_t inv_get_temperature_float (float ∗ *data*)

inv_get_temperature_float is used to get the most recent temperature measurement.

The argument array should only have one element. The value is in units of deg C (degrees Celsius).

**Precondition:**

MLDmpOpen() must have been called.

**Parameters:**

***data*** A pointer to data to be passed back to the user.

**Returns:**

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

#### 4.1.2.42 inv_error_t inv_get_version (unsigned char ** *version*)

inv_get_version is used to get the ML version.

**Precondition:**

inv_get_version can be called at any time.

**Parameters:**

***version*** inv_get_version writes the ML version string pointer to version.

**Returns:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.43 inv_error_t inv_serial_start (char const * *port*)

Open serial connection with the MPU device.

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

**Parameters:**

***port*** System handle for 'port' MPU device is found on. The significance of this parameter varies by platform. It is passed as 'port' to MLSLSerialOpen.

**Returns:**

INV_SUCCESS or error code.

#### 4.1.2.44 inv_error_t inv_serial_stop (void)

Close the serial communication.

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

**Returns:**

INV_SUCCESS; non-zero error code otherwise.

#### 4.1.2.45 inv_error_t inv_set_accel_bias (long * *data*)

inv_set_accel_bias is used to set the accelerometer bias.

The argument array elements are ordered X,Y,Z. The values are scaled in units of g (gravity), where 1 g = 2^16 LSBs.

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

**Precondition:**

MLDmpOpen()

**Parameters:**

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

**Returns:**

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

#### 4.1.2.46 inv_error_t inv_set_accel_bias_float (float * *data*)

inv_set_accel_bias_float is used to set the accelerometer bias.

The argument array elements are ordered X,Y,Z. The values are in units of g (gravity).

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

**Precondition:**

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

**Parameters:**

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

**Returns:**

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

#### 4.1.2.47 inv_error_t inv_set_accel_calibration (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:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

**Precondition:**

inv_dmp_start() must **NOT** have been called.

**See also:**

inv_set_gyro_calibration().
inv_set_compass_calibration().

**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:**

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

#### 4.1.2.48 inv_error_t inv_set_bias_update (unsigned short *function*)

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

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

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() 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:

- INV_NONE or 0,
- INV_BIAS_FROM_NO_MOTION,
- INV_BIAS_FROM_GRAVITY,
- INV_BIAS_FROM_TEMPERATURE,

**Returns:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.49 inv_error_t inv_set_compass_calibration (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:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

**Precondition:**

inv_dmp_start() must have **NOT** been called.

**See also:**

inv_set_gyro_calibration().
inv_set_accel_calibration().

**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:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.50 inv_error_t inv_set_fifo_interrupt (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:**

INV_SUCCESS or non-zero error code

#### 4.1.2.51 inv_error_t inv_set_gyro_bias (long * *data*)

inv_set_gyro_bias is used to set the gyroscope bias.

The argument array elements are ordered X,Y,Z. The values are scaled at 1 dps = 2^16 LSBs.

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

**Precondition:**

MLDmpOpen()

**Parameters:**

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

**Returns:**

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

#### 4.1.2.52 inv_error_t inv_set_gyro_bias_float (float * *data*)

inv_set_gyro_bias_float is used to set the gyroscope bias.

The argument array elements are ordered X,Y,Z. The values are in units of dps (degrees per second).

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

**Precondition:**

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

**Parameters:**

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

**Returns:**

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

#### 4.1.2.53 inv_error_t inv_set_gyro_calibration (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:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

**Precondition:**

inv_dmp_start() must have **NOT** been called.

**See also:**

inv_set_accel_calibration().
inv_set_compass_calibration().

**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:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.54 inv_error_t inv_set_mag_bias (long * *data*)

inv_set_mag_bias is used to set Compass Bias

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

UMPL_NINE_AXIS is required.

**Precondition:**

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

**Parameters:**

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

**Returns:**

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

#### 4.1.2.55 inv_error_t inv_set_mag_bias_float (float * *data*)

inv_set_mag_bias_float is used to set compass bias

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

UMPL_NINE_AXIS required.

**Precondition:**

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

**Parameters:**

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

**Returns:**

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

#### 4.1.2.56 inv_error_t inv_set_motion_interrupt (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:**

INV_SUCCESS or non-zero error code.

#### 4.1.2.57 inv_error_t inv_set_mpu_sensors (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:**

INV_SUCCESS or non-zero error code

#### 4.1.2.58 inv_error_t inv_set_no_motion_thresh (float *thresh*)

inv_set_no_motion_thresh is used to set the threshold for detecting INV_NO_-MOTION

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

**Parameters:**

***thresh*** A threshold scaled in degrees per second.

**Returns:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.59 inv_error_t inv_set_no_motion_threshAccel (long *thresh*)

inv_set_no_motion_threshAccel is used to set the threshold for detecting INV_NO_-MOTION with accelerometers when Gyros have been turned off

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

**Parameters:**

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

**Returns:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.60 inv_error_t inv_set_no_motion_time (float *time*)

inv_set_no_motion_time is used to set the time required for detecting INV_NO_-MOTION

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

must have been called.

**Parameters:**

***time*** A time in seconds.

**Returns:**

INV_SUCCESS if successful or Non-zero error code otherwise.

#### 4.1.2.61 inv_error_t inv_update_data (void)

inv_update_data fetches data from the fifo and updates the motion algorithms.

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must have been called.

**Note:**

Motion algorithm data is constant between calls to inv_update_data

**Returns:**

- INV_SUCCESS
- Non-zero error code

## 4.2 OS Interface for Atmel AVR32

OS Interface for Atmel AVR32.

### Files

- file mlos_at32.c
  *OS Interface.*

### Functions

- unsigned long inv_get_tick_count (void)
  *Get system's internal tick count.*
- void inv_sleep (int mSecs)
  *Sleep function.*

### 4.2.1 Detailed Description

OS Interface for Atmel AVR32.

### 4.2.2 Function Documentation

#### 4.2.2.1 unsigned long inv_get_tick_count (void)

Get system's internal tick count.

Used for time reference.

**Returns:**

Current tick count.

#### 4.2.2.2 void inv_sleep (int *mSecs*)

Sleep function.

**Parameters:**

***mSecs*** Delay in milliseconds

## 4.3 Serial Layer

The Motion Library Serial Layer provides the Motion Library the interface to the system serial functions.

### Files

- file mlsl_at32.c

  *The Motion Library Serial Layer.*

### Functions

- inv_error_t inv_serial_get_cal_length (unsigned int *len)

  *Get the calibration length.*

- inv_error_t inv_serial_open (char const *port, void **sl_handle)

  *stub on at32 platform.*

- inv_error_t inv_serial_read (void *sl_handle, unsigned char slaveAddr, unsigned char registerAddr, unsigned short length, unsigned char *data)

  *used to read multiple bytes of data from registers.*

- inv_error_t inv_serial_read_cal (unsigned char *cal, unsigned int len)

  *used to get the calibration data.*

- inv_error_t inv_serial_read_fifo (void *sl_handle, unsigned char slaveAddr, unsigned short length, unsigned char *data)

  *used to read multiple bytes of data from the fifo.*

- inv_error_t inv_serial_read_mem (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.*

- inv_error_t inv_serial_single_write (void *sl_handle, unsigned char slaveAddr, unsigned char registerAddr, unsigned char data)

  *used to write a single byte to a single register.*

- inv_error_t inv_serial_write (void *sl_handle, unsigned char slaveAddr, unsigned short length, unsigned char const *data)

  *used as generic serial write.*

- inv_error_t inv_serial_write_cal (unsigned char ∗cal, unsigned int len)

  *used to save the calibration data.*

- inv_error_t inv_serial_write_fifo (void ∗sl_handle, unsigned char slaveAddr, unsigned short length, unsigned char const ∗data)

  *used to write multiple bytes of data to the fifo.*

- inv_error_t inv_serial_write_mem (void ∗sl_handle, unsigned char slaveAddr, unsigned short memAddr, unsigned short length, unsigned char const ∗data)

  *used to write multiple bytes of data to DMP memory.*

### 4.3.1 Detailed Description

The Motion Library Serial Layer provides the Motion Library the interface to the system serial functions.

### 4.3.2 Function Documentation

#### 4.3.2.1 inv_error_t inv_serial_get_cal_length (unsigned int ∗ *len*)

Get the calibration length.

**Parameters:**

***len*** lenght to be returned

**Returns:**

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

#### 4.3.2.2 inv_error_t inv_serial_open (char const ∗ *port*, void ∗∗ *sl_handle*)

stub on at32 platform.

The Atmel ASF twim library, which provides twim_read and twim_write, should be configured before this is called.

**Parameters:**

***port*** Unused, pass as NULL

***sl_handle*** Unused, pass as NULL

#### 4.3.2.3 inv_error_t inv_serial_read (void * *sl_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.

**Parameters:**

***sl_handle*** Unused, pass as NULL

***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

#### 4.3.2.4 inv_error_t inv_serial_read_cal (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:**

***cal*** Pointer to the calibration data.

***len*** Length of the calibration data.

**Returns:**

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

#### 4.3.2.5 inv_error_t inv_serial_read_fifo (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.

**Parameters:**

***sl_handle*** Unused, pass as NULL

***slaveAddr*** I2C slave address of device.

***length*** Number of bytes to read from fifo.

***data*** Pointer to write fifo data.

**Returns:**

Zero if successful; an error code otherwise

#### 4.3.2.6 inv_error_t inv_serial_read_mem (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.

**Parameters:**

***sl_handle*** Unused, pass as NULL

***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

#### 4.3.2.7 inv_error_t inv_serial_single_write (void * *sl_handle*, unsigned char *slaveAddr*, unsigned char *registerAddr*, unsigned char *data*)

used to write a single byte to a single register.

**Parameters:**

***sl_handle*** Unused, pass as NULL

***slaveAddr*** I2C slave address of device.

***registerAddr*** Register address to write.

***data*** Single byte of data to write.

**Returns:**

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

#### 4.3.2.8 inv_error_t inv_serial_write (void * *sl_handle*, unsigned char *slaveAddr*, unsigned short *length*, unsigned char const * *data*)

used as generic serial write.

Does not accept a register parameter like the rest of the MLSLSerial functions - if used to write to a register, the register address should be the first member of data This should be sent by I2C.

**Parameters:**

***sl_handle*** Unused, pass as NULL

***slaveAddr*** I2C slave address of device.

***length*** Length of burst of data.

***data*** Pointer to block of data.

**Returns:**

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

#### 4.3.2.9 inv_error_t inv_serial_write_cal (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:**

***cal*** Pointer to the calibration data.

***len*** Length of the calibration data.

**Returns:**

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

#### 4.3.2.10 inv_error_t inv_serial_write_fifo (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.

**Parameters:**

***sl_handle*** Unused, pass as NULL

***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

#### 4.3.2.11 inv_error_t inv_serial_write_mem (void * *sl_handle*, unsigned char *slaveAddr*, unsigned short *memAddr*, unsigned short *length*, unsigned char const * *data*)

used to write multiple bytes of data to DMP memory.

**Parameters:**

***sl_handle*** Unused, pass as NULL

***slaveAddr*** I2C slave address of device.

***memAddr*** The location in the memory to write to. 16 bits.

***length*** Length of burst data. Must not cross 8-bit address boundary.

***data*** Pointer to block of data.

**Returns:**

Zero if successful; an error code otherwise

## 4.4 uMPL Layer

uMPL Layer.

### Files

- file umpl-states.c

  *Controls the UMPL state machine.*

### Functions

- int umplGetState (void)

  *umplGetState returns the current state of UMPL.*

- inv_error_t umplInit (unsigned char *port)

  *umplInit is the entry point to uMPL and must be called before all other functions.*

- inv_error_t umplNotifyInterrupt (unsigned char inv_interrupt)

  *umplNotifyInterrupt is called to handle a recently occurred interrupt.*

- inv_error_t umplOnTimer (void)

  *umplOnTimer calls MLUpdataData which updates all the realtime data from the motion algorithms.*

- inv_error_t umplStartAccelOnly (float freq)

  *umplStartAccelOnly starts the accelerometer.*

- inv_error_t umplStartAccelOnlyLowPower (float freq)

  *umplStartAccelOnlyLowPower starts the accelerometer in low power mode.*

- inv_error_t umplStartMPU (void)

  *umplStartMPU kicks starts the uMPL state machine.*

- inv_error_t umplStop (void)

  *umplStop stops the uMPL state machine.*

### 4.4.1 Detailed Description

uMPL Layer.

umpl-states.c provides a mechanism to control the uMPL state machine. It controls the flow from one state to the next.

### 4.4.2 Function Documentation

#### 4.4.2.1 int umplGetState (void)

umplGetState returns the current state of UMPL.

**Returns:**

The current state of UMPL

#### 4.4.2.2 inv_error_t umplInit (unsigned char * *port*)

umplInit is the entry point to uMPL and must be called before all other functions.

It initializes the platform specific data. It also initializes the MPL state.

**Parameters:**

***port*** A dummy port number. Could be NULL.

#### 4.4.2.3 inv_error_t umplNotifyInterrupt (unsigned char *inv_interrupt*)

umplNotifyInterrupt is called to handle a recently occurred interrupt.

**Precondition:**

umplInit() and umplStartAccelOnly() or umplStartMPU() must have been called.

**Parameters:**

***inv_interrupt*** the occurred interrupt which must be handled

**Returns:**

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

#### 4.4.2.4 inv_error_t umplOnTimer (void)

umplOnTimer calls MLUpdataData which updates all the realtime data from the motion algorithms.

**Precondition:**

umplInit() and umplStartAccelOnly() or umplStartMPU() must have been called.

**Returns:**

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

#### 4.4.2.5 inv_error_t umplStartAccelOnly (float *freq*)

umplStartAccelOnly starts the accelerometer.

**Precondition:**

umplStartMPU() must have been called.

**Parameters:**

***freq*** The update rate can be 18 Hz, 50 Hz and 100 Hz.

**Returns:**

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

#### 4.4.2.6 inv_error_t umplStartAccelOnlyLowPower (float *freq*)

umplStartAccelOnlyLowPower starts the accelerometer in low power mode.

**Precondition:**

umplStartMPU() must have been called.

**Parameters:**

***freq*** The update rate can be 40 Hz, 10 Hz, 2.5 Hz, 1.25 Hz.

**Returns:**

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

#### 4.4.2.7 inv_error_t umplStartMPU (void)

umplStartMPU kicks starts the uMPL state machine.

This function in turn calls the MPL functions like inv_dmp_open() and inv_dmp_start() which starts the motion processing algorithms. This function also enables the required features such as turning on the bias trackers and temperature compensation. This function enables the required type of data to be put in FIFO.

**Precondition:**

umplInit() must have been called.

**Returns:**

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

#### 4.4.2.8 inv_error_t umplStop (void)

umplStop stops the uMPL state machine.

This function in turn calls the necessary MPL functions like inv_dmp_stop() and inv_dmp_close() to stop the motion processing algorithms.

**Precondition:**

umplInit() and umplStartMPU() must have been called.

**Returns:**

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

## 4.5 FIFO Driver

Motion Library - FIFO Driver.

### Files

- file mlFIFO.c

  *FIFO Interface.*

### Functions

- unsigned long inv_accel_sum_of_sqr (void)

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

- inv_error_t inv_close_fifo (void)

  *Close the FIFO usage.*

- long inv_decode_temperature (short tempReg)

  *Converts 16-bit temperature data as read from temperature register into Celcius scaled by 2^16.*

- inv_error_t inv_get_6axis_quaternion (long *data)

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

- inv_error_t inv_get_accel (long *data)

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

- inv_error_t inv_get_accel_float (float *data)

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

- inv_error_t inv_get_cntrl_data (long *data)

  *Returns 4-element vector of control data.*

- inv_error_t inv_get_eis (long *data)

  *Returns 3-element vector of EIS shfit data.*

- inv_error_t inv_get_external_sensor_data (long *data, int size)

  *Returns 3-element vector of external sensor.*

- unsigned short inv_get_fifo_rate (void)

  *Retrieve the current FIFO update divider - 1.*

- inv_error_t inv_get_gravity (long *data)

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

- inv_error_t inv_get_gyro (long *data)

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

- inv_error_t inv_get_gyro_and_accel_sensor (long *data)

  *Returns 6-element vector of gyro and accel data.*

- inv_error_t inv_get_gyro_sensor (long *data)

  *This gets raw gyro data.*

- unsigned long inv_get_gyro_sum_of_sqr (void)

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

- inv_error_t inv_get_linear_accel (long *data)

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

- inv_error_t inv_get_linear_accel_in_world (long *data)

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

- inv_error_t inv_get_quantized_accel (long *data)

  *Get the Quantized Accel data algorithm output from the FIFO.*

- inv_error_t inv_get_quaternion (long *data)

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

- inv_error_t inv_get_quaternion_float (float *data)

  *Returns 4-element quaternion vector.*

- int_fast16_t inv_get_sample_frequency (void)

  *Returns the step size for quaternion type data.*

- int_fast16_t inv_get_sample_step_size_ms (void)

  *Returns the step size for quaternion type data.*

- inv_error_t inv_get_temperature (long *data)

  *Returns 1-element vector of temperature.*

- inv_error_t inv_get_unquantized_accel (long *data)

  *Get the Decoded Accel Data.*

- inv_error_t inv_init_fifo_param (void)

  *Initializes all the internal static variables for the FIFO module.*

- inv_error_t inv_read_and_process_fifo (int_fast8_t numPackets, int_fast8_t *processed)

  *Reads and processes FIFO data.*

- inv_error_t inv_send_accel (uint_fast16_t elements, uint_fast16_t accuracy)

  *Sends accelerometer data to the FIFO.*

- inv_error_t inv_send_cntrl_data (uint_fast16_t elements, uint_fast16_t accuracy)

  *Sends control data to the FIFO.*

- inv_error_t inv_send_external_sensor_data (uint_fast16_t elements, uint_fast16_t accuracy)

  *Sends raw external data to the FIFO.*

- inv_error_t inv_send_gravity (uint_fast16_t elements, uint_fast16_t accuracy)

  *Send the computed gravity vectors into the FIFO.*

- inv_error_t inv_send_gyro (uint_fast16_t elements, uint_fast16_t accuracy)

  *Sends gyro data to the FIFO.*

- inv_error_t inv_send_linear_accel (uint_fast16_t elements, uint_fast16_t accuracy)

  *Sends linear accelerometer data to the FIFO.*

- inv_error_t inv_send_linear_accel_in_world (uint_fast16_t elements, uint_fast16_t accuracy)

  *Sends linear world accelerometer data to the FIFO.*

- inv_error_t inv_send_packet_number (uint_fast16_t accuracy)

  *Adds a rolling counter to the fifo packet.*

- inv_error_t inv_send_quantized_accel (uint_fast16_t elements, uint_fast16_t accuracy)

  *Send the Quantized Accelerometer data into the FIFO.*

- inv_error_t inv_send_quaternion (uint_fast16_t accuracy)

*Sends quaternion data to the FIFO.*

- inv_error_t inv_send_sensor_data (uint_fast16_t elements, uint_fast16_t accuracy)

  *Sends raw data to the FIFO.*

- inv_error_t inv_set_fifo_processed_callback (void(∗func)(void))

  *inv_set_fifo_processed_callback is used to set a processed data callback function.*

- inv_error_t inv_set_fifo_rate (unsigned short fifoRate)

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

- inv_error_t inv_set_gyro_data_source (uint_fast8_t source)

  *Set the gyro source to output to the fifo.*

- inv_error_t inv_set_linear_accel_filter_coef (float coef)

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

### 4.5.1 Detailed Description

Motion Library - FIFO Driver.

The FIFO API interface.

### 4.5.2 Function Documentation

#### 4.5.2.1 unsigned long inv_accel_sum_of_sqr (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.

#### 4.5.2.2 inv_error_t inv_close_fifo (void)

Close the FIFO usage.

**Returns:**

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

#### 4.5.2.3 inv_error_t inv_get_6axis_quaternion (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.

#### 4.5.2.4 inv_error_t inv_get_accel (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.

#### 4.5.2.5 inv_error_t inv_get_accel_float (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.

#### 4.5.2.6 inv_error_t inv_get_cntrl_data (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.

#### 4.5.2.7 inv_error_t inv_get_eis (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.

#### 4.5.2.8 inv_error_t inv_get_external_sensor_data (long * *data*, int *size*)

Returns 3-element vector of external sensor.

**Parameters:**

***data*** 3-element vector of external sensor

***size*** the size of the buffer.

**Returns:**

0 on success or an error code.

#### 4.5.2.9 unsigned short inv_get_fifo_rate (void)

Retrieve the current FIFO update divider - 1.

See inv_set_fifo_rate() 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 INV_INVALID_FIFO_RATE on error.

#### 4.5.2.10 inv_error_t inv_get_gravity (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.

#### 4.5.2.11 inv_error_t inv_get_gyro (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.

#### 4.5.2.12 inv_error_t inv_get_gyro_and_accel_sensor (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.

#### 4.5.2.13 inv_error_t inv_get_gyro_sensor (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

#### 4.5.2.14 unsigned long inv_get_gyro_sum_of_sqr (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.

#### 4.5.2.15 inv_error_t inv_get_linear_accel (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.

#### 4.5.2.16 inv_error_t inv_get_linear_accel_in_world (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.

#### 4.5.2.17 inv_error_t inv_get_quantized_accel (long * *data*)

Get the Quantized Accel data algorithm output from the FIFO.

**Parameters:**

***data*** a buffer to store the quantized data.

**Returns:**

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

#### 4.5.2.18 inv_error_t inv_get_quaternion (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.

#### 4.5.2.19 inv_error_t inv_get_quaternion_float (float * *data*)

Returns 4-element quaternion vector.

**Parameters:**

***data*** 4-element quaternion vector.

**Returns:**

0 on success, an error code otherwise.

#### 4.5.2.20 int_fast16_t inv_get_sample_frequency (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

#### 4.5.2.21 int_fast16_t inv_get_sample_step_size_ms (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

#### 4.5.2.22 inv_error_t inv_get_temperature (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.

#### 4.5.2.23 inv_error_t inv_get_unquantized_accel (long * *data*)

Get the Decoded Accel Data.

**Parameters:**

***data*** a buffer to store the quantized data.

**Returns:**

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

#### 4.5.2.24 inv_error_t inv_init_fifo_param (void)

Initializes all the internal static variables for the FIFO module.

**Note:**

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

**Returns:**

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

#### 4.5.2.25 inv_error_t inv_read_and_process_fifo (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:**

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

#### 4.5.2.26 inv_error_t inv_send_accel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends accelerometer data to the FIFO.

**Parameters:**

***elements*** Which of the 3 elements to send. Use INV_ALL for 3 axis or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

***accuracy*** Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

#### 4.5.2.27 inv_error_t inv_send_cntrl_data (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 INV_ALL for all or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3, INV_ELEMENT_4 or'd together for a subset.

***accuracy*** Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

#### 4.5.2.28 inv_error_t inv_send_external_sensor_data (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends raw external data to the FIFO.

Should be called once after inv_dmp_open() and before inv_dmp_start().

**Parameters:**

***elements*** Which of the 3 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

***accuracy*** INV_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.

#### 4.5.2.29 inv_error_t inv_send_gravity (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 inv_get_gravity(), 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 INV_ALL.

***accuracy*** The number of bits the gravitation vector is expressed into. Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

**Returns:**

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

#### 4.5.2.30 inv_error_t inv_send_gyro (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 inv_dmp_open() and before inv_dmp_start().

**Parameters:**

***elements*** Which of the 3 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

***accuracy*** Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

#### 4.5.2.31 inv_error_t inv_send_linear_accel (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 INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

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

**Parameters:**

***accuracy*** Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data. Set to zero to remove it from the FIFO.

#### 4.5.2.32 inv_error_t inv_send_linear_accel_in_world (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 inv_dmp_open() and before inv_dmp_start().

**Parameters:**

***elements*** Which of the 3 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 or'd together for a subset.

***accuracy*** Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data.

#### 4.5.2.33 inv_error_t inv_send_packet_number (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:**

INV_SUCCESS or error code

#### 4.5.2.34 inv_error_t inv_send_quantized_accel (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Send the Quantized Acceleromter data into the FIFO.

The data can be retrieved using inv_get_quantized_accel() or inv_get_unquantized_accel().

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

**Parameters:**

***elements*** the components bitmask. To send all compoents use INV_ALL.

***accuracy*** Use INV_32_BIT for 32-bit data or INV_16_BIT for 16-bit data. Set to zero to remove it from the FIFO.

**Returns:**

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

#### 4.5.2.35 inv_error_t inv_send_quaternion (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 inv_dmp_open() and before inv_dmp_start().

**Parameters:**

***accuracy*** Set to INV_32_BIT for 32-bit data, or INV_16_BIT for 16 bit data.

#### 4.5.2.36 inv_error_t inv_send_sensor_data (uint_fast16_t *elements*, uint_fast16_t *accuracy*)

Sends raw data to the FIFO.

Should be called once after inv_dmp_open() and before inv_dmp_start().

**Parameters:**

***elements*** Which of the 7 elements to send. Use INV_ALL for all of them or INV_ELEMENT_1, INV_ELEMENT_2, INV_ELEMENT_3 ... INV_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 INV_16_BIT, INV_32_BIT, or 0 to turn off.

**Returns:**

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

#### 4.5.2.37 inv_error_t inv_set_fifo_processed_callback (void(∗)(void) *func*)

inv_set_fifo_processed_callback is used to set a processed data callback function.

inv_set_fifo_processed_callback 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:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must **NOT** have been called.

**Parameters:**

***func*** A user defined callback function.

**Returns:**

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

#### 4.5.2.38 inv_error_t inv_set_fifo_rate (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 == INV_STATE_DMP_STARTED) then inv_run_state_callbacks() will be called to allow features that depend upon fundamental constants to be updated.

**Precondition:**

inv_dmp_open()

or inv_open_low_power_pedometer() or inv_eis_open_dmp()

and inv_dmp_start() must **NOT** have been called.

**Parameters:**

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

**Returns:**

INV_SUCCESS if successful, ML error code on any failure.

#### 4.5.2.39 inv_error_t inv_set_gyro_data_source (uint_fast8_t *source*)

Set the gyro source to output to the fifo.

**Parameters:**

***source*** The source. One of

- INV_GYRO_FROM_RAW
- INV_GYRO_FROM_QUATERNION

**Returns:**

INV_SUCCESS or non-zero error code;

#### 4.5.2.40 inv_error_t inv_set_linear_accel_filter_coef (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.

## 4.6 Memory Storage

### Files

- file ustore_manager.c

  *Accesses nonvolatile memory.*

- file ustore_stub_io.c

  *Implements input and output for "ustore" routines.*

### Functions

- inv_error_t inv_clear_nvmem (void)
- inv_error_t inv_uload_byte (unsigned char *b)
- inv_error_t inv_uload_calibration (void)
- inv_error_t inv_uload_close (void)
- inv_error_t inv_uload_mem (void *b, int length)
- inv_error_t inv_uload_open (void)
- inv_error_t inv_ustore_byte (unsigned char b)
- inv_error_t inv_ustore_calibration (void)
- inv_error_t inv_ustore_close (void)
- inv_error_t inv_ustore_enable (void)
- inv_error_t inv_ustore_mem (const void *b, int length)
- inv_error_t inv_ustore_open (void)
- inv_error_t inv_ustore_register_handler (const struct uloadstoredelegate *d)
- inv_error_t inv_ustoreload_set_max_len (int l)
- inv_error_t nvmem_read (nvmem_addr_t source, void *destination, int len)
- inv_error_t nvmem_write (nvmem_addr_t destination, const void *source, int len)

### 4.6.1 Function Documentation

#### 4.6.1.1 inv_error_t inv_clear_nvmem (void)

TODO: REPLACE IMPLEMENTATION HERE This implementation just overwrites the existing file with zeroes, which should be sufficient to make the ustore_manager reject the heading tags at the beginning of each stored section. If you are using a filesystem for nvmem, you may want to delete the storage file here. If you are checking some sort of flag on calls to inv_uload_open, you may want to clear it here.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.2 inv_error_t inv_uload_byte (unsigned char * *b*)

TODO: IMPLEMENTATION CHOICE AVAILABLE HERE inv_ustore_byte and inv_ustore_mem may be implemented in terms of the other, i.e. inv_ustore_mem is a loop on inv_ustore_byte, or inv_ustore_byte is a call to inv_ustore_mem with length 1. You can implement whichever of these makes more sense for your platform.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.3 inv_error_t inv_uload_calibration (void)

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.4 inv_error_t inv_uload_close (void)

TODO: INSERT IMPLEMENTATION HERE Along the same lines as inv_ustore_close, except this time close the memory from reading.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.5 inv_error_t inv_uload_mem (void * *b*, int *length*)

TODO: IMPLEMENTATION CHOICE AVAILABLE HERE inv_ustore_byte and inv_ustore_mem may be implemented in terms of the other, i.e. inv_ustore_mem is a loop on inv_ustore_byte, or inv_ustore_byte is a call to inv_ustore_mem with length 1. You can implement whichever of these makes more sense for your platform.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.6 inv_error_t inv_uload_open (void)

TODO: INSERT IMPLEMENTATION HERE Along the same lines as inv_ustore_open, except this time prepare the nonvolatile memory for reading. Prepare pcal to point to beginning of memory space.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.7 inv_error_t inv_ustore_byte (unsigned char *b*)

TODO: IMPLEMENTATION CHOICE AVAILABLE HERE inv_ustore_byte and inv_ustore_mem may be implemented in terms of the other, i.e. inv_ustore_mem is a loop on inv_ustore_byte, or inv_ustore_byte is a call to inv_ustore_mem with length 1. You can implement whichever of these makes more sense for your platform.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.8 inv_error_t inv_ustore_calibration (void)

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.9 inv_error_t inv_ustore_close (void)

TODO: INSERT IMPLEMENTATION HERE This function is called after calls to ustore are complete. Finalize any writes at this time.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.10 inv_error_t inv_ustore_enable (void)

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.11 inv_error_t inv_ustore_mem (const void * *b*, int *length*)

TODO: IMPLEMENTATION CHOICE AVAILABLE HERE inv_ustore_byte and inv_ustore_mem may be implemented in terms of the other, i.e. inv_ustore_mem is a loop on inv_ustore_byte, or inv_ustore_byte is a call to inv_ustore_mem with length 1. You can implement whichever of these makes more sense for your platform.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.12 inv_error_t inv_ustore_open (void)

TODO: INSERT IMPLEMENTATION HERE You should do whatever is appropriate to initialize nonvolatile memory for for storage. This may be a call to fopen, powering up an external storage device, unlocking a memory page, etc. etc.

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.13 inv_error_t inv_ustore_register_handler (const struct uloadstoredelegate * *d*)

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.14 inv_error_t inv_ustoreload_set_max_len (int *l*)

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.15 inv_error_t nvmem_read (nvmem_addr_t *source*, void * *destination*, int *len*)

**Returns:**

INV_SUCCESS if successful.

#### 4.6.1.16 inv_error_t nvmem_write (nvmem_addr_t *destination*, const void * *source*, int *len*)

**Returns:**

INV_SUCCESS if successful.

# Index