

**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](http://www.invensense.com)

Doc : SW-MF-REL-5.1

Doc Rev : 1.0

Date : 09/21/2012

# MotionFit SDK v5.1 APIs Specification

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

This information furnished by InvenSense is believed to be accurate and reliable. However, no responsibility is assumed by InvenSense for its use, or for any infringements or patents or other rights of third parties that may result from its use. Specifications are subject to change without notice. Certain intellectual property owned by InvenSense and described in this document is patent protected. No license is granted by Implication or otherwise under any patent or patent rights of InvenSense. This is an unpublished work protected under the United States copyright laws. This work contains proprietary and confidential information of InvenSense Inc. Use, disclosure or reproduction without the express written authorization of InvenSense Inc. is prohibited. Trademarks that are registered trademarks are the property of their respective companies.

This publication supersedes and replaces all information previously supplied. InvenSense sensors should not be used or sold for the development, storing, production and utilization of any conventional or mass-destructive weapons or any other weapons or life threatening applications, as well as to be used in any other life critical applications such as medical, transportation, aerospace, nuclear, undersea, power, disaster and crime prevention equipment.

Copyright ©2010 InvenSense Corporation.

CONFIDENTIAL &amp; PROPRIETARY

[www.invensense.com](http://www.invensense.com)



**MotionFit SDK v5.1  
APIs Specification**

Doc : SW-MF-REL-5.1  
Doc Rev : 1.0  
Date : 09/21/2012



# Chapter 1

## Purpose and Scope

This document is a guide to all of the functions available in the InvenSense MotionFit SDK, and corresponds with MotionFit SDK Release v5.1.

The MotionFit SDK contains the code for controlling the InvenSense devices, 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 MotionFit SDK are described in this document, including all parameters involved in the function calls. The functions are divided into modules as follows:

Module	Name	Description
<a href="#">Data Builder</a>	Builds Sensor Data Structures	Builds the sensor structures and calls functions that need to use them.
<a href="#">HAL Outputs</a>	HAL Outputs	Creates and holds information that a Android HAL layer might want.
<a href="#">Message Layer</a>	Message Layer	Holds Messages
<a href="#">ML Math Func</a>	Math Functions	Support Math Functions.
<a href="#">MPL</a>	MPU Start	Handles init, start, and version properties .
<a href="#">Result_Holder</a>	Result Holder	Holds various output results.
<a href="#">Start_Manager</a>	Start Manager	Sends start events.
<a href="#">Storage_Manager</a>	Store Variables	Stores Internal States.

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. The document is subdivided in the various sections, each describing the main source [Modules](#) composing the MotionFit SDK and implementing specific features.

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 **MotionFit SDK 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 Alphabetical 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:

compass_vector_cal . . . . .	3
fast_no_mot . . . . .	6
nine_axis_fusion . . . . .	10
gyro_tc . . . . .	14
heading_from_gyro . . . . .	16
mag_disturb . . . . .	18
motion_no_motion . . . . .	20
no_gyro_fusion . . . . .	23
quaternion_supervisor . . . . .	25
data_builder . . . . .	27
ml_math_func . . . . .	43
message_layer . . . . .	51
mpl . . . . .	53
results_holder . . . . .	55
start_manager . . . . .	66
storage_manager . . . . .	68
hal_outputs . . . . .	71
MSP430 System Layer . . . . .	75
Sensor Driver Layer . . . . .	82



## MotionFit SDK v5.1 APIs Specification

Doc : SW-MF-REL-5.1  
Doc Rev : 1.0  
Date : 09/21/2012

## Chapter 4

# Module Documentation

### 4.1 compass\_vector\_cal

A compass calibration algorithm that is mutually exclusive with compass\_fit.

#### Files

- file [compass\\_vec\\_cal.c](#)

#### Functions

- `inv_error_t` [inv\\_disable\\_vector\\_compass\\_cal](#) (void)  
*Disables a precise compass bias algorithm.*
- `inv_error_t` [inv\\_enable\\_vector\\_compass\\_cal](#) (void)  
*Enables a precise compass bias algorithm.*
- `inv_error_t` [inv\\_init\\_vector\\_compass\\_cal](#) (void)  
*Initializes/Resets this module.*
- `inv_error_t` [inv\\_start\\_vector\\_compass\\_cal](#) (void)  
*Allows the user to start a precise compass bias algorithm.*
- `inv_error_t` [inv\\_stop\\_vector\\_compass\\_cal](#) (void)  
*Allows the user to stop a precise compass bias algorithm.*

### 4.1.1 Detailed Description

A compass calibration algorithm that is mutually exclusive with `compass_fit`.

### 4.1.2 Function Documentation

#### 4.1.2.1 `inv_error_t inv_disable_vector_compass_cal (void)`

Disables a precise compass bias algorithm.

Should only be called once per library load when you wish to remove this functionality. See [inv\\_stop\\_vector\\_compass\\_cal\(\)](#) if you wish to simply stop the algorithm.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

#### 4.1.2.2 `inv_error_t inv_enable_vector_compass_cal (void)`

Enables a precise compass bias algorithm.

This may be called after [inv\\_init\\_mpl\(\)](#) and before [inv\\_start\\_mpl\(\)](#). It is typically only called once per session. It does not return a motion state. Mutually exclusive with `inv_enable_compass_fit()`.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

#### 4.1.2.3 `inv_error_t inv_init_vector_compass_cal (void)`

Initializes/Resets this module.

Called by [inv\\_enable\\_vector\\_compass\\_cal\(\)](#). If you are calling this for testing, you probably also want to call `inv_init_adv_fusion_obj()`

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

#### 4.1.2.4 `inv_error_t inv_start_vector_compass_cal (void)`

Allows the user to start a precise compass bias algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by [inv\\_stop\\_vector\\_compass\\_cal\(\)](#).





## **4.1 compass\_vector\_cal**

**5**

### **Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

### **4.1.2.5 inv\_error\_t inv\_stop\_vector\_compass\_cal (void)**

Allows the user to stop a precise compass bias algorithm.

To start the algorithm back up call [inv\\_start\\_vector\\_compass\\_cal\(\)](#)

### **Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

## 4.2 fast\_no\_mot

Fast no motion algorithm used to set the gyro bias.

### Files

- file [fast\\_no\\_motion.c](#)  
*Fast no motion algorithm.*

### Functions

- void [inv\\_set\\_fast\\_nomot\\_gyro\\_threshold](#) (float thresh)  
*Sets internal threshold for fast no motion.*
- inv\_error\_t [inv\\_disable\\_fast\\_nomot](#) (void)  
*Turns off a faster Motion/No Motion to set gyro biases (see [inv\\_enable\\_fast\\_nomot\(\)](#)).*
- inv\_error\_t [inv\\_enable\\_fast\\_nomot](#) (void)  
*Turns on a faster Motion/No Motion to set gyro biases.*
- void [inv\\_get\\_fast\\_nomot\\_accel\\_param](#) (long \*cntr, float \*param)  
*This is used to help set [inv\\_set\\_fast\\_nomot\\_accel\\_threshold\(\)](#).*
- void [inv\\_get\\_fast\\_nomot\\_compass\\_param](#) (long \*cntr, float \*param)  
*This is used to help set [inv\\_set\\_fast\\_nomot\\_compass\\_threshold\(\)](#).*
- float [inv\\_get\\_fnm\\_gyro\\_no\\_motion\\_param](#) (void)  
*Get gyro parameters.*
- inv\_error\_t [inv\\_init\\_fast\\_nomot](#) (void)  
*Initializes the fast no motion algorithm.*
- void [inv\\_set\\_default\\_number\\_of\\_samples](#) (int N)  
*Set default number of samples.*
- void [inv\\_set\\_fast\\_nomot\\_accel\\_threshold](#) (float thresh)  
*Used to set internal threshold.*
- void [inv\\_set\\_fast\\_nomot\\_compass\\_threshold](#) (float thresh)  
*Used to set internal threshold.*

## 4.2 fast\_no\_mot

7

- `inv_error_t inv_start_fast_nomot (void)`  
*Allows the user to start the fast no motion algorithm.*
- `inv_error_t inv_stop_fast_nomot (void)`  
*Allows the user to stop the fast no motion algorithm.*

### 4.2.1 Detailed Description

Fast no motion algorithm used to set the gyro bias.

### 4.2.2 Function Documentation

#### 4.2.2.1 `inv_error_t inv_disable_fast_nomot (void)`

Turns off a faster Motion/No Motion to set gyro biases (see [inv\\_enable\\_fast\\_nomot\(\)](#)).

It is typically only called once per session. It does not return a motion state. It is mutually exclusive with [inv\\_enable\\_motion\\_no\\_motion\(\)](#).

#### Returns:

INV\_SUCCESS on success or an error code if call was not successful.

#### 4.2.2.2 `inv_error_t inv_enable_fast_nomot (void)`

Turns on a faster Motion/No Motion to set gyro biases.

This may be called after [inv\\_init\\_mpl\(\)](#) and before [inv\\_start\\_mpl\(\)](#). It is typically only called once per session. It does not return a motion state. It is mutually exclusive with [inv\\_enable\\_motion\\_no\\_motion\(\)](#).

#### Returns:

INV\_SUCCESS on success or an error code if call was not successful.

#### 4.2.2.3 `void inv_get_fast_nomot_accel_param (long * cntr, float * param)`

This is used to help set [inv\\_set\\_fast\\_nomot\\_accel\\_threshold\(\)](#).

*cntr* is incremented each time there is a new value of *param*. 100 new values should be sorted from low to high and the 97th value should be used as the threshold in [inv\\_set\\_fast\\_nomot\\_accel\\_threshold\(\)](#). The compass must be on.

**Parameters:**

*cntr* Counter for when param changes  
*param* Parameter used to help set threshold

**4.2.2.4 void inv\_get\_fast\_nomot\_compass\_param (long \* *cntr*, float \* *param*)**

This is used to help set [inv\\_set\\_fast\\_nomot\\_compass\\_threshold\(\)](#).

*cntr* is incremented each time there is a new value of *param*. 100 new values should be sorted from low to high and the 97th value should be used as the threshold value to be provided to [inv\\_set\\_fast\\_nomot\\_compass\\_threshold\(\)](#). The compass must be on.

**Parameters:**

*cntr* Counter for when param changes  
*param* Parameter used to help set threshold

**4.2.2.5 inv\_error\_t inv\_init\_fast\_nomot (void)**

Initializes the fast no motion algorithm.

Automatically called by [inv\\_enable\\_fast\\_nomot\(\)](#). Not typically called by the user.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.2.2.6 void inv\_set\_default\_number\_of\_samples (int *N*)**

Set default number of samples.

Not typically called by users.

**Parameters:**

*N* Number of samples to use for algorithm

**4.2.2.7 void inv\_set\_fast\_nomot\_accel\_threshold (float *thresh*)**

Used to set internal threshold.

This may need to be set based upon device environment.

**4.2 fast\_no\_mot****9****See also:**

[inv\\_get\\_fast\\_nomot\\_accel\\_param\(\)](#) for a range of values to set this to.

**Parameters:**

*thresh*

**4.2.2.8 void inv\_set\_fast\_nomot\_compass\_threshold (float *thresh*)**

Used to set internal threshold.

This may need to be set based upon device environment.

**See also:**

[inv\\_get\\_fast\\_nomot\\_compass\\_param\(\)](#) for a range of values to set this to.

**Parameters:**

*thresh*

**4.2.2.9 inv\_error\_t inv\_start\_fast\_nomot (void)**

Allows the user to start the fast no motion algorithm.

It is automatically in start mode after an enable. This function only needs to be called to start after a stop command generated by [inv\\_stop\\_fast\\_nomot\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.2.2.10 inv\_error\_t inv\_stop\_fast\_nomot (void)**

Allows the user to stop the fast no motion algorithm.

See [inv\\_start\\_fast\\_nomot\(\)](#) to start the algorithm back up.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

## 4.3 nine\_axis\_fusion

Performs nine axis sensor fusion.

### Files

- file [fusion\\_9axis.c](#)  
*Performs nine axis sensor fusion.*

### Functions

- `inv_error_t inv_9x_fusion_enable_jitter_reduction (int en)`  
*This enables the jitter reduction feature.*
- `inv_error_t inv_9x_fusion_set_mag_fb (double fb)`  
*This sets the magnetic feedback.*
- `inv_error_t inv_9x_fusion_use_timestamps (int en)`  
*Use timestamps when evaluating compass correction gain.*
- `inv_error_t inv_disable_9x_sensor_fusion ()`  
*Disables the 9 axis sensor fusion algorithm.*
- `inv_error_t inv_enable_9x_sensor_fusion (void)`  
*Enables the 9 axis sensor fusion algorithm.*
- `void inv_init_9x_fusion (void)`  
*Initializes the algorithm.*
- `inv_error_t inv_start_9x_sensor_fusion (void)`  
*Starts the 9 axis sensor fusion.*
- `inv_error_t inv_stop_9x_sensor_fusion (void)`  
*Stops the 9 axis sensor fusion from running.*

### 4.3.1 Detailed Description

Performs nine axis sensor fusion.

## 4.3 nine\_axis\_fusion

**11**

### 4.3.2 Function Documentation

#### 4.3.2.1 `inv_error_t inv_9x_fusion_enable_jitter_reduction (int en)`

This enables the jitter reduction feature.

**Parameters:**

*en* Should be non-zero to enable the feature. Initialized to 0, i.e. off

**Returns:**

heading correction angle

#### 4.3.2.2 `inv_error_t inv_9x_fusion_set_mag_fb (double fb)`

This sets the magnetic feedback.

Increasing it results in faster compass correction in the 9 axis quaternion.

**Parameters:**

*fb* Desired magnetic feedback value. Typical value is 1. Also, initialized to 1 in `inv_init_9x_fusion`.

**Returns:**

heading correction angle

#### 4.3.2.3 `inv_error_t inv_9x_fusion_use_timestamps (int en)`

Use timestamps when evaluating compass correction gain.

This feature should be used when the MPL is not receiving compass data at a constant rate.

**Parameters:**

*en* 1 to enable the feature.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.3.2.4 inv\_error\_t inv\_disable\_9x\_sensor\_fusion ()**

Disables the 9 axis sensor fusion algorithm.

Should only be called once per library load when you wish to remove this functionality. See [inv\\_stop\\_9x\\_sensor\\_fusion\(\)](#) if you wish to simply stop the algorithm.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.3.2.5 inv\_error\_t inv\_enable\_9x\_sensor\_fusion (void)**

Enables the 9 axis sensor fusion algorithm.

This should only be called once per library load. See [inv\\_start\\_9x\\_sensor\\_fusion\(\)](#) and [inv\\_stop\\_9x\\_sensor\\_fusion\(\)](#) for starting and stopping. Automatically calls [inv\\_start\\_9x\\_sensor\\_fusion\(\)](#) and [inv\\_init\\_9x\\_fusion\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.3.2.6 void inv\_init\_9x\_fusion (void)**

Initializes the algorithm.

Automatically called by [inv\\_enable\\_9x\\_sensor\\_fusion\(\)](#). Not normally called by users.

**4.3.2.7 inv\_error\_t inv\_start\_9x\_sensor\_fusion (void)**

Starts the 9 axis sensor fusion.

Automatically called by [inv\\_enable\\_9x\\_sensor\\_fusion\(\)](#) and only needs to be called after stopping with [inv\\_stop\\_9x\\_sensor\\_fusion\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.3.2.8 inv\_error\_t inv\_stop\_9x\_sensor\_fusion (void)**

Stops the 9 axis sensor fusion from running.

See [inv\\_start\\_9x\\_sensor\\_fusion\(\)](#) to start it back up again.





#### **4.3 nine\_axis\_fusion**

**13**

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

## 4.4 gyro\_tc

Gyro Temperature Compensation algorithm.

### Files

- file [gyro\\_tc.c](#)  
*Gyro bias temperature compensation.*

### Functions

- `inv_error_t` [inv\\_disable\\_gyro\\_tc](#) (void)  
*Enable the gyro temp comp algorithm.*
- `inv_error_t` [inv\\_enable\\_gyro\\_tc](#) (void)  
*Enable the gyro temp comp algorithm.*
- `inv_error_t` [inv\\_init\\_gyro\\_ts](#) (void)  
*Reset the gyro temp slope.*
- `inv_error_t` [inv\\_start\\_gyro\\_tc](#) (void)  
*Registers callback to receive new temperature data.*
- `inv_error_t` [inv\\_stop\\_gyro\\_tc](#) (void)  
*Unregisters callback.*

### 4.4.1 Detailed Description

Gyro Temperature Compensation algorithm.

### 4.4.2 Function Documentation

#### 4.4.2.1 `inv_error_t` [inv\\_disable\\_gyro\\_tc](#) (void)

Enable the gyro temp comp algorithm.

#### Returns:

INV\_SUCCESS if successful.

#### **4.4 gyro\_tc**

**15**

##### **4.4.2.2 inv\_error\_t inv\_enable\_gyro\_tc (void)**

Enable the gyro temp comp algorithm.

**Returns:**

INV\_SUCCESS if successful.

##### **4.4.2.3 inv\_error\_t inv\_init\_gyro\_ts (void)**

Reset the gyro temp slope.

**Returns:**

INV\_SUCCESS if successful.

##### **4.4.2.4 inv\_error\_t inv\_start\_gyro\_tc (void)**

Registers callback to receive new temperature data.

**Returns:**

INV\_SUCCESS if successful.

##### **4.4.2.5 inv\_error\_t inv\_stop\_gyro\_tc (void)**

Unregisters callback.

**Returns:**

INV\_SUCCESS if successful.

## 4.5 heading\_from\_gyro

A less accurate but fast algorithm for 9 axis sensor fusion.

### Files

- file [heading\\_from\\_gyro.c](#)

### Functions

- `inv_error_t inv_disable_heading_from_gyro (void)`  
*Turns off a heading from gyro.*
- `inv_error_t inv_enable_heading_from_gyro (void)`  
*Turns on a heading from gyro algorithm which performs sensor fusion when the compass bias hasn't been fully solved for.*
- `void inv_init_heading_from_gyro (void)`  
*Initializes/Resets this module.*
- `inv_error_t inv_start_heading_from_gyro (void)`  
*Registers callback to receive gyro and compass data.*
- `inv_error_t inv_stop_heading_from_gyro (void)`  
*Unregisters callback.*

### 4.5.1 Detailed Description

A less accurate but fast algorithm for 9 axis sensor fusion.

### 4.5.2 Function Documentation

#### 4.5.2.1 `inv_error_t inv_disable_heading_from_gyro (void)`

Turns off a heading from gyro.

It is typically only called once per session.

#### Returns:

INV\_SUCCESS if successful.

#### 4.5 heading\_from\_gyro

17

##### 4.5.2.2 `inv_error_t inv_enable_heading_from_gyro (void)`

Turns on a heading from gyro algorithm which performs sensor fusion when the compass bias hasn't been fully solved for.

This may be called after [inv\\_init\\_mpl\(\)](#) and before [inv\\_start\\_mpl\(\)](#). It is typically only called once per session.

**Returns:**

INV\_SUCCESS if successful.

##### 4.5.2.3 `void inv_init_heading_from_gyro (void)`

Initializes/Resets this module.

Called by [inv\\_enable\\_heading\\_from\\_gyro\(\)](#).

**Returns:**

INV\_SUCCESS if successful.

##### 4.5.2.4 `inv_error_t inv_start_heading_from_gyro (void)`

Registers callback to receive gyro and compass data.

**Returns:**

INV\_SUCCESS if successful.

##### 4.5.2.5 `inv_error_t inv_stop_heading_from_gyro (void)`

Unregisters callback.

**Returns:**

INV\_SUCCESS if successful.

## 4.6 mag\_disturb

Determines magnetic disturbances and sets compass accuracy appropriately.

### Files

- file [mag\\_disturb.c](#)

### Functions

- `inv_error_t inv_disable_magnetic_disturbance (void)`  
*Turns off a magnetic disturbance algorithm (see [inv\\_enable\\_magnetic\\_disturbance\(\)](#)).*
- `inv_error_t inv_enable_magnetic_disturbance (void)`  
*Enables a magnetic disturbance algorithm.*
- `inv_error_t inv_start_magnetic_disturbance (void)`  
*Allows the user to start the magnetic disturbance algorithm.*
- `inv_error_t inv_stop_magnetic_disturbance (void)`  
*Allows the user to stop the magnetic disturbance algorithm.*

### 4.6.1 Detailed Description

Determines magnetic disturbances and sets compass accuracy appropriately.

### 4.6.2 Function Documentation

#### 4.6.2.1 `inv_error_t inv_disable_magnetic_disturbance (void)`

Turns off a magnetic disturbance algorithm (see [inv\\_enable\\_magnetic\\_disturbance\(\)](#)).

It is typically only called once per session. See [inv\\_stop\\_magnetic\\_disturbance\(\)](#) to stop the algorithm

#### Returns:

INV\_SUCCESS on success or an error code if call was not successful.

**4.6 mag\_disturb****19****4.6.2.2 inv\_error\_t inv\_enable\_magnetic\_disturbance (void)**

Enables a magnetic disturbance algorithm.

This may be called after [inv\\_init\\_mpl\(\)](#) and before [inv\\_start\\_mpl\(\)](#). It is typically only called once per session. It does not return a motion state.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.6.2.3 inv\_error\_t inv\_start\_magnetic\_disturbance (void)**

Allows the user to start the magnetic disturbance algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by [inv\\_stop\\_magnetic\\_disturbance\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.6.2.4 inv\_error\_t inv\_stop\_magnetic\_disturbance (void)**

Allows the user to stop the magnetic disturbance algorithm.

To start the algorithm back up call [inv\\_start\\_no\\_gyro\\_fusion\(\)](#)

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

## 4.7 motion\_no\_motion

A motion detection algorithm that is used to set gyro bias when the device is not moving.

### Files

- file [motion\\_no\\_motion.c](#)

*A motion detection algorithm that is used to set gyro bias when the device is not moving.*

### Functions

- `inv_error_t inv_disable_motion_no_motion (void)`  
*Turns off Motion/No Motion to set gyro biases (see [inv\\_enable\\_motion\\_no\\_motion\(\)](#)).*
- `inv_error_t inv_enable_motion_no_motion ()`  
*Turns on Motion/No Motion used to set gyro biases.*
- `inv_error_t inv_init_motion_no_motion (void)`  
*Initializes the motion no motion algorithm.*
- `inv_error_t inv_set_no_motion_time (long time_ms)`  
*Allows the user to set the time to be in a no motion state before setting the gyro bias.*
- `inv_error_t inv_start_motion_no_motion (void)`  
*Allows the user to start the no motion algorithm.*
- `inv_error_t inv_stop_motion_no_motion (void)`  
*Allows the user to stop the no motion algorithm.*

### 4.7.1 Detailed Description

A motion detection algorithm that is used to set gyro bias when the device is not moving.



**4.7 motion\_no\_motion****21****4.7.2 Function Documentation****4.7.2.1 `inv_error_t inv_disable_motion_no_motion (void)`**

Turns off Motion/No Motion to set gyro biases (see [inv\\_enable\\_motion\\_no\\_motion\(\)](#)).

It is typically only called once per session. It does not return a motion state. It is mutually exclusive with [inv\\_enable\\_fast\\_nomot\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.7.2.2 `inv_error_t inv_enable_motion_no_motion ()`**

Turns on Motion/No Motion used to set gyro biases.

This may be called after [inv\\_init\\_mpl\(\)](#) and before [inv\\_start\\_mpl\(\)](#). It is typically only called once per session. It does not return a motion state. It is mutually exclusive with [inv\\_enable\\_motion\\_no\\_motion\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.7.2.3 `inv_error_t inv_init_motion_no_motion (void)`**

Initializes the motion no motion algorithm.

Automatically called by [inv\\_enable\\_motion\\_no\\_motion\(\)](#). Not typically called by the user.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.7.2.4 `inv_error_t inv_set_no_motion_time (long time_ms)`**

Allows the user to set the time to be in a no motion state before setting the gyro bias.

**Parameters:**

*time\_ms* Time in milliseconds. Default is 8000ms or 8 seconds.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.7.2.5 inv\_error\_t inv\_start\_motion\_no\_motion (void)**

Allows the user to start the no motion algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by [inv\\_stop\\_motion\\_no\\_motion\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.7.2.6 inv\_error\_t inv\_stop\_motion\_no\_motion (void)**

Allows the user to stop the no motion algorithm.

See [inv\\_start\\_motion\\_no\\_motion\(\)](#) to start the algorithm back up.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

## 4.8 no\_gyro\_fusion

Accel/Compass Sensor fusion.

### Files

- file [no\\_gyro\\_fusion.c](#)  
*Accel/Compass Sensor fusion.*

### Functions

- `inv_error_t inv_disable_no_gyro_fusion (void)`  
*Turns off a sensor fusion using accel and compass only (see [inv\\_enable\\_no\\_gyro\\_fusion\(\)](#)).*
- `inv_error_t inv_enable_no_gyro_fusion (void)`  
*Enables a sensor fusion using accel and compass only.*
- `inv_error_t inv_init_no_gyro_fusion (void)`  
*Initializes the algorithm.*
- `inv_error_t inv_start_no_gyro_fusion (void)`  
*Allows the user to start the sensor fusion using accel and compass only algorithm.*
- `inv_error_t inv_stop_no_gyro_fusion (void)`  
*Allows the user to stop the sensor fusion using accel and compass only algorithm.*

### 4.8.1 Detailed Description

Accel/Compass Sensor fusion.

### 4.8.2 Function Documentation

#### 4.8.2.1 `inv_error_t inv_disable_no_gyro_fusion (void)`

Turns off a sensor fusion using accel and compass only (see [inv\\_enable\\_no\\_gyro\\_fusion\(\)](#)).

It is typically only called once per session. See [inv\\_stop\\_no\\_gyro\\_fusion\(\)](#) to stop the algorithm

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.8.2.2 inv\_error\_t inv\_enable\_no\_gyro\_fusion (void)**

Enables a sensor fusion using accel and compass only.

This may be called after [inv\\_init\\_mpl\(\)](#) and before [inv\\_start\\_mpl\(\)](#). It is typically only called once per session. It does not return a motion state.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.8.2.3 inv\_error\_t inv\_init\_no\_gyro\_fusion (void)**

Initializes the algorithm.

Automatically called by the enable function.

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.8.2.4 inv\_error\_t inv\_start\_no\_gyro\_fusion (void)**

Allows the user to start the sensor fusion using accel and compass only algorithm.

It is automatically called in start mode after an enable. This function only needs to be called to start after a stop command generated by [inv\\_stop\\_no\\_gyro\\_fusion\(\)](#).

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.8.2.5 inv\_error\_t inv\_stop\_no\_gyro\_fusion (void)**

Allows the user to stop the sensor fusion using accel and compass only algorithm.

See [inv\\_start\\_no\\_gyro\\_fusion\(\)](#) to start the algorithm back up call [inv\\_start\\_no\\_gyro\\_fusion\(\)](#)

**Returns:**

INV\_SUCCESS on success or an error code if call was not successful.

**4.9 quaternion\_supervisor****25****4.9 quaternion\_supervisor**

Motion Library - Generates the 6-axis quaternion.

**Files**

- file [quaternion\\_supervisor.c](#)  
*Performs the quaternion fusion.*

**Functions**

- `inv_error_t` [inv\\_disable\\_quaternion](#) (void)  
*Disables generating the gyro and accel quaternion.*
- `inv_error_t` [inv\\_enable\\_quaternion](#) ()  
*Turns on quaternion computation.*
- `inv_error_t` [inv\\_init\\_quaternion](#) (void)  
*Initializes all quaternion data.*
- `inv_error_t` [inv\\_start\\_quaternion](#) (void)  
*Starts gyro and accel quaternion generation.*
- `inv_error_t` [inv\\_stop\\_quaternion](#) (void)  
*Stops gyro and accel quaternion generation.*

**4.9.1 Detailed Description**

Motion Library - Generates the 6-axis quaternion.

**4.9.2 Function Documentation****4.9.2.1 `inv_error_t inv_enable_quaternion` ()**

Turns on quaternion computation.

This must be called after [inv\\_init\\_mpl\(\)](#) and before [inv\\_start\\_mpl\(\)](#). It is typically only called once per session. [inv\\_start\\_quaternion\(\)](#) and [inv\\_stop\\_quaternion\(\)](#) are used

to start and stop this feature. This feature is started automatically and `inv_start_quaternion()` would only need to be called after turning this feature off with `inv_stop_quaternion()`.

**Returns:**

INV\_SUCCESS=0 on success, a non-zero error code otherwise.

**4.9.2.2 `inv_error_t inv_init_quaternion (void)`**

Initializes all quaternion data.

This is called automatically by the enable function. It may be called any time the feature is enabled, but is typically not needed to be called by outside callers.

**Returns:**

INV\_SUCCESS=0 on success, a non-zero error code otherwise.

**4.9.2.3 `inv_error_t inv_start_quaternion (void)`**

Starts gyro and accel quaternion generation.

Automatically called by `inv_enable_quaternion()` and therefor would only need to be called after `inv_stop_quaternion()`.

**Returns:**

INV\_SUCCESS=0 on success, a non-zero error code otherwise.

**4.9.2.4 `inv_error_t inv_stop_quaternion (void)`**

Stops gyro and accel quaternion generation.

Call `inv_start_quaternion()` to turn this back on after the stop command.

**Returns:**

INV\_SUCCESS=0 on success, a non-zero error code otherwise.

## 4.10 data\_builder

Motion Library - Data Builder Constructs and Creates the data for MPL.

### Files

- file [data\\_builder.c](#)  
*Data Builder.*

### Defines

- #define [INV\\_DB\\_SAVE\\_KEY](#) 53395  
*Change this key if the data being stored by this file changes.*

### Functions

- void [inv\\_accel\\_was\\_turned\\_off](#) ()  
*This should be called when the accel has been turned off.*
- void [inv\\_apply\\_calibration](#) (struct [inv\\_single\\_sensor\\_t](#) \*sensor, const long \*bias)  
*Takes raw data stored in the sensor, removes bias, and converts it to calibrated data in the body frame.*
- [inv\\_error\\_t](#) [inv\\_build\\_accel](#) (const long \*accel, int status, [inv\\_time\\_t](#) timestamp)  
*Record new accel data for use when [inv\\_execute\\_on\\_data\(\)](#) is called.*
- [inv\\_error\\_t](#) [inv\\_build\\_compass](#) (const long \*compass, int status, [inv\\_time\\_t](#) timestamp)  
*Record new compass data for use when [inv\\_execute\\_on\\_data\(\)](#) is called.*
- [inv\\_error\\_t](#) [inv\\_build\\_gyro](#) (const short \*gyro, [inv\\_time\\_t](#) timestamp)  
*Record new gyro data and calls [inv\\_execute\\_on\\_data\(\)](#) if previous sample has not been processed.*
- [inv\\_error\\_t](#) [inv\\_build\\_quat](#) (const long \*quat, int status, [inv\\_time\\_t](#) timestamp)  
*quaternion data*
- [inv\\_error\\_t](#) [inv\\_build\\_temp](#) (const long temp, [inv\\_time\\_t](#) timestamp)

*Record new temperature data for use when [inv\\_execute\\_on\\_data\(\)](#) is called.*

- void [inv\\_compass\\_was\\_turned\\_off](#) ()  
*This should be called when the compass has been turned off.*
- inv\_error\_t [inv\\_execute\\_on\\_data](#) (void)  
*After at least one of [inv\\_build\\_gyro\(\)](#), [inv\\_build\\_accel\(\)](#), or [inv\\_build\\_compass\(\)](#) has been called, this function should be called.*
- int [inv\\_get\\_accel\\_accuracy](#) (void)  
*Returns accuracy of accel.*
- void [inv\\_get\\_accel\\_bias](#) (long \*bias, long \*temp)  
*Get Accel Bias.*
- int [inv\\_get\\_accel\\_on](#) ()  
*Helper function stating whether the accelerometer is on or off.*
- long [inv\\_get\\_accel\\_sensitivity](#) (void)  
*Accel sensitivity.*
- void [inv\\_get\\_accel\\_set](#) (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Gets a whole set of accel data including data, accuracy and timestamp.*
- void [inv\\_get\\_compass\\_bias](#) (long \*bias)  
*Returns the current bias for the compass.*
- int [inv\\_get\\_compass\\_on](#) ()  
*Helper function stating whether the compass is on or off.*
- long [inv\\_get\\_compass\\_sensitivity](#) (void)  
*Compass sensitivity.*
- void [inv\\_get\\_compass\\_set](#) (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Gets a whole set of compass data including data, accuracy and timestamp.*
- void [inv\\_get\\_gyro](#) (long \*gyro)  
*Get's latest gyro data.*
- int [inv\\_get\\_gyro\\_accuracy](#) (void)  
*Returns accuracy of gyro.*



**4.10 data\_builder****29**

- void [inv\\_get\\_gyro\\_bias](#) (long \*bias, long \*temp)  
*Get the gyro biases and temperature record from MPL.*
- int [inv\\_get\\_gyro\\_on](#) ()  
*Helper function stating whether the gyro is on or off.*
- long [inv\\_get\\_gyro\\_sensitivity](#) ()  
*Gyro sensitivity.*
- void [inv\\_get\\_gyro\\_set](#) (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Gets a whole set of gyro data including data, accuracy and timestamp.*
- inv\_time\_t [inv\\_get\\_last\\_timestamp](#) ()  
*Get last timestamp across all 3 sensors that are on.*
- int [inv\\_get\\_mag\\_accuracy](#) (void)  
*Returns accuracy of compass.*
- void [inv\\_get\\_temp\\_set](#) (long \*data, int \*accuracy, inv\_time\_t \*timestamp)  
*Gets a whole set of temperature data including data, accuracy and timestamp.*
- void [inv\\_gyro\\_was\\_turned\\_off](#) ()  
*This should be called when the gyro has been turned off.*
- inv\_error\_t [inv\\_init\\_data\\_builder](#) (void)  
*Initialize the data builder.*
- void [inv\\_quaternion\\_sensor\\_was\\_turned\\_off](#) (void)  
*This should be called when the quaternion data from the DMP has been turned off.*
- inv\_error\_t [inv\\_register\\_data\\_cb](#) (inv\_error\_t(\*func)(struct inv\_sensor\_cal\_t \*data), int priority, int sensor\_type)  
*Registers to receive a callback when there is new sensor data.*
- void [inv\\_set\\_accel\\_bandwidth](#) (int bandwidth\_hz)  
*Set Accel Bandwidth in Hz.*
- void [inv\\_set\\_accel\\_bias](#) (const long \*bias, int accuracy)  
*Sets the accel bias.*
- void [inv\\_set\\_accel\\_bias\\_mask](#) (const long \*bias, int accuracy, int mask)  
*Sets the accel bias with control over which axis.*

- void [inv\\_set\\_accel\\_orientation\\_and\\_scale](#) (int orientation, long sensitivity)  
*Sets the orientation and sensitivity of the gyro data.*
- void [inv\\_set\\_accel\\_sample\\_rate](#) (long sample\_rate\_us)  
*Set Accel Sample rate in micro seconds.*
- void [inv\\_set\\_compass\\_bandwidth](#) (int bandwidth\_hz)  
*Set Compass Bandwidth in Hz.*
- void [inv\\_set\\_compass\\_disturbance](#) (int dist)  
*Set the state of a compass disturbance.*
- void [inv\\_set\\_compass\\_orientation\\_and\\_scale](#) (int orientation, long sensitivity)  
*Sets the Orientation and Sensitivity of the gyro data.*
- void [inv\\_set\\_compass\\_sample\\_rate](#) (long sample\_rate\_us)  
*Set Compass Sample rate in micro seconds.*
- void [inv\\_set\\_gyro\\_bandwidth](#) (int bandwidth\_hz)  
*Set Gyro Bandwidth in Hz.*
- void [inv\\_set\\_gyro\\_bias](#) (const long \*bias, int accuracy)  
*Sets the gyro bias.*
- void [inv\\_set\\_gyro\\_orientation\\_and\\_scale](#) (int orientation, long sensitivity)  
*Sets the Orientation and Sensitivity of the gyro data.*
- void [inv\\_set\\_gyro\\_sample\\_rate](#) (long sample\_rate\_us)  
*Set Gyro Sample rate in micro seconds.*
- void [inv\\_set\\_quat\\_sample\\_rate](#) (long sample\_rate\_us)  
*Set Quat Sample rate in micro seconds.*
- void [inv\\_temperature\\_was\\_turned\\_off](#) ()  
*This should be called when the temperature sensor has been turned off.*
- inv\_error\_t [inv\\_unregister\\_data\\_cb](#) (inv\_error\_t(\*func)(struct inv\_sensor\_cal\_t \*data))  
*Unregisters the callback that happens when new sensor data is received.*
- void [set\\_sensor\\_orientation\\_and\\_scale](#) (struct inv\_single\_sensor\_t \*sensor, int orientation, long sensitivity)

**4.10 data\_builder****31**

*Sets orientation and sensitivity field for a sensor.*

**4.10.1 Detailed Description**

Motion Library - Data Builder Constructs and Creates the data for MPL.

**4.10.2 Function Documentation****4.10.2.1 void inv\_accel\_was\_turned\_off ()**

This should be called when the accel has been turned off.

This is so that we will know if the data is contiguous.

**4.10.2.2 void inv\_apply\_calibration (struct inv\_single\_sensor\_t \* *sensor*, const long \* *bias*)**

Takes raw data stored in the sensor, removes bias, and converts it to calibrated data in the body frame.

**Parameters:**

*sensor* structure to modify

*bias* bias in the mounting frame, in hardware units scaled by  $2^{16}$ . Length 3.

**4.10.2.3 inv\_error\_t inv\_build\_accel (const long \* *accel*, int *status*, inv\_time\_t *timestamp*)**

Record new accel data for use when [inv\\_execute\\_on\\_data\(\)](#) is called.

**Parameters:**

*accel* accel data. Length 3. Calibrated data is in  $m/s^2$  scaled by  $2^{16}$  in body frame. Raw data is in device units in chip mounting frame.

*status* Lower 2 bits are the accuracy, with 0 being inaccurate, and 3 being most accurate. The upper bit INV\_CALIBRATED, is set if the data was calibrated outside MPL and it is not set if the data being passed is raw. Raw data should be in device units, typically in a 16-bit range.

*timestamp* Monotonic time stamp, for Android it's in nanoseconds.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.10.2.4 `inv_error_t inv_build_compass (const long * compass, int status,  
inv_time_t timestamp)`**

Record new compass data for use when [inv\\_execute\\_on\\_data\(\)](#) is called.

**Parameters:**

***compass*** Compass data, if it was calibrated outside MPL, the units are uT scaled by  $2^{16}$ . Length 3.

***status*** Lower 2 bits are the accuracy, with 0 being inaccurate, and 3 being most accurate. The upper bit INV\_CALIBRATED, is set if the data was calibrated outside MPL and it is not set if the data being passed is raw. Raw data should be in device units, typically in a 16-bit range.

***timestamp*** Monotonic time stamp, for Android it's in nanoseconds.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.10.2.5 `inv_error_t inv_build_gyro (const short * gyro, inv_time_t timestamp)`**

Record new gyro data and calls [inv\\_execute\\_on\\_data\(\)](#) if previous sample has not been processed.

**Parameters:**

***gyro*** Data is in device units. Length 3.

***timestamp*** Monotonic time stamp, for Android it's in nanoseconds.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.10.2.6 `inv_error_t inv_build_quat (const long * quat, int status, inv_time_t  
timestamp)`**

quaternion data

**Parameters:**

***quat*** Quaternion data.  $2^{30} = 1.0$  or  $2^{14} = 1$  for 16-bit data. Real part first. Length 4.

***status*** number of axis, 16-bit or 32-bit

**4.10 data\_builder****33***timestamp**timestamp* Monotonic time stamp; for Android it's in nanoseconds.**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.10.2.7 inv\_error\_t inv\_build\_temp (const long temp, inv\_time\_t timestamp)**Record new temperature data for use when [inv\\_execute\\_on\\_data\(\)](#) is called.**Parameters:***temp* Temperature data in q16 format.*timestamp* Monotonic time stamp; for Android it's in nanoseconds.**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.10.2.8 void inv\_compass\_was\_turned\_off ()**

This should be called when the compass has been turned off.

This is so that we will know if the data is contiguous.

**4.10.2.9 inv\_error\_t inv\_execute\_on\_data (void)**After at least one of [inv\\_build\\_gyro\(\)](#), [inv\\_build\\_accel\(\)](#), or [inv\\_build\\_compass\(\)](#) has been called, this function should be called.

It will process the data it has received and update all the internal states and features that have been turned on.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.10.2.10 int inv\_get\_accel\_accuracy (void)**

Returns accuracy of accel.

**Returns:**

Accuracy of accel with 0 being not accurate, and 3 being most accurate.

#### 4.10.2.11 void inv\_get\_accel\_bias (long \* *bias*, long \* *temp*)

Get Accel Bias.

**Parameters:**

*bias* Accel bias where

*temp* Temperature where  $1\text{ C} = 2^{16}$

#### 4.10.2.12 int inv\_get\_accel\_on ()

Helper function stating whether the accelerometer is on or off.

**Returns:**

TRUE if accel is on, 0 if accel is off

#### 4.10.2.13 long inv\_get\_accel\_sensitivity (void)

Accel sensitivity.

**Returns:**

A scale factor to convert device units to g's scaled by  $2^{16}$  such that  $g\_s = \text{device\_units} * \text{sensitivity} / 2^{30}$ . Typically it works out to be the maximum accel value in g's \*  $2^{15}$ .

#### 4.10.2.14 void inv\_get\_accel\_set (long \* *data*, int8\_t \* *accuracy*, inv\_time\_t \* *timestamp*)

Gets a whole set of accel data including data, accuracy and timestamp.

**Parameters:**

*data* Accel Data where  $1g = 2^{16}$

*accuracy* Accuracy 0 being not accurate, and 3 being most accurate.

*timestamp* The timestamp of the data sample.

**4.10 data\_builder****35****4.10.2.15 void inv\_get\_compass\_bias (long \* *bias*)**

Returns the current bias for the compass.

**Parameters:**

*bias* Compass bias in hardware units scaled by  $2^{16}$ . In mounting frame. Length 3.

**4.10.2.16 int inv\_get\_compass\_on ()**

Helper function stating whether the compass is on or off.

**Returns:**

TRUE if compass if on, 0 if compass if off

**4.10.2.17 long inv\_get\_compass\_sensitivity (void)**

Compass sensitivity.

**Returns:**

A scale factor to convert device units to micro Tesla scaled by  $2^{16}$  such that  $uT = device\_units * sensitivity / 2^{30}$ . Typically it works out to be the maximum  $uT * 2^{15}$ .

**4.10.2.18 void inv\_get\_compass\_set (long \* *data*, int8\_t \* *accuracy*, inv\_time\_t \* *timestamp*)**

Gets a whole set of compass data including data, accuracy and timestamp.

**Parameters:**

*data* Compass Data where  $1 uT = 2^{16}$

*accuracy* Accuracy 0 being not accurate, and 3 being most accurate.

*timestamp* The timestamp of the data sample.

#### 4.10.2.19 void inv\_get\_gyro (long \* gyro)

Get's latest gyro data.

**Parameters:**

*gyro* Gyro Data, Length 3. 1 dps =  $2^{16}$ .

#### 4.10.2.20 int inv\_get\_gyro\_accuracy (void)

Returns accuracy of gyro.

**Returns:**

Accuracy of gyro with 0 being not accurate, and 3 being most accurate.

#### 4.10.2.21 void inv\_get\_gyro\_bias (long \* bias, long \* temp)

Get the gyro biases and temperature record from MPL.

**Parameters:**

*bias* Gyro bias in hardware units scaled by  $2^{16}$ . In chip mounting frame. Length 3.

*temp* Temperature in degrees C.

#### 4.10.2.22 int inv\_get\_gyro\_on ()

Helper function stating whether the gyro is on or off.

**Returns:**

TRUE if gyro if on, 0 if gyro if off

#### 4.10.2.23 long inv\_get\_gyro\_sensitivity ()

Gyro sensitivity.

**Returns:**

A scale factor to convert device units to degrees per second scaled by  $2^{16}$  such that  $\text{degrees\_per\_second} = \text{device\_units} * \text{sensitivity} / 2^{30}$ . Typically it works out to be the maximum rate \*  $2^{15}$ .



**4.10 data\_builder****37****4.10.2.24 void inv\_get\_gyro\_set (long \* *data*, int8\_t \* *accuracy*, inv\_time\_t \* *timestamp*)**

Gets a whole set of gyro data including data, accuracy and timestamp.

**Parameters:**

*data* Gyro Data where 1 dps =  $2^{16}$

*accuracy* Accuracy 0 being not accurate, and 3 being most accurate.

*timestamp* The timestamp of the data sample.

**4.10.2.25 inv\_time\_t inv\_get\_last\_timestamp ()**

Get last timestamp across all 3 sensors that are on.

This find out which timestamp has the largest value for sensors that are on.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.10.2.26 int inv\_get\_mag\_accuracy (void)**

Returns accuracy of compass.

**Returns:**

Accuracy of compass with 0 being not accurate, and 3 being most accurate.

**4.10.2.27 void inv\_get\_temp\_set (long \* *data*, int \* *accuracy*, inv\_time\_t \* *timestamp*)**

Gets a whole set of temperature data including data, accuracy and timestamp.

**Parameters:**

*data* Temperature data where 1 degree C =  $2^{16}$

*accuracy* 0 to 3, where 3 is most accurate.

*timestamp* The timestamp of the data sample.

#### 4.10.2.28 void inv\_gyro\_was\_turned\_off ()

This should be called when the gyro has been turned off.

This is so that we will know if the data is contiguous.

#### 4.10.2.29 void inv\_quaternion\_sensor\_was\_turned\_off (void)

This should be called when the quaternion data from the DMP has been turned off.

This is so that we will know if the data is contiguous.

#### 4.10.2.30 void inv\_set\_accel\_bandwidth (int *bandwidth\_hz*)

Set Accel Bandwidth in Hz.

##### Parameters:

*bandwidth\_hz* Gyro bandwidth in Hz

#### 4.10.2.31 void inv\_set\_accel\_bias (const long \* *bias*, int *accuracy*)

Sets the accel bias.

##### Parameters:

*bias* Accel bias, length 3. In HW units scaled by  $2^{16}$  in body frame

*accuracy* Accuracy rating from 0 to 3, with 3 being most accurate.

#### 4.10.2.32 void inv\_set\_accel\_bias\_mask (const long \* *bias*, int *accuracy*, int *mask*)

Sets the accel bias with control over which axis.

##### Parameters:

*bias* Accel bias, length 3. In HW units scaled by  $2^{16}$  in body frame

*accuracy* Accuracy rating from 0 to 3, with 3 being most accurate.

*mask* Mask to select axis to apply bias set.

**4.10 data\_builder****39****4.10.2.33 void inv\_set\_accel\_orientation\_and\_scale (int *orientation*, long *sensitivity*)**

Sets the orientation and sensitivity of the gyro data.

**Parameters:**

***orientation*** A scalar defining the transformation from chip mounting to the body frame. The function [inv\\_orientation\\_matrix\\_to\\_scalar\(\)](#) can convert the transformation matrix to this scalar and describes the scalar in further detail.

***sensitivity*** A scale factor to convert device units to g's such that g's = device\_units \* sensitivity / 2<sup>30</sup>. Typically it works out to be the maximum g\_value \* 2<sup>15</sup>.

**4.10.2.34 void inv\_set\_accel\_sample\_rate (long *sample\_rate\_us*)**

Set Accel Sample rate in micro seconds.

**Parameters:**

***sample\_rate\_us*** Set Accel Sample rate in us

**4.10.2.35 void inv\_set\_compass\_bandwidth (int *bandwidth\_hz*)**

Set Compass Bandwidth in Hz.

**Parameters:**

***bandwidth\_hz*** Gyro bandwidth in Hz

**4.10.2.36 void inv\_set\_compass\_disturbance (int *dist*)**

Set the state of a compass disturbance.

**Parameters:**

***dist*** 1=disturbance, 0=no disturbance

**4.10.2.37 void inv\_set\_compass\_orientation\_and\_scale (int *orientation*, long *sensitivity*)**

Sets the Orientation and Sensitivity of the gyro data.

**Parameters:**

***orientation*** A scalar defining the transformation from chip mounting to the body frame. The function [inv\\_orientation\\_matrix\\_to\\_scalar\(\)](#) can convert the transformation matrix to this scalar and describes the scalar in further detail.

***sensitivity*** A scale factor to convert device units to uT such that  $uT = device\_units * sensitivity / 2^{30}$ . Typically it works out to be the maximum  $uT\_value * 2^{15}$ .

**4.10.2.38 void inv\_set\_compass\_sample\_rate (long *sample\_rate\_us*)**

Set Compass Sample rate in micro seconds.

**Parameters:**

***sample\_rate\_us*** Set Gyro Sample rate in micro seconds.

**4.10.2.39 void inv\_set\_gyro\_bandwidth (int *bandwidth\_hz*)**

Set Gyro Bandwidth in Hz.

**Parameters:**

***bandwidth\_hz*** Gyro bandwidth in Hz

**4.10.2.40 void inv\_set\_gyro\_bias (const long \* *bias*, int *accuracy*)**

Sets the gyro bias.

**Parameters:**

***bias*** Gyro bias in hardware units scaled by  $2^{16}$ . In chip mounting frame. Length 3.

***accuracy*** Accuracy of bias. 0 = least accurate, 3 = most accurate.

**4.10 data\_builder****41****4.10.2.41 void inv\_set\_gyro\_orientation\_and\_scale (int *orientation*, long *sensitivity*)**

Sets the Orientation and Sensitivity of the gyro data.

**Parameters:**

***orientation*** A scalar defining the transformation from chip mounting to the body frame. The function [inv\\_orientation\\_matrix\\_to\\_scalar\(\)](#) can convert the transformation matrix to this scalar and describes the scalar in further detail.

***sensitivity*** A scale factor to convert device units to degrees per second scaled by  $2^{16}$  such that  $\text{degrees\_per\_second} = \text{device\_units} * \text{sensitivity} / 2^{30}$ . Typically it works out to be the maximum rate \*  $2^{15}$ .

**4.10.2.42 void inv\_set\_gyro\_sample\_rate (long *sample\_rate\_us*)**

Set Gyro Sample rate in micro seconds.

**Parameters:**

***sample\_rate\_us*** Set Gyro Sample rate in us

**4.10.2.43 void inv\_set\_quat\_sample\_rate (long *sample\_rate\_us*)**

Set Quat Sample rate in micro seconds.

**Parameters:**

***sample\_rate\_us*** Set Quat Sample rate in us

**4.10.2.44 void inv\_temperature\_was\_turned\_off ()**

This should be called when the temperature sensor has been turned off.

This is so that we will know if the data is contiguous.

**4.10.2.45 void set\_sensor\_orientation\_and\_scale (struct inv\_single\_sensor\_t \* *sensor*, int *orientation*, long *sensitivity*)**

Sets orientation and sensitivity field for a sensor.



**Parameters:**

*sensor* Structure to apply settings to

*orientation* Orientation description of how part is mounted.

*sensitivity* A Scale factor to convert from hardware units to standard units (dps, uT, g).

## 4.11 ml\_math\_func

Motion Library - Math Functions Common math functions the Motion Library.

### Files

- file [ml\\_math\\_func.c](#)  
*Math Functions.*

### Functions

- float [inv\\_angle\\_diff](#) (float ang1, float ang2)  
*Finds the minimum angle difference ang1-ang2 such that difference is between [-M\_PI, M\_PI].*
- short [inv\\_big8\\_to\\_int16](#) (const unsigned char \*big8)  
*Converts a big endian byte stream into a 16-bit integer (short).*
- long [inv\\_big8\\_to\\_int32](#) (const unsigned char \*big8)  
*Converts a big endian byte stream into a 32-bit long.*
- uint32\_t [inv\\_checksum](#) (const unsigned char \*str, int len)  
*bernstein hash, derived from public domain source*
- void [inv\\_convert\\_to\\_body](#) (unsigned short orientation, const long \*input, long \*output)  
*Uses the scalar orientation value to convert from chip frame to body frame.*
- void [inv\\_convert\\_to\\_body\\_with\\_scale](#) (unsigned short orientation, long sensitivity, const long \*input, long \*output)  
*Uses the scalar orientation value to convert from chip frame to body frame and apply appropriate scaling.*
- void [inv\\_convert\\_to\\_chip](#) (unsigned short orientation, const long \*input, long \*output)  
*Uses the scalar orientation value to convert from body frame to chip frame.*
- unsigned long [inv\\_get\\_gyro\\_sum\\_of\\_sqr](#) (const long \*gyro)  
*The gyro data magnitude squared :  $(1 \text{ degree per second})^2 = 2^6 = 2^{\text{GYRO\_MAG\_SQR\_SHIFT}}$ .*

- unsigned char \* [inv\\_int16\\_to\\_big8](#) (short x, unsigned char \*big8)  
*Converts a 16-bit short to a big endian byte stream.*
- unsigned char \* [inv\\_int32\\_to\\_big8](#) (long x, unsigned char \*big8)  
*Converts a 32-bit long to a big endian byte stream.*
- short [inv\\_little8\\_to\\_int16](#) (const unsigned char \*little8)  
*Converts a little endian byte stream into a 16-bit integer (short).*
- unsigned short [inv\\_orientation\\_matrix\\_to\\_scalar](#) (const signed char \*mtx)  
*Converts an orientation matrix made up of 0,+1,and -1 to a scalar representation.*
- long [inv\\_q29\\_mult](#) (long a, long b)  
*Performs a multiply and shift by 29.*
- long [inv\\_q30\\_mult](#) (long a, long b)  
*Performs a multiply and shift by 30.*
- void [inv\\_q\\_add](#) (long \*q1, long \*q2, long \*qSum)  
*Performs a fixed point quaternion addition.*
- void [inv\\_q\\_mult](#) (const long \*q1, const long \*q2, long \*qProd)  
*Performs a fixed point quaternion multiply.*
- void [inv\\_q\\_norm4](#) (float \*q)  
*Performs a length 4 vector normalization with a square root.*
- void [inv\\_q\\_rotate](#) (const long \*q, const long \*in, long \*out)  
*Rotates a 3-element vector by Rotation defined by Q.*
- long [inv\\_q\\_shift\\_mult](#) (long a, long b, int shift)  
*Performs a multiply and shift by shift.*
- void [inv\\_quaternion\\_to\\_rotation](#) (const long \*quat, long \*rot)  
*Converts a quaternion to a rotation matrix.*
- void [inv\\_quaternion\\_to\\_rotation\\_vector](#) (const long \*quat, long \*rot)  
*Converts a quaternion to a rotation vector.*
- double [inv\\_vector\\_norm](#) (const float \*x)  
*find a norm for a vector*



**4.11 ml\_math\_func****45**

- float **inv\_wrap\_angle** (float ang)  
*Wraps angle from  $(-M\_PI, M\_PI]$ .*

**4.11.1 Detailed Description**

Motion Library - Math Functions Common math functions the Motion Library.

**4.11.2 Function Documentation****4.11.2.1 float inv\_angle\_diff (float *ang1*, float *ang2*)**

Finds the minimum angle difference  $ang1 - ang2$  such that difference is between  $[-M\_PI, M\_PI]$ .

**Parameters:**

*ang1*

*ang2*

**Returns:**

angle difference  $ang1 - ang2$

**4.11.2.2 void inv\_convert\_to\_body (unsigned short *orientation*, const long \* *input*, long \* *output*)**

Uses the scalar orientation value to convert from chip frame to body frame.

**Parameters:**

*orientation* A scalar that represent how to go from chip to body frame

*input* Input vector, length 3

*output* Output vector, length 3

**4.11.2.3 void inv\_convert\_to\_body\_with\_scale (unsigned short *orientation*, long *sensitivity*, const long \* *input*, long \* *output*)**

Uses the scalar orientation value to convert from chip frame to body frame and apply appropriate scaling.

**Parameters:**

*orientation* A scalar that represent how to go from chip to body frame

*sensitivity* Sensitivity scale

*input* Input vector, length 3

*output* Output vector, length 3

**4.11.2.4 void inv\_convert\_to\_chip (unsigned short *orientation*, const long \*  
*input*, long \* *output*)**

Uses the scalar orientation value to convert from body frame to chip frame.

**Parameters:**

*orientation* A scalar that represent how to go from chip to body frame

*input* Input vector, length 3

*output* Output vector, length 3

**4.11.2.5 unsigned long inv\_get\_gyro\_sum\_of\_sqr (const long \* *gyro*)**

The gyro data magnitude squared :  $(1 \text{ degree per second})^2 = 2^6 = 2^{\text{GYRO\_MAG\_SQR\_SHIFT}}$ .

**Parameters:**

*gyro* Gyro data scaled with  $1 \text{ dps} = 2^{16}$

**Returns:**

the computed magnitude squared output of the gyroscope.

**4.11.2.6 unsigned short inv\_orientation\_matrix\_to\_scalar (const signed char \*  
*mtx*)**

Converts an orientation matrix made up of 0,+1,and -1 to a scalar representation.

**Parameters:**

*mtx* Orientation matrix to convert to a scalar.

**4.11 ml\_math\_func****47****Returns:**

Description of orientation matrix. The lowest 2 bits (0 and 1) represent the column the one is on for the first row, with the bit number 2 being the sign. The next 2 bits (3 and 4) represent the column the one is on for the second row with bit number 5 being the sign. The next 2 bits (6 and 7) represent the column the one is on for the third row with bit number 8 being the sign. In binary the identity matrix would therefor be: 010\_001\_000 or 0x88 in hex.

**4.11.2.7 long inv\_q29\_mult (long *a*, long *b*)**

Performs a multiply and shift by 29.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

**Parameters:***a**b***Returns:**

$((\text{long long})a*b)>>29$

**4.11.2.8 long inv\_q30\_mult (long *a*, long *b*)**

Performs a multiply and shift by 30.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

**Parameters:***a**b***Returns:**

$((\text{long long})a*b)>>30$

**4.11.2.9 void inv\_q\_add (long \**q1*, long \**q2*, long \**qSum*)**

Performs a fixed point quaternion addition.

**Parameters:**

- q1* First Quaternion term, length 4. 1.0 scaled to  $2^{30}$
- q2* Second Quaternion term, length 4. 1.0 scaled to  $2^{30}$
- qSum* Sum after quaternion summation. Length 4. 1.0 scaled to  $2^{30}$ .

**4.11.2.10 void inv\_q\_mult (const long \* *q1*, const long \* *q2*, long \* *qProd*)**

Performs a fixed point quaternion multiply.

**Parameters:**

- q1* First Quaternion Multicand, length 4. 1.0 scaled to  $2^{30}$
- q2* Second Quaternion Multicand, length 4. 1.0 scaled to  $2^{30}$
- qProd* Product after quaternion multiply. Length 4. 1.0 scaled to  $2^{30}$ .

**4.11.2.11 void inv\_q\_norm4 (float \* *q*)**

Performs a length 4 vector normalization with a square root.

**Parameters:**

- q* vector to normalize. Returns [1,0,0,0] if magnitude is zero.

**4.11.2.12 long inv\_q\_shift\_mult (long *a*, long *b*, int *shift*)**

Performs a multiply and shift by shift.

These are good functions to write in assembly on with devices with small memory where you want to get rid of the long long which some assemblers don't handle well

**Parameters:**

- a* First multicand
- b* Second multicand
- shift* Shift amount after multiplying

**Returns:**

$((\text{long long})a*b) << \text{shift}$

**4.11 ml\_math\_func****49****4.11.2.13 void inv\_quaternion\_to\_rotation (const long \* *quat*, long \* *rot*)**

Converts a quaternion to a rotation matrix.

**Parameters:**

*quat* 4-element quaternion in fixed point. One is  $2^{30}$ .

*rot* Rotation matrix in fixed point. One is  $2^{30}$ . The First 3 elements of the rotation matrix, represent the first row of the matrix. Rotation matrix multiplied by a 3 element column vector transform a vector from Body to World.

**4.11.2.14 void inv\_quaternion\_to\_rotation\_vector (const long \* *quat*, long \* *rot*)**

Converts a quaternion to a rotation vector.

A rotation vector is a method to represent a 4-element quaternion vector in 3-elements. To get the quaternion from the 3-elements, The last 3-elements of the quaternion will be the given rotation vector. The first element of the quaternion will be the positive value that will be required to make the magnitude of the quaternion 1.0 or  $2^{30}$  in fixed point units.

**Parameters:**

*quat* 4-element quaternion in fixed point. One is  $2^{30}$ .

*rot* Rotation vector in fixed point. One is  $2^{30}$ .

**4.11.2.15 double inv\_vector\_norm (const float \* *x*)**

find a norm for a vector

**Parameters:**

*x* a vector [3x1]

**Returns:**

the norm of the input vector

**4.11.2.16 float inv\_wrap\_angle (float *ang*)**

Wraps angle from  $(-M\_PI, M\_PI]$ .

**Parameters:**

*ang* Angle in radians to wrap



**Returns:**

Wrapped angle from  $(-M\_PI, M\_PI]$

## 4.12 message\_layer

Motion Library - Message Layer Holds Low Occurance messages.

### Files

- file [message\\_layer.c](#)  
*Holds Low Occurance Messages.*

### Functions

- long [inv\\_get\\_message\\_level\\_0](#) (int clear)  
*Returns Message Flags for Level 0 Messages.*
- void [inv\\_set\\_message](#) (long set, long clear, int level)  
*Sets a message.*

#### 4.12.1 Detailed Description

Motion Library - Message Layer Holds Low Occurance messages.

#### 4.12.2 Function Documentation

##### 4.12.2.1 long [inv\\_get\\_message\\_level\\_0](#) (int *clear*)

Returns Message Flags for Level 0 Messages.

Levels are to allow expansion of more messages in the future.

#### Parameters:

*clear* If set, will clear the message. Typically this will be set for one reader, so that you don't get the same message over and over.

#### Returns:

bit field to corresponding message.

#### 4.12.2.2 void inv\_set\_message (long *set*, long *clear*, int *level*)

Sets a message.

**Parameters:**

*set* The flags to set.

*clear* Before setting anything this will clear these messages, which is useful for mutually exclusive messages such a motion or no motion message.

*level* Level of the messages. It starts at 0, and may increase in the future to allow more messages if the bit storage runs out.



## 4.13 mpl

Motion Library - Start Point Initializes MPL.

### Files

- file [mpl.c](#)  
*MPL start point.*

### Functions

- `inv_error_t inv_get_version` (char \*\*version)  
*used to get the MPL version.*
- `inv_error_t inv_init_mpl` (void)  
*Initializes the MPL.*
- `inv_error_t inv_start_mpl` (void)  
*Starts the MPL.*

#### 4.13.1 Detailed Description

Motion Library - Start Point Initializes MPL.

#### 4.13.2 Function Documentation

##### 4.13.2.1 `inv_error_t inv_get_version` (char \*\* version)

used to get the MPL version.

#### Parameters:

**version** a string where the MPL version gets stored.

#### Returns:

INV\_SUCCESS if successful or a non-zero error code otherwise.

#### **4.13.2.2 inv\_error\_t inv\_init\_mpl (void)**

Initializes the MPL.

Should be called first and once

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

#### **4.13.2.3 inv\_error\_t inv\_start\_mpl (void)**

Starts the MPL.

Typically called after [inv\\_init\\_mpl\(\)](#) or after a [inv\\_stop\\_mpl\(\)](#) to start the MPL back up an running.

**Returns:**

INV\_SUCCESS if successful or a non-zero error code otherwise.

## 4.14 results\_holder

Motion Library - Results Holder Holds the data for MPL.

### Files

- file [results\\_holder.c](#)  
*Results Holder for HAL.*

### Functions

- `inv_error_t inv_enable_results_holder ()`  
*Turns on storage of results.*
- `inv_error_t inv_generate_results (struct inv_sensor_cal_t *sensor_cal)`  
*Callback that gets called everytime there is new data.*
- `inv_error_t inv_get_6axis_quaternion (long *data)`  
*Returns a quaternion based only on gyro and accel.*
- `int inv_get_acc_state ()`  
*Gets the accel state set by [inv\\_set\\_acc\\_state\(\)](#).*
- `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 float data.*
- `void inv_get_compass_bias_error (long *bias_error)`  
*Get's compass bias error.*
- `int inv_get_compass_state ()`  
*Get's the compass state.*
- `inv_error_t inv_get_gravity (long *data)`  
*Gets gravity vector.*
- `inv_error_t inv_get_gyro_float (float *data)`  
*Returns 3-element vector of gyro float data.*

- float [inv\\_get\\_heading\\_confidence\\_interval](#) (void)  
*Get 9 axis 95% heading confidence interval for quaternion.*
- int [inv\\_get\\_large\\_mag\\_field](#) ()  
*Returns non-zero if there is a large magnetic field.*
- 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\\_float](#) (float \*data)  
*Returns 3-element vector of linear accel float data.*
- void [inv\\_get\\_local\\_field](#) (long \*data)  
*Gets the local earth's magnetic field.*
- void [inv\\_get\\_mag\\_scale](#) (long \*data)  
*Gets the compass sensitivity.*
- int [inv\\_get\\_motion\\_state](#) (unsigned int \*cntr)  
*Returns the motion state.*
- inv\_error\_t [inv\\_get\\_quaternion](#) (long \*data)  
*Returns a quaternion.*
- inv\_error\_t [inv\\_get\\_quaternion\\_float](#) (float \*data)  
*Returns a quaternion.*
- void [inv\\_get\\_quaternion\\_set](#) (long \*data, int \*accuracy, inv\_time\_t \*timestamp)  
*Returns a quaternion with accuracy and timestamp.*
- int [inv\\_got\\_compass\\_bias](#) ()  
*Sets state of if we know the compass bias.*
- inv\_error\_t [inv\\_init\\_results\\_holder](#) (void)  
*Initializes results holder.*
- void [inv\\_set\\_acc\\_state](#) (int state)  
*Sets the accel state.*
- void [inv\\_set\\_compass\\_bias\\_error](#) (const long \*bias\_error)

**4.14 results\_holder****57**

*Set compass bias error.*

- void [inv\\_set\\_compass\\_bias\\_found](#) (int state)  
*Sets whether we know the compass bias.*
- void [inv\\_set\\_compass\\_state](#) (int state)  
*Sets the compass state.*
- void [inv\\_set\\_heading\\_confidence\\_interval](#) (float ci)  
*Set 9 axis 95% heading confidence interval for quaternion.*
- void [inv\\_set\\_large\\_mag\\_field](#) (int state)  
*Set to non-zero if there as a large magnetic field.*
- void [inv\\_set\\_local\\_field](#) (const long \*data)  
*Sets the local earth's magnetic field.*
- void [inv\\_set\\_mag\\_scale](#) (const long \*data)  
*Sets the compass sensitivity.*
- void [inv\\_set\\_motion\\_state](#) (unsigned char state)  
*Sets the motion state.*
- [inv\\_error\\_t inv\\_start\\_results\\_holder](#) (void)  
*Function to turn on this module.*

**4.14.1 Detailed Description**

Motion Library - Results Holder Holds the data for MPL.

**4.14.2 Function Documentation****4.14.2.1 [inv\\_error\\_t inv\\_generate\\_results](#) (struct [inv\\_sensor\\_cal\\_t](#) \* *sensor\_cal*)**

Callback that gets called everytime there is new data.

It is registered by [inv\\_start\\_results\\_holder\(\)](#).

**Parameters:**

*sensor\_cal* New sensor data to process.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.14.2.2 `inv_error_t inv_get_6axis_quaternion (long * data)`**

Returns a quaternion based only on gyro and accel.

**Parameters:**

*data* 6-axis gyro and accel quaternion scaled such that  $1.0 = 2^{30}$ .

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.14.2.3 `int inv_get_acc_state ()`**

Gets the accel state set by [inv\\_set\\_acc\\_state\(\)](#).

**Returns:**

accel state.

**4.14.2.4 `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

**Returns:**

INV\_SUCCESS if successful INV\_ERROR\_INVALID\_PARAMETER if invalid  
input pointer

**4.14.2.5 `inv_error_t inv_get_accel_float (float * data)`**

Returns 3-element vector of accelerometer float data.

**Parameters:**

*data* 3-element vector of accelerometer float data

**4.14 results\_holder****59****Returns:**

INV\_SUCCESS if successful INV\_ERROR\_INVALID\_PARAMETER if invalid input pointer

**4.14.2.6 void inv\_get\_compass\_bias\_error (long \* *bias\_error*)**

Get's compass bias error.

See [inv\\_set\\_compass\\_bias\\_error\(\)](#) for setting.

**Parameters:**

*bias\_error* Accuracy as to how well the compass bias is known. It is the error squared.

**4.14.2.7 int inv\_get\_compass\_state ()**

Get's the compass state.

**Returns:**

the compass state that was set with [inv\\_set\\_compass\\_state\(\)](#)

**4.14.2.8 inv\_error\_t inv\_get\_gravity (long \* *data*)**

Gets gravity vector.

**Parameters:**

*data* gravity vector in body frame scaled such that  $1.0 = 2^{30}$ .

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.14.2.9 inv\_error\_t inv\_get\_gyro\_float (float \* *data*)**

Returns 3-element vector of gyro float data.

**Parameters:**

*data* 3-element vector of gyro float data

**Returns:**

INV\_SUCCESS if successful INV\_ERROR\_INVALID\_PARAMETER if invalid input pointer

**4.14.2.10 float inv\_get\_heading\_confidence\_interval (void)**

Get 9 axis 95% heading confidence interval for quaternion.

**Returns:**

Confidence interval in radians.

**4.14.2.11 int inv\_get\_large\_mag\_field ()**

Returns non-zero if there is a large magnetic field.

See [inv\\_set\\_large\\_mag\\_field\(\)](#) for setting this variable.

**Returns:**

Returns non-zero if there is a large magnetic field.

**4.14.2.12 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

**Returns:**

INV\_SUCCESS if successful INV\_ERROR\_INVALID\_PARAMETER if invalid input pointer

**4.14.2.13 inv\_error\_t inv\_get\_linear\_accel\_float (float \* data)**

Returns 3-element vector of linear accel float data.

**Parameters:**

*data* 3-element vector of linear accel float data



**4.14 results\_holder****61****Returns:**

INV\_SUCCESS if successful INV\_ERROR\_INVALID\_PARAMETER if invalid input pointer

**4.14.2.14 void inv\_get\_local\_field (long \* data)**

Gets the local earth's magnetic field.

**Parameters:**

*data* Local earth's magnetic field in uT scaled by  $2^{16}$ . Length = 3. Y typically points north, Z typically points down in northern hemisphere and up in southern hemisphere.

**4.14.2.15 void inv\_get\_mag\_scale (long \* data)**

Gets the compass sensitivity.

**Parameters:**

*data* Length 3, sensitivity for each compass axis scaled such that  $1.0 = 2^{30}$ .

**4.14.2.16 int inv\_get\_motion\_state (unsigned int \* cntr)**

Returns the motion state.

**Parameters:**

*cntr* Number of previous times a no motion event has occurred in a row.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.14.2.17 inv\_error\_t inv\_get\_quaternion (long \* data)**

Returns a quaternion.

**Parameters:**

*data* 9-axis quaternion scaled such that  $1.0 = 2^{30}$ .

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.14.2.18 inv\_error\_t inv\_get\_quaternion\_float (float \* *data*)**

Returns a quaternion.

**Parameters:**

*data* 9-axis quaternion.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.14.2.19 void inv\_get\_quaternion\_set (long \* *data*, int \* *accuracy*, inv\_time\_t \* *timestamp*)**

Returns a quaternion with accuracy and timestamp.

**Parameters:**

*data* 9-axis quaternion scaled such that  $1.0 = 2^{30}$ .

*accuracy* Accuracy of quaternion, 0-3, where 3 is most accurate.

*timestamp* Timestamp of this quaternion in nanoseconds

**4.14.2.20 int inv\_get\_compass\_bias ()**

Sets state of if we know the compass bias.

**Returns:**

return 1 if we know the compass bias, 0 if not. it is set with [inv\\_set\\_compass\\_bias\\_found\(\)](#)

**4.14.2.21 inv\_error\_t inv\_init\_results\_holder (void)**

Initializes results holder.

This is called automatically by the enable function [inv\\_enable\\_results\\_holder\(\)](#). It may be called any time the feature is enabled, but is typically not needed to be called by outside callers.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.14 results\_holder****63****4.14.2.22 void inv\_set\_acc\_state (int state)**

Sets the accel state.

See [inv\\_get\\_acc\\_state\(\)](#) to get the value.

**Parameters:**

*state* value to set accel state to.

**4.14.2.23 void inv\_set\_compass\_bias\_error (const long \* bias\_error)**

Set compass bias error.

See [inv\\_get\\_compass\\_bias\\_error\(\)](#)

**Parameters:**

*bias\_error* Set's how accurate we know the compass bias. It is the error squared.

**4.14.2.24 void inv\_set\_compass\_bias\_found (int state)**

Sets whether we know the compass bias.

**Parameters:**

*state* Set to 1 if we know the compass bias. Can be retrieved with [inv\\_get\\_compass\\_bias\(\)](#)

**4.14.2.25 void inv\_set\_compass\_state (int state)**

Sets the compass state.

**Parameters:**

*state* Compass state. It can be retrieved with [inv\\_get\\_compass\\_state\(\)](#).

**4.14.2.26 void inv\_set\_heading\_confidence\_interval (float ci)**

Set 9 axis 95% heading confidence interval for quaternion.

**Parameters:**

*ci* Confidence interval in radians.

**4.14.2.27 void inv\_set\_large\_mag\_field (int *state*)**

Set to non-zero if there as a large magnetic field.

See [inv\\_get\\_large\\_mag\\_field\(\)](#) for getting this variable.

**Parameters:**

*state* value to set for magnetic field strength. Should be non-zero if it is large.

**4.14.2.28 void inv\_set\_local\_field (const long \* *data*)**

Sets the local earth's magnetic field.

**Parameters:**

*data* Local earth's magnetic field in uT scaled by  $2^{16}$ . Length = 3. Y typically points north, Z typically points down in northern hemisphere and up in southern hemisphere.

**4.14.2.29 void inv\_set\_mag\_scale (const long \* *data*)**

Sets the compass sensitivity.

**Parameters:**

*data* Length 3, sensitivity for each compass axis scaled such that  $1.0 = 2^{30}$ .

**4.14.2.30 void inv\_set\_motion\_state (unsigned char *state*)**

Sets the motion state.

**Parameters:**

*state* motion state where INV\_NO\_MOTION is not moving and INV\_MOTION is moving.

**4.14.2.31 inv\_error\_t inv\_start\_results\_holder (void)**

Function to turn on this module.

This is automatically called by [inv\\_enable\\_results\\_holder\(\)](#). Typically not called by users.



#### **4.14 results\_holder**

**65**

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

## 4.15 start\_manager

Motion Library - Start Manager Start Manager.

### Files

- file [start\\_manager.c](#)

*This handles all the callbacks when [inv\\_start\\_mpl\(\)](#) is called.*

### Functions

- `inv_error_t inv\_execute\_mpl\_start\_notification (void)`  
*Callback all the functions that want to be notified when [inv\\_start\\_mpl\(\)](#) was called.*
- `inv_error_t inv\_init\_start\_manager (void)`  
*Initilize the start manager.*
- `inv_error_t inv\_register\_mpl\_start\_notification (inv_error_t(*start_cb)(void))`  
*Register a callback to receive when [inv\\_start\\_mpl\(\)](#) is called.*
- `inv_error_t inv\_unregister\_mpl\_start\_notification (inv_error_t(*start_cb)(void))`  
*Removes a callback from start notification.*

### 4.15.1 Detailed Description

Motion Library - Start Manager Start Manager.

### 4.15.2 Function Documentation

#### 4.15.2.1 `inv_error_t inv_execute_mpl_start_notification (void)`

Callback all the functions that want to be notified when [inv\\_start\\_mpl\(\)](#) was called.

#### Returns:

Returns INV\_SUCCESS if successful or an error code if not.

**4.15 start\_manager****67****4.15.2.2 inv\_error\_t inv\_init\_start\_manager (void)**

Initialize the start manager.

Typically called by [inv\\_start\\_mpl\(\)](#);

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.15.2.3 inv\_error\_t inv\_register\_mpl\_start\_notification (inv\_error\_t(\*) (void) start\_cb)**

Register a callback to receive when [inv\\_start\\_mpl\(\)](#) is called.

**Parameters:**

*start\_cb* Function callback that will be called when [inv\\_start\\_mpl\(\)](#) is called.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.15.2.4 inv\_error\_t inv\_unregister\_mpl\_start\_notification (inv\_error\_t(\*) (void) start\_cb)**

Removes a callback from start notification.

**Parameters:**

*start\_cb* function to remove from start notification

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

## 4.16 storage\_manager

Motion Library - Stores Data for functions.

### Files

- file [storage\\_manager.c](#)  
*Load and Store Manager.*

### Defines

- #define [NUM\\_STORAGE\\_BOXES](#) 20  
*Max number of entites that can be stored.*

### Functions

- `inv_error_t inv\_get\_mpl\_state\_size (size_t *size)`  
*Returns the memory size needed to perform a store.*
- `void inv\_init\_storage\_manager ()`  
*Should be called once before using any of the storage methods.*
- `inv_error_t inv\_load\_mpl\_states (const unsigned char *data, size_t length)`  
*This function takes a block of data that has been saved in non-volatile memory and pushes to the proper locations.*
- `inv_error_t inv\_register\_load\_store (inv_error_t(*load_func)(const unsigned char *data), inv_error_t(*save_func)(unsigned char *data), size_t size, unsigned int key)`  
*Used to register your mechanism to load and store non-volatile data.*
- `inv_error_t inv\_save\_mpl\_states (unsigned char *data, size_t sz)`  
*This function fills up a block of memory to be stored in non-volatile memory.*

#### 4.16.1 Detailed Description

Motion Library - Stores Data for functions.



**4.16 storage\_manager****69****4.16.2 Function Documentation****4.16.2.1 `inv_error_t inv_get_mpl_state_size (size_t * size)`**

Returns the memory size needed to perform a store.

**Parameters:**

*size* Size in bytes of memory needed to store.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.16.2.2 `void inv_init_storage_manager ()`**

Should be called once before using any of the storage methods.

Typically called first by [inv\\_init\\_mpl\(\)](#).

**4.16.2.3 `inv_error_t inv_load_mpl_states (const unsigned char * data, size_t length)`**

This function takes a block of data that has been saved in non-volatile memory and pushes to the proper locations.

Multiple error checks are performed on the data.

**Parameters:**

*data* Data that was saved to be loaded up by MPL

*length* Length of data vector in bytes

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.16.2.4 `inv_error_t inv_register_load_store (inv_error_t(*) (const unsigned char *data) load_func, inv_error_t(*) (unsigned char *data) save_func, size_t size, unsigned int key)`**

Used to register your mechanism to load and store non-volatile data.

This should typically be called during the enable function for your feature.

**Parameters:**

- load\_func* function pointer you will use to receive data that was stored for you.
- save\_func* function pointer you will use to save any data you want saved to non-volatile memory between runs.
- size* The size in bytes of the amount of data you want loaded and saved.
- key* The key associated with your data type should be unique across MPL. The key should change when your type of data for storage changes.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

**4.16.2.5 `inv_error_t inv_save_mpl_states (unsigned char * data, size_t sz)`**

This function fills up a block of memory to be stored in non-volatile memory.

**Parameters:**

- data* Place to store data, size of sz, must be at least size returned by [inv\\_get\\_mpl\\_state\\_size\(\)](#)
- sz* Size of data.

**Returns:**

Returns INV\_SUCCESS if successful or an error code if not.

## 4.17 hal\_outputs

Motion Library - HAL Outputs Sets up common outputs for HAL.

### Files

- file [eMPL\\_outputs.c](#)  
*Embedded MPL outputs.*

### Functions

- `inv_error_t inv_disable_eMPL_outputs` (void)  
*Turns off creation and storage of HAL type results.*
- `inv_error_t inv_enable_eMPL_outputs` (void)  
*Turns on creation and storage of HAL type results.*
- `int inv_get_sensor_type_accel` (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Acceleration (g's) in body frame.*
- `int inv_get_sensor_type_compass` (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Magnetic field strength in body frame.*
- `int inv_get_sensor_type_euler` (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Body-to-world frame euler angles.*
- `int inv_get_sensor_type_gyro` (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Angular velocity (degrees per second) in body frame.*
- `int inv_get_sensor_type_heading` (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Quaternion-derived heading.*
- `int inv_get_sensor_type_quat` (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)  
*Body-to-world frame quaternion.*

- int `inv_get_sensor_type_rot_mat` (long \*data, int8\_t \*accuracy, inv\_time\_t \*timestamp)

*Body-to-world frame rotation matrix.*

#### 4.17.1 Detailed Description

Motion Library - HAL Outputs Sets up common outputs for HAL.

#### 4.17.2 Function Documentation

##### 4.17.2.1 int `inv_get_sensor_type_accel` (long \* data, int8\_t \* accuracy, inv\_time\_t \* timestamp)

Acceleration (g's) in body frame.

Embedded MPL defines gravity as positive acceleration pointing away from the Earth.

**Parameters:**

*data* Acceleration in g's, q16 fixed point.

*accuracy* Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

*timestamp* The time in milliseconds when this sensor was read.

**Returns:**

1 if data was updated.

##### 4.17.2.2 int `inv_get_sensor_type_compass` (long \* data, int8\_t \* accuracy, inv\_time\_t \* timestamp)

Magnetic field strength in body frame.

**Parameters:**

*data* Field strength in microteslas, q16 fixed point.

*accuracy* Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

*timestamp* The time in milliseconds when this sensor was read.

**Returns:**

1 if data was updated.

**4.17 hal\_outputs****73****4.17.2.3 int inv\_get\_sensor\_type\_euler (long \* *data*, int8\_t \* *accuracy*,  
inv\_time\_t \* *timestamp*)**

Body-to-world frame euler angles.

The euler angles are output with the following convention: Pitch: -180 to 180 Roll: -90 to 90 Yaw: -180 to 180

**Parameters:**

*data* Euler angles in degrees, q16 fixed point.

*accuracy* Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

*timestamp* The time in milliseconds when this sensor was read.

**Returns:**

1 if data was updated.

**4.17.2.4 int inv\_get\_sensor\_type\_gyro (long \* *data*, int8\_t \* *accuracy*,  
inv\_time\_t \* *timestamp*)**

Angular velocity (degrees per second) in body frame.

**Parameters:**

*data* Angular velocity in dps, q16 fixed point.

*accuracy* Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

*timestamp* The time in milliseconds when this sensor was read.

**Returns:**

1 if data was updated.

**4.17.2.5 int inv\_get\_sensor\_type\_heading (long \* *data*, int8\_t \* *accuracy*,  
inv\_time\_t \* *timestamp*)**

Quaternion-derived heading.

**Parameters:**

*data* Heading in degrees, q16 fixed point.

*accuracy* Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

*timestamp* The time in milliseconds when this sensor was read.

**Returns:**

1 if data was updated.

**4.17.2.6 int inv\_get\_sensor\_type\_quat (long \* *data*, int8\_t \* *accuracy*,  
inv\_time\_t \* *timestamp*)**

Body-to-world frame quaternion.

The elements are output in the following order: W, X, Y, Z.

**Parameters:**

*data* Quaternion, q30 fixed point.

*accuracy* Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

*timestamp* The time in milliseconds when this sensor was read.

**Returns:**

1 if data was updated.

**4.17.2.7 int inv\_get\_sensor\_type\_rot\_mat (long \* *data*, int8\_t \* *accuracy*,  
inv\_time\_t \* *timestamp*)**

Body-to-world frame rotation matrix.

**Parameters:**

*data* Rotation matrix, q30 fixed point.

*accuracy* Accuracy of the measurement from 0 (least accurate) to 3 (most accurate).

*timestamp* The time in milliseconds when this sensor was read.

**Returns:**

1 if data was updated.

## 4.18 MSP430 System Layer

MSP430 System Layer APIs.

### Files

- file [msp430\\_clock.h](#)  
*Functions to configure the MSP430 system clock to settings required for eMPL.*
- file [msp430\\_i2c.h](#)  
*Serial communication functions needed by eMPL to communicate to the MPU devices.*
- file [msp430\\_interrupt.h](#)  
*Supports common interrupt vectors using callbacks.*
- file [packet.h](#)  
*Defines needed for sending data/debug packets via USB.*

### Functions

- void [eMPL\\_send\\_data](#) (unsigned char type, long \*data)  
*Send a data packet via USB.*
- void [eMPL\\_send\\_quat](#) (long \*quat)  
*Send a quaternion packet via USB.*
- int [msp430\\_clock\\_disable](#) (void)  
*Disable the millisecond timer.*
- int [msp430\\_clock\\_enable](#) (void)  
*Enable the millisecond timer.*
- int [msp430\\_clock\\_init](#) (unsigned long mclk, unsigned char xt)  
*Set the frequency of MCLK, SMCLK, and ACLK.*
- int [msp430\\_delay\\_ms](#) (unsigned long num\_ms)  
*Perform a blocking delay.*
- int [msp430\\_get\\_clock\\_ms](#) (unsigned long \*count)  
*Get current clock count.*

- int [msp430\\_get\\_smclk\\_freq](#) (unsigned long \*smclk)  
*Get frequency of SMCLK.*
- int [msp430\\_i2c\\_disable](#) (void)  
*Disable I2C communication.*
- int [msp430\\_i2c\\_enable](#) (void)  
*Set up the I2C port and configure the MSP430 as the master.*
- int [msp430\\_i2c\\_read](#) (unsigned char slave\_addr, unsigned char reg\_addr, unsigned char length, unsigned char \*data)  
*Read from a device.*
- int [msp430\\_i2c\\_write](#) (unsigned char slave\_addr, unsigned char reg\_addr, unsigned char length, unsigned char const \*data)  
*Write to a device register.*
- int [msp430\\_int\\_disable](#) (void)  
*Disable interrupts.*
- int [msp430\\_int\\_enable](#) (void)  
*Enable interrupts.*
- int [msp430\\_int\\_init](#) (unsigned char active\_low)  
*Set up shared interrupt vectors.*
- int [msp430\\_reg\\_int\\_cb](#) (void(\*cb)(void), unsigned char port, unsigned char pin, unsigned char lp\_exit)  
*Register callback for a particular interrupt pin.*
- int [msp430\\_slow\\_timer](#) (unsigned char slow)  
*Slow down the timer.*

#### **4.18.1 Detailed Description**

MSP430 System Layer APIs.

To interface with any platform, eMPL needs access to various system layer functions.



**4.18 MSP430 System Layer****77****4.18.2 Function Documentation****4.18.2.1 void eMPL\_send\_data (unsigned char *type*, long \* *data*)**

Send a data packet via USB.

**Parameters:**

*type* Contents of packet (PACKET\_DATA\_ACCEL, etc).

*data* Data (length dependent on contents).

**4.18.2.2 void eMPL\_send\_quat (long \* *quat*)**

Send a quaternion packet via USB.

The host is expected to use the data in this packet to graphically represent the device orientation. To send quaternion in the same manner as any other data packet, use eMPL\_send\_data.

**Parameters:**

*quat* Quaternion data.

**4.18.2.3 int msp430\_clock\_disable (void)**

Disable the millisecond timer.

This function should be used prior to entering a low-power mode.

**Returns:**

0 if successful.

**4.18.2.4 int msp430\_clock\_enable (void)**

Enable the millisecond timer.

This function is automatically called by *msp430\_clock\_init*. It should be used to re-enable the timer after *msp430\_clock\_disable* is called.

**Returns:**

0 if successful.

**4.18.2.5 int msp430\_clock\_init (unsigned long *mclk*, unsigned char *xt*)**

Set the frequency of MCLK, SMCLK, and ACLK.

**Parameters:**

*mclk* Frequency of master clock.

*xt* 1 if XT1 is present, 2 if XT2 is present, 0 otherwise.

**Returns:**

0 if successful.

**4.18.2.6 int msp430\_delay\_ms (unsigned long *num\_ms*)**

Perform a blocking delay.

**Parameters:**

*num\_ms* Number of milliseconds to delay.

**Returns:**

0 if successful.

**4.18.2.7 int msp430\_get\_clock\_ms (unsigned long \* *count*)**

Get current clock count.

Timer overflow will occur after  $2^{32}$  milliseconds.

**Parameters:**

*count* Timer count in milliseconds.

**Returns:**

0 if successful.

**4.18.2.8 int msp430\_get\_smclk\_freq (unsigned long \* *smclk*)**

Get frequency of SMCLK.

Currently, the sub-master clock and the master clock are the same frequency.

**4.18 MSP430 System Layer****79****Parameters:**

*smclk* SMCLK frequency.

**Returns:**

0 if successful.

**4.18.2.9 int msp430\_i2c\_disable (void)**

Disable I2C communication.

This function will disable the I2C hardware and should be called prior to entering low-power mode.

**Returns:**

0 if successful.

**4.18.2.10 int msp430\_i2c\_enable (void)**

Set up the I2C port and configure the MSP430 as the master.

**Returns:**

0 if successful.

**4.18.2.11 int msp430\_i2c\_read (unsigned char *slave\_addr*, unsigned char *reg\_addr*, unsigned char *length*, unsigned char \* *data*)**

Read from a device.

**Parameters:**

*slave\_addr* Slave address of device.

*reg\_addr* Slave register to be read from.

*length* Number of bytes to read.

*data* Data from register.

**Returns:**

0 if successful.

**4.18.2.12 int msp430\_i2c\_write (unsigned char *slave\_addr*, unsigned char *reg\_addr*, unsigned char *length*, unsigned char const \* *data*)**

Write to a device register.

**Parameters:**

*slave\_addr* Slave address of device.  
*reg\_addr* Slave register to be written to.  
*length* Number of bytes to write.  
*data* Data to be written to register.

**Returns:**

0 if successful.

**4.18.2.13 int msp430\_int\_disable (void)**

Disable interrupts.

**Returns:**

0 if successful.

**4.18.2.14 int msp430\_int\_enable (void)**

Enable interrupts.

**Returns:**

0 if successful.

**4.18.2.15 int msp430\_int\_init (unsigned char *active\_low*)**

Set up shared interrupt vectors.

This function will automatically call *msp430\_int\_enable* before returning.

**Parameters:**

*active\_low* 1 if interrupts are active low.

**Returns:**

0 if successful.

**4.18 MSP430 System Layer****81****4.18.2.16 int msp430\_reg\_int\_cb (void(\*) (void) *cb*, unsigned char *port*, unsigned char *pin*, unsigned char *lp\_exit*)**

Register callback for a particular interrupt pin.

This function will override any function already registered.

**Parameters:**

*cb* Function executed for this interrupt.

*port* Port number.

*pin* Pin number.

*lp\_exit* Low-power mode exited after this interrupt.

**Returns:**

0 if successful.

**4.18.2.17 int msp430\_slow\_timer (unsigned char *slow*)**

Slow down the timer.

By default, a millisecond timer is used for timing/scheduling purposes. This API can be used to slow down the interval at which this clock is updated, saving power by reducing interrupts.

**Parameters:**

*slow* 1 to slow down timer.

**Returns:**

0 if successful.

## 4.19 Sensor Driver Layer

Hardware drivers to communicate with sensors via I2C.

### Files

- file [inv\\_gyro.c](#)  
*An I2C-based driver for Invensense gyroscopes.*
- file [inv\\_gyro\\_dmp\\_android.c](#)  
*DMP image and interface functions.*

### Functions

- int [dmp\\_enable\\_6x\\_lp\\_quat](#) (unsigned char enable)  
*Generate 6-axis quaternions from the DMP.*
- int [dmp\\_enable\\_feature](#) (unsigned char mask)  
*Enable DMP features.*
- int [dmp\\_enable\\_lp\\_quat](#) (unsigned char enable)  
*Generate 3-axis quaternions from the DMP.*
- int [dmp\\_get\\_fifo\\_rate](#) (unsigned short \*rate)  
*Get DMP output rate.*
- int [dmp\\_get\\_pedometer\\_step\\_count](#) (unsigned long \*count)  
*Get current step count.*
- int [dmp\\_get\\_pedometer\\_walk\\_time](#) (unsigned long \*time)  
*Get duration of walking time.*
- int [dmp\\_load\\_android\\_firmware](#) (void)  
*Load the DMP with this image.*
- int [dmp\\_read\\_fifo](#) (short \*gyro, short \*accel, long \*quat, unsigned long \*timestamp, short \*sensors, unsigned char \*more)  
*Get one packet from the FIFO.*
- int [dmp\\_register\\_display\\_orient\\_cb](#) (void(\*func)(unsigned char))

**4.19 Sensor Driver Layer****83**

*Register a function to be executed on a display orientation event.*

- int [dmp\\_register\\_orient\\_cb](#) (void(\*func)(unsigned char))  
*Register a function to be executed on an orientation event.*
- int [dmp\\_register\\_tap\\_cb](#) (void(\*func)(unsigned char, unsigned char))  
*Register a function to be executed on a tap event.*
- int [dmp\\_set\\_fifo\\_rate](#) (unsigned short rate)  
*Set DMP output rate.*
- int [dmp\\_set\\_gyro\\_bias](#) (long \*bias)  
*Push gyro biases to the DMP.*
- int [dmp\\_set\\_interrupt\\_mode](#) (unsigned char mode)  
*Specify when a DMP interrupt should occur.*
- int [dmp\\_set\\_orient\\_axes](#) (unsigned char axis)  
*Set which orientations will trigger an event.*
- int [dmp\\_set\\_orient\\_thresh](#) (float angle)  
*Set orientation angle threshold.*
- int [dmp\\_set\\_orient\\_time](#) (unsigned short time)  
*Set orientation time.*
- int [dmp\\_set\\_orientation](#) (unsigned short orient)  
*Push gyro and accel orientation to the DMP.*
- int [dmp\\_set\\_pedometer\\_step\\_count](#) (unsigned long count)  
*Overwrite current step count.*
- int [dmp\\_set\\_pedometer\\_walk\\_time](#) (unsigned long time)  
*Overwrite current walk time.*
- int [dmp\\_set\\_shake\\_reject\\_thresh](#) (long sf, unsigned short thresh)  
*Set shake rejection threshold.*
- int [dmp\\_set\\_shake\\_reject\\_time](#) (unsigned short time)  
*Set shake rejection time.*
- int [dmp\\_set\\_shake\\_reject\\_timeout](#) (unsigned short time)

*Set shake rejection timeout.*

- int [dmp\\_set\\_tap\\_axes](#) (unsigned char axis)  
*Set which axes will register a tap.*
- int [dmp\\_set\\_tap\\_count](#) (unsigned char min\_taps)  
*Set minimum number of taps needed for an interrupt.*
- int [dmp\\_set\\_tap\\_thresh](#) (unsigned char axis, unsigned short thresh)  
*Set tap threshold for a specific axis.*
- int [dmp\\_set\\_tap\\_time](#) (unsigned short time)  
*Set length between valid taps.*
- int [dmp\\_set\\_tap\\_time\\_multi](#) (unsigned short time)  
*Set max time between taps to register as a multi-tap.*
- int [gyro\\_configure\\_fifo](#) (unsigned char sensors)  
*Select which sensors are pushed to FIFO.*
- int [gyro\\_get\\_accel\\_fsr](#) (unsigned char \*fsr)  
*Get the accel full-scale range.*
- int [gyro\\_get\\_accel\\_reg](#) (short \*data, unsigned long \*timestamp)  
*Read raw accel data directly from the registers.*
- int [gyro\\_get\\_accel\\_sens](#) (unsigned short \*sens)  
*Get accel sensitivity scale factor.*
- int [gyro\\_get\\_compass\\_sample\\_rate](#) (unsigned short \*rate)  
*Get compass sampling rate.*
- int [gyro\\_get\\_fifo\\_config](#) (unsigned char \*sensors)  
*Get current FIFO configuration.*
- int [gyro\\_get\\_gyro\\_fsr](#) (unsigned short \*fsr)  
*Get the gyro full-scale range.*
- int [gyro\\_get\\_gyro\\_reg](#) (short \*data, unsigned long \*timestamp)  
*Read raw gyro data directly from the registers.*
- int [gyro\\_get\\_gyro\\_sens](#) (float \*sens)



**4.19 Sensor Driver Layer****85**

*Get gyro sensitivity scale factor.*

- int [gyro\\_get\\_lpf](#) (unsigned short \*lpf)

*Get the current DLPF setting.*

- int [gyro\\_get\\_power\\_state](#) (unsigned char \*power\_on)

*Get current power state.*

- int [gyro\\_get\\_sample\\_rate](#) (unsigned short \*rate)

*Get sampling rate.*

- int [gyro\\_get\\_temperature](#) (long \*data, unsigned long \*timestamp)

*Read temperature data directly from the registers.*

- int [gyro\\_init](#) (struct int\_param\_s \*int\_param)

*Initialize hardware.*

- int [gyro\\_lp\\_accel\\_mode](#) (unsigned char rate)

*Enter low-power accel-only mode.*

- int [gyro\\_read\\_fifo](#) (short \*gyro, short \*accel, unsigned long \*timestamp, unsigned char \*sensors, unsigned char \*more)

*Get one packet from the FIFO.*

- int [gyro\\_read\\_fifo\\_stream](#) (unsigned short length, unsigned char \*data, unsigned char \*more)

*Get one unparsed packet from the FIFO.*

- int [gyro\\_read\\_reg](#) (unsigned char reg, unsigned char \*data)

*Read from a single register.*

- int [gyro\\_reg\\_dump](#) (void)

*Register dump for testing.*

- int [gyro\\_reset\\_fifo](#) (void)

*Reset FIFO read/write pointers.*

- int [gyro\\_run\\_self\\_test](#) (long \*gyro, long \*accel)

*Trigger gyro/accel/compass self-test.*

- int [gyro\\_set\\_accel\\_bias](#) (const long \*accel\_bias)

*Push biases to the accel bias registers.*

- int [gyro\\_set\\_accel\\_fsr](#) (unsigned char fsr)  
*Set the accel full-scale range.*
- int [gyro\\_set\\_bypass](#) (unsigned char bypass\_on)  
*Set device to bypass mode.*
- int [gyro\\_set\\_compass\\_sample\\_rate](#) (unsigned short rate)  
*Set compass sampling rate.*
- int [gyro\\_set\\_gyro\\_fsr](#) (unsigned short fsr)  
*Set the gyro full-scale range.*
- int [gyro\\_set\\_int\\_latched](#) (unsigned char enable)  
*Enable latched interrupts.*
- int [gyro\\_set\\_int\\_level](#) (unsigned char active\_low)  
*Set interrupt level.*
- int [gyro\\_set\\_lpf](#) (unsigned short lpf)  
*Set digital low pass filter.*
- int [gyro\\_set\\_sample\\_rate](#) (unsigned short rate)  
*Set sampling rate.*
- int [gyro\\_set\\_sensors](#) (unsigned char sensors)  
*Turn specific sensors on/off.*

#### **4.19.1 Detailed Description**

Hardware drivers to communicate with sensors via I2C.

#### **4.19.2 Function Documentation**

##### **4.19.2.1 int [dmp\\_enable\\_6x\\_lp\\_quat](#) (unsigned char *enable*)**

Generate 6-axis quaternions from the DMP.

In this driver, the 3-axis and 6-axis DMP quaternion features are mutually exclusive.

##### **Parameters:**

*enable* 1 to enable 6-axis quaternion.

**4.19 Sensor Driver Layer****87****Returns:**

0 if successful.

**4.19.2.2 int dmp\_enable\_feature (unsigned char *mask*)**

Enable DMP features.

The following #define's are used in the input mask:

DMP\_FEATURE\_TAP

DMP\_FEATURE\_ORIENTATION

DMP\_FEATURE\_LP\_QUAT

DMP\_FEATURE\_6X\_LP\_QUAT

NOTE: Gyro and accel data are always put into the FIFO.

NOTE: DMP\_FEATURE\_LP\_QUAT and DMP\_FEATURE\_6X\_LP\_QUAT are mutually exclusive.

**Parameters:**

*mask* Mask of features to enable.

**Returns:**

0 if successful.

**4.19.2.3 int dmp\_enable\_lp\_quat (unsigned char *enable*)**

Generate 3-axis quaternions from the DMP.

In this driver, the 3-axis and 6-axis DMP quaternion features are mutually exclusive.

**Parameters:**

*enable* 1 to enable 3-axis quaternion.

**Returns:**

0 if successful.

**4.19.2.4 int dmp\_get\_fifo\_rate (unsigned short \* *rate*)**

Get DMP output rate.

**Parameters:**

*rate* Current fifo rate (Hz).

**Returns:**

0 if successful.

**4.19.2.5 int dmp\_get\_pedometer\_step\_count (unsigned long \* *count*)**

Get current step count.

**Parameters:**

*count* Number of steps detected.

**Returns:**

0 if successful.

**4.19.2.6 int dmp\_get\_pedometer\_walk\_time (unsigned long \* *time*)**

Get duration of walking time.

**Parameters:**

*time* Walk time in milliseconds.

**Returns:**

0 if successful.

**4.19.2.7 int dmp\_load\_android\_firmware (void)**

Load the DMP with this image.

**Returns:**

0 if successful.

**4.19 Sensor Driver Layer****89****4.19.2.8 int dmp\_read\_fifo (short \* *gyro*, short \* *accel*, long \* *quat*, unsigned long \* *timestamp*, short \* *sensors*, unsigned char \* *more*)**

Get one packet from the FIFO.

If *sensors* does not contain a particular sensor, disregard the data returned to that pointer.

*sensors* can contain a combination of the following flags:

INV\_X\_GYRO, INV\_Y\_GYRO, INV\_Z\_GYRO

INV\_XYZ\_GYRO

INV\_XYZ\_ACCEL

INV\_WXYZ\_QUAT

If the FIFO has no new data, *sensors* will be zero.

If the FIFO is disabled, *sensors* will be zero and this function will return a non-zero error code.

**Parameters:**

*gyro* Gyro data in hardware units.

*accel* Accel data in hardware units.

*quat* 3-axis quaternion data in hardware units.

*timestamp* Timestamp in milliseconds.

*sensors* Mask of sensors read from FIFO.

*more* Number of remaining packets.

**Returns:**

0 if successful.

**4.19.2.9 int dmp\_register\_display\_orient\_cb (void(\*) (unsigned char) *func*)**

Register a function to be executed on a display orientation event.

**Parameters:**

*func* Callback function.

**Returns:**

0 if successful.

**4.19.2.10 int dmp\_register\_orient\_cb (void(\*)(unsigned char) *func*)**

Register a function to be executed on an orientation event.

**Parameters:**

*func* Callback function.

**Returns:**

0 if successful.

**4.19.2.11 int dmp\_register\_tap\_cb (void(\*)(unsigned char, unsigned char) *func*)**

Register a function to be executed on a tap event.

The tap direction is represented by one of the following:

TAP\_X\_UP

TAP\_X\_DOWN

TAP\_Y\_UP

TAP\_Y\_DOWN

TAP\_Z\_UP

TAP\_Z\_DOWN

**Parameters:**

*func* Callback function.

**Returns:**

0 if successful.

**4.19.2.12 int dmp\_set\_fifo\_rate (unsigned short *rate*)**

Set DMP output rate.

Only used when DMP is on.

**Parameters:**

*rate* Desired fifo rate (Hz).

**Returns:**

0 if successful.

**4.19 Sensor Driver Layer****91****4.19.2.13 int dmp\_set\_gyro\_bias (long \* *bias*)**

Push gyro biases to the DMP.

Because the gyro integration is handled in the DMP, any gyro biases calculated by the MPL should be pushed down to DMP memory to remove 3-axis quaternion drift.

**Parameters:**

*bias* Gyro biases in q16.

**Returns:**

0 if successful.

**4.19.2.14 int dmp\_set\_interrupt\_mode (unsigned char *mode*)**

Specify when a DMP interrupt should occur.

A DMP interrupt can be configured to trigger on either of the two conditions below:

- One FIFO period has elapsed (set by *gyro\_set\_sample\_rate*).
- A tap event has been detected.

By default, condition *a* is selected when *gyro\_enable\_tap* is called.

**Parameters:**

*mode* DMP\_INT\_GESTURE or DMP\_INT\_CONTINUOUS.

**Returns:**

0 if successful.

**4.19.2.15 int dmp\_set\_orient\_axes (unsigned char *axis*)**

Set which orientations will trigger an event.

This function expects a mask containing a combination of the following macros: ORIENTATION\_X\_UP ORIENTATION\_X\_DOWN ORIENTATION\_Y\_UP ORIENTATION\_Y\_DOWN ORIENTATION\_Z\_UP ORIENTATION\_Z\_DOWN ORIENTATION\_FLIP

**Parameters:**

*axis* 1, 2, and 4 for XYZ, respectively.

**Returns:**

0 if successful.

#### 4.19.2.16 int dmp\_set\_orient\_thresh (float *angle*)

Set orientation angle threshold.

**Parameters:**

*angle* Angle where orientation changes.

**Returns:**

0 if successful.

#### 4.19.2.17 int dmp\_set\_orient\_time (unsigned short *time*)

Set orientation time.

Sets the length of time that the device must remain in the same orientation before a DMP orientation event will occur. A mandatory 60 ms is added to this parameter.

**Parameters:**

*time* Time in milliseconds.

**Returns:**

0 if successful.

#### 4.19.2.18 int dmp\_set\_orientation (unsigned short *orient*)

Push gyro and accel orientation to the DMP.

The orientation is represented here as the output of *inv\_orientation\_matrix\_to\_scalar*.

**Parameters:**

*orient* Gyro and accel orientation in body frame.

**Returns:**

0 if successful.

#### 4.19.2.19 int dmp\_set\_pedometer\_step\_count (unsigned long *count*)

Overwrite current step count.

WARNING: This function writes to DMP memory and could potentially encounter a race condition if called while the pedometer is enabled.



**4.19 Sensor Driver Layer****93****Parameters:**

*count* New step count.

**Returns:**

0 if successful.

**4.19.2.20 int dmp\_set\_pedometer\_walk\_time (unsigned long *time*)**

Overwrite current walk time.

WARNING: This function writes to DMP memory and could potentially encounter a race condition if called while the pedometer is enabled.

**Parameters:**

*time* New walk time in milliseconds.

**4.19.2.21 int dmp\_set\_shake\_reject\_thresh (long *sf*, unsigned short *thresh*)**

Set shake rejection threshold.

If the DMP detects a gyro sample larger than *thresh*, taps are rejected.

**Parameters:**

*sf* Gyro scale factor.

*thresh* Gyro threshold in dps.

**Returns:**

0 if successful.

**4.19.2.22 int dmp\_set\_shake\_reject\_time (unsigned short *time*)**

Set shake rejection time.

Sets the length of time that the gyro must be outside of the threshold set by *gyro\_set\_shake\_reject\_thresh* before taps are rejected. A mandatory 60 ms is added to this parameter.

**Parameters:**

*time* Time in milliseconds.

**Returns:**

0 if successful.

**4.19.2.23 int dmp\_set\_shake\_reject\_timeout (unsigned short *time*)**

Set shake rejection timeout.

Sets the length of time after a shake rejection that the gyro must stay inside of the threshold before taps can be detected again. A mandatory 60 ms is added to this parameter.

**Parameters:**

*time* Time in milliseconds.

**Returns:**

0 if successful.

**4.19.2.24 int dmp\_set\_tap\_axes (unsigned char *axis*)**

Set which axes will register a tap.

**Parameters:**

*axis* 1, 2, and 4 for XYZ, respectively.

**Returns:**

0 if successful.

**4.19.2.25 int dmp\_set\_tap\_count (unsigned char *min\_taps*)**

Set minimum number of taps needed for an interrupt.

**Parameters:**

*min\_taps* Minimum consecutive taps (1-4).

**Returns:**

0 if successful.

**4.19.2.26 int dmp\_set\_tap\_thresh (unsigned char *axis*, unsigned short *thresh*)**

Set tap threshold for a specific axis.

#### 4.19 Sensor Driver Layer

95

##### Parameters:

*axis* 1, 2, and 4 for XYZ accel, respectively.

*thresh* Tap threshold, in mg/ms.

##### Returns:

0 if successful.

##### 4.19.2.27 int dmp\_set\_tap\_time (unsigned short time)

Set length between valid taps.

##### Parameters:

*time* Milliseconds between taps.

##### Returns:

0 if successful.

##### 4.19.2.28 int dmp\_set\_tap\_time\_multi (unsigned short time)

Set max time between taps to register as a multi-tap.

##### Parameters:

*time* Max milliseconds between taps.

##### Returns:

0 if successful.

##### 4.19.2.29 int gyro\_configure\_fifo (unsigned char sensors)

Select which sensors are pushed to FIFO.

*sensors* can contain a combination of the following flags:

INV\_X\_GYRO, INV\_Y\_GYRO, INV\_Z\_GYRO

INV\_XYZ\_GYRO

INV\_XYZ\_ACCEL

##### Parameters:

*sensors* Mask of sensors to push to FIFO.

**Returns:**

0 if successful.

**4.19.2.30 int gyro\_get\_accel\_fsr (unsigned char \* *fsr*)**

Get the accel full-scale range.

**Parameters:**

*fsr* Current full-scale range.

**Returns:**

0 if successful.

**4.19.2.31 int gyro\_get\_accel\_reg (short \* *data*, unsigned long \* *timestamp*)**

Read raw accel data directly from the registers.

**Parameters:**

*data* Raw data in hardware units.

*timestamp* Timestamp in milliseconds. Null if not needed.

**Returns:**

0 if successful.

**4.19.2.32 int gyro\_get\_accel\_sens (unsigned short \* *sens*)**

Get accel sensitivity scale factor.

**Parameters:**

*sens* Conversion from hardware units to g's.

**Returns:**

0 if successful.

**4.19 Sensor Driver Layer****97****4.19.2.33 int gyro\_get\_compass\_sample\_rate (unsigned short \* *rate*)**

Get compass sampling rate.

**Parameters:**

*rate* Current compass sampling rate (Hz).

**Returns:**

0 if successful.

**4.19.2.34 int gyro\_get\_fifo\_config (unsigned char \* *sensors*)**

Get current FIFO configuration.

*sensors* can contain a combination of the following flags:

INV\_X\_GYRO, INV\_Y\_GYRO, INV\_Z\_GYRO

INV\_XYZ\_GYRO

INV\_XYZ\_ACCEL

**Parameters:**

*sensors* Mask of sensors in FIFO.

**Returns:**

0 if successful.

**4.19.2.35 int gyro\_get\_gyro\_fsr (unsigned short \* *fsr*)**

Get the gyro full-scale range.

**Parameters:**

*fsr* Current full-scale range.

**Returns:**

0 if successful.

**4.19.2.36 int gyro\_get\_gyro\_reg (short \* *data*, unsigned long \* *timestamp*)**

Read raw gyro data directly from the registers.

**Parameters:**

*data* Raw data in hardware units.

*timestamp* Timestamp in milliseconds. Null if not needed.

**Returns:**

0 if successful.

**4.19.2.37 int gyro\_get\_gyro\_sens (float \* *sens*)**

Get gyro sensitivity scale factor.

**Parameters:**

*sens* Conversion from hardware units to dps.

**Returns:**

0 if successful.

**4.19.2.38 int gyro\_get\_lpf (unsigned short \* *lpf*)**

Get the current DLPF setting.

**Parameters:**

*lpf* Current LPF setting. 0 if successful.

**4.19.2.39 int gyro\_get\_power\_state (unsigned char \* *power\_on*)**

Get current power state.

**Parameters:**

*power\_on* 1 if turned on, 0 if suspended.

**Returns:**

0 if successful.

**4.19 Sensor Driver Layer****99****4.19.2.40 int gyro\_get\_sample\_rate (unsigned short \* *rate*)**

Get sampling rate.

**Parameters:**

*rate* Current sampling rate (Hz).

**Returns:**

0 if successful.

**4.19.2.41 int gyro\_get\_temperature (long \* *data*, unsigned long \* *timestamp*)**

Read temperature data directly from the registers.

**Parameters:**

*data* Data in q16 format.

*timestamp* Timestamp in milliseconds. Null if not needed.

**Returns:**

0 if successful.

**4.19.2.42 int gyro\_init (struct int\_param\_s \* *int\_param*)**

Initialize hardware.

Initial configuration:

Gyro FSR: +/- 2000DPS

Accel FSR +/- 2G

DLPF: 42Hz

FIFO rate: 50Hz

Clock source: Gyro PLL

FIFO: Disabled.

Data ready interrupt: Disabled, active low, unlatched.

**Parameters:**

*int\_param* Platform-specific parameters to interrupt API.

**Returns:**

0 if successful.

**4.19.2.43 int gyro\_lp\_accel\_mode (unsigned char *rate*)**

Enter low-power accel-only mode.

In low-power accel mode, the chip goes to sleep and only wakes up to sample the accelerometer at one of the following frequencies:

1.25Hz, 5Hz, 20Hz, 40Hz

If the requested rate is not one of the four listed above, the device will be set to the next highest rate. Requesting a rate above 40Hz will result in an error.

NOTE: To select a wake-up frequency of 1.25Hz, set the *rate* parameter to 1.

**Parameters:**

*rate* Minimum sampling rate, or zero to disable LP accel mode.

**Returns:**

0 if successful.

**4.19.2.44 int gyro\_read\_fifo (short \* *gyro*, short \* *accel*, unsigned long \* *timestamp*, unsigned char \* *sensors*, unsigned char \* *more*)**

Get one packet from the FIFO.

If *sensors* does not contain a particular sensor, disregard the data returned to that pointer.

*sensors* can contain a combination of the following flags:

INV\_X\_GYRO, INV\_Y\_GYRO, INV\_Z\_GYRO

INV\_XYZ\_GYRO

INV\_XYZ\_ACCEL

If the FIFO has no new data, *sensors* will be zero.

If the FIFO is disabled, *sensors* will be zero and this function will return a non-zero error code.

**Parameters:**

*gyro* Gyro data in hardware units.

*accel* Accel data in hardware units.

*timestamp* Timestamp in milliseconds.

*sensors* Mask of sensors read from FIFO.

*more* Number of remaining packets.



**4.19 Sensor Driver Layer****101****Returns:**

0 if successful.

**4.19.2.45 int gyro\_read\_fifo\_stream (unsigned short *length*, unsigned char \*  
*data*, unsigned char \* *more*)**

Get one unparsed packet from the FIFO.

This function should be used if the packet is to be parsed elsewhere.

**Parameters:**

*length* Length of one FIFO packet.

*data* FIFO packet.

*more* Number of remaining packets.

**4.19.2.46 int gyro\_read\_reg (unsigned char *reg*, unsigned char \* *data*)**

Read from a single register.

NOTE: The memory and FIFO read/write registers cannot be accessed.

**Parameters:**

*reg* Register address.

*data* Register data.

**Returns:**

0 if successful.

**4.19.2.47 int gyro\_reg\_dump (void)**

Register dump for testing.

**Returns:**

0 if successful.

**4.19.2.48 int gyro\_reset\_fifo (void)**

Reset FIFO read/write pointers.

**Returns:**

0 if successful.

**4.19.2.49 int gyro\_run\_self\_test (long \* gyro, long \* accel)**

Trigger gyro/accel/compass self-test.

On error, the self-test returns a mask representing the sensor(s) that failed. The mask is defined as follows:

Bit 0: X gyro.

Bit 1: Y gyro.

Bit 2: Z gyro.

Bit 3: X accel.

Bit 4: Y accel.

Bit 5: Z accel.

Bit 6: X compass.

Bit 7: Y compass.

Bit 8: Z compass.

Bit 9: I2C error.

**Parameters:**

*gyro* Gyro biases in q16 format.

*accel* Accel biases (if applicable) in q16 format.

**Returns:**

0 if successful.

**4.19.2.50 int gyro\_set\_accel\_bias (const long \* accel\_bias)**

Push biases to the accel bias registers.

This function expects biases relative to the current sensor output, and these biases will be added to the factory-supplied values.

#### 4.19 Sensor Driver Layer

103

**Parameters:**

*accel\_bias* New biases.

**Returns:**

0 if successful.

##### 4.19.2.51 `int gyro_set_accel_fsr (unsigned char fsr)`

Set the accel full-scale range.

**Parameters:**

*fsr* Desired full-scale range.

**Returns:**

0 if successful.

##### 4.19.2.52 `int gyro_set_bypass (unsigned char bypass_on)`

Set device to bypass mode.

**Parameters:**

*bypass\_on* 1 to enable bypass mode.

**Returns:**

0 if successful.

##### 4.19.2.53 `int gyro_set_compass_sample_rate (unsigned short rate)`

Set compass sampling rate.

The compass on the auxiliary I2C bus is read by the MPU hardware at a maximum of 100Hz. The actual rate can be set to a fraction of the gyro sampling rate.

WARNING: The new rate may be different than what was requested. Call `gyro_get_compass_sample_rate` to check the actual setting.

**Parameters:**

*rate* Desired compass sampling rate (Hz).

**Returns:**

0 if successful.

#### 4.19.2.54 int gyro\_set\_gyro\_fsr (unsigned short *fsr*)

Set the gyro full-scale range.

**Parameters:**

*fsr* Desired full-scale range.

**Returns:**

0 if successful.

#### 4.19.2.55 int gyro\_set\_int\_latched (unsigned char *enable*)

Enable latched interrupts.

Any MPU register will clear the interrupt.

**Parameters:**

*enable* 1 to enable, 0 to disable.

**Returns:**

0 if successful.

#### 4.19.2.56 int gyro\_set\_int\_level (unsigned char *active\_low*)

Set interrupt level.

**Parameters:**

*active\_low* 1 for active low, 0 for active high.

**Returns:**

0 if successful.

#### 4.19.2.57 int gyro\_set\_lpf (unsigned short *lpf*)

Set digital low pass filter.

The following LPF settings are supported: 188, 98, 42, 20, 10, 5.

**Parameters:**

*lpf* Desired LPF setting.

#### 4.19 Sensor Driver Layer

105

**Returns:**

0 if successful.

##### 4.19.2.58 int gyro\_set\_sample\_rate (unsigned short *rate*)

Set sampling rate.

Sampling rate must be between 4Hz and 1kHz.

**Parameters:**

*rate* Desired sampling rate (Hz).

**Returns:**

0 if successful.

##### 4.19.2.59 int gyro\_set\_sensors (unsigned char *sensors*)

Turn specific sensors on/off.

*sensors* can contain a combination of the following flags:

INV\_X\_GYRO, INV\_Y\_GYRO, INV\_Z\_GYRO

INV\_XYZ\_GYRO

INV\_XYZ\_ACCEL

INV\_XYZ\_COMPASS

**Parameters:**

*sensors* Mask of sensors to wake.

**Returns:**

0 if successful.