

# InvenSense Device Driver library

3.8.4

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Architecture overview . . . . .	1
1.3	Importing and building libIDD . . . . .	2
1.3.1	Requested library components . . . . .	2
1.3.2	Building the library . . . . .	3
1.3.3	About debugging . . . . .	3
<b>2</b>	<b>Integration Guide</b>	<b>5</b>
2.1	Adapter . . . . .	5
2.2	Application . . . . .	6
<b>3</b>	<b>Supported devices</b>	<b>7</b>
3.1	ICM20602 . . . . .	7
3.2	ICM20603 . . . . .	8
3.3	ICM20690 . . . . .	8
3.4	SmartMotion . . . . .	8
<b>4</b>	<b>Deprecated List</b>	<b>9</b>
<b>5</b>	<b>Module Index</b>	<b>11</b>
5.1	Modules . . . . .	11
<b>6</b>	<b>Class Index</b>	<b>13</b>
6.1	Class List . . . . .	13

<b>7</b>	<b>Module Documentation</b>	<b>17</b>
7.1	Error code	17
7.1.1	Detailed Description	17
7.1.2	Enumeration Type Documentation	17
7.1.2.1	inv_error	17
7.2	Sensor types	18
7.2.1	Detailed Description	19
7.2.2	Macro Definition Documentation	19
7.2.2.1	INV_SENSOR_TYPE_ENERGY_EXPENDITURE	19
7.2.2.2	INV_SENSOR_TYPE_GYROMETER	19
7.2.2.3	INV_SENSOR_TYPE_META_DATA	19
7.2.2.4	INV_SENSOR_TYPE_UNCAL_GYROMETER	19
7.2.3	Typedef Documentation	19
7.2.3.1	inv_sensor_listener_event_cb_t	19
7.2.4	Enumeration Type Documentation	20
7.2.4.1	inv_sensor_status	20
7.2.4.2	inv_sensor_type	20
7.3	Sensor Configuration	22
7.3.1	Detailed Description	23
7.3.2	Typedef Documentation	23
7.3.2.1	inv_sensor_config_bac_t	23
7.3.2.2	inv_sensor_config_BSCD_t	24
7.3.2.3	inv_sensor_config_distance_t	24
7.3.2.4	inv_sensor_config_double_tap_t	24
7.3.2.5	inv_sensor_config_energy_expenditure_t	24
7.3.2.6	inv_sensor_config_mounting_mtx_t	25
7.3.2.7	inv_sensor_config_offset_t	25
7.3.2.8	inv_sensor_config_shake_wrist_t	25
7.3.2.9	inv_sensor_config_stepc_t	25
7.3.3	Enumeration Type Documentation	26

7.3.3.1	<code>inv_sensor_config</code>	26
7.4	Device	27
7.4.1	Detailed Description	29
7.4.2	Macro Definition Documentation	29
7.4.2.1	<code>inv_device_enable</code>	29
7.4.3	Function Documentation	29
7.4.3.1	<code>inv_device_cleanup(const inv_device_t *dev)</code>	29
7.4.3.2	<code>inv_device_enable_sensor(const inv_device_t *dev, int sensor, inv_bool_t start)</code>	30
7.4.3.3	<code>inv_device_flush_sensor(const inv_device_t *dev, int sensor)</code>	30
7.4.3.4	<code>inv_device_get_fw_info(const inv_device_t *dev, struct inv_fw_version *version)</code>	31
7.4.3.5	<code>inv_device_get_sensor_bias(const inv_device_t *dev, int sensor, float bias[3])</code>	31
7.4.3.6	<code>inv_device_get_sensor_config(const inv_device_t *dev, int sensor, int settings, void *value, uint16_t size)</code>	32
7.4.3.7	<code>inv_device_get_sensor_data(const inv_device_t *dev, int sensor, inv_sensor_event_t *event)</code>	32
7.4.3.8	<code>inv_device_load(const inv_device_t *dev, int type, const uint8_t *image, uint32_t size, inv_bool_t verify, inv_bool_t force)</code>	33
7.4.3.9	<code>inv_device_ping_sensor(const inv_device_t *dev, int sensor)</code>	33
7.4.3.10	<code>inv_device_poll(const inv_device_t *dev)</code>	34
7.4.3.11	<code>inv_device_read_mems_register(const inv_device_t *dev, int sensor, uint16_t reg_addr, void *data, uint16_t len)</code>	34
7.4.3.12	<code>inv_device_reset(const inv_device_t *dev)</code>	35
7.4.3.13	<code>inv_device_self_test(const inv_device_t *dev, int sensor)</code>	35
7.4.3.14	<code>inv_device_set_running_state(const inv_device_t *dev, inv_bool_t state)</code>	36
7.4.3.15	<code>inv_device_set_sensor_bias(const inv_device_t *dev, int sensor, const float bias[3])</code>	36
7.4.3.16	<code>inv_device_set_sensor_config(const inv_device_t *dev, int sensor, int settings, const void *arg, uint16_t size)</code>	37
7.4.3.17	<code>inv_device_set_sensor_mounting_matrix(const inv_device_t *dev, int sensor, const float matrix[9])</code>	38
7.4.3.18	<code>inv_device_set_sensor_period(const inv_device_t *dev, int sensor, uint32_t period)</code>	38
7.4.3.19	<code>inv_device_set_sensor_period_us(const inv_device_t *dev, int sensor, uint32_t period)</code>	39
7.4.3.20	<code>inv_device_set_sensor_timeout(const inv_device_t *dev, int sensor, uint32_t timeout)</code>	40

7.4.3.21	<code>inv_device_set_sensor_timeout_us(const inv_device_t *dev, int sensor, uint32_t timeout)</code>	40
7.4.3.22	<code>inv_device_setup(const inv_device_t *dev)</code>	41
7.4.3.23	<code>inv_device_start_sensor(const inv_device_t *dev, int sensor)</code>	41
7.4.3.24	<code>inv_device_stop_sensor(const inv_device_t *dev, int sensor)</code>	42
7.4.3.25	<code>inv_device_whoami(const inv_device_t *dev, uint8_t *whoami)</code>	42
7.4.3.26	<code>inv_device_write_mems_register(const inv_device_t *dev, int sensor, uint16_t reg_addr, const void *data, uint16_t len)</code>	43
7.5	DeviceSmartMotion	44
7.5.1	Detailed Description	46
7.5.2	Enumeration Type Documentation	47
7.5.2.1	<code>inv_smartmotion_config_setting</code>	47
7.5.3	Function Documentation	47
7.5.3.1	<code>inv_device_smart_motion_init(inv_device_smart_motion_t *self, inv_device_t *hw_device)</code>	47
7.5.3.2	<code>inv_device_smart_motion_init2(inv_device_smart_motion_t *self, inv_device_t *hw_device, void *opt)</code>	47
7.5.3.3	<code>inv_device_smart_motion_setup(void *context)</code>	48
7.5.4	Variable Documentation	48
7.5.4.1	<code>I</code>	48
7.6	DeviceEmdWrapper	49
7.6.1	Detailed Description	49
7.7	DeviceIcm20602	50
7.7.1	Detailed Description	51
7.7.2	Function Documentation	51
7.7.2.1	<code>inv_device_icm20602_init(inv_device_icm20602_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)</code>	51
7.7.2.2	<code>inv_device_icm20602_init2(inv_device_icm20602_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener)</code>	51
7.7.2.3	<code>inv_device_icm20602_init_aux_compass(inv_device_icm20602_t *self, int aux_compass_id, uint8_t aux_compass_addr)</code>	51
7.7.2.4	<code>inv_device_icm20602_init_serif_ois(inv_device_icm20602_t *self, const inv_host_serif_t *serif_ois)</code>	52

7.7.2.5	<code>inv_device_icm20602_init_serif_ois2(inv_device_icm20602_t *self, const inv_serif_hal_t *serif_ois)</code>	52
7.8	DeviceIcm20603	53
7.8.1	Detailed Description	54
7.8.2	Function Documentation	54
7.8.2.1	<code>inv_device_icm20603_init(inv_device_icm20603_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)</code>	54
7.8.2.2	<code>inv_device_icm20603_init2(inv_device_icm20603_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener)</code>	54
7.8.2.3	<code>inv_device_icm20603_init_aux_compass(inv_device_icm20603_t *self, int aux_compass_id, uint8_t aux_compass_addr)</code>	54
7.8.2.4	<code>inv_device_icm20603_init_serif_ois(inv_device_icm20603_t *self, const inv_host_serif_t *serif_ois)</code>	55
7.8.2.5	<code>inv_device_icm20603_init_serif_ois2(inv_device_icm20603_t *self, const inv_serif_hal_t *serif_ois)</code>	55
7.9	DeviceIcm20690	56
7.9.1	Detailed Description	57
7.9.2	Function Documentation	57
7.9.2.1	<code>inv_device_icm20690_init(inv_device_icm20690_t *self, const inv_host_serif_t *serif, const inv_sensor_listener_t *listener)</code>	57
7.9.2.2	<code>inv_device_icm20690_init2(inv_device_icm20690_t *self, const inv_serif_hal_t *serif, const inv_sensor_listener_t *listener)</code>	57
7.9.2.3	<code>inv_device_icm20690_init_aux_compass(inv_device_icm20690_t *self, int aux_compass_id, uint8_t aux_compass_addr)</code>	57
7.9.2.4	<code>inv_device_icm20690_init_serif_ois(inv_device_icm20690_t *self, const inv_host_serif_t *serif_ois)</code>	58
7.9.2.5	<code>inv_device_icm20690_init_serif_ois2(inv_device_icm20690_t *self, const inv_serif_hal_t *serif_ois)</code>	58
7.10	Host Serial Interface	59
7.10.1	Detailed Description	60
7.11	Data Converter	61
7.11.1	Detailed Description	61
7.11.2	Function Documentation	61
7.11.2.1	<code>inv_dc_float_to_sf32(const float *in, uint32_t len, uint8_t qx, int32_t *out)</code>	61
7.11.2.2	<code>inv_dc_sf32_to_float(const int32_t *in, uint32_t len, uint8_t qx, float *out)</code>	61

7.12	Error Helper	63
7.12.1	Detailed Description	63
7.12.2	Function Documentation	63
7.12.2.1	inv_error_str(int error)	63
7.13	Message	64
7.13.1	Detailed Description	65
7.13.2	Macro Definition Documentation	65
7.13.2.1	INV_MSG	65
7.13.2.2	INV_MSG_LEVEL	65
7.13.3	Function Documentation	65
7.13.3.1	inv_msg(int level, const char *str,...)	65
7.13.3.2	inv_msg_get_level(void)	65
7.13.3.3	inv_msg_printer_default(int level, const char *str, va_list ap)	66
7.13.3.4	inv_msg_setup(int level, inv_msg_printer_t printer)	66
7.13.3.5	inv_msg_setup_default(void)	66
7.13.3.6	inv_msg_setup_level(int level)	66
7.14	Icm20602 driver	68
7.14.1	Detailed Description	69
7.14.2	Function Documentation	69
7.14.2.1	inv_icm20602_get_dataready_interrupt_time_us(void)	69
7.14.2.2	inv_icm20602_get_time_us(void)	69
7.14.2.3	inv_icm20602_reset_states(struct inv_icm20602 *s, const struct inv_icm20602_serif *serif)	69
7.14.2.4	inv_icm20602_reset_states_serif_ois(struct inv_icm20602 *s, const struct inv_icm20602_serif_ois *serif_ois)	70
7.14.2.5	inv_icm20602_sleep(int ms)	70
7.14.2.6	inv_icm20602_sleep_us(int us)	70
7.15	Icm20602 akm compass support	71
7.15.1	Detailed Description	71
7.15.2	Enumeration Type Documentation	72
7.15.2.1	inv_icm20602_compass_id	72



7.15.3	Function Documentation	72
7.15.3.1	inv_icm20602_check_akm_self_test(struct inv_icm20602 *s)	72
7.15.3.2	inv_icm20602_compass_getstate(struct inv_icm20602 *s)	72
7.15.3.3	inv_icm20602_get_compass_bytes(struct inv_icm20602 *s)	72
7.15.3.4	inv_icm20602_get_compass_data(struct inv_icm20602 *s, const unsigned char *packet, unsigned char *raw_compass)	72
7.15.3.5	inv_icm20602_read_akm_scale(struct inv_icm20602 *s, int *scale)	73
7.15.3.6	inv_icm20602_register_aux_compass(struct inv_icm20602 *s, enum inv_icm20602_compass_id compass_id, uint8_t compass_i2c_addr)	73
7.15.3.7	inv_icm20602_resume_akm(struct inv_icm20602 *s)	73
7.15.3.8	inv_icm20602_setup_compass_akm(struct inv_icm20602 *s)	74
7.15.3.9	inv_icm20602_suspend_akm(struct inv_icm20602 *s)	74
7.16	Icm20602 secondary driver transport	75
7.16.1	Detailed Description	75
7.16.2	Macro Definition Documentation	76
7.16.2.1	COMPASS_I2C_SLV_READ	76
7.16.3	Function Documentation	76
7.16.3.1	inv_icm20602_execute_read_secondary(struct inv_icm20602 *s, int index, unsigned char addr, int reg, int len, unsigned char *d)	76
7.16.3.2	inv_icm20602_execute_write_secondary(struct inv_icm20602 *s, int index, unsigned char addr, int reg, unsigned char v)	76
7.16.3.3	inv_icm20602_read_secondary(struct inv_icm20602 *s, int index, unsigned char addr, unsigned char reg, char len)	77
7.16.3.4	inv_icm20602_secondary_disable_i2c(struct inv_icm20602 *s)	77
7.16.3.5	inv_icm20602_secondary_enable_i2c(struct inv_icm20602 *s)	77
7.16.3.6	inv_icm20602_secondary_stop_channel(struct inv_icm20602 *s, int index)	77
7.16.3.7	inv_icm20602_write_secondary(struct inv_icm20602 *s, int index, unsigned char addr, unsigned char reg, char v)	78
7.17	Icm20602 control	79
7.17.1	Detailed Description	80
7.17.2	Function Documentation	80
7.17.2.1	inv_icm20602_all_sensors_off(struct inv_icm20602 *s)	80
7.17.2.2	inv_icm20602_check_drdy(struct inv_icm20602 *s, uint8_t int_status)	80

7.17.2.3	<code>inv_icm20602_check_wom_status(struct inv_icm20602 *s, uint8_t int_status)</code> . . .	81
7.17.2.4	<code>inv_icm20602_configure_accel_wom(struct inv_icm20602 *s, uint8_t wom_↵ threshold)</code> . . . . .	81
7.17.2.5	<code>inv_icm20602_disable_fifo(struct inv_icm20602 *s)</code> . . . . .	81
7.17.2.6	<code>inv_icm20602_enable_fifo(struct inv_icm20602 *s, int bit_mask)</code> . . . . .	81
7.17.2.7	<code>inv_icm20602_enable_mems(struct inv_icm20602 *s, int bit_mask, uint16_↵ t smplr_t_divider)</code> . . . . .	82
7.17.2.8	<code>inv_icm20602_enable_sensor(struct inv_icm20602 *s, enum inv_icm20602_↵ sensor sensor, uint8_t enable)</code> . . . . .	82
7.17.2.9	<code>inv_icm20602_get_int_status(struct inv_icm20602 *s, uint8_t *int_status)</code> . . . .	82
7.17.2.10	<code>inv_icm20602_get_st_bias(struct inv_icm20602 *s, int *st_bias)</code> . . . . .	83
7.17.2.11	<code>inv_icm20602_has_data_ready(struct inv_icm20602 *s)</code> . . . . .	83
7.17.2.12	<code>inv_icm20602_is_sensor_enabled(struct inv_icm20602 *s, enum inv_↵ icm20602_sensor sensor)</code> . . . . .	83
7.17.2.13	<code>inv_icm20602_poll_delay_count(struct inv_icm20602 *s, int16_t *delay_count)</code> .	83
7.17.2.14	<code>inv_icm20602_poll_fifo_data(struct inv_icm20602 *s, struct inv_icm20602_fifo_↵ _states *states, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3], int16_t compass_data[3])</code> . . . . .	84
7.17.2.15	<code>inv_icm20602_poll_fifo_data_setup(struct inv_icm20602 *s, struct inv_↵ icm20602_fifo_states *states, uint8_t int_status)</code> . . . . .	84
7.17.2.16	<code>inv_icm20602_poll_ois_gyro_data(struct inv_icm20602 *s, int16_t gyro_data[3])</code>	84
7.17.2.17	<code>inv_icm20602_poll_sensor_data_reg(struct inv_icm20602 *s, int16_t acc_↵ data[3], int16_t *temp_data, int16_t gyro_data[3])</code> . . . . .	84
7.17.2.18	<code>inv_icm20602_reset_fifo(struct inv_icm20602 *s)</code> . . . . .	85
7.17.2.19	<code>inv_icm20602_run_selftest(struct inv_icm20602 *s)</code> . . . . .	85
7.17.2.20	<code>inv_icm20602_set_sensor_period(struct inv_icm20602 *s, enum inv_↵ icm20602_sensor sensor, uint16_t delayInMs)</code> . . . . .	85
7.17.2.21	<code>inv_icm20602_set_slave_compass_id(struct inv_icm20602 *s)</code> . . . . .	86
7.17.2.22	<code>inv_icm20602_set_st_bias(struct inv_icm20602 *s, int *st_bias)</code> . . . . .	86
7.18	Icm20602 driver serif . . . . .	87
7.18.1	Detailed Description . . . . .	87
7.19	Icm20602 driver setup . . . . .	88
7.19.1	Detailed Description . . . . .	89
7.19.2	Function Documentation . . . . .	89

7.19.2.1	<code>inv_icm20602_accel_fsr_2_reg(int32_t fsr)</code>	89
7.19.2.2	<code>inv_icm20602_get_accel_bandwidth(struct inv_icm20602 *s)</code>	89
7.19.2.3	<code>inv_icm20602_get_accel_fullscale(struct inv_icm20602 *s)</code>	90
7.19.2.4	<code>inv_icm20602_get_accel_ois_fullscale(struct inv_icm20602 *s)</code>	90
7.19.2.5	<code>inv_icm20602_get_chip_base_sample_rate(struct inv_icm20602 *s)</code>	90
7.19.2.6	<code>inv_icm20602_get_chip_info(struct inv_icm20602 *s, uint8_t chip_info[3])</code>	90
7.19.2.7	<code>inv_icm20602_get_chip_power_state(struct inv_icm20602 *s)</code>	90
7.19.2.8	<code>inv_icm20602_get_gyro_bandwidth(struct inv_icm20602 *s)</code>	91
7.19.2.9	<code>inv_icm20602_get_gyro_fullscale(struct inv_icm20602 *s)</code>	91
7.19.2.10	<code>inv_icm20602_get_gyro_ois_fullscale(struct inv_icm20602 *s)</code>	91
7.19.2.11	<code>inv_icm20602_get_whoami(struct inv_icm20602 *s, uint8_t *whoami)</code>	91
7.19.2.12	<code>inv_icm20602_gyro_fsr_2_reg(int32_t fsr)</code>	91
7.19.2.13	<code>inv_icm20602_initialize(struct inv_icm20602 *s)</code>	92
7.19.2.14	<code>inv_icm20602_is_advanced_features_supported(void)</code>	92
7.19.2.15	<code>inv_icm20602_reg_2_accel_fsr(uint8_t reg)</code>	92
7.19.2.16	<code>inv_icm20602_reg_2_gyro_fsr(uint8_t reg)</code>	92
7.19.2.17	<code>inv_icm20602_set_accel_bandwidth(struct inv_icm20602 *s, int level)</code>	92
7.19.2.18	<code>inv_icm20602_set_accel_fullscale(struct inv_icm20602 *s, int level)</code>	93
7.19.2.19	<code>inv_icm20602_set_accel_ois_fullscale(struct inv_icm20602 *s, int level)</code>	93
7.19.2.20	<code>inv_icm20602_set_chip_power_state(struct inv_icm20602 *s, uint8_t func, uint8_t on_off)</code>	93
7.19.2.21	<code>inv_icm20602_set_divider(struct inv_icm20602 *s, uint8_t idiv)</code>	93
7.19.2.22	<code>inv_icm20602_set_fsync_bit_location(struct inv_icm20602 *s, int bit_location)</code>	94
7.19.2.23	<code>inv_icm20602_set_gyro_bandwidth(struct inv_icm20602 *s, int level)</code>	94
7.19.2.24	<code>inv_icm20602_set_gyro_fullscale(struct inv_icm20602 *s, int level)</code>	94
7.19.2.25	<code>inv_icm20602_set_gyro_ois_fullscale(struct inv_icm20602 *s, int level)</code>	94
7.19.2.26	<code>inv_icm20602_soft_reset(struct inv_icm20602 *s)</code>	95
7.20	Icm20602 driver transport	96
7.20.1	Detailed Description	96
7.20.2	Function Documentation	96

7.20.2.1	<code>inv_icm20602_mems_read_reg(struct inv_icm20602 *s, uint16_t reg, uint32_t length, uint8_t *data)</code>	96
7.20.2.2	<code>inv_icm20602_mems_write_reg(struct inv_icm20602 *s, uint16_t reg, uint32_t length, const uint8_t *data)</code>	97
7.20.2.3	<code>inv_icm20602_mems_write_reg_one(struct inv_icm20602 *s, uint16_t reg, uint8_t data)</code>	97
7.21	Icm20603 driver	98
7.21.1	Detailed Description	99
7.21.2	Function Documentation	99
7.21.2.1	<code>inv_icm20603_get_dataready_interrupt_time_us(void)</code>	99
7.21.2.2	<code>inv_icm20603_get_time_us(void)</code>	99
7.21.2.3	<code>inv_icm20603_reset_states(struct inv_icm20603 *s, const struct inv_icm20603_serif *serif)</code>	99
7.21.2.4	<code>inv_icm20603_reset_states_serif_ois(struct inv_icm20603 *s, const struct inv_icm20603_serif *serif_ois)</code>	100
7.21.2.5	<code>inv_icm20603_sleep(int ms)</code>	100
7.21.2.6	<code>inv_icm20603_sleep_us(int us)</code>	100
7.22	Icm20603 akm compass support	101
7.22.1	Detailed Description	101
7.22.2	Enumeration Type Documentation	102
7.22.2.1	<code>inv_icm20603_compass_id</code>	102
7.22.3	Function Documentation	102
7.22.3.1	<code>inv_icm20603_check_akm_self_test(struct inv_icm20603 *s)</code>	102
7.22.3.2	<code>inv_icm20603_compass_getstate(struct inv_icm20603 *s)</code>	102
7.22.3.3	<code>inv_icm20603_get_compass_bytes(struct inv_icm20603 *s)</code>	102
7.22.3.4	<code>inv_icm20603_get_compass_data(struct inv_icm20603 *s, const unsigned char *packet, unsigned char *raw_compass)</code>	102
7.22.3.5	<code>inv_icm20603_read_akm_scale(struct inv_icm20603 *s, int *scale)</code>	103
7.22.3.6	<code>inv_icm20603_register_aux_compass(struct inv_icm20603 *s, enum inv_icm20603_compass_id compass_id, uint8_t compass_i2c_addr)</code>	103
7.22.3.7	<code>inv_icm20603_resume_akm(struct inv_icm20603 *s)</code>	103
7.22.3.8	<code>inv_icm20603_setup_compass_akm(struct inv_icm20603 *s)</code>	104
7.22.3.9	<code>inv_icm20603_suspend_akm(struct inv_icm20603 *s)</code>	104

7.23 Icm20603 secondary driver transport . . . . .	105
7.23.1 Detailed Description . . . . .	105
7.23.2 Macro Definition Documentation . . . . .	106
7.23.2.1 COMPASS_I2C_SLV_READ . . . . .	106
7.23.3 Function Documentation . . . . .	106
7.23.3.1 inv_icm20603_execute_read_secondary(struct inv_icm20603 *s, int index, unsigned char addr, int reg, int len, unsigned char *d) . . . . .	106
7.23.3.2 inv_icm20603_execute_write_secondary(struct inv_icm20603 *s, int index, unsigned char addr, int reg, unsigned char v) . . . . .	106
7.23.3.3 inv_icm20603_read_secondary(struct inv_icm20603 *s, int index, unsigned char addr, unsigned char reg, char len) . . . . .	107
7.23.3.4 inv_icm20603_secondary_disable_i2c(struct inv_icm20603 *s) . . . . .	107
7.23.3.5 inv_icm20603_secondary_enable_i2c(struct inv_icm20603 *s) . . . . .	107
7.23.3.6 inv_icm20603_secondary_stop_channel(struct inv_icm20603 *s, int index) . . . . .	107
7.23.3.7 inv_icm20603_write_secondary(struct inv_icm20603 *s, int index, unsigned char addr, unsigned char reg, char v) . . . . .	108
7.24 Icm20603 control . . . . .	109
7.24.1 Detailed Description . . . . .	110
7.24.2 Function Documentation . . . . .	110
7.24.2.1 inv_icm20603_all_sensors_off(struct inv_icm20603 *s) . . . . .	110
7.24.2.2 inv_icm20603_check_drdy(struct inv_icm20603 *s, uint8_t int_status) . . . . .	110
7.24.2.3 inv_icm20603_check_wom_status(struct inv_icm20603 *s, uint8_t int_status) . . . . .	111
7.24.2.4 inv_icm20603_configure_accel_wom(struct inv_icm20603 *s, uint8_t wom_↵ threshold) . . . . .	111
7.24.2.5 inv_icm20603_disable_fifo(struct inv_icm20603 *s) . . . . .	111
7.24.2.6 inv_icm20603_enable_fifo(struct inv_icm20603 *s, int bit_mask) . . . . .	111
7.24.2.7 inv_icm20603_enable_mems(struct inv_icm20603 *s, int bit_mask, uint16_↵ t smplr_t_divider) . . . . .	112
7.24.2.8 inv_icm20603_enable_sensor(struct inv_icm20603 *s, enum inv_icm20603_↵ sensor sensor, uint8_t enable) . . . . .	112
7.24.2.9 inv_icm20603_get_int_status(struct inv_icm20603 *s, uint8_t *int_status) . . . . .	112
7.24.2.10 inv_icm20603_get_st_bias(struct inv_icm20603 *s, int *st_bias) . . . . .	113
7.24.2.11 inv_icm20603_has_data_ready(struct inv_icm20603 *s) . . . . .	113

7.24.2.12	<code>inv_icm20603_is_sensor_enabled(struct inv_icm20603 *s, enum inv_icm20603_sensor sensor)</code>	113
7.24.2.13	<code>inv_icm20603_poll_delay_count(struct inv_icm20603 *s, int16_t *delay_count)</code>	113
7.24.2.14	<code>inv_icm20603_poll_fifo_data(struct inv_icm20603 *s, struct inv_icm20603_fifo_data *states, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3], int16_t compass_data[3])</code>	114
7.24.2.15	<code>inv_icm20603_poll_fifo_data_setup(struct inv_icm20603 *s, struct inv_icm20603_fifo_states *states, uint8_t int_status)</code>	114
7.24.2.16	<code>inv_icm20603_poll_ois_gyro_data(struct inv_icm20603 *s, int16_t gyro_data[3])</code>	114
7.24.2.17	<code>inv_icm20603_poll_sensor_data_reg(struct inv_icm20603 *s, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3])</code>	114
7.24.2.18	<code>inv_icm20603_reset_fifo(struct inv_icm20603 *s)</code>	115
7.24.2.19	<code>inv_icm20603_run_selftest(struct inv_icm20603 *s)</code>	115
7.24.2.20	<code>inv_icm20603_set_sensor_period(struct inv_icm20603 *s, enum inv_icm20603_sensor sensor, uint16_t delayInMs)</code>	115
7.24.2.21	<code>inv_icm20603_set_slave_compass_id(struct inv_icm20603 *s)</code>	116
7.24.2.22	<code>inv_icm20603_set_st_bias(struct inv_icm20603 *s, int *st_bias)</code>	116
7.25	Icm20603 driver serif	117
7.25.1	Detailed Description	117
7.26	Icm20603 driver setup	118
7.26.1	Detailed Description	119
7.26.2	Function Documentation	119
7.26.2.1	<code>inv_icm20603_accel_fsr_2_reg(int32_t fsr)</code>	119
7.26.2.2	<code>inv_icm20603_get_accel_bandwidth(struct inv_icm20603 *s)</code>	119
7.26.2.3	<code>inv_icm20603_get_accel_fullscale(struct inv_icm20603 *s)</code>	120
7.26.2.4	<code>inv_icm20603_get_accel_ois_fullscale(struct inv_icm20603 *s)</code>	120
7.26.2.5	<code>inv_icm20603_get_chip_base_sample_rate(struct inv_icm20603 *s)</code>	120
7.26.2.6	<code>inv_icm20603_get_chip_info(struct inv_icm20603 *s, uint8_t chip_info[3])</code>	120
7.26.2.7	<code>inv_icm20603_get_chip_power_state(struct inv_icm20603 *s)</code>	120
7.26.2.8	<code>inv_icm20603_get_gyro_bandwidth(struct inv_icm20603 *s)</code>	121
7.26.2.9	<code>inv_icm20603_get_gyro_fullscale(struct inv_icm20603 *s)</code>	121
7.26.2.10	<code>inv_icm20603_get_gyro_ois_fullscale(struct inv_icm20603 *s)</code>	121
7.26.2.11	<code>inv_icm20603_get_whoami(struct inv_icm20603 *s, uint8_t *whoami)</code>	121

7.26.2.12	<code>inv_icm20603_gyro_fsr_2_reg(int32_t fsr)</code> . . . . .	121
7.26.2.13	<code>inv_icm20603_initialize(struct inv_icm20603 *s)</code> . . . . .	122
7.26.2.14	<code>inv_icm20603_is_advanced_features_supported(void)</code> . . . . .	122
7.26.2.15	<code>inv_icm20603_reg_2_accel_fsr(uint8_t reg)</code> . . . . .	122
7.26.2.16	<code>inv_icm20603_reg_2_gyro_fsr(uint8_t reg)</code> . . . . .	122
7.26.2.17	<code>inv_icm20603_set_accel_bandwidth(struct inv_icm20603 *s, int level)</code> . . . . .	122
7.26.2.18	<code>inv_icm20603_set_accel_fullscale(struct inv_icm20603 *s, int level)</code> . . . . .	123
7.26.2.19	<code>inv_icm20603_set_accel_ois_fullscale(struct inv_icm20603 *s, int level)</code> . . . . .	123
7.26.2.20	<code>inv_icm20603_set_chip_power_state(struct inv_icm20603 *s, uint8_t func, uint8_t on_off)</code> . . . . .	123
7.26.2.21	<code>inv_icm20603_set_divider(struct inv_icm20603 *s, uint8_t idiv)</code> . . . . .	123
7.26.2.22	<code>inv_icm20603_set_fsyc_bit_location(struct inv_icm20603 *s, int bit_location)</code> . . . . .	124
7.26.2.23	<code>inv_icm20603_set_gyro_bandwidth(struct inv_icm20603 *s, int level)</code> . . . . .	124
7.26.2.24	<code>inv_icm20603_set_gyro_fullscale(struct inv_icm20603 *s, int level)</code> . . . . .	124
7.26.2.25	<code>inv_icm20603_set_gyro_ois_fullscale(struct inv_icm20603 *s, int level)</code> . . . . .	124
7.26.2.26	<code>inv_icm20603_soft_reset(struct inv_icm20603 *s)</code> . . . . .	125
7.27	Icm20603 driver transport . . . . .	126
7.27.1	Detailed Description . . . . .	126
7.27.2	Function Documentation . . . . .	126
7.27.2.1	<code>inv_icm20603_mems_read_reg(struct inv_icm20603 *s, uint16_t reg, uint32_t ← length, uint8_t *data)</code> . . . . .	126
7.27.2.2	<code>inv_icm20603_mems_write_reg(struct inv_icm20603 *s, uint16_t reg, uint32_t ← length, const uint8_t *data)</code> . . . . .	127
7.27.2.3	<code>inv_icm20603_mems_write_reg_one(struct inv_icm20603 *s, uint16_t reg, uint8_t data)</code> . . . . .	127
7.28	Icm20690 driver . . . . .	128
7.28.1	Detailed Description . . . . .	129
7.28.2	Function Documentation . . . . .	129
7.28.2.1	<code>inv_icm20690_get_dataready_interrupt_time_us(void)</code> . . . . .	129
7.28.2.2	<code>inv_icm20690_get_time_us(void)</code> . . . . .	129
7.28.2.3	<code>inv_icm20690_reset_states(struct inv_icm20690 *s, const struct inv_icm20690 ← _serif *serif)</code> . . . . .	129

7.28.2.4	<code>inv_icm20690_reset_states_serif_ois(struct inv_icm20690 *s, const struct inv_icm20690_serif *serif_ois)</code>	130
7.28.2.5	<code>inv_icm20690_sleep(int ms)</code>	130
7.28.2.6	<code>inv_icm20690_sleep_us(int us)</code>	130
7.29	Icm20690 akm compass support	131
7.29.1	Detailed Description	131
7.29.2	Enumeration Type Documentation	132
7.29.2.1	<code>inv_icm20690_compass_id</code>	132
7.29.3	Function Documentation	132
7.29.3.1	<code>inv_icm20690_check_akm_self_test(struct inv_icm20690 *s)</code>	132
7.29.3.2	<code>inv_icm20690_compass_getstate(struct inv_icm20690 *s)</code>	132
7.29.3.3	<code>inv_icm20690_get_compass_bytes(struct inv_icm20690 *s)</code>	132
7.29.3.4	<code>inv_icm20690_get_compass_data(struct inv_icm20690 *s, const unsigned char *packet, unsigned char *raw_compass)</code>	132
7.29.3.5	<code>inv_icm20690_read_akm_scale(struct inv_icm20690 *s, int *scale)</code>	133
7.29.3.6	<code>inv_icm20690_register_aux_compass(struct inv_icm20690 *s, enum inv_icm20690_compass_id compass_id, uint8_t compass_i2c_addr)</code>	133
7.29.3.7	<code>inv_icm20690_resume_akm(struct inv_icm20690 *s)</code>	133
7.29.3.8	<code>inv_icm20690_setup_compass_akm(struct inv_icm20690 *s)</code>	134
7.29.3.9	<code>inv_icm20690_suspend_akm(struct inv_icm20690 *s)</code>	134
7.30	Icm20690 secondary driver transport	135
7.30.1	Detailed Description	135
7.30.2	Macro Definition Documentation	136
7.30.2.1	<code>COMPASS_I2C_SLV_READ</code>	136
7.30.3	Function Documentation	136
7.30.3.1	<code>inv_icm20690_execute_read_secondary(struct inv_icm20690 *s, int index, unsigned char addr, int reg, int len, unsigned char *d)</code>	136
7.30.3.2	<code>inv_icm20690_execute_write_secondary(struct inv_icm20690 *s, int index, unsigned char addr, int reg, unsigned char v)</code>	136
7.30.3.3	<code>inv_icm20690_read_secondary(struct inv_icm20690 *s, int index, unsigned char addr, unsigned char reg, char len)</code>	137
7.30.3.4	<code>inv_icm20690_secondary_disable_i2c(struct inv_icm20690 *s)</code>	137
7.30.3.5	<code>inv_icm20690_secondary_enable_i2c(struct inv_icm20690 *s)</code>	137



7.30.3.6	<code>inv_icm20690_secondary_stop_channel(struct inv_icm20690 *s, int index)</code>	137
7.30.3.7	<code>inv_icm20690_write_secondary(struct inv_icm20690 *s, int index, unsigned char addr, unsigned char reg, char v)</code>	138
7.31	Icm20690 control	139
7.31.1	Detailed Description	140
7.31.2	Function Documentation	140
7.31.2.1	<code>inv_icm20690_all_sensors_off(struct inv_icm20690 *s)</code>	140
7.31.2.2	<code>inv_icm20690_check_drdy(struct inv_icm20690 *s, uint8_t int_status)</code>	140
7.31.2.3	<code>inv_icm20690_check_wom_status(struct inv_icm20690 *s, uint8_t int_status)</code>	141
7.31.2.4	<code>inv_icm20690_configure_accel_wom(struct inv_icm20690 *s, uint8_t wom_threshold)</code>	141
7.31.2.5	<code>inv_icm20690_disable_fifo(struct inv_icm20690 *s)</code>	141
7.31.2.6	<code>inv_icm20690_enable_fifo(struct inv_icm20690 *s, int bit_mask)</code>	141
7.31.2.7	<code>inv_icm20690_enable_mems(struct inv_icm20690 *s, int bit_mask, uint16_t smplr_divider)</code>	142
7.31.2.8	<code>inv_icm20690_enable_sensor(struct inv_icm20690 *s, enum inv_icm20690_sensor sensor, uint8_t enable)</code>	142
7.31.2.9	<code>inv_icm20690_get_int_status(struct inv_icm20690 *s, uint8_t *int_status)</code>	142
7.31.2.10	<code>inv_icm20690_get_st_bias(struct inv_icm20690 *s, int *st_bias)</code>	143
7.31.2.11	<code>inv_icm20690_has_data_ready(struct inv_icm20690 *s)</code>	143
7.31.2.12	<code>inv_icm20690_is_sensor_enabled(struct inv_icm20690 *s, enum inv_icm20690_sensor sensor)</code>	143
7.31.2.13	<code>inv_icm20690_poll_delay_count(struct inv_icm20690 *s, int16_t *delay_count)</code>	143
7.31.2.14	<code>inv_icm20690_poll_fifo_data(struct inv_icm20690 *s, struct inv_icm20690_fifo_states *states, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3], int16_t compass_data[3])</code>	144
7.31.2.15	<code>inv_icm20690_poll_fifo_data_setup(struct inv_icm20690 *s, struct inv_icm20690_fifo_states *states, uint8_t int_status)</code>	144
7.31.2.16	<code>inv_icm20690_poll_ois_gyro_data(struct inv_icm20690 *s, int16_t gyro_data[3])</code>	144
7.31.2.17	<code>inv_icm20690_poll_sensor_data_reg(struct inv_icm20690 *s, int16_t acc_data[3], int16_t *temp_data, int16_t gyro_data[3])</code>	144
7.31.2.18	<code>inv_icm20690_reset_fifo(struct inv_icm20690 *s)</code>	145
7.31.2.19	<code>inv_icm20690_run_selftest(struct inv_icm20690 *s)</code>	145
7.31.2.20	<code>inv_icm20690_set_sensor_period(struct inv_icm20690 *s, enum inv_icm20690_sensor sensor, uint16_t delayInMs)</code>	145

7.31.2.21 <code>inv_icm20690_set_slave_compass_id(struct inv_icm20690 *s)</code> . . . . .	146
7.31.2.22 <code>inv_icm20690_set_st_bias(struct inv_icm20690 *s, int *st_bias)</code> . . . . .	146
7.32 Icm20690 driver serif . . . . .	147
7.32.1 Detailed Description . . . . .	147
7.33 Icm20690 driver setup . . . . .	148
7.33.1 Detailed Description . . . . .	149
7.33.2 Function Documentation . . . . .	149
7.33.2.1 <code>inv_icm20690_accel_fsr_2_reg(int32_t fsr)</code> . . . . .	149
7.33.2.2 <code>inv_icm20690_get_accel_bandwidth(struct inv_icm20690 *s)</code> . . . . .	149
7.33.2.3 <code>inv_icm20690_get_accel_fullscale(struct inv_icm20690 *s)</code> . . . . .	150
7.33.2.4 <code>inv_icm20690_get_accel_ois_fullscale(struct inv_icm20690 *s)</code> . . . . .	150
7.33.2.5 <code>inv_icm20690_get_chip_base_sample_rate(struct inv_icm20690 *s)</code> . . . . .	150
7.33.2.6 <code>inv_icm20690_get_chip_info(struct inv_icm20690 *s, uint8_t chip_info[3])</code> . . . . .	150
7.33.2.7 <code>inv_icm20690_get_chip_power_state(struct inv_icm20690 *s)</code> . . . . .	150
7.33.2.8 <code>inv_icm20690_get_gyro_bandwidth(struct inv_icm20690 *s)</code> . . . . .	151
7.33.2.9 <code>inv_icm20690_get_gyro_fullscale(struct inv_icm20690 *s)</code> . . . . .	151
7.33.2.10 <code>inv_icm20690_get_gyro_ois_fullscale(struct inv_icm20690 *s)</code> . . . . .	151
7.33.2.11 <code>inv_icm20690_get_whoami(struct inv_icm20690 *s, uint8_t *whoami)</code> . . . . .	151
7.33.2.12 <code>inv_icm20690_gyro_fsr_2_reg(int32_t fsr)</code> . . . . .	151
7.33.2.13 <code>inv_icm20690_initialize(struct inv_icm20690 *s)</code> . . . . .	152
7.33.2.14 <code>inv_icm20690_is_advanced_features_supported(void)</code> . . . . .	152
7.33.2.15 <code>inv_icm20690_reg_2_accel_fsr(uint8_t reg)</code> . . . . .	152
7.33.2.16 <code>inv_icm20690_reg_2_gyro_fsr(uint8_t reg)</code> . . . . .	152
7.33.2.17 <code>inv_icm20690_set_accel_bandwidth(struct inv_icm20690 *s, int level)</code> . . . . .	152
7.33.2.18 <code>inv_icm20690_set_accel_fullscale(struct inv_icm20690 *s, int level)</code> . . . . .	153
7.33.2.19 <code>inv_icm20690_set_accel_ois_fullscale(struct inv_icm20690 *s, int level)</code> . . . . .	153
7.33.2.20 <code>inv_icm20690_set_chip_power_state(struct inv_icm20690 *s, uint8_t func, uint8_t on_off)</code> . . . . .	153
7.33.2.21 <code>inv_icm20690_set_divider(struct inv_icm20690 *s, uint8_t idiv)</code> . . . . .	153
7.33.2.22 <code>inv_icm20690_set_fsnc_bit_location(struct inv_icm20690 *s, int bit_location)</code> . . . . .	154
7.33.2.23 <code>inv_icm20690_set_gyro_bandwidth(struct inv_icm20690 *s, int level)</code> . . . . .	154

7.33.2.24	<code>inv_icm20690_set_gyro_fullscale(struct inv_icm20690 *s, int level)</code>	154
7.33.2.25	<code>inv_icm20690_set_gyro_ois_fullscale(struct inv_icm20690 *s, int level)</code>	154
7.33.2.26	<code>inv_icm20690_soft_reset(struct inv_icm20690 *s)</code>	155
7.34	Icm20690 driver transport	156
7.34.1	Detailed Description	156
7.34.2	Function Documentation	156
7.34.2.1	<code>inv_icm20690_mems_read_reg(struct inv_icm20690 *s, uint16_t reg, uint32_t length, uint8_t *data)</code>	156
7.34.2.2	<code>inv_icm20690_mems_write_reg(struct inv_icm20690 *s, uint16_t reg, uint32_t length, const uint8_t *data)</code>	157
7.34.2.3	<code>inv_icm20690_mems_write_reg_one(struct inv_icm20690 *s, uint16_t reg, uint8_t data)</code>	157
7.35	Drivers	158
7.35.1	Detailed Description	158
7.36	Utils	159
<b>8</b>	<b>Class Documentation</b>	<b>161</b>
8.1	<code>inv_device</code> Struct Reference	161
8.1.1	Detailed Description	162
8.2	<code>inv_device_emd_wrap_icm20xxx</code> Struct Reference	162
8.3	<code>inv_device_emd_wrap_icm20xxx_serial</code> Struct Reference	163
8.4	<code>inv_device_icm20602</code> Struct Reference	163
8.4.1	Detailed Description	163
8.5	<code>inv_device_icm20602_config_bias_st</code> Struct Reference	163
8.5.1	Detailed Description	164
8.6	<code>inv_device_icm20603</code> Struct Reference	164
8.6.1	Detailed Description	164
8.7	<code>inv_device_icm20603_config_bias_st</code> Struct Reference	165
8.7.1	Detailed Description	165
8.8	<code>inv_device_icm20690</code> Struct Reference	165
8.8.1	Detailed Description	166
8.9	<code>inv_device_icm20690_config_bias_st</code> Struct Reference	166

8.9.1 Detailed Description . . . . .	166
8.10 inv_device_smart_motion Struct Reference . . . . .	167
8.10.1 Detailed Description . . . . .	168
8.11 inv_device_smart_motion_vlistener Struct Reference . . . . .	168
8.11.1 Detailed Description . . . . .	168
8.12 inv_device_smart_motion_vsensor Struct Reference . . . . .	168
8.12.1 Detailed Description . . . . .	169
8.13 inv_device_vt Struct Reference . . . . .	169
8.13.1 Detailed Description . . . . .	169
8.14 inv_fw_version Struct Reference . . . . .	169
8.14.1 Detailed Description . . . . .	169
8.15 inv_host_serif Struct Reference . . . . .	170
8.15.1 Detailed Description . . . . .	170
8.15.2 Member Data Documentation . . . . .	170
8.15.2.1 close . . . . .	170
8.15.2.2 open . . . . .	170
8.15.2.3 read_reg . . . . .	170
8.15.2.4 register_interrupt_callback . . . . .	171
8.15.2.5 write_reg . . . . .	171
8.16 inv_icm20602 Struct Reference . . . . .	171
8.16.1 Detailed Description . . . . .	172
8.17 inv_icm20602_fifo_states Struct Reference . . . . .	172
8.17.1 Detailed Description . . . . .	172
8.18 inv_icm20602::inv_icm20602_secondary_states::inv_icm20602_secondary_reg Struct Reference . . . . .	173
8.18.1 Detailed Description . . . . .	173
8.19 inv_icm20602::inv_icm20602_secondary_states Struct Reference . . . . .	173
8.19.1 Detailed Description . . . . .	174
8.20 inv_icm20602_serif Struct Reference . . . . .	174
8.20.1 Detailed Description . . . . .	174
8.21 inv_icm20602::inv_icm20602_states Struct Reference . . . . .	174

8.21.1 Detailed Description . . . . .	174
8.22 inv_icm20603 Struct Reference . . . . .	175
8.22.1 Detailed Description . . . . .	175
8.23 inv_icm20603_fifo_states Struct Reference . . . . .	175
8.23.1 Detailed Description . . . . .	176
8.24 inv_icm20603::inv_icm20603_secondary_states::inv_icm20603_secondary_reg Struct Reference . . . . .	176
8.24.1 Detailed Description . . . . .	176
8.25 inv_icm20603::inv_icm20603_secondary_states Struct Reference . . . . .	176
8.25.1 Detailed Description . . . . .	177
8.26 inv_icm20603_serif Struct Reference . . . . .	177
8.26.1 Detailed Description . . . . .	177
8.27 inv_icm20603::inv_icm20603_states Struct Reference . . . . .	177
8.27.1 Detailed Description . . . . .	177
8.28 inv_icm20690 Struct Reference . . . . .	178
8.28.1 Detailed Description . . . . .	178
8.29 inv_icm20690_fifo_states Struct Reference . . . . .	178
8.29.1 Detailed Description . . . . .	179
8.30 inv_icm20690::inv_icm20690_secondary_states::inv_icm20690_secondary_reg Struct Reference . . . . .	179
8.30.1 Detailed Description . . . . .	179
8.31 inv_icm20690::inv_icm20690_secondary_states Struct Reference . . . . .	179
8.31.1 Detailed Description . . . . .	180
8.32 inv_icm20690_serif Struct Reference . . . . .	180
8.32.1 Detailed Description . . . . .	180
8.33 inv_icm20690::inv_icm20690_states Struct Reference . . . . .	180
8.33.1 Detailed Description . . . . .	180
8.34 inv_sensor_config_bac Struct Reference . . . . .	180
8.34.1 Detailed Description . . . . .	180
8.35 inv_sensor_config_BSCD Struct Reference . . . . .	181
8.35.1 Detailed Description . . . . .	181
8.36 inv_sensor_config_context Struct Reference . . . . .	181

8.36.1 Detailed Description . . . . .	181
8.37 inv_sensor_config_distance Struct Reference . . . . .	182
8.37.1 Detailed Description . . . . .	182
8.38 inv_sensor_config_double_tap Struct Reference . . . . .	182
8.38.1 Detailed Description . . . . .	182
8.39 inv_sensor_config_energy_expenditure Struct Reference . . . . .	182
8.39.1 Detailed Description . . . . .	183
8.40 inv_sensor_config_fsr Struct Reference . . . . .	183
8.40.1 Detailed Description . . . . .	183
8.41 inv_sensor_config_gain Struct Reference . . . . .	183
8.41.1 Detailed Description . . . . .	183
8.42 inv_sensor_config_mounting_mtx Struct Reference . . . . .	184
8.42.1 Detailed Description . . . . .	184
8.43 inv_sensor_config_offset Struct Reference . . . . .	184
8.43.1 Detailed Description . . . . .	184
8.44 inv_sensor_config_powermode Struct Reference . . . . .	184
8.44.1 Detailed Description . . . . .	185
8.45 inv_sensor_config_shake_wrist Struct Reference . . . . .	185
8.45.1 Detailed Description . . . . .	185
8.46 inv_sensor_config_stepc Struct Reference . . . . .	185
8.46.1 Detailed Description . . . . .	185
8.47 inv_sensor_event Struct Reference . . . . .	186
8.47.1 Detailed Description . . . . .	191
8.47.2 Member Data Documentation . . . . .	191
8.47.2.1 accuracy_flag . . . . .	191
8.47.2.2 audio_buffer . . . . .	191
8.47.2.3 bias . . . . .	191
8.47.2.4 cumulativeEEkcal . . . . .	191
8.47.2.5 cumulativeEEemets . . . . .	191
8.47.2.6 eis . . . . .	192
8.47.2.7 event . . . . .	192
8.47.2.8 floorsDown . . . . .	192
8.47.2.9 floorsUp . . . . .	192
8.47.2.10 hrm . . . . .	192
8.47.2.11 instantEEkcal . . . . .	192
8.47.2.12 instantEEemets . . . . .	192
8.47.2.13 touch_status . . . . .	193
8.48 inv_sensor_listener Struct Reference . . . . .	193
8.48.1 Detailed Description . . . . .	193

<b>9 Example Documentation</b>	<b>195</b>
9.1 ExampleDeviceIcm20602EMD.c . . . . .	195
9.2 ExampleDeviceIcm20603EMD.c . . . . .	197
9.3 ExampleDeviceIcm20690EMD.c . . . . .	200
9.4 ExampleSerifHal.c . . . . .	202
<b>Index</b>	<b>205</b>





# Chapter 1

## Main Page

### 1.1 Introduction

The InvenSense Device Driver library (aka libIDD) is dedicated to provide a unified API to control and retrieve data from InvenSense devices.

The library is composed of several layers which abstract specific features (protocol communication, registers access, ...) to the end user.

This library intends to be:

- Easy to use
- Highly modular
- C99 ANSI compliant

See the [Supported devices](#) page for the exhaustive list of supported device by libIDD.

### 1.2 Architecture overview

The main purpose of libIDD is to make any InvenSense devices easy to use, by mean of an abstract interface.

libIDD was designed to be used in embedded context, hence any system specific facility (SPI, I2C, delay, IRQ, ...) are abstracted. This also ease integration and portability.

The main API to be called from the application, in order to access to an InvenSense device is the [Device](#) API. The concrete implementation for the Device will call the associated low-level device driver.

Device and driver implementation may require to access to low-level system ressources and HW buses. This is achieved by mean of the [Host Serial Interface](#). User is in charge to implement it (see ExampleHostSerif.c) depending on system capability and device interface.

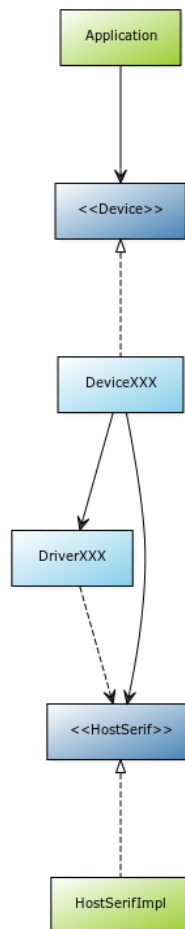


Figure 1.1 High level class diagram of libIDD

Legend:

- Green: user code
- Dark blue: abstract interface
- Light blue: specific device implementation

## 1.3 Importing and building libIDD

### 1.3.1 Requested library components

The library is delivered as sources. The directory structure should be preserved when imported into user project.

If only one device support is required, only files or directory with the device reference in it must be imported, in addition to the following files:

- `Invn/IDDVersion.h`
- `Invn/InvBool.h`

- [Invn/InvError.h](#)
- [Invn/Devices/Device.h](#)
- [Invn/Devices/HostSerif.c](#)
- [Invn/Devices/HostSerif.h](#)
- [Invn/Devices/Sensor.c](#)
- [Invn/Devices/SensorTypes.h](#)
- [Invn/EmbUtils/DataConverter.c](#)
- [Invn/EmbUtils/DataConverter.h](#)
- [Invn/EmbUtils/ErrorHelper.c](#)
- [Invn/EmbUtils/ErrorHelper.h](#)
- [Invn/EmbUtils/Message.c](#)
- [Invn/EmbUtils/Message.h](#)

### 1.3.2 Building the library

libIDD is coded in plain ANSI C99 without any uses of compiler extension. libIDD was built under gcc, vs2010, linaro for ARM and IAR for ARM.

#### Warning

libIDD uses 64bits integer and was not compiled for 8bits or 16bits architecture.

### 1.3.3 About debugging

Some library modules can output traces using the [Message](#) facility. Refer to dedicated page for help on how to use or disable this feature.



## Chapter 2

# Integration Guide

The library is using a serial interface instance to manage communication between targeted MCU and your InvenSense device.

The main goal will be to address common communication functionalities.

### 2.1 Adapter

The first step is to create the adapter (also referred to as HostSerifImpl in the high level class diagram) . This file will describe how to connect to your specific hardware abstraction layer.

Please refer to ExampleHostSerif.c for a complete example of this file.

4 functions must be defined here :

```
int my_adapter_open(void) //Implementation of SPI/I2c open
{
    return Serial_open_low_level_driver()
}

int my_adapter_close(void) //Implementation of SPI/I2c close
{
    return Serial_close_low_level_driver()
}

int my_adapter_read_reg(uint8_t reg, uint8_t * rbuffer, uint32_t rlen) //Implementation of SPI/I2c read
{
    return Serial_read_low_level_driver(reg, rbuffer, rlen);
}

int my_adapter_write_reg(uint8_t reg, const uint8_t * wbuffer, uint32_t wlen) //Implementation of SPI/I2c
write
{
    return Serial_write_low_level_driver(reg, wbuffer, wlen);
}
```

For advanced usage, more functions could be implemented such as an interrupt callback. This won't be covered in this guide.

## 2.2 Application

Once the Adapter is ready, we can start building the application.

Please refer to these files to have more complete example :

- [ExampleDeviceIcm20602EMD.c](#)
- [ExampleDeviceIcm20603EMD.c](#)
- [ExampleDeviceIcm20690EMD.c](#)

The application requires a instance of type `inv_host_serif_t` containing pointer to functions defined in the Adapter. In addition to the functions, this instance should contain :

- `max_transaction_size` : Hardware dependent value defining how many bytes are allowed per serial transaction
- `serif_type` : to be chosen amongst the `inv_host_serif_type` enumeration

```
// definition of the instance
static const inv_host_serif_t my_serif_instance = {
    my_adapter_open,
    my_adapter_close,
    my_adapter_read_reg,
    my_adapter_write_reg,
    MY_ADAPTER_SERIF_MAX_TRANSACTION_SIZE,
    MY_ADAPTER_SERIF_TYPE,
};
```

The structure being defined, it can now be used from the main to communicate with your InvenSense device. It then your responsibility to open your serial interface at the very beginning and close it at the end depending on your needs (it might not be necessary to do it for your hardware) :

```
int main(void)
{
    rc = inv_host_serif_open(&my_serif_instance);

    // Your code here

    rc = inv_host_serif_close(&my_serif_instance);
}
```

You can then implement whatever you need using the LibIDD API as described in the example files.

## Chapter 3

# Supported devices

This page lists all devices supported by libIDD.

Two Base-Sensor devices are also supported:

- The ICM20602, a 6-axis MEMS able to report raw accelerometer and raw gyroscope data.
- The ICM20603, the next generation of 6-axis MEMS able to report raw accelerometer and raw gyroscope data
- The ICM20690, an advanced MEMS able to report raw accelerometer and raw gyroscope data and is also designed to support OIS and EIS feature.

The SmartMotion device is a virtual device that connects to other IDD devices and emulates missing motion features using software libraries.

### 3.1 ICM20602

The ICM20602 gives you access to raw accelerometer and raw gyroscope sensor. This device can be connected to the SmartMotion devices to emulate missing motion features using software libraries.

Primary interface:

- SPI: up to 6MHz, MSB first, CPOL=CPHA=1 (mode 3)
- I2C: 400 KHz, slave @ 0x68

GPIOs interrupts pinout;

- Data interrupt : The GPIO generates a active high pulse to indicate data ready.

For an example on how to use ICM20602 device with libIDD please see [ExampleDeviceIcm20602EMD.c](#).

## 3.2 ICM20603

The ICM20603 gives you access to raw accelerometer and raw gyroscope sensor. This device can be connected to the SmartMotion devices to emulate missing motion features using software libraries.

Primary interface:

- SPI: up to 6MHz, MSB first, CPOL=CPHA=1 (mode 3)
- I2C: 400 KHz, slave @ 0x68

GPIOs interrupts pinout;

- Data interrupt : The GPIO generates a active high pulse to indicate data ready.

For an example on how to use ICM20603 device with libIDD please see [ExampleDeviceIcm20603EMD.c](#).

## 3.3 ICM20690

The ICM20690 gives you access to raw accelerometer and raw gyroscope sensor. This parts is dedicated to O↔IS support. The OIS feature allow to outputted gyroscope data to a camera module at high frequency for image stabilization. A FSYNC pin is also available to support the EIS feature and synchronize the gyroscope sampling. The ICM20690 supports auxiliary sensor such as magnetometer. This device can be connected to the SmartMotion devices to emulate missing motion features using software libraries.

Primary interface:

- SPI: up to 6MHz, MSB first, CPOL=CPHA=1 (mode 3)
- I2C: 400 KHz, slave @ 0x68

GPIOs interrupts pinout;

- Data interrupt : The GPIO generates a active high pulse to indicate data ready.

For an example on how to use ICM20690 device with libIDD please see [ExampleDeviceIcm20690EMD.c](#).

## 3.4 SmartMotion

This Device can be connected to other IDD devices and emulates missing motion features using software libraries. It uses the InvenSense VSensor framework to connect a HW device to a InvenSense CModel processing graph.



## Chapter 4

# Deprecated List

### Module **HostSerif**

Use SerifHal.h instead

### Member **inv\_device\_enable**

use inv\_device\_enable\_sensor

### Member **inv\_device\_icm20602\_init** (inv\_device\_icm20602\_t \*self, const inv\_host\_serif\_t \*serif, const inv↔\_sensor\_listener\_t \*listener)

Use innv\_device\_icm20602\_init2() instead.

### Member **inv\_device\_icm20602\_init\_serif\_ois** (inv\_device\_icm20602\_t \*self, const inv\_host\_serif\_t \*serif↔\_ois)

Use innv\_device\_icm20602\_init\_serif\_ois2() instead.

### Member **inv\_device\_icm20603\_init** (inv\_device\_icm20603\_t \*self, const inv\_host\_serif\_t \*serif, const inv↔\_sensor\_listener\_t \*listener)

Use innv\_device\_icm20603\_init2() instead.

### Member **inv\_device\_icm20603\_init\_serif\_ois** (inv\_device\_icm20603\_t \*self, const inv\_host\_serif\_t \*serif↔\_ois)

Use innv\_device\_icm20603\_init\_serif\_ois2() instead.

### Member **inv\_device\_icm20690\_init** (inv\_device\_icm20690\_t \*self, const inv\_host\_serif\_t \*serif, const inv↔\_sensor\_listener\_t \*listener)

Use innv\_device\_icm20690\_init2() instead.

### Member **inv\_device\_icm20690\_init\_serif\_ois** (inv\_device\_icm20690\_t \*self, const inv\_host\_serif\_t \*serif↔\_ois)

Use innv\_device\_icm20690\_init\_serif\_ois2() instead.

### Member **INV\_SENSOR\_TYPE\_ENERGY\_EXPANDITURE**

### Member **INV\_SENSOR\_TYPE\_GYROMETER**

### Member **INV\_SENSOR\_TYPE\_META\_DATA**

### Member **INV\_SENSOR\_TYPE\_UNCAL\_GYROMETER**



## Chapter 5

# Module Index

### 5.1 Modules

Here is a list of all modules:

Error code . . . . .	17
Device . . . . .	27
DeviceSmartMotion . . . . .	44
DeviceEmdWrapper . . . . .	49
DeviceIcm20602 . . . . .	50
DeviceIcm20603 . . . . .	53
DeviceIcm20690 . . . . .	56
Data Converter . . . . .	61
Error Helper . . . . .	63
Message . . . . .	64
Drivers . . . . .	158
Sensor types . . . . .	18
Sensor Configuration . . . . .	22
Host Serial Interface . . . . .	59
Icm20602 driver . . . . .	68
Icm20602 akm compass support . . . . .	71
Icm20602 secondary driver transport . . . . .	75
Icm20602 control . . . . .	79
Icm20602 driver serif . . . . .	87
Icm20602 driver setup . . . . .	88
Icm20602 driver transport . . . . .	96
Icm20603 driver . . . . .	98
Icm20603 akm compass support . . . . .	101
Icm20603 secondary driver transport . . . . .	105
Icm20603 control . . . . .	109
Icm20603 driver serif . . . . .	117
Icm20603 driver setup . . . . .	118
Icm20603 driver transport . . . . .	126
Icm20690 driver . . . . .	128
Icm20690 akm compass support . . . . .	131
Icm20690 secondary driver transport . . . . .	135
Icm20690 control . . . . .	139
Icm20690 driver serif . . . . .	147
Icm20690 driver setup . . . . .	148
Icm20690 driver transport . . . . .	156
Utils . . . . .	159



## Chapter 6

# Class Index

### 6.1 Class List

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

<a href="#">inv_device</a>	
Abstract device object definition	161
<a href="#">inv_device_emd_wrap_icm20xxx</a>	162
<a href="#">inv_device_emd_wrap_icm20xxx_serial</a>	163
<a href="#">inv_device_icm20602</a>	
States for Icm20602 device	163
<a href="#">inv_device_icm20602_config_bias_st</a>	
Bias collected during self-test (config INV_DEVICE_ICM20602_CONFIG_BIAS_ST) Value is scaled by $2^{16}$ , accel is gee and gyro is dps	163
<a href="#">inv_device_icm20603</a>	
States for Icm20603 device	164
<a href="#">inv_device_icm20603_config_bias_st</a>	
Bias collected during self-test (config INV_DEVICE_ICM20603_CONFIG_BIAS_ST) Value is scaled by $2^{16}$ , accel is gee and gyro is dps	165
<a href="#">inv_device_icm20690</a>	
States for Icm20690 device	165
<a href="#">inv_device_icm20690_config_bias_st</a>	
Bias collected during self-test (config INV_DEVICE_ICM20690_CONFIG_BIAS_ST) Value is scaled by $2^{16}$ , accel is gee and gyro is dps	166
<a href="#">inv_device_smart_motion</a>	
States for SmartMotion device implementation	167
<a href="#">inv_device_smart_motion_vlistener</a>	
Internal definition for graph leaves (based on VSensorListener)	168
<a href="#">inv_device_smart_motion_vsensor</a>	
Internal definition for graph roots (based on VSensor)	168
<a href="#">inv_device_vt</a>	
Device virtual table definition	169
<a href="#">inv_fw_version</a>	
FW version structure definition	169
<a href="#">inv_host_serif</a>	
Serial Interface interface definition	170
<a href="#">inv_icm20602</a>	
Icm20602 driver states definition	171
<a href="#">inv_icm20602_fifo_states</a>	
Check and retrieve for new data	172

<a href="#">inv_icm20602::inv_icm20602_secondary_states::inv_icm20602_secondary_reg</a>	
Secondary register mapping	173
<a href="#">inv_icm20602::inv_icm20602_secondary_states</a>	
Secondary device support	173
<a href="#">inv_icm20602_serif</a>	
Basesensor serial interface	174
<a href="#">inv_icm20602::inv_icm20602_states</a>	
Icm20602 internal state structure	174
<a href="#">inv_icm20603</a>	
Icm20603 driver states definition	175
<a href="#">inv_icm20603_fifo_states</a>	
Check and retrieve for new data	175
<a href="#">inv_icm20603::inv_icm20603_secondary_states::inv_icm20603_secondary_reg</a>	
Secondary register mapping	176
<a href="#">inv_icm20603::inv_icm20603_secondary_states</a>	
Secondary device support	176
<a href="#">inv_icm20603_serif</a>	
Basesensor serial interface	177
<a href="#">inv_icm20603::inv_icm20603_states</a>	
Icm20603 internal state structure	177
<a href="#">inv_icm20690</a>	
Icm20690 driver states definition	178
<a href="#">inv_icm20690_fifo_states</a>	
Check and retrieve for new data	178
<a href="#">inv_icm20690::inv_icm20690_secondary_states::inv_icm20690_secondary_reg</a>	
Secondary register mapping	179
<a href="#">inv_icm20690::inv_icm20690_secondary_states</a>	
Secondary device support	179
<a href="#">inv_icm20690_serif</a>	
Basesensor serial interface	180
<a href="#">inv_icm20690::inv_icm20690_states</a>	
Icm20690 internal state structure	180
<a href="#">inv_sensor_config_bac</a>	
Define the configuration for BAC	180
<a href="#">inv_sensor_config_BSCD</a>	
Define the configuration for the BSCD virtual sensor	181
<a href="#">inv_sensor_config_context</a>	
Define configuration context value (associated with INV_SENSOR_CONFIG_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implementation	181
<a href="#">inv_sensor_config_distance</a>	
Define the configuration for the distance's algorithm	182
<a href="#">inv_sensor_config_double_tap</a>	
Define the configuration for the double tap's algorithm	182
<a href="#">inv_sensor_config_energy_expenditure</a>	
Define the configuration for the energy expenditure's algorithm	182
<a href="#">inv_sensor_config_fsr</a>	
Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV_SENSOR_CONFIG_FSR config ID) Value is expected to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000	183
<a href="#">inv_sensor_config_gain</a>	
Define gain matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect	183
<a href="#">inv_sensor_config_mounting_mtx</a>	
Define mounting matrix value for 3-axis sensors (associated with INV_SENSOR_CONFIG_MOUNTING_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference	184

<a href="#">inv_sensor_config_offset</a>	
Define offset vector value for 3-axis sensors (associated with INV_SENSOR_CONFIG_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect . . . . .	184
<a href="#">inv_sensor_config_powermode</a>	
Define chip power mode (associated with INV_SENSOR_CONFIG_POWER_MODE config ID) Value is expetcted to be 0 for low power or 1 for low noise . . . . .	184
<a href="#">inv_sensor_config_shake_wrist</a>	
Define the configuration for the shake wrist's algorithm . . . . .	185
<a href="#">inv_sensor_config_stepc</a>	
Define the configuration for steps counter . . . . .	185
<a href="#">inv_sensor_event</a>	
Sensor event definition . . . . .	186
<a href="#">inv_sensor_listener</a>	
Sensor event listener definition . . . . .	193





## Chapter 7

# Module Documentation

### 7.1 Error code

Common error code.

#### Enumerations

#### 7.1.1 Detailed Description

Common error code.

#### 7.1.2 Enumeration Type Documentation

##### 7.1.2.1 enum inv\_error

Common error code definition.

#### Enumerator

**INV\_ERROR\_SUCCESS** no error

**INV\_ERROR** unspecified error

**INV\_ERROR\_NIMPL** function not implemented for given arguments

**INV\_ERROR\_TRANSPORT** error occurred at transport level

**INV\_ERROR\_TIMEOUT** action did not complete in the expected time window

**INV\_ERROR\_SIZE** size/length of given arguments is not suitable to complete requested action

**INV\_ERROR\_OS** error related to OS

**INV\_ERROR\_IO** error related to IO operation

**INV\_ERROR\_MEM** not enough memory to complete requested action

**INV\_ERROR\_HW** error at HW level

**INV\_ERROR\_BAD\_ARG** provided arguments are not good to perform request action

**INV\_ERROR\_UNEXPECTED** something unexpected happened

**INV\_ERROR\_FILE** cannot access file or unexpected format

**INV\_ERROR\_PATH** invalid file path

**INV\_ERROR\_IMAGE\_TYPE** error when image type is not managed

**INV\_ERROR\_WATCHDOG** error when device doesn't respond to ping

## 7.2 Sensor types

Sensor related types definitions.

Collaboration diagram for Sensor types:



### Classes

- struct `inv_sensor_event`  
*Sensor event definition.*
- struct `inv_sensor_listener`  
*Sensor event listener definition.*

### Macros

- `#define INV_SENSOR_TYPE_META_DATA INV_SENSOR_TYPE_RESERVED`
- `#define INV_SENSOR_TYPE_GYROMETER INV_SENSOR_TYPE_GYROSCOPE`
- `#define INV_SENSOR_TYPE_UNCAL_GYROMETER INV_SENSOR_TYPE_UNCAL_GYROSCOPE`
- `#define INV_SENSOR_TYPE_ENERGY_EXPENDITURE INV_SENSOR_TYPE_ENERGY_EXPENDITURE`
- `#define INV_SENSOR_TYPE_WU_FLAG (unsigned int)(0x80000000)`  
*Helper flag to indicate if sensor is a Wake-Up sensor.*
- `#define INV_SENSOR_EVENT_DATA_SIZE 64`  
*Maximum size of an event data.*
- `#define INV_SENSOR_EVENT_DATA_SIZE INV_SENSOR_EVENT_DATA_SIZE`  
*For backward compatibility only - do not use.*
- `#define INV_SENSOR_ID_TO_TYPE(sensor) (((unsigned int)(sensor) & ~INV_SENSOR_TYPE_WU_FLAG)`  
*Helper macro to retrieve sensor type (without wake-up flag) from a sensor id.*
- `#define INV_SENSOR_IS_VALID(sensor) (INV_SENSOR_ID_TO_TYPE(sensor) < INV_SENSOR_TYPE_MAX)`  
*Helper macro that check if given sensor is of known type.*
- `#define INV_SENSOR_IS_WU(sensor) (((int)(sensor) & INV_SENSOR_TYPE_WU_FLAG) != 0)`  
*Helper macro that check if given sensor is a wake-up sensor.*
- `#define inv_sensor_2str inv_sensor_str`  
*Alias for inv\_sensor\_str.*

## Typedefs

- typedef struct [inv\\_sensor\\_event](#) [inv\\_sensor\\_event\\_t](#)  
*Sensor event definition.*
- typedef void(\* [inv\\_sensor\\_listener\\_event\\_cb\\_t](#)) (const [inv\\_sensor\\_event\\_t](#) \*event, void \*context)  
*Sensor listener event callback definition.*
- typedef struct [inv\\_sensor\\_listener](#) [inv\\_sensor\\_listener\\_t](#)  
*Sensor event listener definition.*

## Enumerations

## Functions

- static void [inv\\_sensor\\_listener\\_init](#) ([inv\\_sensor\\_listener\\_t](#) \*listener, [inv\\_sensor\\_listener\\_event\\_cb\\_t](#) event\_cb, void \*context)  
*Helper to initialize a listener object.*
- static void [inv\\_sensor\\_listener\\_notify](#) (const [inv\\_sensor\\_listener\\_t](#) \*listener, const [inv\\_sensor\\_event\\_t](#) \*event)  
*Helper to notify a listener of a new sensor event.*
- const char INV\_EXPORT \* [inv\\_sensor\\_str](#) (int sensor)  
*Utility function that returns a string from a sensor id Empty string is returned if sensor is invalid.*

### 7.2.1 Detailed Description

Sensor related types definitions.

### 7.2.2 Macro Definition Documentation

7.2.2.1 `#define INV_SENSOR_TYPE_ENERGY_EXPENDITURE INV_SENSOR_TYPE_ENERGY_EXPENDITURE`

**Deprecated**

7.2.2.2 `#define INV_SENSOR_TYPE_GYROMETER INV_SENSOR_TYPE_GYROSCOPE`

**Deprecated**

7.2.2.3 `#define INV_SENSOR_TYPE_META_DATA INV_SENSOR_TYPE_RESERVED`

**Deprecated**

7.2.2.4 `#define INV_SENSOR_TYPE_UNCAL_GYROMETER INV_SENSOR_TYPE_UNCAL_GYROSCOPE`

**Deprecated**

### 7.2.3 Typedef Documentation

7.2.3.1 `typedef void(* inv_sensor_listener_event_cb_t) (const inv_sensor_event_t *event, void *context)`

Sensor listener event callback definition.

## Parameters

in	<i>event</i>	reference to sensor event
in	<i>context</i>	listener context

## Returns

none

## 7.2.4 Enumeration Type Documentation

### 7.2.4.1 enum inv\_sensor\_status

Sensor status definition.

## Enumerator

**INV\_SENSOR\_STATUS\_DATA\_UPDATED** new sensor data  
**INV\_SENSOR\_STATUS\_STATE\_CHANGED** dummy sensor data indicating to a change in sensor state  
**INV\_SENSOR\_STATUS\_FLUSH\_COMPLETE** dummy sensor data indicating a end of batch after a manual flush  
**INV\_SENSOR\_STATUS\_POLLED\_DATA** sensor data value after manual request

### 7.2.4.2 enum inv\_sensor\_type

Sensor type identifier definition.

## Enumerator

**INV\_SENSOR\_TYPE\_RESERVED** Reserved ID: do not use.  
**INV\_SENSOR\_TYPE\_ACCELEROMETER** Accelerometer.  
**INV\_SENSOR\_TYPE\_MAGNETOMETER** Magnetic field.  
**INV\_SENSOR\_TYPE\_ORIENTATION** Deprecated orientation.  
**INV\_SENSOR\_TYPE\_GYROSCOPE** Gyroscope.  
**INV\_SENSOR\_TYPE\_LIGHT** Ambient light sensor.  
**INV\_SENSOR\_TYPE\_PRESSURE** Barometer.  
**INV\_SENSOR\_TYPE\_TEMPERATURE** Temperature.  
**INV\_SENSOR\_TYPE\_PROXIMITY** Proximity.  
**INV\_SENSOR\_TYPE\_GRAVITY** Gravity.  
**INV\_SENSOR\_TYPE\_LINEAR\_ACCELERATION** Linear acceleration.  
**INV\_SENSOR\_TYPE\_ROTATION\_VECTOR** Rotation vector.  
**INV\_SENSOR\_TYPE\_HUMIDITY** Relative humidity.  
**INV\_SENSOR\_TYPE\_AMBIENT\_TEMPERATURE** Ambient temperature.  
**INV\_SENSOR\_TYPE\_UNCAL\_MAGNETOMETER** Uncalibrated magnetic field.  
**INV\_SENSOR\_TYPE\_GAME\_ROTATION\_VECTOR** Game rotation vector.  
**INV\_SENSOR\_TYPE\_UNCAL\_GYROSCOPE** Uncalibrated gyroscope.

**INV\_SENSOR\_TYPE\_SMD** Significant motion detection.

**INV\_SENSOR\_TYPE\_STEP\_DETECTOR** Step detector.

**INV\_SENSOR\_TYPE\_STEP\_COUNTER** Step counter.

**INV\_SENSOR\_TYPE\_GEOMAG\_ROTATION\_VECTOR** Geomagnetic rotation vector.

**INV\_SENSOR\_TYPE\_HEART\_RATE** Heart rate.

**INV\_SENSOR\_TYPE\_TILT\_DETECTOR** Tilt detector.

**INV\_SENSOR\_TYPE\_WAKE\_GESTURE** Wake-up gesture.

**INV\_SENSOR\_TYPE\_GLANCE\_GESTURE** Glance gesture.

**INV\_SENSOR\_TYPE\_PICK\_UP\_GESTURE** Pick-up gesture.

**INV\_SENSOR\_TYPE\_BAC** Basic Activity Classifier.

**INV\_SENSOR\_TYPE\_PDR** Pedestrian Dead Reckoning.

**INV\_SENSOR\_TYPE\_B2S** Bring to see.

**INV\_SENSOR\_TYPE\_3AXIS** 3 Axis sensor

**INV\_SENSOR\_TYPE\_EIS** Electronic Image Stabilization.

**INV\_SENSOR\_TYPE\_OIS** Optical Image Stabilization.

**INV\_SENSOR\_TYPE\_RAW\_ACCELEROMETER** Raw accelerometer.

**INV\_SENSOR\_TYPE\_RAW\_GYROSCOPE** Raw gyroscope.

**INV\_SENSOR\_TYPE\_RAW\_MAGNETOMETER** Raw magnetometer.

**INV\_SENSOR\_TYPE\_RAW\_TEMPERATURE** Raw temperature.

**INV\_SENSOR\_TYPE\_CUSTOM\_PRESSURE** Custom Pressure Sensor.

**INV\_SENSOR\_TYPE\_MIC** Stream audio from microphone.

**INV\_SENSOR\_TYPE\_TSIMU** TS-IMU.

**INV\_SENSOR\_TYPE\_RAW\_PPG** Raw Photoplethysmogram.

**INV\_SENSOR\_TYPE\_HRV** Heart rate variability.

**INV\_SENSOR\_TYPE\_SLEEP\_ANALYSIS** Sleep analysis.

**INV\_SENSOR\_TYPE\_BAC\_EXTENDED** Basic Activity Classifier Extended.

**INV\_SENSOR\_TYPE\_BAC\_STATISTICS** Basic Activity Classifier Statistics.

**INV\_SENSOR\_TYPE\_FLOOR\_CLIMB\_COUNTER** Floor Climbed Counter.

**INV\_SENSOR\_TYPE\_ENERGY\_EXPENDITURE** Energy Expenditure.

**INV\_SENSOR\_TYPE\_DISTANCE** Distance.

**INV\_SENSOR\_TYPE\_SHAKE** Shake Gesture.

**INV\_SENSOR\_TYPE\_DOUBLE\_TAP** Double Tap.

**INV\_SENSOR\_TYPE\_CUSTOM0** Custom sensor ID 0.

**INV\_SENSOR\_TYPE\_CUSTOM1** Custom sensor ID 1.

**INV\_SENSOR\_TYPE\_CUSTOM2** Custom sensor ID 2.

**INV\_SENSOR\_TYPE\_CUSTOM3** Custom sensor ID 3.

**INV\_SENSOR\_TYPE\_CUSTOM4** Custom sensor ID 4.

**INV\_SENSOR\_TYPE\_CUSTOM5** Custom sensor ID 5.

**INV\_SENSOR\_TYPE\_CUSTOM6** Custom sensor ID 6.

**INV\_SENSOR\_TYPE\_CUSTOM7** Custom sensor ID 7.

**INV\_SENSOR\_TYPE\_WOM** Wake-up on motion.

**INV\_SENSOR\_TYPE\_SEDENTARY\_REMIND** Sedentary Remind.

**INV\_SENSOR\_TYPE\_DATA\_ENCRYPTION** Data Encryption.

**INV\_SENSOR\_TYPE\_FSYNC\_EVENT** FSYNC event.

**INV\_SENSOR\_TYPE\_HIGH\_RATE\_GYRO** High Rate Gyro.

**INV\_SENSOR\_TYPE\_CUSTOM\_BSCD** Custom BAC StepCounter Calorie counter and Distance counter.

**INV\_SENSOR\_TYPE\_HRM\_LOGGER** HRM output for logger.

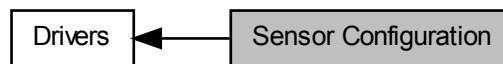
**INV\_SENSOR\_TYPE\_PREDICTIVE\_QUATERNION** Predictive Quaternion.

**INV\_SENSOR\_TYPE\_MAX** sentinel value for sensor type

## 7.3 Sensor Configuration

General sensor configuration types definitions.

Collaboration diagram for Sensor Configuration:



### Classes

- struct [inv\\_sensor\\_config\\_mounting\\_mtx](#)  
*Define mounting matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.*
- struct [inv\\_sensor\\_config\\_gain](#)  
*Define gain matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.*
- struct [inv\\_sensor\\_config\\_offset](#)  
*Define offset vector value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.*
- struct [inv\\_sensor\\_config\\_context](#)  
*Define configuration context value (associated with INV\_SENSOR\_CONFIG\_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implementation.*
- struct [inv\\_sensor\\_config\\_fsr](#)  
*Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV\_SENSOR\_CONFIG\_FSR config ID) Value is expetcted to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.*
- struct [inv\\_sensor\\_config\\_powermode](#)  
*Define chip power mode (associated with INV\_SENSOR\_CONFIG\_POWER\_MODE config ID) Value is expetcted to be 0 for low power or 1 for low noise.*
- struct [inv\\_sensor\\_config\\_energy\\_expenditure](#)  
*Define the configuration for the energy expenditure's algorithm.*
- struct [inv\\_sensor\\_config\\_distance](#)  
*Define the configuration for the distance's algorithm.*
- struct [inv\\_sensor\\_config\\_bac](#)  
*Define the configuration for BAC.*
- struct [inv\\_sensor\\_config\\_stepc](#)  
*Define the configuration for steps counter.*
- struct [inv\\_sensor\\_config\\_shake\\_wrist](#)  
*Define the configuration for the shake wrist's algorithm.*
- struct [inv\\_sensor\\_config\\_double\\_tap](#)  
*Define the configuration for the double tap's algorithm.*
- struct [inv\\_sensor\\_config\\_BSCD](#)  
*Define the configuration for the BSCD virtual sensor.*

## Typedefs

- typedef struct [inv\\_sensor\\_config\\_mounting\\_mtx](#) [inv\\_sensor\\_config\\_mounting\\_mtx\\_t](#)  
*Define mounting matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.*
- typedef struct [inv\\_sensor\\_config\\_gain](#) [inv\\_sensor\\_config\\_gain\\_t](#)  
*Define gain matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.*
- typedef struct [inv\\_sensor\\_config\\_offset](#) [inv\\_sensor\\_config\\_offset\\_t](#)  
*Define offset vector value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.*
- typedef struct [inv\\_sensor\\_config\\_context](#) [inv\\_sensor\\_config\\_context\\_t](#)  
*Define configuration context value (associated with INV\_SENSOR\_CONFIG\_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implementation.*
- typedef struct [inv\\_sensor\\_config\\_fsr](#) [inv\\_sensor\\_config\\_fsr\\_t](#)  
*Define full-scale range value for accelero, gyro or magnetometer based sensor (associated with INV\_SENSOR\_CONFIG\_FSR config ID) Value is expected to be expressed in mg, dps and uT for accelero, gyro or magnetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.*
- typedef struct [inv\\_sensor\\_config\\_powermode](#) [inv\\_sensor\\_config\\_powermode\\_t](#)  
*Define chip power mode (associated with INV\_SENSOR\_CONFIG\_POWER\_MODE config ID) Value is expected to be 0 for low power or 1 for low noise.*
- typedef struct [inv\\_sensor\\_config\\_energy\\_expenditure](#) [inv\\_sensor\\_config\\_energy\\_expenditure\\_t](#)  
*Define the configuration for the energy expenditure's algorithm.*
- typedef struct [inv\\_sensor\\_config\\_distance](#) [inv\\_sensor\\_config\\_distance\\_t](#)  
*Define the configuration for the distance's algorithm.*
- typedef struct [inv\\_sensor\\_config\\_bac](#) [inv\\_sensor\\_config\\_bac\\_t](#)  
*Define the configuration for BAC.*
- typedef struct [inv\\_sensor\\_config\\_stepc](#) [inv\\_sensor\\_config\\_stepc\\_t](#)  
*Define the configuration for steps counter.*
- typedef struct [inv\\_sensor\\_config\\_shake\\_wrist](#) [inv\\_sensor\\_config\\_shake\\_wrist\\_t](#)  
*Define the configuration for the shake wrist's algorithm.*
- typedef struct [inv\\_sensor\\_config\\_double\\_tap](#) [inv\\_sensor\\_config\\_double\\_tap\\_t](#)  
*Define the configuration for the double tap's algorithm.*
- typedef struct [inv\\_sensor\\_config\\_BSCD](#) [inv\\_sensor\\_config\\_BSCD\\_t](#)  
*Define the configuration for the BSCD virtual sensor.*

## Enumerations

### 7.3.1 Detailed Description

General sensor configuration types definitions.

### 7.3.2 Typedef Documentation

#### 7.3.2.1 typedef struct [inv\\_sensor\\_config\\_bac](#) [inv\\_sensor\\_config\\_bac\\_t](#)

Define the configuration for BAC.

## Parameters

<i>enableNotify</i>	enable disable notify
---------------------	-----------------------

7.3.2.2 `typedef struct inv_sensor_config_BSCD inv_sensor_config_BSCD_t`

Define the configuration for the BSCD virtual sensor.

## Parameters

<i>Age</i>	age in year; Range is (0;100). Default is 35.
<i>Gender</i>	gender is 0 for men, 1 for female. Default is 0
<i>Height</i>	height in centimeter; Range is (50;250). Default is 175.
<i>Weight</i>	weight in kg; Range is (3;300). Default is 75
<i>enableNotify</i>	bitmask to enable/disable notify on a a specific sensor event bit 0 (1): enable/disable notify on BAC event bit 1 (2): enable/disable notify on step counter event bit 2 (4): enable/disable notify on energy expenditure event bit 3 (8): enable/disable notify on distance event

7.3.2.3 `typedef struct inv_sensor_config_distance inv_sensor_config_distance_t`

Define the configuration for the distance's algorithm.

## Parameters

<i>user_height</i>	height of the user in cm
<i>enableNotify</i>	enable disable notify

7.3.2.4 `typedef struct inv_sensor_config_double_tap inv_sensor_config_double_tap_t`

Define the configuration for the double tap's algorithm.

## Parameters

<i>minimum_threshold</i>	This parameter sets the minimum threshold to reach in order to start a Tap detection. Default value is 2000, recommended range [500 ; 2500]
<i>t_max</i>	This parameter sets the maximum time after a Tap event in [sample]. Default value is 100, recommended range [30 ; 200].

7.3.2.5 `typedef struct inv_sensor_config_energy_expenditure inv_sensor_config_energy_expenditure_t`

Define the configuration for the energy expenditure's algorithm.



## Parameters

<i>age</i>	age in year; Range is (0;100).
<i>gender</i>	gender is 0 for men, 1 for female.
<i>height</i>	height in centimeter; Range is (50;250)
<i>weight</i>	weight in kg; Range is (3;300)
<i>enableNotify</i>	enable disable notify

## 7.3.2.6 typedef struct inv\_sensor\_config\_mounting\_mtx inv\_sensor\_config\_mounting\_mtx\_t

Define mounting matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.

Value is expected to be a rotation matrix.

## 7.3.2.7 typedef struct inv\_sensor\_config\_offset inv\_sensor\_config\_offset\_t

Define offset vector value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.

If applied to RAW sensor, value is expected to be in lsb. If applied to other sensor, value is expected to be in sensor unit (g, uT or dps).

## 7.3.2.8 typedef struct inv\_sensor\_config\_shake\_wrist inv\_sensor\_config\_shake\_wrist\_t

Define the configuration for the shake wrist's algorithm.

## Parameters

<i>max_period</i>	This parameter sets the maximal duration for half oscillation to detect a Shake wrist. The default value is 20, recommend range [15 ; 40], 15 for the lower sensitivity and 40 for the higher sensitivity. Notice that increasing the sensitivity will increase the number of false detection, and also slightly increase response time.
<i>dummy_padding</i>	Dummy byte for padding. Set it to 0.

## 7.3.2.9 typedef struct inv\_sensor\_config\_stepc inv\_sensor\_config\_stepc\_t

Define the configuration for steps counter.

## Parameters

<i>enableNotify</i>	enable disable notify
---------------------	-----------------------

### 7.3.3 Enumeration Type Documentation

#### 7.3.3.1 enum inv\_sensor\_config

Sensor type identifier definition.

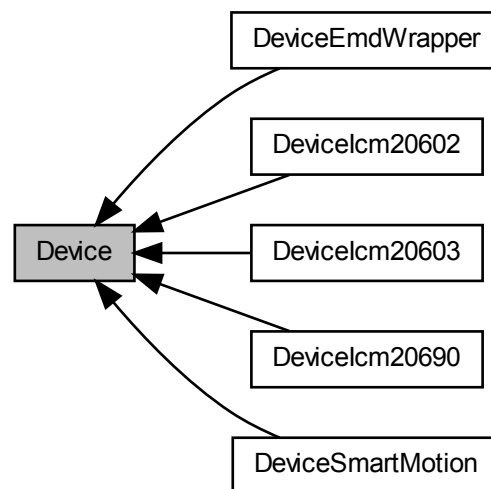
Enumerator

**INV\_SENSOR\_CONFIG\_RESERVED** Reserved config ID: do not use.  
**INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX** 3x3 mounting matrix  
**INV\_SENSOR\_CONFIG\_GAIN** 3x3 gain matrix (to correct for cross-axis defect)  
**INV\_SENSOR\_CONFIG\_OFFSET** 3d offset vector  
**INV\_SENSOR\_CONFIG\_CONTEXT** arbitrary context buffer  
**INV\_SENSOR\_CONFIG\_FSR** Full scale range.  
**INV\_SENSOR\_CONFIG\_RESET** Reset the specified service.  
**INV\_SENSOR\_CONFIG\_POWER\_MODE** Low Power or Low Noise mode.  
**INV\_SENSOR\_CONFIG\_CUSTOM** Configuration ID above this value are device specific.  
**INV\_SENSOR\_CONFIG\_MAX** Absolute maximum value for sensor config.

## 7.4 Device

Abstract device interface definition.

Collaboration diagram for Device:



### Modules

- [DeviceSmartMotion](#)  
*Device implementation that connects to other IDD devices and emulates missing motion features using software libraries.*
- [DeviceEmdWrapper](#)  
*emd wrapper implementation for device interface*
- [DeviceIcm20602](#)  
*Concrete implementation of the 'Device' interface for lcm20602 devices.*
- [DeviceIcm20603](#)  
*Concrete implementation of the 'Device' interface for lcm20603 devices.*
- [DeviceIcm20690](#)  
*Concrete implementation of the 'Device' interface for lcm20690 devices.*

### Classes

- struct [inv\\_fw\\_version](#)  
*FW version structure definition.*
- struct [inv\\_device\\_vt](#)  
*Device virtual table definition.*
- struct [inv\\_device](#)  
*Abstract device object definition.*

## Macros

- `#define inv_device_enable inv_device_enable_sensor`  
Alias of `inv_device_enable_sensor()` for backward compatibility.

## Typedefs

- typedef struct `inv_fw_version` `inv_fw_version_t`  
FW version structure definition.
- typedef struct `inv_device_vt` `inv_device_vt_t`  
Device virtual table definition.
- typedef struct `inv_device` `inv_device_t`  
Abstract device object definition.

## Functions

- static int `inv_device_whoami` (const `inv_device_t` \*dev, uint8\_t \*whoami)  
Gets WHO AM I value.
- static int `inv_device_reset` (const `inv_device_t` \*dev)  
Resets the device to a known state.
- static int `inv_device_setup` (const `inv_device_t` \*dev)  
Performs basic device initialization.
- static int `inv_device_cleanup` (const `inv_device_t` \*dev)  
Shutdown the device and clean-up internal states.
- static int `inv_device_poll` (const `inv_device_t` \*dev)  
Polls the device for data.
- static int `inv_device_load` (const `inv_device_t` \*dev, int type, const uint8\_t \*image, uint32\_t size, inv\_bool\_t verify, inv\_bool\_t force)  
Begins loading procedure for device's image(s)
- static int `inv_device_get_fw_info` (const `inv_device_t` \*dev, struct `inv_fw_version` \*version)  
Gets device FW version.
- static int `inv_device_set_running_state` (const `inv_device_t` \*dev, inv\_bool\_t state)  
Indicates to device current RUN/SUSPEND state of the host.
- static int `inv_device_ping_sensor` (const `inv_device_t` \*dev, int sensor)  
Checks if a sensor is supported by the device.
- static int `inv_device_enable_sensor` (const `inv_device_t` \*dev, int sensor, inv\_bool\_t start)  
Enable/Disable a sensor.
- static int `inv_device_start_sensor` (const `inv_device_t` \*dev, int sensor)  
Starts a sensor.
- static int `inv_device_stop_sensor` (const `inv_device_t` \*dev, int sensor)  
Stops a sensor.
- static int `inv_device_set_sensor_period_us` (const `inv_device_t` \*dev, int sensor, uint32\_t period)  
Configure sensor output data period.
- static int `inv_device_set_sensor_period` (const `inv_device_t` \*dev, int sensor, uint32\_t period)  
Configure sensor output data period.
- static int `inv_device_set_sensor_timeout` (const `inv_device_t` \*dev, int sensor, uint32\_t timeout)  
Configure sensor output timeout.
- static int `inv_device_set_sensor_timeout_us` (const `inv_device_t` \*dev, int sensor, uint32\_t timeout)  
Configure sensor output timeout.
- static int `inv_device_flush_sensor` (const `inv_device_t` \*dev, int sensor)

- Forces flush of devices's internal buffers.*
- static int `inv_device_set_sensor_bias` (const `inv_device_t` \*dev, int sensor, const float bias[3])  
*Configure bias value for a sensor.*
- static int `inv_device_get_sensor_bias` (const `inv_device_t` \*dev, int sensor, float bias[3])  
*Gets bias value for a sensor.*
- static int `inv_device_set_sensor_mounting_matrix` (const `inv_device_t` \*dev, int sensor, const float matrix[9])  
*Sets the mounting matrix information for a multi-axis sensor.*
- static int `inv_device_get_sensor_data` (const `inv_device_t` \*dev, int sensor, `inv_sensor_event_t` \*event)  
*Retrieve last known sensor event for a sensor.*
- static int `inv_device_self_test` (const `inv_device_t` \*dev, int sensor)  
*Perform self-test procedure for MEMS component of the device.*
- static int `inv_device_set_sensor_config` (const `inv_device_t` \*dev, int sensor, int settings, const void \*arg, uint16\_t size)  
*Generic method to configure a sensor.*
- static int `inv_device_get_sensor_config` (const `inv_device_t` \*dev, int sensor, int settings, void \*value, uint16\_t size)  
*Generic method to retrieve configuration value for a sensor.*
- static int `inv_device_write_mems_register` (const `inv_device_t` \*dev, int sensor, uint16\_t reg\_addr, const void \*data, uint16\_t len)  
*Set the MEMS register.*
- static int `inv_device_read_mems_register` (const `inv_device_t` \*dev, int sensor, uint16\_t reg\_addr, void \*data, uint16\_t len)  
*Read register of underlying MEMS or HW sensor.*

### 7.4.1 Detailed Description

Abstract device interface definition.

All functions declared in this file are virtual. They aim to provide a unified way of accessing InvenSense devices. All functions shall return a int for which 0 indicates success and a negative value indicates an error as described by enum `inv_error`

If a particular device implementation does not support any of the method declared here, it shall return `INV_ERROR_NIMPL`.

Implementation is not expected to be thread-safe.

Refer to concrete device implementation for additionnal and specific information about API usage related to a particular device.

### 7.4.2 Macro Definition Documentation

#### 7.4.2.1 #define `inv_device_enable inv_device_enable_sensor`

Alias of `inv_device_enable_sensor()` for backward compatibility.

**Deprecated** use `inv_device_enable_sensor`

### 7.4.3 Function Documentation

#### 7.4.3.1 static int `inv_device_cleanup` ( const `inv_device_t` \*dev ) [inline], [static]

Shutdown the device and clean-up internal states.

**Parameters**

in	<i>dev</i>	pointer to device object instance
----	------------	-----------------------------------

**Returns**

0 on success INV\_ERROR\_TIMEOUT if clean-up does not complete in time INV\_ERROR\_TRANSPORT in case of low level serial error

**Examples:**

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

**7.4.3.2** `static int inv_device_enable_sensor ( const inv_device_t * dev, int sensor, inv_bool_t start )` `[inline]`,  
`[static]`

Enable/Disable a sensor.

Send a command to start or stop a sensor. See [inv\\_device\\_start\\_sensor\(\)](#) and [inv\\_device\\_stop\\_sensor\(\)](#)

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

**See also**

`inv_sensor_type_t`)

**Parameters**

in	<i>start</i>	true to start the sensor, false to stop the sensor
----	--------------	--

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**7.4.3.3** `static int inv_device_flush_sensor ( const inv_device_t * dev, int sensor )` `[inline]`, `[static]`

Forces flush of devices's internal buffers.

Send a command a flush command to device. Device will imediatly send all sensor events that may be store in its internal buffers.

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

[inv\\_sensor\\_type\\_t](#))

## Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**7.4.3.4** `static int inv_device_get_fw_info ( const inv_device_t * dev, struct inv_fw_version * version ) [inline], [static]`

Gets device FW version.

## Parameters

in	<i>dev</i>	pointer to device object instance
out	<i>version</i>	version structure placeholder

## Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time

**7.4.3.5** `static int inv_device_get_sensor_bias ( const inv_device_t * dev, int sensor, float bias[3] ) [inline], [static]`

Gets bias value for a sensor.

Bias configuration makes sense only for few sensor types:

- INV\_SENSOR\_TYPE\_ACCELEROMETER
- INV\_SENSOR\_TYPE\_MAGNETOMETER
- INV\_SENSOR\_TYPE\_GYROSCOPE Bias unit is the same as the corresponding sensor unit.

## See also

[inv\\_sensor\\_event\\_t](#) for details.

## Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

[inv\\_sensor\\_type\\_t](#))

**Parameters**

out	<i>bias</i>	returned bias
-----	-------------	---------------

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**7.4.3.6** static int inv\_device\_get\_sensor\_config ( const inv\_device\_t \* dev, int sensor, int settings, void \* value, uint16\_t size ) [inline],[static]

Generic method to retrieve configuration value for a sensor.

For common settings, setting value is expected to be a value from

**See also**

enum [inv\\_sensor\\_config](#). Settings data is expected to point the proper type as describes in [SensorConfig.h](#)

For specific settings, refer to concrete device implementation for supported sensor and available configuration settings parameters.

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

**See also**

inv\_sensor\_type\_t)

**Parameters**

in	<i>settings</i>	settings to configure
out	<i>value</i>	the value for the corresponding setting
in	<i>size</i>	maximum buffer size pointed by value

**Returns**

0 on success >0 indicating success and the number of byte written to value INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation INV\_ERROR\_SIZE provided buffer is not big enough INV\_ERROR if configuration has failed

**7.4.3.7** static int inv\_device\_get\_sensor\_data ( const inv\_device\_t \* dev, int sensor, inv\_sensor\_event\_t \* event ) [inline],[static]

Retrieve last known sensor event for a sensor.



Depending on device capability, a call to this function may have no effect or return an error.

#### Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

#### See also

inv\_sensor\_type\_t)

#### Parameters

out	<i>event</i>	last known event data
-----	--------------	-----------------------

#### Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation INV\_ERROR if and event was received but unmanaged by the implementation

**7.4.3.8** static int inv\_device\_load ( const inv\_device\_t \* *dev*, int *type*, const uint8\_t \* *image*, uint32\_t *size*, inv\_bool\_t *verify*, inv\_bool\_t *force* ) [inline],[static]

Begins loading procedure for device's image(s)

Will start the process of loading an image to device's memory. Type of images to load will depend on the device type and/or FW version.

#### Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>type</i>	type of image to load. Can vary from one implementation to another. Refer to specific implementation for details.
in	<i>image</i>	pointer to image (or image chunk) data
in	<i>size</i>	size of image (or size of image chunk)
in	<i>verify</i>	true to perform image integrity verification, false to skip it
in	<i>force</i>	true to load image even if identical to current image, false to compare image first

#### Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_SIZE if image size does not fit in device memory

**7.4.3.9** static int inv\_device\_ping\_sensor ( const inv\_device\_t \* *dev*, int *sensor* ) [inline],[static]

Checks if a sensor is supported by the device.

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

**See also**

`inv_sensor_type_t`)

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR if sensor is not supported by the device INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**Examples:**

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

**7.4.3.10** `static int inv_device_poll ( const inv_device_t * dev ) [inline], [static]`

Polls the device for data.

Will read device interrupt registers and data registers or FIFOs. Will parse data and called sensor events handler provided at init time. Handler will be called in the same context of this function.

**Warning**

Care should be taken regarding concurrency. If this function is called in a dedicated thread, suitable protection must be used to avoid concurrent calls to poll() or any other device methods.

**Parameters**

in	<i>dev</i>	pointer to device object instance
----	------------	-----------------------------------

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_UNEXPECTED in case of bad formatted or un-handled data frame

**Examples:**

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

**7.4.3.11** `static int inv_device_read_mems_register ( const inv_device_t * dev, int sensor, uint16_t reg_addr, void * data, uint16_t len ) [inline], [static]`

Read register of underlying MEMS or HW sensor.

## Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

inv\_sensor\_type\_t)

## Parameters

in	<i>reg_addr</i>	the register that should be read
in	<i>data</i>	pointer to buffer to hold read data
in	<i>length</i>	length of data to read

## Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation INV\_ERROR\_SIZE request length is above device capability INV\_ERROR if configuration has failed

**7.4.3.12** static int inv\_device\_reset ( const inv\_device\_t \* dev ) [inline],[static]

Resets the device to a known state.

Will perform an HW and SW reset of device, and reset internal driver states To know value. Should be called before setup or when device state is unknown.

## Parameters

in	<i>dev</i>	pointer to device object instance
----	------------	-----------------------------------

## Returns

0 on success INV\_ERROR\_TIMEOUT if reset does not complete in time INV\_ERROR\_TRANSPORT in case of low level serial error

**7.4.3.13** static int inv\_device\_self\_test ( const inv\_device\_t \* dev, int sensor ) [inline],[static]

Perform self-test procedure for MEMS component of the device.

Available MEMS vary depend on the device. Use following sensor type for the various MEMS:

- INV\_SENSOR\_TYPE\_ACCELEROMETER: for HW accelerometer sensor
- INV\_SENSOR\_TYPE\_MAGNETOMETER : for HW magnetometer sensor
- INV\_SENSOR\_TYPE\_GYROSCOPE : for HW gyroscope sensor
- INV\_SENSOR\_TYPE\_PRESSURE : for HW pressure sensor

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

**See also**

`inv_sensor_type_t`)

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR if self test has failed

**7.4.3.14** `static int inv_device_set_running_state ( const inv_device_t * dev, inv_bool_t state )` `[inline], [static]`

Indicates to device current RUN/SUSPEND state of the host.

If SUSPEND state (false) is set, device should not notify any sensor events (besides event coming from a wake-up source). If RUNNING state (true) is set, all sensor events will be notify to host. Device will consider host to be in RUNNING state after a reset/setup.

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>state</i>	RUNNING (true) or SUSPEND (false) state

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error

**7.4.3.15** `static int inv_device_set_sensor_bias ( const inv_device_t * dev, int sensor, const float bias[3] )` `[inline], [static]`

Configure bias value for a sensor.

Bias configuration makes sense only for few sensor types:

- INV\_SENSOR\_TYPE\_ACCELEROMETER
- INV\_SENSOR\_TYPE\_MAGNETOMETER
- INV\_SENSOR\_TYPE\_GYROSCOPE Bias unit is the same as the corresponding sensor unit.

**See also**

[inv\\_sensor\\_event\\_t](#) for details.

If this feature is supported by the implementation but not by the device, behavior is undefined (but will most probably have no effect).

## Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

inv\_sensor\_type\_t)

## Parameters

in	<i>bias</i>	bias to set
----	-------------	-------------

## Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**7.4.3.16** static int inv\_device\_set\_sensor\_config ( const inv\_device\_t \* dev, int sensor, int settings, const void \* arg, uint16\_t size ) [inline],[static]

Generic method to configure a sensor.

Allow to configure a sensor (HW or virtual), such as FSR, BW, ...

For common settings, setting value is expected to be a value from

## See also

enum [inv\\_sensor\\_config](#). Settings data is expected to point the proper type as describes in [SensorConfig.h](#)

For specific settings, refer to concrete device implementation for supported sensor and available configuration settings parameters.

## Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

inv\_sensor\_type\_t)

## Parameters

in	<i>settings</i>	settings to configure
in	<i>arg</i>	pointer to settings value
in	<i>size</i>	settings value size

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation INV\_ERROR\_SIZE size is above internal buffer size / device capability INV\_ERROR if configuration has failed

**7.4.3.17** `static int inv_device_set_sensor_mounting_matrix ( const inv_device_t * dev, int sensor, const float matrix[9] )` `[inline], [static]`

Sets the mounting matrix information for a multi-axis sensor.

Allow to specify the mounting matrix for multi-axis sensor in order to align axis of several sensors in the same reference frame. Sensor types allowed:

- INV\_SENSOR\_TYPE\_ACCELEROMETER
- INV\_SENSOR\_TYPE\_MAGNETOMETER
- INV\_SENSOR\_TYPE\_GYROSCOPE Depending on device capability, called to this function may have no effect.

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

**See also**

`inv_sensor_type_t)`

**Parameters**

in	<i>matrix</i>	mounting matrix to apply
----	---------------	--------------------------

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation or if one of the mounting matrix value is not in the [-1;1] range

**7.4.3.18** `static int inv_device_set_sensor_period ( const inv_device_t * dev, int sensor, uint32_t period )` `[inline], [static]`

Configure sensor output data period.

Similar to [inv\\_device\\_set\\_sensor\\_period\\_us\(\)](#) except period is specified in ms. Will simply call [inv\\_device\\_set\\_sensor\\_period\\_us\(\)](#) after converting input period.

## Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

inv\_sensor\_type\_t)

## Parameters

in	<i>period</i>	requested data period in ms
----	---------------	-----------------------------

## Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

## Examples:

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

```
7.4.3.19 static int inv_device_set_sensor_period_us( const inv_device_t * dev, int sensor, uint32_t period ) [inline],
[static]
```

Configure sensor output data period.

Send a command to set sensor output data period. Period is a hint only. Depending on sensor type or device capability, the effective output data might be different. User shall refer to sensor events timestamp to determine effective output data period.

## Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

inv\_sensor\_type\_t)

## Parameters

in	<i>period</i>	requested data period in us
----	---------------	-----------------------------

## Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**7.4.3.20** `static int inv_device_set_sensor_timeout ( const inv_device_t * dev, int sensor, uint32_t timeout )` `[inline]`, `[static]`

Configure sensor output timeout.

Send a command to set sensor maximum report latency (or batch timeout). This allows to enable batch mode. Provided timeout is a hint only and sensor events may be notified at a faster rate depending on sensor type or device capability or other active sensors.

#### Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

#### See also

`inv_sensor_type_t`)

#### Parameters

in	<i>timeout</i>	allowed timeout in ms
----	----------------	-----------------------

#### Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**7.4.3.21** `static int inv_device_set_sensor_timeout_us ( const inv_device_t * dev, int sensor, uint32_t timeout )` `[inline]`, `[static]`

Configure sensor output timeout.

Similar to `inv_device_set_sensor_timeout()` except period is specified in ms. Will simply call `inv_device_set_sensor_timeout()` after converting input period.

#### Warning

If input timeout is < 1000, value will be rounded to 0.

#### Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

#### See also

`inv_sensor_type_t`)



## Parameters

in	<i>timeout</i>	allowed timeout in us
----	----------------	-----------------------

## Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**7.4.3.22** `static int inv_device_setup ( const inv_device_t * dev ) [inline],[static]`

Performs basic device initialization.

Except if device's flash memory is outdated, device should be able to handle request after setup() is complete. If device's flash memory need to be updated, load\_begin()/load\_continue()/ load\_end() methods must be called first with suitable argument.

## Parameters

in	<i>dev</i>	pointer to device object instance
----	------------	-----------------------------------

## Returns

0 on success INV\_ERROR\_TIMEOUT if setup does not complete in time INV\_ERROR\_TRANSPORT in case of low level serial error

## Examples:

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

**7.4.3.23** `static int inv_device_start_sensor ( const inv_device_t * dev, int sensor ) [inline],[static]`

Starts a sensor.

Send a command to start a sensor. Device will start sending events if sensor is supported (ie: ping() returns 0 for this sensor type).

## Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

## See also

inv\_sensor\_type\_t)

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**Examples:**

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

**7.4.3.24** `static int inv_device_stop_sensor ( const inv_device_t * dev, int sensor ) [inline],[static]`

Stops a sensor.

Send a command to stop a sensor. Device will stop sending events if sensor was previously started.

**Parameters**

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

**See also**

`inv_sensor_type_t`)

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation

**Examples:**

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

**7.4.3.25** `static int inv_device_whoami ( const inv_device_t * dev, uint8_t * whoami ) [inline],[static]`

Gets WHO AM I value.

Can be called before performing device setup

**Parameters**

in	<i>dev</i>	pointer to device object instance
out	<i>whoami</i>	WHO AM I value

**Returns**

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error

**Examples:**

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

7.4.3.26 `static int inv_device_write_mems_register ( const inv_device_t * dev, int sensor, uint16_t reg_addr, const void * data, uint16_t len )` `[inline], [static]`

Set the MEMS register.

#### Parameters

in	<i>dev</i>	pointer to device object instance
in	<i>sensor</i>	sensor type (as defined by

#### See also

`inv_sensor_type_t`)

#### Parameters

in	<i>reg_addr</i>	the register that should be written
in	<i>data</i>	data to write at <i>reg_addr</i>
in	<i>length</i>	length of data to write

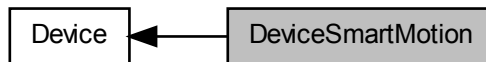
#### Returns

0 on success INV\_ERROR\_TRANSPORT in case of low level serial error INV\_ERROR\_TIMEOUT if device does not respond in time INV\_ERROR\_BAD\_ARG if sensor is not supported by the implementation INV\_ERROR\_SIZE request length is above device capability INV\_ERROR if configuration has failed

## 7.5 DeviceSmartMotion

Device implementation that connects to other IDD devices and emulates missing motion features using software libraries.

Collaboration diagram for DeviceSmartMotion:



### Classes

- struct [inv\\_device\\_smart\\_motion\\_vlistener](#)  
*Internal definition for graph leaves (based on VSensorListener)*
- struct [inv\\_device\\_smart\\_motion\\_vsensor](#)  
*Internal definition for graph roots (based on VSensor)*
- struct [inv\\_device\\_smart\\_motion](#)  
*States for SmartMotion device implementation.*

### Typedefs

- typedef struct [inv\\_device\\_smart\\_motion](#) [inv\\_device\\_smart\\_motion\\_t](#)  
*States for SmartMotion device implementation.*
- typedef uint32\_t [inv\\_smartmotion\\_config\\_value\\_sensitivity\\_t](#)  
*Expected type for SENSITIVITY config settings Value is coded in Q16 and expressed SI unit / LSB SI unit being g for ACC, uT for MAG and dps for GYR.*

### Enumerations

### Functions

- void INV\_EXPORT [inv\\_device\\_smart\\_motion\\_init](#) ([inv\\_device\\_smart\\_motion\\_t](#) \*self, [inv\\_device\\_t](#) \*hw\_↔ device)  
*Initialize states for SmartMotion device implementation.*
- void INV\_EXPORT [inv\\_device\\_smart\\_motion\\_init2](#) ([inv\\_device\\_smart\\_motion\\_t](#) \*self, [inv\\_device\\_t](#) \*hw\_↔ device, void \*opt)  
*Initialize states for SmartMotion device implementation.*
- static [inv\\_device\\_t](#) \* [inv\\_device\\_smart\\_motion\\_get\\_base](#) ([inv\\_device\\_smart\\_motion\\_t](#) \*self)  
*Return generic handle to base inv\_device\_t.*
- int INV\_EXPORT [inv\\_device\\_smart\\_motion\\_setup](#) (void \*context)  
*Initialize SmartMotion graph.*
- void INV\_EXPORT [inv\\_device\\_smart\\_motion\\_init\\_aar](#) (void \*context)  
*Initialize SmartMotion leaves related to Advanced Activity Recognition.*
- void INV\_EXPORT [inv\\_device\\_smart\\_motion\\_init\\_pickup](#) (void \*context)  
*Initialize SmartMotion graph leaves related to Pickup.*
- void INV\_EXPORT [inv\\_device\\_smart\\_motion\\_init\\_eis](#) (void \*context)  
*Initialize SmartMotion graph leaves related to Electronic Image Stabilisation.*

## Variables

- VSensorListener [inv\\_device\\_smart\\_motion\\_vlistener::vlistener](#)  
*base VSensorListener states*
- void(\* [inv\\_device\\_smart\\_motion\\_vlistener::build\\_event](#) )(const void \*, [inv\\_sensor\\_event\\_t](#) \*)  
*callback to convert VSensorData to IDD sensor event*
- uint8\_t [inv\\_device\\_smart\\_motion\\_vlistener::idd\\_type](#)  
*corresponding IDD sensor type*
- InvList [inv\\_device\\_smart\\_motion\\_vlistener::node](#)  
*pointer to next element when stored in a list*
- VSensor [inv\\_device\\_smart\\_motion\\_vsensor::vsensor](#)  
*base VSensor states*
- void(\* [inv\\_device\\_smart\\_motion\\_vsensor::build\\_vsensor\\_data](#) )(const [inv\\_sensor\\_event\\_t](#) \*, void \*)  
*callback to convert IDD sensor event to VSensorData*
- uint8\_t [inv\\_device\\_smart\\_motion\\_vsensor::idd\\_type](#)  
*corresponding IDD sensor type*
- [inv\\_device\\_t](#) [inv\\_device\\_smart\\_motion::base](#)  
*base inv\_device\_t states*
- [inv\\_device\\_t](#) \* [inv\\_device\\_smart\\_motion::hw\\_device](#)  
*reference to inv\_device\_t that will provide sensor data for base sensor*
- uint8\_t [inv\\_device\\_smart\\_motion::base\\_chip\\_info](#) [3]  
*parameter given at init time to identify underlying ICM*
- [inv\\_sensor\\_listener\\_t](#) [inv\\_device\\_smart\\_motion::private\\_idd\\_listener](#)  
*internal IDD listener to catch events from HW device*
- InvList [inv\\_device\\_smart\\_motion::leaves\\_list](#)  
*list of graph leaves (corresponds to sensors not present at hardware level and that are emulated)*
- struct [inv\\_device\\_smart\\_motion\\_vsensor](#) [inv\\_device\\_smart\\_motion::s](#)  
*placeholder for roots VSensor*
- struct [inv\\_device\\_smart\\_motion\\_vlistener](#) [inv\\_device\\_smart\\_motion::l](#)  
*placeholder for leaves corresponding to root VSensor*
- struct {  
    struct [inv\\_device\\_smart\\_motion\\_vsensor](#) [inv\\_device\\_smart\\_motion::s](#)  
        *placeholder for roots VSensor*  
    struct [inv\\_device\\_smart\\_motion\\_vlistener](#) [inv\\_device\\_smart\\_motion::l](#)  
        *placeholder for leaves corresponding to root VSensor*  
} [inv\\_device\\_smart\\_motion::graph\\_roots](#) [5]  
  
*placeholder holder for graph roots objects*
- struct [inv\\_device\\_smart\\_motion\\_vlistener](#) [inv\\_device\\_smart\\_motion::l](#)  
*placeholder for leaves VSensorListener*
- struct {  
    struct [inv\\_device\\_smart\\_motion\\_vlistener](#) [inv\\_device\\_smart\\_motion::l](#)  
        *placeholder for leaves VSensorListener*  
} [inv\\_device\\_smart\\_motion::graph\\_leaves](#) [24]  
  
*placeholder for leaves corresponding to emulated sensors*
- uint64\_t [inv\\_device\\_smart\\_motion::hw\\_sensor\\_available\\_mask](#)  
*mask to keep track of HW available sensor from underlying device*

### 7.5.1 Detailed Description

Device implementation that connects to other IDD devices and emulates missing motion features using software libraries.

It uses the InvenSense VSensor framework to connect a HW device to a InvenSense CModel processing graph.

The graph consists in:

- Few roots that get data from underlying IDD device
- Leaves for all emulated sensors
- Intermediary nodes that call SmartMotion library

Upon SmartMotion device initialization, you must provide a reference to an already initialized and setup `inv_device_t` (eg: ICM20602).

This device implementation takes ownership of the underlying device. Except for very specific methods (eg: `setup()`, `load()`, `self_test()`, ...), handle to SmartMotion device should be used to control sensors.

When calling `setup()`, SmartMotion device will ping all sensors for the underlying HW device and keep track their availability.

When starting a sensor, if it is present at HW level, SmartMotion device will forward the request to the underlying device. If not, it will enable the corresponding algorithm and start required base sensor to provide requested data. Eg: if user requests 'GAME\_ROTATION\_VECTOR' and it is not directly available from the HW device, both 'RAW\_GYROSCOPE' and 'RAW\_ACCELEROMETER' will be started at HW level, and 'GAME\_ROTATION\_VECTOR' algorithm enabled.

SmartMotion device only emulates the following sensors:

- `INV_SENSOR_TYPE_ACCELEROMETER`
- `INV_SENSOR_TYPE_GYROSCOPE`
- `INV_SENSOR_TYPE_UNCAL_GYROSCOPE`
- `INV_SENSOR_TYPE_MAGNETOMETER`
- `INV_SENSOR_TYPE_UNCAL_MAGNETOMETER`
- `INV_SENSOR_TYPE_GAME_ROTATION_VECTOR`
- `INV_SENSOR_TYPE_ROTATION_VECTOR`
- `INV_SENSOR_TYPE_GEOMAG_ROTATION_VECTOR`
- `INV_SENSOR_TYPE_BAC`
- `INV_SENSOR_TYPE_STEP_COUNTER`
- `INV_SENSOR_TYPE_STEP_DETECTOR`
- `INV_SENSOR_TYPE_SMD`
- `INV_SENSOR_TYPE_TILT_DETECTOR`
- `INV_SENSOR_TYPE_PICK_UP_GESTURE`
- `INV_SENSOR_TYPE_GRAVITY`

- INV\_SENSOR\_TYPE\_LINEAR\_ACCELERATION
- INV\_SENSOR\_TYPE\_ORIENTATION

Required base sensor are:

- INV\_SENSOR\_TYPE\_RAW\_ACCELEROMETER
- INV\_SENSOR\_TYPE\_RAW\_GYROSCOPE
- INV\_SENSOR\_TYPE\_RAW\_MAGNETOMETER

#### Warning

Batching is currently not emulated

Only one instance of this device can be used in the same application SmartMotion algorithms are currently integrated using static variables. Only one instance of this device must be used at once, to avoid re-entrancy issue.

## 7.5.2 Enumeration Type Documentation

### 7.5.2.1 enum inv\_smartmotion\_config\_setting

Allowed config settings for DeviceSmartMotion.

#### Enumerator

**INV\_SMARTMOTION\_CONFIG\_SENSITIVITY** sensitivity (LSB to SI unit factor)

## 7.5.3 Function Documentation

**7.5.3.1** void INV\_EXPORT inv\_device\_smart\_motion\_init ( inv\_device\_smart\_motion\_t \* *self*, inv\_device\_t \* *hw\_device* )

Initialize states for SmartMotion device implementation.

Assumes underlying device is already initialized. Will take ownership of the underlying device and re-use its listener. No HW access are done in this function.

#### Parameters

in	<i>self</i>	handle to current device
in	<i>hw_device</i>	reference to inv_device_t used to retrieve base data

**7.5.3.2** void INV\_EXPORT inv\_device\_smart\_motion\_init2 ( inv\_device\_smart\_motion\_t \* *self*, inv\_device\_t \* *hw\_device*, void \* *opt* )

Initialize states for SmartMotion device implementation.

Assumes underlying device is already initialized. Will take ownership of the underlying device and re-use its listener. No HW access are done in this function.

#### Parameters

in	<i>self</i>	handle to current device
in	<i>hw_device</i>	reference to <code>inv_device_t</code> used to retrieve base data
in	<i>opt</i>	optional parameter given at init time (can be NULL)

#### 7.5.3.3 `int INV_EXPORT inv_device_smart_motion_setup ( void * context )`

Initialize SmartMotion graph.

Will perform HW access through underlying device (eg: calling `ping()`). Underlying device should be setup and all FW/images loaded before calling this method

### 7.5.4 Variable Documentation

#### 7.5.4.1 `struct inv_device_smart_motion_vlistener inv_device_smart_motion::l`

placeholder for leaves corresponding to root VSensor

placeholder for leaves VSensorListener



## 7.6 DeviceEmdWrapper

emd wrapper implementation for device interface

Collaboration diagram for DeviceEmdWrapper:



### Classes

- struct [inv\\_device\\_emd\\_wrap\\_icm20xxx](#)
- struct [inv\\_device\\_emd\\_wrap\\_icm20xxx\\_serial](#)

### Functions

- void INV\_EXPORT [inv\\_device\\_emd\\_wrap\\_icm20xxx\\_init](#) ([inv\\_device\\_emd\\_wrap\\_icm20xxx\\_t](#) \*self, const [inv\\_sensor\\_listener\\_t](#) \*listener, const struct [inv\\_device\\_emd\\_wrap\\_icm20xxx\\_serial](#) \*serial, void \*serial\_cookie)  
*constructor-like function for EMD Wrapper device*

### 7.6.1 Detailed Description

emd wrapper implementation for device interface

## 7.7 DeviceIcm20602

Concrete implementation of the 'Device' interface for Icm20602 devices.

Collaboration diagram for DeviceIcm20602:



### Classes

- struct [inv\\_device\\_icm20602](#)  
*States for Icm20602 device.*
- struct [inv\\_device\\_icm20602\\_config\\_bias\\_st](#)  
*Bias collected during self-test (config INV\_DEVICE\_ICM20602\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.*

### Typedefs

- typedef struct [inv\\_device\\_icm20602](#) [inv\\_device\\_icm20602\\_t](#)  
*States for Icm20602 device.*
- typedef struct [inv\\_device\\_icm20602\\_config\\_bias\\_st](#) [inv\\_device\\_icm20602\\_config\\_bias\\_st\\_t](#)  
*Bias collected during self-test (config INV\_DEVICE\_ICM20602\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.*
- typedef int32\_t [inv\\_device\\_icm20602\\_config\\_wom\\_threshold\\_t](#)  
*WOM threshold value (expressed in LSB)*

### Enumerations

### Functions

- void INV\_EXPORT [inv\\_device\\_icm20602\\_init2](#) ([inv\\_device\\_icm20602\\_t](#) \*self, const [inv\\_serif\\_hal\\_t](#) \*serif, const [inv\\_sensor\\_listener\\_t](#) \*listener)  
*constructor-like function for basesensor device*
- void INV\_EXPORT [inv\\_device\\_icm20602\\_init](#) ([inv\\_device\\_icm20602\\_t](#) \*self, const [inv\\_host\\_serif\\_t](#) \*serif, const [inv\\_sensor\\_listener\\_t](#) \*listener)  
*constructor-like function for basesensor device*
- void INV\_EXPORT [inv\\_device\\_icm20602\\_init\\_serif\\_ois2](#) ([inv\\_device\\_icm20602\\_t](#) \*self, const [inv\\_serif\\_hal\\_t](#) \*serif, [inv\\_serif\\_ois\\_t](#) \*serif\_ois)  
*Set reference to Serial Interface object for OIS interface.*
- void INV\_EXPORT [inv\\_device\\_icm20602\\_init\\_serif\\_ois](#) ([inv\\_device\\_icm20602\\_t](#) \*self, const [inv\\_host\\_serif\\_t](#) \*serif, [inv\\_serif\\_ois\\_t](#) \*serif\_ois)  
*Set reference to Serial Interface object for OIS interface.*
- void INV\_EXPORT [inv\\_device\\_icm20602\\_init\\_aux\\_compass](#) ([inv\\_device\\_icm20602\\_t](#) \*self, int aux\_compass\_id, uint8\_t aux\_compass\_addr)  
*Register a compass as AUX sensor.*
- static [inv\\_device\\_t](#) \* [inv\\_device\\_icm20602\\_get\\_base](#) ([inv\\_device\\_icm20602\\_t](#) \*self)  
*Helper function to get handle to base object.*

### 7.7.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20602 devices.

See ExampleDeviceIcm20602.c example.

### 7.7.2 Function Documentation

**7.7.2.1** void INV\_EXPORT inv\_device\_icm20602\_init ( inv\_device\_icm20602\_t \* *self*, const inv\_host\_serif\_t \* *serif*, const inv\_sensor\_listener\_t \* *listener* )

constructor-like function for basesensor device

**Deprecated** Use inv\_device\_icm20602\_init2() instead.

Will initialize inv\_device\_icm20602\_t object states to default value for basesensor.

#### Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to legacy Host Serial Interface object
in	<i>listener</i>	reference to Sensor Event Listener object

**7.7.2.2** void INV\_EXPORT inv\_device\_icm20602\_init2 ( inv\_device\_icm20602\_t \* *self*, const inv\_serif\_hal\_t \* *serif*, const inv\_sensor\_listener\_t \* *listener* )

constructor-like function for basesensor device

Will initialize inv\_device\_icm20602\_t object states to default value for basesensor.

#### Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to Serial Interface object
in	<i>listener</i>	reference to Sensor Event Listener object

#### Examples:

[ExampleDeviceIcm20602EMD.c](#).

**7.7.2.3** void INV\_EXPORT inv\_device\_icm20602\_init\_aux\_compass ( inv\_device\_icm20602\_t \* *self*, int *aux\_compass\_id*, uint8\_t *aux\_compass\_addr* )

Register a compass as AUX sensor.

Should be called after [inv\\_device\\_icm20602\\_init\(\)](#) but before device setup.

## Parameters

in	<i>aux_compass_id</i>	auxiliary compass id (as of enum <code>inv_icm20602_compass_id</code> )
in	<i>aux_compass_add</i>	I2C slave address for compass

7.7.2.4 `void INV_EXPORT inv_device_icm20602_init_serif_ois ( inv_device_icm20602_t * self, const inv_host_serif_t * serif_ois )`

Set reference to Serial Interface object for OIS interface.

**Deprecated** Use `inv_device_icm20602_init_serif_ois2()` instead.

## Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to legacy Host Serial Interface object for OIS interface

7.7.2.5 `void INV_EXPORT inv_device_icm20602_init_serif_ois2 ( inv_device_icm20602_t * self, const inv_serif_hal_t * serif_ois )`

Set reference to Serial Interface object for OIS interface.

When set, a call to the `poll()` method will also retrieve and report OIS data (if enabled)

Should be called after `inv_device_icm20602_init2()` but before device setup.

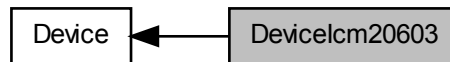
## Parameters

in	<i>self</i>	handle to device
in	<i>serif_ois</i>	reference to Serial Interface object for OIS interface

## 7.8 DeviceIcm20603

Concrete implementation of the 'Device' interface for Icm20603 devices.

Collaboration diagram for DeviceIcm20603:



### Classes

- struct [inv\\_device\\_icm20603](#)  
*States for Icm20603 device.*
- struct [inv\\_device\\_icm20603\\_config\\_bias\\_st](#)  
*Bias collected during self-test (config INV\_DEVICE\_ICM20603\_CONFIG\_BIAS\_ST) Value is scaled by 2<sup>16</sup>, accel is gee and gyro is dps.*

### Typedefs

- typedef struct [inv\\_device\\_icm20603](#) [inv\\_device\\_icm20603\\_t](#)  
*States for Icm20603 device.*
- typedef struct [inv\\_device\\_icm20603\\_config\\_bias\\_st](#) [inv\\_device\\_icm20603\\_config\\_bias\\_st\\_t](#)  
*Bias collected during self-test (config INV\_DEVICE\_ICM20603\_CONFIG\_BIAS\_ST) Value is scaled by 2<sup>16</sup>, accel is gee and gyro is dps.*
- typedef int32\_t [inv\\_device\\_icm20603\\_config\\_wom\\_threshold\\_t](#)  
*WOM threshold value (expressed in LSB)*

### Enumerations

### Functions

- void INV\_EXPORT [inv\\_device\\_icm20603\\_init2](#) ([inv\\_device\\_icm20603\\_t](#) \*self, const [inv\\_serif\\_hal\\_t](#) \*serif, const [inv\\_sensor\\_listener\\_t](#) \*listener)  
*constructor-like function for basesensor device*
- void INV\_EXPORT [inv\\_device\\_icm20603\\_init](#) ([inv\\_device\\_icm20603\\_t](#) \*self, const [inv\\_host\\_serif\\_t](#) \*serif, const [inv\\_sensor\\_listener\\_t](#) \*listener)  
*constructor-like function for basesensor device*
- void INV\_EXPORT [inv\\_device\\_icm20603\\_init\\_serif\\_ois2](#) ([inv\\_device\\_icm20603\\_t](#) \*self, const [inv\\_serif\\_hal\\_t](#) \*serif\_ois)  
*Set reference to Serial Interface object for OIS interface.*
- void INV\_EXPORT [inv\\_device\\_icm20603\\_init\\_serif\\_ois](#) ([inv\\_device\\_icm20603\\_t](#) \*self, const [inv\\_host\\_serif\\_t](#) \*serif\_ois)  
*Set reference to Serial Interface object for OIS interface.*
- void INV\_EXPORT [inv\\_device\\_icm20603\\_init\\_aux\\_compass](#) ([inv\\_device\\_icm20603\\_t](#) \*self, int aux\_compass\_id, uint8\_t aux\_compass\_addr)  
*Register a compass as AUX sensor.*
- static [inv\\_device\\_t](#) \* [inv\\_device\\_icm20603\\_get\\_base](#) ([inv\\_device\\_icm20603\\_t](#) \*self)  
*Helper function to get handle to base object.*

### 7.8.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20603 devices.

See `ExampleDeviceIcm20603.c` example.

### 7.8.2 Function Documentation

**7.8.2.1** `void INV_EXPORT inv_device_icm20603_init ( inv_device_icm20603_t * self, const inv_host_serif_t * serif, const inv_sensor_listener_t * listener )`

constructor-like function for basesensor device

**Deprecated** Use `inv_device_icm20603_init2()` instead.

Will initialize `inv_device_icm20603_t` object states to default value for basesensor.

#### Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to legacy Host Serial Interface object
in	<i>listener</i>	reference to Sensor Event Listener object

**7.8.2.2** `void INV_EXPORT inv_device_icm20603_init2 ( inv_device_icm20603_t * self, const inv_serif_hal_t * serif, const inv_sensor_listener_t * listener )`

constructor-like function for basesensor device

Will initialize `inv_device_icm20603_t` object states to default value for basesensor.

#### Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to Serial Interface object
in	<i>listener</i>	reference to Sensor Event Listener object

#### Examples:

[ExampleDeviceIcm20603EMD.c](#).

**7.8.2.3** `void INV_EXPORT inv_device_icm20603_init_aux_compass ( inv_device_icm20603_t * self, int aux_compass_id, uint8_t aux_compass_addr )`

Register a compass as AUX sensor.

Should be called after `inv_device_icm20603_init()` but before device setup.

## Parameters

in	<i>aux_compass_id</i>	auxiliary compass id (as of enum <code>inv_icm20603_compass_id</code> )
in	<i>aux_compass_add</i>	I2C slave address for compass

7.8.2.4 `void INV_EXPORT inv_device_icm20603_init_serif_ois ( inv_device_icm20603_t * self, const inv_host_serif_t * serif_ois )`

Set reference to Serial Interface object for OIS interface.

**Deprecated** Use `inv_device_icm20603_init_serif_ois2()` instead.

## Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to legacy Host Serial Interface object for OIS interface

7.8.2.5 `void INV_EXPORT inv_device_icm20603_init_serif_ois2 ( inv_device_icm20603_t * self, const inv_serif_hal_t * serif_ois )`

Set reference to Serial Interface object for OIS interface.

When set, a call to the `poll()` method will also retrieve and report OIS data (if enabled)

Should be called after `inv_device_icm20603_init2()` but before device setup.

## Parameters

in	<i>self</i>	handle to device
in	<i>serif_ois</i>	reference to Serial Interface object for OIS interface

## 7.9 DeviceIcm20690

Concrete implementation of the 'Device' interface for Icm20690 devices.

Collaboration diagram for DeviceIcm20690:



### Classes

- struct [inv\\_device\\_icm20690](#)  
*States for Icm20690 device.*
- struct [inv\\_device\\_icm20690\\_config\\_bias\\_st](#)  
*Bias collected during self-test (config INV\_DEVICE\_ICM20690\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.*

### Typedefs

- typedef struct [inv\\_device\\_icm20690](#) [inv\\_device\\_icm20690\\_t](#)  
*States for Icm20690 device.*
- typedef struct [inv\\_device\\_icm20690\\_config\\_bias\\_st](#) [inv\\_device\\_icm20690\\_config\\_bias\\_st\\_t](#)  
*Bias collected during self-test (config INV\_DEVICE\_ICM20690\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.*
- typedef int32\_t [inv\\_device\\_icm20690\\_config\\_wom\\_threshold\\_t](#)  
*WOM threshold value (expressed in LSB)*

### Enumerations

### Functions

- void INV\_EXPORT [inv\\_device\\_icm20690\\_init2](#) ([inv\\_device\\_icm20690\\_t](#) \*self, const [inv\\_serif\\_hal\\_t](#) \*serif, const [inv\\_sensor\\_listener\\_t](#) \*listener)  
*constructor-like function for basesensor device*
- void INV\_EXPORT [inv\\_device\\_icm20690\\_init](#) ([inv\\_device\\_icm20690\\_t](#) \*self, const [inv\\_host\\_serif\\_t](#) \*serif, const [inv\\_sensor\\_listener\\_t](#) \*listener)  
*constructor-like function for basesensor device*
- void INV\_EXPORT [inv\\_device\\_icm20690\\_init\\_serif\\_ois2](#) ([inv\\_device\\_icm20690\\_t](#) \*self, const [inv\\_serif\\_hal\\_t](#) \*serif, [inv\\_serif\\_ois\\_t](#) \*serif\_ois)  
*Set reference to Serial Interface object for OIS interface.*
- void INV\_EXPORT [inv\\_device\\_icm20690\\_init\\_serif\\_ois](#) ([inv\\_device\\_icm20690\\_t](#) \*self, const [inv\\_host\\_serif\\_t](#) \*serif, [inv\\_serif\\_ois\\_t](#) \*serif\_ois)  
*Set reference to Serial Interface object for OIS interface.*
- void INV\_EXPORT [inv\\_device\\_icm20690\\_init\\_aux\\_compass](#) ([inv\\_device\\_icm20690\\_t](#) \*self, int aux\_compass\_id, uint8\_t aux\_compass\_addr)  
*Register a compass as AUX sensor.*
- static [inv\\_device\\_t](#) \* [inv\\_device\\_icm20690\\_get\\_base](#) ([inv\\_device\\_icm20690\\_t](#) \*self)  
*Helper function to get handle to base object.*



### 7.9.1 Detailed Description

Concrete implementation of the 'Device' interface for Icm20690 devices.

See ExampleDeviceIcm20690.c example.

### 7.9.2 Function Documentation

**7.9.2.1** void INV\_EXPORT inv\_device\_icm20690\_init ( inv\_device\_icm20690\_t \* *self*, const inv\_host\_serif\_t \* *serif*, const inv\_sensor\_listener\_t \* *listener* )

constructor-like function for basesensor device

**Deprecated** Use inv\_device\_icm20690\_init2() instead.

Will initialize inv\_device\_icm20690\_t object states to default value for basesensor.

#### Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to legacy Host Serial Interface object
in	<i>listener</i>	reference to Sensor Event Listener object

**7.9.2.2** void INV\_EXPORT inv\_device\_icm20690\_init2 ( inv\_device\_icm20690\_t \* *self*, const inv\_serif\_hal\_t \* *serif*, const inv\_sensor\_listener\_t \* *listener* )

constructor-like function for basesensor device

Will initialize inv\_device\_icm20690\_t object states to default value for basesensor.

#### Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to Serial Interface object
in	<i>listener</i>	reference to Sensor Event Listener object

#### Examples:

[ExampleDeviceIcm20690EMD.c](#).

**7.9.2.3** void INV\_EXPORT inv\_device\_icm20690\_init\_aux\_compass ( inv\_device\_icm20690\_t \* *self*, int *aux\_compass\_id*, uint8\_t *aux\_compass\_addr* )

Register a compass as AUX sensor.

Should be called after [inv\\_device\\_icm20690\\_init\(\)](#) but before device setup.

## Parameters

in	<i>aux_compass_id</i>	auxiliary compass id (as of enum <code>inv_icm20690_compass_id</code> )
in	<i>aux_compass_add</i>	I2C slave address for compass

7.9.2.4 `void INV_EXPORT inv_device_icm20690_init_serif_ois ( inv_device_icm20690_t * self, const inv_host_serif_t * serif_ois )`

Set reference to Serial Interface object for OIS interface.

**Deprecated** Use `inv_device_icm20690_init_serif_ois2()` instead.

## Parameters

in	<i>self</i>	handle to device
in	<i>serif</i>	reference to legacy Host Serial Interface object for OIS interface

7.9.2.5 `void INV_EXPORT inv_device_icm20690_init_serif_ois2 ( inv_device_icm20690_t * self, const inv_serif_hal_t * serif_ois )`

Set reference to Serial Interface object for OIS interface.

When set, a call to the `poll()` method will also retrieve and report OIS data (if enabled)

Should be called after `inv_device_icm20690_init2()` but before device setup.

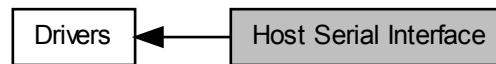
## Parameters

in	<i>self</i>	handle to device
in	<i>serif_ois</i>	reference to Serial Interface object for OIS interface

## 7.10 Host Serial Interface

Virtual abstraction of host adapter for serial interface.

Collaboration diagram for Host Serial Interface:



### Classes

- struct [inv\\_host\\_serif](#)  
*Serial Interface interface definition.*

### Typedefs

- typedef struct [inv\\_host\\_serif](#) [inv\\_host\\_serif\\_t](#)  
*Serial Interface interface definition.*

### Enumerations

### Functions

- void INV\_EXPORT [inv\\_host\\_serif\\_set\\_instance](#) (const [inv\\_host\\_serif\\_t](#) \*instance)  
*Set global instance for Serial Interface.*
- const [inv\\_host\\_serif\\_t](#) \* [inv\\_host\\_serif\\_get\\_instance](#) (void)  
*Return global instance for Serial Interface.*
- static int [inv\\_host\\_serif\\_open](#) (const [inv\\_host\\_serif\\_t](#) \*instance)  
*Helper method to call open() method of a Serial Interface object.*
- static int [inv\\_host\\_serif\\_close](#) (const [inv\\_host\\_serif\\_t](#) \*instance)  
*Helper method to call close() method of a Serial Interface object.*
- static int [inv\\_host\\_serif\\_read\\_reg](#) (const [inv\\_host\\_serif\\_t](#) \*instance, uint8\_t reg, uint8\_t \*data, uint32\_t len)  
*Helper method to call read\_reg() method of a Serial Interface object.*
- static int [inv\\_host\\_serif\\_write\\_reg](#) (const [inv\\_host\\_serif\\_t](#) \*instance, uint8\_t reg, const uint8\_t \*data, uint32\_t len)  
*Helper method to call write\_reg() method of a Serial Interface object.*
- static int [inv\\_host\\_serif\\_register\\_interrupt\\_callback](#) (const [inv\\_host\\_serif\\_t](#) \*instance, void(\*interrupt\_cb)(void \*context, int int\_num), void \*context)  
*Helper method to call register\_interrupt\_callback() method of a Serial Interface object.*
- static int [inv\\_host\\_serif\\_get\\_type](#) (const [inv\\_host\\_serif\\_t](#) \*instance)  
*Helper method to get serial interface type of a Serial Interface object.*
- static uint32\_t [inv\\_host\\_serif\\_get\\_max\\_read\\_transaction\\_size](#) (const [inv\\_host\\_serif\\_t](#) \*instance)

*Helper method to get max read size value of a Serial Interface object.*

- static uint32\_t [inv\\_host\\_serif\\_get\\_max\\_write\\_transaction\\_size](#) (const [inv\\_host\\_serif\\_t](#) \*instance)

*Helper method to get max write size value of a Serial Interface object.*

- static int [inv\\_host\\_serif\\_is\\_i2c](#) (const [inv\\_host\\_serif\\_t](#) \*instance)

*Helper method to check if serial interface type is I2C for a Serial Interface object.*

- static int [inv\\_host\\_serif\\_is\\_spi](#) (const [inv\\_host\\_serif\\_t](#) \*instance)

*Helper method to check if serial interface type is SPI for a Serial Interface object.*

### 7.10.1 Detailed Description

Virtual abstraction of host adapter for serial interface.

**Deprecated** Use SerifHal.h instead

## 7.11 Data Converter

Helper functions to convert integer.

### Functions

- `uint8_t INV_EXPORT * inv_dc_int32_to_little8 (int32_t x, uint8_t *little8)`  
*Converts a 32-bit long to a little endian byte stream.*
- `uint8_t INV_EXPORT * inv_dc_int16_to_little8 (int16_t x, uint8_t *little8)`  
*Converts a 16-bit integer to a little endian byte stream.*
- `uint8_t INV_EXPORT * inv_dc_int32_to_big8 (int32_t x, uint8_t *big8)`  
*Converts a 32-bit long to a big endian byte stream.*
- `int32_t INV_EXPORT inv_dc_little8_to_int32 (const uint8_t *little8)`  
*Converts a little endian byte stream into a 32-bit integer.*
- `int16_t INV_EXPORT inv_dc_le_to_int16 (const uint8_t *little8)`  
*Converts a little endian byte stream into a 16-bit integer.*
- `int16_t INV_EXPORT inv_dc_big16_to_int16 (uint8_t *data)`  
*Converts big endian on 16 bits into an unsigned short.*
- `void INV_EXPORT inv_dc_sfix32_to_float (const int32_t *in, uint32_t len, uint8_t qx, float *out)`  
*Converts an array of 32-bit signed fixed-point integers to an array of floats.*
- `void INV_EXPORT inv_dc_float_to_sfix32 (const float *in, uint32_t len, uint8_t qx, int32_t *out)`  
*Converts an array of floats to an array of 32-bit signed fixed-point integers.*

### 7.11.1 Detailed Description

Helper functions to convert integer.

### 7.11.2 Function Documentation

**7.11.2.1** `void INV_EXPORT inv_dc_float_to_sfix32 ( const float * in, uint32_t len, uint8_t qx, int32_t * out )`

Converts an array of floats to an array of 32-bit signed fixed-point integers.

#### Parameters

<code>in</code>	<i>in</i>	Pointer to the first element of the array of floats
<code>in</code>	<i>len</i>	Length of the array
<code>in</code>	<i>qx</i>	Number of bits used to represent the decimal part of the fixed-point integers
<code>out</code>	<i>out</i>	Pointer to the memory area where the output will be stored

**7.11.2.2** `void INV_EXPORT inv_dc_sfix32_to_float ( const int32_t * in, uint32_t len, uint8_t qx, float * out )`

Converts an array of 32-bit signed fixed-point integers to an array of floats.

**Parameters**

in	<i>in</i>	Pointer to the first element of the array of 32-bit signed fixed-point integers
in	<i>len</i>	Length of the array
in	<i>qx</i>	Number of bits used to represent the decimal part of the fixed-point integers
out	<i>out</i>	Pointer to the memory area where the output will be stored

## 7.12 Error Helper

Helper functions related to error code.

### Functions

- `const char INV_EXPORT * inv_error_str (int error)`  
*Returns string describing error number.*

### 7.12.1 Detailed Description

Helper functions related to error code.

### 7.12.2 Function Documentation

#### 7.12.2.1 `const char INV_EXPORT* inv_error_str ( int error )`

Returns string describing error number.

#### See also

enum [inv\\_error](#)

## 7.13 Message

Utility functions to display and redirect diagnostic messages.

### Macros

- `#define INV_MSG_DISABLE 1`  
*For eMD target, disable log by default. If compile switch is set for a compilation unit messages will be totally disabled by default.*
- `#define INV_MSG(level, ...) _INV_MSG(level, __VA_ARGS__)`  
*Allow to force enabling messaging using INV\_MSG\_ENABLE define.*
- `#define INV_MSG_SETUP(level, printer) _INV_MSG_SETUP(level, printer)`  
*Helper macro for calling `inv_msg_setup()`. If INV\_MSG\_DISABLE compile switch is set for a compilation unit messages will be totally disabled.*
- `#define INV_MSG_SETUP_LEVEL(level) _INV_MSG_SETUP_LEVEL(level)`  
*Helper macro for calling `inv_msg_setup_level()`. If INV\_MSG\_DISABLE compile switch is set for a compilation unit messages will be totally disabled.*
- `#define INV_MSG_SETUP_DEFAULT() _INV_MSG_SETUP_DEFAULT()`  
*Helper macro for calling `inv_msg_setup_default()`. If INV\_MSG\_DISABLE compile switch is set for a compilation unit messages will be totally disabled.*
- `#define INV_MSG_LEVEL _INV_MSG_LEVEL`  
*Return current level.*

### Typedefs

- `typedef void(* inv_msg_printer_t) (int level, const char *str, va_list ap)`  
*Prototype for print routine function.*

### Enumerations

### Functions

- `void INV_EXPORT inv_msg_setup (int level, inv_msg_printer_t printer)`  
*Set message level and printer function.*
- `void INV_EXPORT inv_msg_printer_default (int level, const char *str, va_list ap)`  
*Default printer function that display messages to stderr. Function uses stdio.*
- `static void inv_msg_setup_level (int level)`  
*Set message level. Default printer function will be used.*
- `static void inv_msg_setup_default (void)`  
*Set default message level and printer.*
- `int INV_EXPORT inv_msg_get_level (void)`  
*Return current message level.*
- `void INV_EXPORT inv_msg (int level, const char *str,...)`  
*Display a message (through means of printer function)*



### 7.13.1 Detailed Description

Utility functions to display and redirect diagnostic messages.

Use `INV_MSG_DISABLE` or `INV_MSG_ENABLE` define before including this header to enable/disable messages for a compilation unit.

Under Linux, Windows or Arduino, messages are enabled by default. Use `INV_MSG_DISABLE` to disable them.

Under orther environmment, message are disabled by default. Use `INV_MSG_ENABLE` to disable them.

### 7.13.2 Macro Definition Documentation

#### 7.13.2.1 `#define INV_MSG( level, ... ) _INV_MSG(level, __VA_ARGS__)`

Allow to force enabling messaging using `INV_MSG_ENABLE` define.

Helper macro for calling `inv_msg()` If `INV_MSG_DISABLE` compile switch is set for a compilation unit messages will be totally disabled

#### 7.13.2.2 `#define INV_MSG_LEVEL _INV_MSG_LEVEL`

Return current level.

#### Warning

This macro may expand as a function call

### 7.13.3 Function Documentation

#### 7.13.3.1 `void INV_EXPORT inv_msg ( int level, const char * str, ... )`

Display a message (through means of printer function)

#### Parameters

in	<i>level</i>	for the message
in	<i>str</i>	message string
in	...	optional arguments

#### Returns

none

#### 7.13.3.2 `int INV_EXPORT inv_msg_get_level ( void )`

Return current message level.

**Returns**

current message level

**7.13.3.3** void INV\_EXPORT inv\_msg\_printer\_default ( int *level*, const char \* *str*, va\_list *ap* )

Default printer function that display messages to stderr Function uses stdio.

Care must be taken on embeded platfrom. Function does nothing with IAR compiler.

**Returns**

none

**7.13.3.4** void INV\_EXPORT inv\_msg\_setup ( int *level*, inv\_msg\_printer\_t *printer* )

Set message level and printer function.

**Parameters**

in	<i>level</i>	only message above level will be passed to printer function
in	<i>printer</i>	user provided function in charge printing message

**Returns**

none

**7.13.3.5** static void inv\_msg\_setup\_default ( void ) [inline],[static]

Set default message level and printer.

**Returns**

none

**7.13.3.6** static void inv\_msg\_setup\_level ( int *level* ) [inline],[static]

Set message level Default printer function will be used.

**Parameters**

in	<i>level</i>	only message above level will be passed to printer function
----	--------------	---

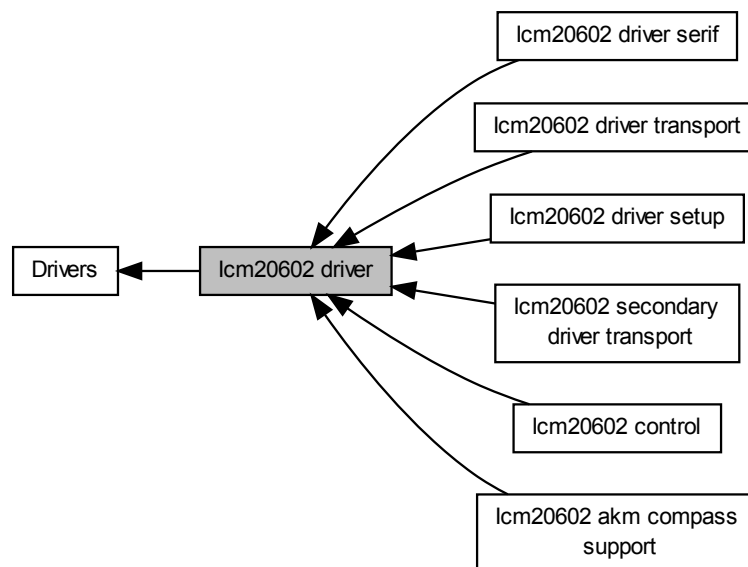
Returns

none

## 7.14 lcm20602 driver

Low-level driver for lcm20602 devices.

Collaboration diagram for lcm20602 driver:



### Modules

- [lcm20602 akm compass support](#)  
*Low-level lcm20602 aux sensor access.*
- [lcm20602 secondary driver transport](#)  
*Low-level lcm20602 secondary interface access.*
- [lcm20602 control](#)  
*Low-level function to control a lcm20602 device.*
- [lcm20602 driver serif](#)  
*Interface for low-level serial (I2C/SPI) access.*
- [lcm20602 driver setup](#)  
*Low-level function to setup an lcm20602 device.*
- [lcm20602 driver transport](#)  
*Low-level lcm20602 register access.*

### Classes

- struct [inv\\_lcm20602](#)  
*lcm20602 driver states definition.*

## Typedefs

- typedef struct [inv\\_icm20602](#) [inv\\_icm20602\\_t](#)  
*icm20602 driver states definition.*

## Functions

- void [inv\\_icm20602\\_sleep](#) (int ms)  
*Hook for low-level system sleep() function to be implemented by upper layer.*
- void [inv\\_icm20602\\_sleep\\_us](#) (int us)  
*Hook for low-level high res system sleep() function to be implemented by upper layer ~ 100us resolution is sufficient.*
- uint64\_t [inv\\_icm20602\\_get\\_time\\_us](#) (void)  
*Hook for low-level system time() function to be implemented by upper layer.*
- uint64\_t [inv\\_icm20602\\_get\\_dataready\\_interrupt\\_time\\_us](#) (void)  
*Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get\_time\_us()*
- static void [inv\\_icm20602\\_reset\\_states](#) (struct [inv\\_icm20602](#) \*s, const struct [inv\\_icm20602\\_serif](#) \*serif)  
*Reset and initialize driver states.*
- static void [inv\\_icm20602\\_reset\\_states\\_serif\\_ois](#) (struct [inv\\_icm20602](#) \*s, const struct [inv\\_icm20602\\_serif](#) \*serif\_ois)  
*Register secondary SERIF object for OIS access Must be called after [inv\\_icm20602\\_reset\\_states\(\)](#)*

### 7.14.1 Detailed Description

Low-level driver for Icm20602 devices.

### 7.14.2 Function Documentation

#### 7.14.2.1 uint64\_t [inv\\_icm20602\\_get\\_dataready\\_interrupt\\_time\\_us](#) ( void )

Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than [get\\_time\\_us\(\)](#)

#### Returns

data ready interrupt timestamp in us

#### 7.14.2.2 uint64\_t [inv\\_icm20602\\_get\\_time\\_us](#) ( void )

Hook for low-level system time() function to be implemented by upper layer.

#### Returns

monotonic timestamp in us

#### 7.14.2.3 static void [inv\\_icm20602\\_reset\\_states](#) ( struct [inv\\_icm20602](#) \* s, const struct [inv\\_icm20602\\_serif](#) \* serif ) [inline],[static]

Reset and initialize driver states.

**Parameters**

in	<i>s</i>	handle to driver states structure
in	<i>serif</i>	handle to SERIF object for underlying register access

**7.14.2.4** `static void inv_icm20602_reset_states_serif_ois ( struct inv_icm20602 * s, const struct inv_icm20602_serif * serif_ois ) [inline],[static]`

Register secondary SERIF object for OIS access Must be called after [inv\\_icm20602\\_reset\\_states\(\)](#)

**Parameters**

in	<i>s</i>	handle to driver states structure
in	<i>serif</i>	handle to SERIF object for underlying register access to OIS

**7.14.2.5** `void inv_icm20602_sleep ( int ms )`

Hook for low-level system sleep() function to be implemented by upper layer.

**Parameters**

in	<i>ms</i>	number of millisecond the calling thread should sleep
----	-----------	---

**7.14.2.6** `void inv_icm20602_sleep_us ( int us )`

Hook for low-level high res system sleep() function to be implemented by upper layer ~100us resolution is sufficient.

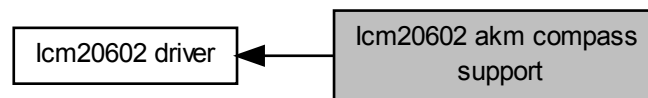
**Parameters**

in	<i>us</i>	number of us the calling thread should sleep
----	-----------	--

## 7.15 lcm20602 akm compass support

Low-level lcm20602 aux sensor access.

Collaboration diagram for lcm20602 akm compass support:



### Enumerations

### Functions

- void INV\_EXPORT [inv\\_icm20602\\_register\\_aux\\_compass](#) (struct [inv\\_icm20602](#) \*s, enum [inv\\_icm20602\\_compass\\_id](#) compass\_id, uint8\_t compass\_i2c\_addr)  
*Register AUX compass.*
- int INV\_EXPORT [inv\\_icm20602\\_is\\_compass\\_registered](#) (struct [inv\\_icm20602](#) \*s)  
*Return non-zero value is AUX compass was regitered, 0 if not If compass was registered but setup failed, will also return 0.*
- int INV\_EXPORT [inv\\_icm20602\\_setup\\_compass\\_akm](#) (struct [inv\\_icm20602](#) \*s)  
*Initializes the compass.*
- int INV\_EXPORT [inv\\_icm20602\\_check\\_akm\\_self\\_test](#) (struct [inv\\_icm20602](#) \*s)  
*Self test for the compass.*
- int INV\_EXPORT [inv\\_icm20602\\_read\\_akm\\_scale](#) (struct [inv\\_icm20602](#) \*s, int \*scale)  
*Reads the scale of the compass.*
- int INV\_EXPORT [inv\\_icm20602\\_suspend\\_akm](#) (struct [inv\\_icm20602](#) \*s)  
*Stops the compass.*
- int INV\_EXPORT [inv\\_icm20602\\_resume\\_akm](#) (struct [inv\\_icm20602](#) \*s)  
*Starts the compass.*
- char INV\_EXPORT [inv\\_icm20602\\_compass\\_getstate](#) (struct [inv\\_icm20602](#) \*s)  
*Get compass power status.*
- int INV\_EXPORT [inv\\_icm20602\\_get\\_compass\\_data](#) (struct [inv\\_icm20602](#) \*s, const unsigned char \*packet, unsigned char \*raw\_compass)  
*Parse compass data packet.*
- int INV\_EXPORT [inv\\_icm20602\\_get\\_compass\\_bytes](#) (struct [inv\\_icm20602](#) \*s)  
*Get data packet size according to comass type.*

### 7.15.1 Detailed Description

Low-level lcm20602 aux sensor access.

## 7.15.2 Enumeration Type Documentation

### 7.15.2.1 enum inv\_icm20602\_compass\_id

Supported auxiliary compass identifier.

Enumerator

**INV\_ICM20602\_COMPASS\_ID\_NONE** no compass  
**INV\_ICM20602\_COMPASS\_ID\_AK09911** AKM AK09911.  
**INV\_ICM20602\_COMPASS\_ID\_AK09912** AKM AK09912.

## 7.15.3 Function Documentation

### 7.15.3.1 int INV\_EXPORT inv\_icm20602\_check\_akm\_self\_test ( struct inv\_icm20602 \* s )

Self test for the compass.

Returns

0 in case of success, -1 for any error

### 7.15.3.2 char INV\_EXPORT inv\_icm20602\_compass\_getstate ( struct inv\_icm20602 \* s )

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

### 7.15.3.3 int INV\_EXPORT inv\_icm20602\_get\_compass\_bytes ( struct inv\_icm20602 \* s )

Get data packet size according to compass type.

Returns

size in bytes

### 7.15.3.4 int INV\_EXPORT inv\_icm20602\_get\_compass\_data ( struct inv\_icm20602 \* s, const unsigned char \* packet, unsigned char \* raw\_compass )

Parse compass data packet.



## Parameters

in	<i>data</i>	packet pointer to compass data packet
in	<i>raw_compass</i>	pointer to raw compass. big endian 2 bytes for each axis.

## Returns

0 in case of success, -1 for any error

7.15.3.5 `int INV_EXPORT inv_icm20602_read_akm_scale ( struct inv_icm20602 * s, int * scale )`

Reads the scale of the compass.

## Parameters

out	<i>scale</i>	pointer to recuperate the scale
-----	--------------	---------------------------------

## Returns

0 in case of success, -1 for any error

7.15.3.6 `void INV_EXPORT inv_icm20602_register_aux_compass ( struct inv_icm20602 * s, enum inv_icm20602_compass_id compass_id, uint8_t compass_i2c_addr )`

Register AUX compass.

Will only set internal states and won't perform any transaction on the bus. Must be called before [inv\\_icm20602\\_initialize\(\)](#).

## Parameters

in	<i>compass_id</i>	Compass ID
in	<i>compass_i2c_addr</i>	Compass I2C address

## Returns

0 on success, negative value on error

7.15.3.7 `int INV_EXPORT inv_icm20602_resume_akm ( struct inv_icm20602 * s )`

Starts the compass.

## Returns

0 in case of success, -1 for any error

7.15.3.8 `int INV_EXPORT inv_icm20602_setup_compass_akm ( struct inv_icm20602 * s )`

Initializes the compass.

**Returns**

0 in case of success, -1 for any error

7.15.3.9 `int INV_EXPORT inv_icm20602_suspend_akm ( struct inv_icm20602 * s )`

Stops the compass.

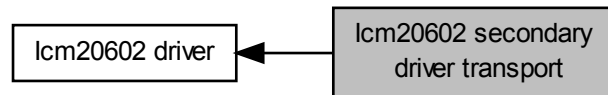
**Returns**

0 in case of success, -1 for any error

## 7.16 lcm20602 secondary driver transport

Low-level lcm20602 secondary interface access.

Collaboration diagram for lcm20602 secondary driver transport:



### Macros

- #define `COMPASS_I2C_SLV_READ` 0  
Secondary I2C channel usage :

### Functions

- void `inv_icm20602_init_secondary` (struct `inv_icm20602` \*s)  
*Initializes the register for the i2c communication.*
- int `inv_icm20602_read_secondary` (struct `inv_icm20602` \*s, int index, unsigned char addr, unsigned char reg, char len)  
*Reads data in i2c a secondary device.*
- int `inv_icm20602_execute_read_secondary` (struct `inv_icm20602` \*s, int index, unsigned char addr, int reg, int len, unsigned char \*d)  
*Reads data in i2c a secondary device directly.*
- int `inv_icm20602_write_secondary` (struct `inv_icm20602` \*s, int index, unsigned char addr, unsigned char reg, char v)  
*Writes data in i2c a secondary device.*
- int `inv_icm20602_execute_write_secondary` (struct `inv_icm20602` \*s, int index, unsigned char addr, int reg, unsigned char v)  
*Writes data in i2c a secondary device directly.*
- int `inv_icm20602_secondary_stop_channel` (struct `inv_icm20602` \*s, int index)  
*Stop one secondary I2C channel by writing 0 in its control register.*
- int `inv_icm20602_secondary_enable_i2c` (struct `inv_icm20602` \*s)  
*Enable secondary I2C interface.*
- int `inv_icm20602_secondary_disable_i2c` (struct `inv_icm20602` \*s)  
*Stop secondary I2C interface.*

#### 7.16.1 Detailed Description

Low-level lcm20602 secondary interface access.

## 7.16.2 Macro Definition Documentation

### 7.16.2.1 #define COMPASS\_I2C\_SLV\_READ 0

Secondary I2C channel usage :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register

## 7.16.3 Function Documentation

### 7.16.3.1 int inv\_icm20602\_execute\_read\_secondary ( struct inv\_icm20602 \* *s*, int *index*, unsigned char *addr*, int *reg*, int *len*, unsigned char \* *d* )

Reads data in i2c a secondary device directly.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be read on the secondary device
in	<i>len</i>	Size of data to be read
out	<i>d</i>	pointer to the data to be read

#### Returns

0 in case of success, -1 for any error

### 7.16.3.2 int inv\_icm20602\_execute\_write\_secondary ( struct inv\_icm20602 \* *s*, int *index*, unsigned char *addr*, int *reg*, unsigned char *v* )

Writes data in i2c a secondary device directly.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be write on the secondary device
in	<i>v</i>	the data to be written

#### Returns

0 in case of success, -1 for any error

7.16.3.3 `int inv_icm20602_read_secondary ( struct inv_icm20602 * s, int index, unsigned char addr, unsigned char reg, char len )`

Reads data in i2c a secondary device.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	I2c address slave of the secondary slave
in	<i>reg</i>	the register to be read on the secondary device
in	<i>len</i>	Size of data to be read

#### Returns

0 in case of success, -1 for any error

7.16.3.4 `int inv_icm20602_secondary_disable_i2c ( struct inv_icm20602 * s )`

Stop secondary I2C interface.

#### Returns

0 in case of success, -1 for any error

#### Warning

It stops all I2C transactions, whatever the channel status

7.16.3.5 `int inv_icm20602_secondary_enable_i2c ( struct inv_icm20602 * s )`

Enable secondary I2C interface.

#### Returns

0 in case of success, -1 for any error

7.16.3.6 `int inv_icm20602_secondary_stop_channel ( struct inv_icm20602 * s, int index )`

Stop one secondary I2C channel by writing 0 in its control register.

#### Parameters

in	<i>index</i>	the channel id to be stopped
----	--------------	------------------------------

**Returns**

0 in case of success, -1 for any error

**Warning**

It does not stop I2C secondary interface, just one channel

**7.16.3.7** `int inv_icm20602_write_secondary ( struct inv_icm20602 * s, int index, unsigned char addr, unsigned char reg, char v )`

Writes data in i2c a secondary device.

**Parameters**

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be write on the secondary device
in	<i>v</i>	the data to be written

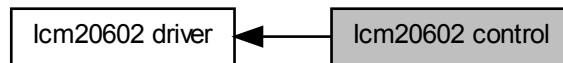
**Returns**

0 in case of success, -1 for any error

## 7.17 lcm20602 control

Low-level function to control a lcm20602 device.

Collaboration diagram for lcm20602 control:



### Classes

- struct [inv\\_icm20602\\_fifo\\_states](#)  
*Check and retrieve for new data.*

### Enumerations

### Functions

- int INV\_EXPORT [inv\\_icm20602\\_enable\\_mems](#) (struct [inv\\_icm20602](#) \*s, int bit\_mask, uint16\_t smplr\_t, ← divider)  
*Enables accel and/or gyro and/or pressure if integrated with gyro and accel.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_sensor\\_period](#) (struct [inv\\_icm20602](#) \*s, enum [inv\\_icm20602\\_sensor](#) sensor, uint16\_t delayInMs)  
*Sets the odr for a sensor.*
- int INV\_EXPORT [inv\\_icm20602\\_is\\_sensor\\_enabled](#) (struct [inv\\_icm20602](#) \*s, enum [inv\\_icm20602\\_sensor](#) sensor)  
*Knows if the sensor is enabled or disabled.*
- int INV\_EXPORT [inv\\_icm20602\\_enable\\_sensor](#) (struct [inv\\_icm20602](#) \*s, enum [inv\\_icm20602\\_sensor](#) sensor, uint8\_t enable)  
*Enables / disables a sensor.*
- int INV\_EXPORT [inv\\_icm20602\\_configure\\_accel\\_wom](#) (struct [inv\\_icm20602](#) \*s, uint8\_t wom\_threshold)  
*Configures accel WOM.*
- int INV\_EXPORT [inv\\_icm20602\\_poll\\_sensor\\_data\\_reg](#) (struct [inv\\_icm20602](#) \*s, int16\_t acc\_data[3], int16\_t ← \*temp\_data, int16\_t gyro\_data[3])  
*Check and retrieve for new data.*
- int INV\_EXPORT [inv\\_icm20602\\_poll\\_fifo\\_data\\_setup](#) (struct [inv\\_icm20602](#) \*s, struct [inv\\_icm20602\\_fifo\\_← states](#) \*states, uint8\_t int\_status)  
*Initializes states for FIFO reading and parsing.*
- int INV\_EXPORT [inv\\_icm20602\\_poll\\_fifo\\_data](#) (struct [inv\\_icm20602](#) \*s, struct [inv\\_icm20602\\_fifo\\_states](#) \*states, int16\_t acc\_data[3], int16\_t \*temp\_data, int16\_t gyro\_data[3], int16\_t compass\_data[3])  
*Read one packet from the FIFO (packet corresponding to data for all current active sensor)*
- int INV\_EXPORT [inv\\_icm20602\\_has\\_data\\_ready](#) (struct [inv\\_icm20602](#) \*s)  
*Read interrupt status and check for DDRY flag.*
- int INV\_EXPORT [inv\\_icm20602\\_get\\_int\\_status](#) (struct [inv\\_icm20602](#) \*s, uint8\_t \*int\_status)

- Read interrupt status and return its value.*
- int INV\_EXPORT [inv\\_icm20602\\_check\\_drdy](#) (struct [inv\\_icm20602](#) \*s, uint8\_t int\_status)  
*Check for DRDY flag in INT register value.*
- int INV\_EXPORT [inv\\_icm20602\\_check\\_wom\\_status](#) (struct [inv\\_icm20602](#) \*s, uint8\_t int\_status)  
*Check for WOM bits in INT register value.*
- int INV\_EXPORT [inv\\_icm20602\\_disable\\_fifo](#) (struct [inv\\_icm20602](#) \*s)  
*Disable FIFO.*
- int INV\_EXPORT [inv\\_icm20602\\_enable\\_fifo](#) (struct [inv\\_icm20602](#) \*s, int bit\_mask)  
*Enable FIFO.*
- int INV\_EXPORT [inv\\_icm20602\\_reset\\_fifo](#) (struct [inv\\_icm20602](#) \*s)  
*Reset FIFO.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_slave\\_compass\\_id](#) (struct [inv\\_icm20602](#) \*s)  
*Initialize AUX I2C and Compass.*
- int INV\_EXPORT [inv\\_icm20602\\_poll\\_ois\\_gyro\\_data](#) (struct [inv\\_icm20602](#) \*s, int16\_t gyro\_data[3])  
*Poll OIS gyro data.*
- int INV\_EXPORT [inv\\_icm20602\\_poll\\_delay\\_count](#) (struct [inv\\_icm20602](#) \*s, int16\_t \*delay\_count)  
*Poll EIS flag and delay counter.*
- int INV\_EXPORT [inv\\_icm20602\\_all\\_sensors\\_off](#) (struct [inv\\_icm20602](#) \*s)  
*test if all sensors are off*
- int [inv\\_icm20602\\_run\\_selftest](#) (struct [inv\\_icm20602](#) \*s)  
*Perform hardware self-test for Accel, Gyro and Compass.*
- void INV\_EXPORT [inv\\_icm20602\\_get\\_st\\_bias](#) (struct [inv\\_icm20602](#) \*s, int \*st\_bias)  
*Retrieve bias collected by self-test.*
- void [inv\\_icm20602\\_set\\_st\\_bias](#) (struct [inv\\_icm20602](#) \*s, int \*st\_bias)  
*Apply bias.*

### 7.17.1 Detailed Description

Low-level function to control a Icm20602 device.

### 7.17.2 Function Documentation

#### 7.17.2.1 int INV\_EXPORT inv\_icm20602\_all\_sensors\_off ( struct inv\_icm20602 \* s )

test if all sensors are off

##### Returns

true if all sensors are off, false otherwise

#### 7.17.2.2 int INV\_EXPORT inv\_icm20602\_check\_drdy ( struct inv\_icm20602 \* s, uint8\_t int\_status )

Check for DRDY flag in INT register value.

##### Parameters

in	<i>int_status</i>	INT status register value
----	-------------------	---------------------------



**Returns**

1 if DDRDY is set, 0 otherwise

7.17.2.3 `int INV_EXPORT inv_icm20602_check_wom_status ( struct inv_icm20602 * s, uint8_t int_status )`

Check for WOM bits in INT register value.

**Parameters**

in	<i>int_status</i>	INT status register value
----	-------------------	---------------------------

**Returns**

bit mask corresponding to x, y, z axis that caused WOM (0x01: x axis, 0x02: y axis, 0x01: z axis)

7.17.2.4 `int INV_EXPORT inv_icm20602_configure_accel_wom ( struct inv_icm20602 * s, uint8_t wom_threshold )`

Configure accel WOM.

**Parameters**

in	<i>wom_threshold</i>	threshold value for X,Y,Z axis that should trigger a WOM interrupt (0 to disable WOM)
----	----------------------	---

**Returns**

0 on success, negative value on error.

7.17.2.5 `int INV_EXPORT inv_icm20602_disable_fifo ( struct inv_icm20602 * s )`

Disable FIFO.

**Returns**

0 on success, negative value on error.

7.17.2.6 `int INV_EXPORT inv_icm20602_enable_fifo ( struct inv_icm20602 * s, int bit_mask )`

Enable FIFO.

**Parameters**

in	<i>bit_mask</i>	A mask of sensor to push to FIFO.
----	-----------------	-----------------------------------

**Returns**

0 on success, negative value on error.

7.17.2.7 `int INV_EXPORT inv_icm20602_enable_mems ( struct inv_icm20602 * s, int bit_mask, uint16_t smplr_t_divider )`

Enables accel and/or gyro and/or pressure if integrated with gyro and accel.

**Parameters**

in	<i>bit_mask</i>	A mask where 2 means turn on accel, 1 means turn on gyro, 4 is for pressure. By default, this only turns on a sensor if all sensors are off otherwise the DMP controls this register including turning off a sensor. To override this behavior add in a mask of 128.
in	<i>smplr_t_divider</i>	The divider which was applied to internal sample rate based on field <i>sample_rate</i> from <i>base_driver_t</i> to get minimum ODR for accel and gyro

**Returns**

0 on success, negative value on error

7.17.2.8 `int INV_EXPORT inv_icm20602_enable_sensor ( struct inv_icm20602 * s, enum inv_icm20602_sensor sensor, uint8_t enable )`

Enables / disables a sensor.

**Parameters**

in	<i>androidSensor</i>	Sensor Identity
in	<i>enable</i>	0=off, 1=on

**Returns**

0 in case of success, negative value on error

7.17.2.9 `int INV_EXPORT inv_icm20602_get_int_status ( struct inv_icm20602 * s, uint8_t * int_status )`

Read interrupt status and return its value.

**Parameters**

out	<i>int_status</i>	INT status register value
-----	-------------------	---------------------------

**Returns**

0 on success, negative value on error

7.17.2.10 void INV\_EXPORT inv\_icm20602\_get\_st\_bias ( struct inv\_icm20602 \* s, int \* st\_bias )

Retrieve bias collected by self-test.

#### Parameters

out	st_bias	bias scaled by 2 <sup>16</sup> , accel is gee and gyro is dps. The buffer will be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z
-----	---------	--

7.17.2.11 int INV\_EXPORT inv\_icm20602\_has\_data\_ready ( struct inv\_icm20602 \* s )

Read interrupt status and check for DDRY flag.

#### Returns

1 if DDRY inteript is set, 0 otherwise and negative value on error

7.17.2.12 int INV\_EXPORT inv\_icm20602\_is\_sensor\_enabled ( struct inv\_icm20602 \* s, enum inv\_icm20602\_sensor sensor )

Knows if the sensor is enabled or disbaled.

#### Parameters

in	sensor	the sensor is enabled or not
----	--------	------------------------------

#### Returns

1 if active, 0 otherwise

7.17.2.13 int INV\_EXPORT inv\_icm20602\_poll\_delay\_count ( struct inv\_icm20602 \* s, int16\_t \* delay\_count )

Poll EIS flag and delay counter.

#### Parameters

out	delay_count	delay time in us between the FSYNC event (before the gyro data event) and the gyro data event
-----	-------------	---

#### Returns

0x8 if FSYNC event reported, 0 otherwise, negative value on error.

7.17.2.14 `int INV_EXPORT inv_icm20602_poll_fifo_data ( struct inv_icm20602 * s, struct inv_icm20602_fifo_states * states, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3], int16_t compass_data[3] )`

Read one packet from the FIFO (packet corresponding to data for all current active sensor)

#### Parameters

in	<i>states</i>	placeholder to FIFO states
out	<i>acc_data</i>	raw gyro data
out	<i>temp_data</i>	raw temp data
out	<i>gyro_data</i>	raw gyro data
out	<i>compass_data</i>	raw compass data

#### Returns

0 : FIFO empty bit0 : gyro data output set bit1 : acc data output set bit2 : compass output set < 0 : error

7.17.2.15 `int INV_EXPORT inv_icm20602_poll_fifo_data_setup ( struct inv_icm20602 * s, struct inv_icm20602_fifo_states * states, uint8_t int_status )`

Initializes states for FIFO reading and parsing.

#### Parameters

in	<i>states</i>	placeholder to FIFO states
in	<i>int_status</i>	int status register value (used to check for overflow)

#### Returns

0 for success, 1 for overflow detected, negative value for other errors

7.17.2.16 `int INV_EXPORT inv_icm20602_poll_ois_gyro_data ( struct inv_icm20602 * s, int16_t gyro_data[3] )`

Poll OIS gyro data.

#### Returns

0x10 if OIS data reported, 0 otherwise, negative value on error.

7.17.2.17 `int INV_EXPORT inv_icm20602_poll_sensor_data_reg ( struct inv_icm20602 * s, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3] )`

Check and retrieve for new data.

## Parameters

out	<i>acc_data</i>	raw gyro data
out	<i>temp_data</i>	raw temp data
out	<i>gyro_data</i>	raw gyro data

## Returns

0 : not data ready 0x01 : gyro data output set 0x02 : acc data output set 0x03 : gyro and acc data output set  
 < 0 : error

7.17.2.18 int INV\_EXPORT inv\_icm20602\_reset\_fifo ( struct inv\_icm20602 \* s )

Reset FIFO.

## Returns

0 on success, negative value on error.

7.17.2.19 int inv\_icm20602\_run\_selftest ( struct inv\_icm20602 \* s )

Perform hardware self-test for Accel, Gyro and Compass.

## Parameters

in	<i>None</i>	
----	-------------	--

## Returns

COMPASS\_SUCCESS<<2 | ACCEL\_SUCCESS<<1 | GYRO\_SUCCESS so 3

7.17.2.20 int INV\_EXPORT inv\_icm20602\_set\_sensor\_period ( struct inv\_icm20602 \* s, enum inv\_icm20602\_sensor sensor, uint16\_t delayInMs )

Sets the odr for a sensor.

## Parameters

in	<i>sensor</i>	Sensor Identity
in	<i>delayInMs</i>	the delay between two values in ms

## Returns

0 in case of success, -1 for any error

7.17.2.21 `int INV_EXPORT inv_icm20602_set_slave_compass_id ( struct inv_icm20602 * s )`

Initialize AUX I2C and Compass.

#### Returns

0 on success, negative value on error.

7.17.2.22 `void inv_icm20602_set_st_bias ( struct inv_icm20602 * s, int * st_bias )`

Apply bias.

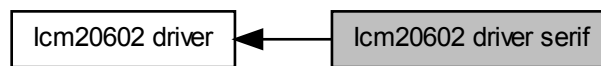
#### Parameters

in	<i>st_bias</i>	bias scaled by $2^{16}$ , accel is gee and gyro is dps. The buffer should be be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z
----	----------------	--

## 7.18 lcm20602 driver serif

Interface for low-level serial (I2C/SPI) access.

Collaboration diagram for lcm20602 driver serif:



### Classes

- struct [inv\\_lcm20602\\_serif](#)  
*basesensor serial interface*

### 7.18.1 Detailed Description

Interface for low-level serial (I2C/SPI) access.

## 7.19 Icm20602 driver setup

Low-level function to setup an Icm20602 device.

Collaboration diagram for Icm20602 driver setup:



### Functions

- int INV\_EXPORT [inv\\_icm20602\\_get\\_whoami](#) (struct [inv\\_icm20602](#) \*s, uint8\_t \*whoami)  
*return WHOAMI value*
- int INV\_EXPORT [inv\\_icm20602\\_get\\_chip\\_info](#) (struct [inv\\_icm20602](#) \*s, uint8\_t chip\_info[3])  
*return WHOAMI value*
- int INV\_EXPORT [inv\\_icm20602\\_initialize](#) (struct [inv\\_icm20602](#) \*s)  
*Initialize the device.*
- int INV\_EXPORT [inv\\_icm20602\\_soft\\_reset](#) (struct [inv\\_icm20602](#) \*s)  
*Perform a soft reset of the device.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_divider](#) (struct [inv\\_icm20602](#) \*s, uint8\_t idiv)  
*Set the mpu sample rate.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_chip\\_power\\_state](#) (struct [inv\\_icm20602](#) \*s, uint8\_t func, uint8\_t on\_off)  
*Sets the power state of the Ivory chip loop.*
- uint8\_t INV\_EXPORT [inv\\_icm20602\\_get\\_chip\\_power\\_state](#) (struct [inv\\_icm20602](#) \*s)  
*Current wake status of the Mems chip.*
- uint16\_t INV\_EXPORT [inv\\_icm20602\\_get\\_chip\\_base\\_sample\\_rate](#) (struct [inv\\_icm20602](#) \*s)  
*Get internal sample rate currently configured.*
- int INV\_EXPORT [inv\\_icm20602\\_accel\\_fsr\\_2\\_reg](#) (int32\_t fsr)  
*Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.*
- int INV\_EXPORT [inv\\_icm20602\\_reg\\_2\\_accel\\_fsr](#) (uint8\_t reg)  
*Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_accel\\_fullscale](#) (struct [inv\\_icm20602](#) \*s, int level)  
*Sets fullscale range of accel in hardware.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_accel\\_ois\\_fullscale](#) (struct [inv\\_icm20602](#) \*s, int level)  
*Sets fullscale range of OIS accel in hardware.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_accel\\_bandwidth](#) (struct [inv\\_icm20602](#) \*s, int level)  
*Sets bandwidth range of accel in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20602\\_get\\_accel\\_fullscale](#) (struct [inv\\_icm20602](#) \*s)  
*Returns fullscale range of accelerometer in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20602\\_get\\_accel\\_ois\\_fullscale](#) (struct [inv\\_icm20602](#) \*s)  
*Returns fullscale range of OIS accelerometer in hardware.*
- uint16\_t INV\_EXPORT [inv\\_icm20602\\_get\\_accel\\_bandwidth](#) (struct [inv\\_icm20602](#) \*s)



- Returns bandwidth range of accelerometer in hardware.*
- int INV\_EXPORT [inv\\_icm20602\\_gyro\\_fsr\\_2\\_reg](#) (int32\_t fsr)
 

*Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)*
- int INV\_EXPORT [inv\\_icm20602\\_reg\\_2\\_gyro\\_fsr](#) (uint8\_t reg)
 

*Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_gyro\\_fullscale](#) (struct [inv\\_icm20602](#) \*s, int level)
 

*Sets fullscale range of gyro in hardware.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_gyro\\_ois\\_fullscale](#) (struct [inv\\_icm20602](#) \*s, int level)
 

*Sets fullscale range of OIS gyro in hardware.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_gyro\\_bandwidth](#) (struct [inv\\_icm20602](#) \*s, int level)
 

*Sets bandwidth range of gyro in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20602\\_get\\_gyro\\_fullscale](#) (struct [inv\\_icm20602](#) \*s)
 

*Returns fullscale range of gyroscope in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20602\\_get\\_gyro\\_ois\\_fullscale](#) (struct [inv\\_icm20602](#) \*s)
 

*Returns fullscale range of OIS gyroscope in hardware.*
- uint16\_t INV\_EXPORT [inv\\_icm20602\\_get\\_gyro\\_bandwidth](#) (struct [inv\\_icm20602](#) \*s)
 

*Returns bandwidth range of gyroscope in hardware.*
- int INV\_EXPORT [inv\\_icm20602\\_is\\_advanced\\_features\\_supported](#) (void)
 

*Get the available features according the base sensor device.*
- int INV\_EXPORT [inv\\_icm20602\\_set\\_fsycn\\_bit\\_location](#) (struct [inv\\_icm20602](#) \*s, int bit\_location)
 

*Set FSYN bit location.*

### 7.19.1 Detailed Description

Low-level function to setup an Icm20602 device.

### 7.19.2 Function Documentation

#### 7.19.2.1 int INV\_EXPORT inv\_icm20602\_accel\_fsr\_2\_reg ( int32\_t fsr )

Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.

#### Returns

internal register value for FSR configuration

#### 7.19.2.2 uint16\_t INV\_EXPORT inv\_icm20602\_get\_accel\_bandwidth ( struct inv\_icm20602 \* s )

Returns bandwidth range of accelerometer in hardware.

#### Returns

the bandwidth range

7.19.2.3 `uint8_t INV_EXPORT inv_icm20602_get_accel_fullscale ( struct inv_icm20602 * s )`

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.19.2.4 `uint8_t INV_EXPORT inv_icm20602_get_accel_ois_fullscale ( struct inv_icm20602 * s )`

Returns fullscale range of OIS accelerometer in hardware.

Returns

the fullscale range

7.19.2.5 `uint16_t INV_EXPORT inv_icm20602_get_chip_base_sample_rate ( struct inv_icm20602 * s )`

Get internal sample rate currently configured.

Returns

Internal sample rate currently configured in MEMS registers in Hz

7.19.2.6 `int INV_EXPORT inv_icm20602_get_chip_info ( struct inv_icm20602 * s, uint8_t chip_info[3] )`

return WHOAMI value

Parameters

out	<i>chip_info</i> [0]	WHOAMI for device
out	<i>chip_info</i> [1]	MANUFACTURER_ID for device
out	<i>chip_info</i> [2]	CHIP_ID for device

Returns

0 on success, negative value on error

7.19.2.7 `uint8_t INV_EXPORT inv_icm20602_get_chip_power_state ( struct inv_icm20602 * s )`

Current wake status of the Mems chip.

Returns

the wake status

7.19.2.8 `uint16_t INV_EXPORT inv_icm20602_get_gyro_bandwidth ( struct inv_icm20602 * s )`

Returns bandwidth range of gyroscope in hardware.

Returns

the bandwidth range

7.19.2.9 `uint8_t INV_EXPORT inv_icm20602_get_gyro_fullscale ( struct inv_icm20602 * s )`

Returns fullscale range of gyroscope in hardware.

Returns

the fullscale range

7.19.2.10 `uint8_t INV_EXPORT inv_icm20602_get_gyro_ois_fullscale ( struct inv_icm20602 * s )`

Returns fullscale range of OIS gyroscope in hardware.

Returns

the fullscale range

7.19.2.11 `int INV_EXPORT inv_icm20602_get_whoami ( struct inv_icm20602 * s, uint8_t * whoami )`

return WHOAMI value

Parameters

out	whoami	WHOAMI for device
-----	--------	-------------------

Returns

0 on success, negative value on error

7.19.2.12 `int INV_EXPORT inv_icm20602_gyro_fsr_2_reg ( int32_t fsr )`

Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)

Returns

internal register value for FSR configuration

7.19.2.13 `int INV_EXPORT inv_icm20602_initialize ( struct inv_icm20602 * s )`

Initialize the device.

#### Returns

0 on success, negative value on error.

7.19.2.14 `int INV_EXPORT inv_icm20602_is_advanced_features_supported ( void )`

Get the available features according the base sensor device.

#### Parameters

<i>in</i>	<i>in</i>	0: No advanced features supported 1: OIS sensor and FSYNC behavior supported (for icm20690 only)
-----------	-----------	--

7.19.2.15 `int INV_EXPORT inv_icm20602_reg_2_accel_fsr ( uint8_t reg )`

Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),..

#### Returns

FSR value in mg

7.19.2.16 `int INV_EXPORT inv_icm20602_reg_2_gyro_fsr ( uint8_t reg )`

Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)

#### Returns

FSR value in dps

7.19.2.17 `int INV_EXPORT inv_icm20602_set_accel_bandwidth ( struct inv_icm20602 * s, int level )`

Sets bandwidth range of accel in hardware.

#### Parameters

<i>in</i>	<i>level</i>	See mpu_accel_bw.
-----------	--------------	-------------------

**Returns**

0 on success, negative value on error.

7.19.2.18 `int INV_EXPORT inv_icm20602_set_accel_fullscale ( struct inv_icm20602 * s, int level )`

Sets fullscale range of accel in hardware.

**Parameters**

in	<i>level</i>	See mpu_accel_fs.
----	--------------	-------------------

**Returns**

0 on success, negative value on error.

7.19.2.19 `int INV_EXPORT inv_icm20602_set_accel_ois_fullscale ( struct inv_icm20602 * s, int level )`

Sets fullscale range of OIS accel in hardware.

**Parameters**

in	<i>level</i>	See mpu_accel_ois_fs.
----	--------------	-----------------------

**Returns**

0 on success, negative value on error.

7.19.2.20 `int INV_EXPORT inv_icm20602_set_chip_power_state ( struct inv_icm20602 * s, uint8_t func, uint8_t on_off )`

Sets the power state of the Ivory chip loop.

**Parameters**

in	<i>func</i>	CHIP_AWAKE, CHIP_LP_ENABLE
in	<i>on_off</i>	The functions are enabled if previously disabled and disabled if previously enabled based on the value of On/Off.

**Returns**

0 on success, negative value on error.

7.19.2.21 `int INV_EXPORT inv_icm20602_set_divider ( struct inv_icm20602 * s, uint8_t idiv )`

Set the mpu sample rate.

**Returns**

Value written to MPUREG\_SMPLRT\_DIV register.

7.19.2.22 `int INV_EXPORT inv_icm20602_set_fsycn_bit_location ( struct inv_icm20602 * s, int bit_location )`

Set FSYNC bit location.

**Parameters**

<code>in</code>	<code>in</code>	0: Disable FSYNC pin data to be sampled 1: Enable FSYNC pin data to be sampled at TEMP_OUT_L[0]
-----------------	-----------------	---

7.19.2.23 `int INV_EXPORT inv_icm20602_set_gyro_bandwidth ( struct inv_icm20602 * s, int level )`

Sets bandwidth range of gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_bw.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

7.19.2.24 `int INV_EXPORT inv_icm20602_set_gyro_fullscale ( struct inv_icm20602 * s, int level )`

Sets fullscale range of gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_fs.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

7.19.2.25 `int INV_EXPORT inv_icm20602_set_gyro_ois_fullscale ( struct inv_icm20602 * s, int level )`

Sets fullscale range of OIS gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_fs.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

7.19.2.26 `int INV_EXPORT inv_icm20602_soft_reset ( struct inv_icm20602 * s )`

Perform a soft reset of the device.

**Returns**

0 on success, negative value on error.

## 7.20 Icm20602 driver transport

Low-level Icm20602 register access.

Collaboration diagram for Icm20602 driver transport:



### Functions

- int INV\_EXPORT [inv\\_icm20602\\_mems\\_write\\_reg](#) (struct [inv\\_icm20602](#) \*s, uint16\_t reg, uint32\_t length, const uint8\_t \*data)  
*Writes data from a register on mems.*
- int INV\_EXPORT [inv\\_icm20602\\_mems\\_write\\_reg\\_one](#) (struct [inv\\_icm20602](#) \*s, uint16\_t reg, uint8\_t data)  
*Writes a single byte of data from a register on mems with no power control.*
- int INV\_EXPORT [inv\\_icm20602\\_mems\\_read\\_reg](#) (struct [inv\\_icm20602](#) \*s, uint16\_t reg, uint32\_t length, uint8\_t \*data)  
*Reads data from a register on mems.*

### 7.20.1 Detailed Description

Low-level Icm20602 register access.

### 7.20.2 Function Documentation

7.20.2.1 int INV\_EXPORT [inv\\_icm20602\\_mems\\_read\\_reg](#) ( struct [inv\\_icm20602](#) \* s, uint16\_t *reg*, uint32\_t *length*, uint8\_t \* *data* )

Reads data from a register on mems.

#### Parameters

in	<i>reg</i>	register address
in	<i>length</i>	length of data
out	<i>data</i>	output data from the register

#### Returns

0 in case of success, -1 for any error



7.20.2.2 `int INV_EXPORT inv_icm20602_mems_write_reg ( struct inv_icm20602 * s, uint16_t reg, uint32_t length, const uint8_t * data )`

Writes data from a register on mems.

#### Parameters

in	<i>length</i>	number of byte to be written
out	<i>data</i>	output data from the register

#### Returns

0 in case of success, -1 for any error

7.20.2.3 `int INV_EXPORT inv_icm20602_mems_write_reg_one ( struct inv_icm20602 * s, uint16_t reg, uint8_t data )`

Writes a single byte of data from a register on mems with no power control.

#### Parameters

in	<i>reg</i>	Register address
out	<i>data</i>	Data to be written

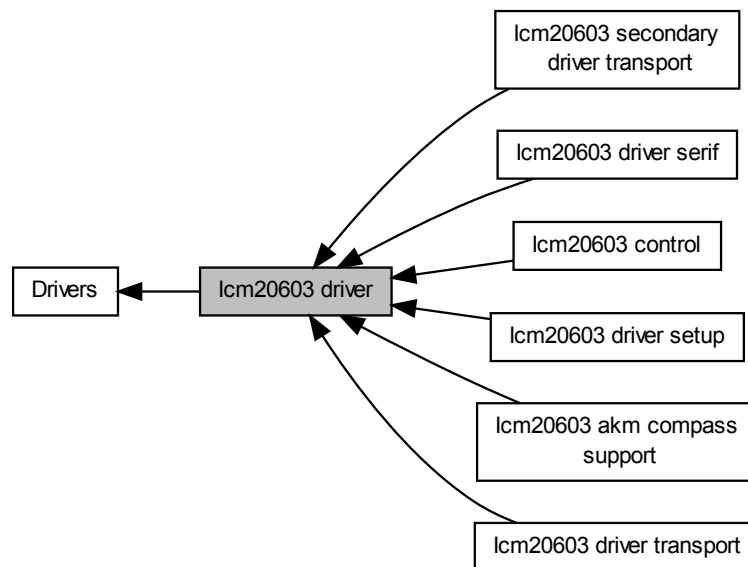
#### Returns

0 in case of success, -1 for any error

## 7.21 lcm20603 driver

Low-level driver for lcm20603 devices.

Collaboration diagram for lcm20603 driver:



### Modules

- [lcm20603 akm compass support](#)  
*Low-level lcm20603 aux sensor access.*
- [lcm20603 secondary driver transport](#)  
*Low-level lcm20603 secondary interface access.*
- [lcm20603 control](#)  
*Low-level function to control a lcm20603 device.*
- [lcm20603 driver serif](#)  
*Interface for low-level serial (I2C/SPI) access.*
- [lcm20603 driver setup](#)  
*Low-level function to setup an lcm20603 device.*
- [lcm20603 driver transport](#)  
*Low-level lcm20603 register access.*

### Classes

- struct [inv\\_lcm20603](#)  
*lcm20603 driver states definition.*

## Typedefs

- typedef struct `inv_icm20603` `inv_icm20603_t`  
*Icm20603 driver states definition.*

## Functions

- void `inv_icm20603_sleep` (int ms)  
*Hook for low-level system sleep() function to be implemented by upper layer.*
- void `inv_icm20603_sleep_us` (int us)  
*Hook for low-level high res system sleep() function to be implemented by upper layer ~ 100us resolution is sufficient.*
- uint64\_t `inv_icm20603_get_time_us` (void)  
*Hook for low-level system time() function to be implemented by upper layer.*
- uint64\_t `inv_icm20603_get_dataready_interrupt_time_us` (void)  
*Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get\_time\_us()*
- static void `inv_icm20603_reset_states` (struct `inv_icm20603` \*s, const struct `inv_icm20603_serif` \*serif)  
*Reset and initialize driver states.*
- static void `inv_icm20603_reset_states_serif_ois` (struct `inv_icm20603` \*s, const struct `inv_icm20603_serif` \*serif\_ois)  
*Register secondary SERIF object for OIS access Must be called after inv\_icm20603\_reset\_states()*

### 7.21.1 Detailed Description

Low-level driver for Icm20603 devices.

### 7.21.2 Function Documentation

#### 7.21.2.1 uint64\_t inv\_icm20603\_get\_dataready\_interrupt\_time\_us ( void )

Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get\_time\_us()

#### Returns

data ready interrupt timestamp in us

#### 7.21.2.2 uint64\_t inv\_icm20603\_get\_time\_us ( void )

Hook for low-level system time() function to be implemented by upper layer.

#### Returns

monotonic timestamp in us

#### 7.21.2.3 static void inv\_icm20603\_reset\_states ( struct inv\_icm20603 \* s, const struct inv\_icm20603\_serif \* serif ) [inline],[static]

Reset and initialize driver states.

**Parameters**

in	<i>s</i>	handle to driver states structure
in	<i>serif</i>	handle to SERIF object for underlying register access

**7.21.2.4** `static void inv_icm20603_reset_states_serif_ois ( struct inv_icm20603 * s, const struct inv_icm20603_serif * serif_ois ) [inline],[static]`

Register secondary SERIF object for OIS access Must be called after [inv\\_icm20603\\_reset\\_states\(\)](#)

**Parameters**

in	<i>s</i>	handle to driver states structure
in	<i>serif</i>	handle to SERIF object for underlying register access to OIS

**7.21.2.5** `void inv_icm20603_sleep ( int ms )`

Hook for low-level system sleep() function to be implemented by upper layer.

**Parameters**

in	<i>ms</i>	number of millisecond the calling thread should sleep
----	-----------	---

**7.21.2.6** `void inv_icm20603_sleep_us ( int us )`

Hook for low-level high res system sleep() function to be implemented by upper layer ~100us resolution is sufficient.

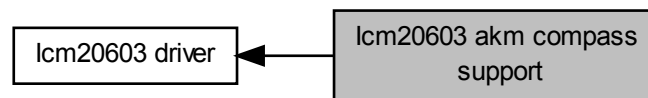
**Parameters**

in	<i>us</i>	number of us the calling thread should sleep
----	-----------	--

## 7.22 lcm20603 akm compass support

Low-level lcm20603 aux sensor access.

Collaboration diagram for lcm20603 akm compass support:



### Enumerations

### Functions

- void INV\_EXPORT [inv\\_icm20603\\_register\\_aux\\_compass](#) (struct [inv\\_icm20603](#) \*s, enum [inv\\_icm20603\\_compass\\_id](#) compass\_id, uint8\_t compass\_i2c\_addr)  
*Register AUX compass.*
- int INV\_EXPORT [inv\\_icm20603\\_is\\_compass\\_registered](#) (struct [inv\\_icm20603](#) \*s)  
*Return non-zero value is AUX compass was regitered, 0 if not If compass was registered but setup failed, will also return 0.*
- int INV\_EXPORT [inv\\_icm20603\\_setup\\_compass\\_akm](#) (struct [inv\\_icm20603](#) \*s)  
*Initializes the compass.*
- int INV\_EXPORT [inv\\_icm20603\\_check\\_akm\\_self\\_test](#) (struct [inv\\_icm20603](#) \*s)  
*Self test for the compass.*
- int INV\_EXPORT [inv\\_icm20603\\_read\\_akm\\_scale](#) (struct [inv\\_icm20603](#) \*s, int \*scale)  
*Reads the scale of the compass.*
- int INV\_EXPORT [inv\\_icm20603\\_suspend\\_akm](#) (struct [inv\\_icm20603](#) \*s)  
*Stops the compass.*
- int INV\_EXPORT [inv\\_icm20603\\_resume\\_akm](#) (struct [inv\\_icm20603](#) \*s)  
*Starts the compass.*
- char INV\_EXPORT [inv\\_icm20603\\_compass\\_getstate](#) (struct [inv\\_icm20603](#) \*s)  
*Get compass power status.*
- int INV\_EXPORT [inv\\_icm20603\\_get\\_compass\\_data](#) (struct [inv\\_icm20603](#) \*s, const unsigned char \*packet, unsigned char \*raw\_compass)  
*Parse compass data packet.*
- int INV\_EXPORT [inv\\_icm20603\\_get\\_compass\\_bytes](#) (struct [inv\\_icm20603](#) \*s)  
*Get data packet size according to comass type.*

### 7.22.1 Detailed Description

Low-level lcm20603 aux sensor access.

## 7.22.2 Enumeration Type Documentation

### 7.22.2.1 enum inv\_icm20603\_compass\_id

Supported auxiliary compass identifier.

Enumerator

**INV\_ICM20603\_COMPASS\_ID\_NONE** no compass  
**INV\_ICM20603\_COMPASS\_ID\_AK09911** AKM AK09911.  
**INV\_ICM20603\_COMPASS\_ID\_AK09912** AKM AK09912.

## 7.22.3 Function Documentation

### 7.22.3.1 int INV\_EXPORT inv\_icm20603\_check\_akm\_self\_test ( struct inv\_icm20603 \* s )

Self test for the compass.

Returns

0 in case of success, -1 for any error

### 7.22.3.2 char INV\_EXPORT inv\_icm20603\_compass\_getstate ( struct inv\_icm20603 \* s )

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

### 7.22.3.3 int INV\_EXPORT inv\_icm20603\_get\_compass\_bytes ( struct inv\_icm20603 \* s )

Get data packet size according to compass type.

Returns

size in bytes

### 7.22.3.4 int INV\_EXPORT inv\_icm20603\_get\_compass\_data ( struct inv\_icm20603 \* s, const unsigned char \* packet, unsigned char \* raw\_compass )

Parse compass data packet.

## Parameters

in	<i>data</i>	packet pointer to compass data packet
in	<i>raw_compass</i>	pointer to raw compass. big endian 2 bytes for each axis.

## Returns

0 in case of success, -1 for any error

7.22.3.5 `int INV_EXPORT inv_icm20603_read_akm_scale ( struct inv_icm20603 * s, int * scale )`

Reads the scale of the compass.

## Parameters

out	<i>scale</i>	pointer to recuperate the scale
-----	--------------	---------------------------------

## Returns

0 in case of success, -1 for any error

7.22.3.6 `void INV_EXPORT inv_icm20603_register_aux_compass ( struct inv_icm20603 * s, enum inv_icm20603_compass_id compass_id, uint8_t compass_i2c_addr )`

Register AUX compass.

Will only set internal states and won't perform any transaction on the bus. Must be called before [inv\\_icm20603\\_↔ initialize\(\)](#).

## Parameters

in	<i>compass_id</i>	Compass ID
in	<i>compass_i2c_addr</i>	Compass I2C address

## Returns

0 on success, negative value on error

7.22.3.7 `int INV_EXPORT inv_icm20603_resume_akm ( struct inv_icm20603 * s )`

Starts the compass.

## Returns

0 in case of success, -1 for any error

7.22.3.8 `int INV_EXPORT inv_icm20603_setup_compass_akm ( struct inv_icm20603 * s )`

Initializes the compass.

**Returns**

0 in case of success, -1 for any error

7.22.3.9 `int INV_EXPORT inv_icm20603_suspend_akm ( struct inv_icm20603 * s )`

Stops the compass.

**Returns**

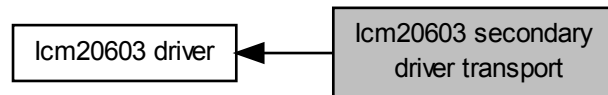
0 in case of success, -1 for any error



## 7.23 lcm20603 secondary driver transport

Low-level lcm20603 secondary interface access.

Collaboration diagram for lcm20603 secondary driver transport:



### Macros

- #define `COMPASS_I2C_SLV_READ` 0  
Secondary I2C channel usage :

### Functions

- void `inv_icm20603_init_secondary` (struct `inv_icm20603` \*s)  
*Initializes the register for the i2c communication.*
- int `inv_icm20603_read_secondary` (struct `inv_icm20603` \*s, int index, unsigned char addr, unsigned char reg, char len)  
*Reads data in i2c a secondary device.*
- int `inv_icm20603_execute_read_secondary` (struct `inv_icm20603` \*s, int index, unsigned char addr, int reg, int len, unsigned char \*d)  
*Reads data in i2c a secondary device directly.*
- int `inv_icm20603_write_secondary` (struct `inv_icm20603` \*s, int index, unsigned char addr, unsigned char reg, char v)  
*Writes data in i2c a secondary device.*
- int `inv_icm20603_execute_write_secondary` (struct `inv_icm20603` \*s, int index, unsigned char addr, int reg, unsigned char v)  
*Writes data in i2c a secondary device directly.*
- int `inv_icm20603_secondary_stop_channel` (struct `inv_icm20603` \*s, int index)  
*Stop one secondary I2C channel by writing 0 in its control register.*
- int `inv_icm20603_secondary_enable_i2c` (struct `inv_icm20603` \*s)  
*Enable secondary I2C interface.*
- int `inv_icm20603_secondary_disable_i2c` (struct `inv_icm20603` \*s)  
*Stop secondary I2C interface.*

#### 7.23.1 Detailed Description

Low-level lcm20603 secondary interface access.

## 7.23.2 Macro Definition Documentation

### 7.23.2.1 #define COMPASS\_I2C\_SLV\_READ 0

Secondary I2C channel usage :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register

## 7.23.3 Function Documentation

### 7.23.3.1 int inv\_icm20603\_execute\_read\_secondary ( struct inv\_icm20603 \* *s*, int *index*, unsigned char *addr*, int *reg*, int *len*, unsigned char \* *d* )

Reads data in i2c a secondary device directly.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be read on the secondary device
in	<i>len</i>	Size of data to be read
out	<i>d</i>	pointer to the data to be read

#### Returns

0 in case of success, -1 for any error

### 7.23.3.2 int inv\_icm20603\_execute\_write\_secondary ( struct inv\_icm20603 \* *s*, int *index*, unsigned char *addr*, int *reg*, unsigned char *v* )

Writes data in i2c a secondary device directly.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be write on the secondary device
in	<i>v</i>	the data to be written

#### Returns

0 in case of success, -1 for any error

7.23.3.3 `int inv_icm20603_read_secondary ( struct inv_icm20603 * s, int index, unsigned char addr, unsigned char reg, char len )`

Reads data in i2c a secondary device.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	I2c address slave of the secondary slave
in	<i>reg</i>	the register to be read on the secondary device
in	<i>len</i>	Size of data to be read

#### Returns

0 in case of success, -1 for any error

7.23.3.4 `int inv_icm20603_secondary_disable_i2c ( struct inv_icm20603 * s )`

Stop secondary I2C interface.

#### Returns

0 in case of success, -1 for any error

#### Warning

It stops all I2C transactions, whatever the channel status

7.23.3.5 `int inv_icm20603_secondary_enable_i2c ( struct inv_icm20603 * s )`

Enable secondary I2C interface.

#### Returns

0 in case of success, -1 for any error

7.23.3.6 `int inv_icm20603_secondary_stop_channel ( struct inv_icm20603 * s, int index )`

Stop one secondary I2C channel by writing 0 in its control register.

#### Parameters

in	<i>index</i>	the channel id to be stopped
----	--------------	------------------------------

**Returns**

0 in case of success, -1 for any error

**Warning**

It does not stop I2C secondary interface, just one channel

**7.23.3.7** `int inv_icm20603_write_secondary ( struct inv_icm20603 * s, int index, unsigned char addr, unsigned char reg, char v )`

Writes data in i2c a secondary device.

**Parameters**

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be write on the secondary device
in	<i>v</i>	the data to be written

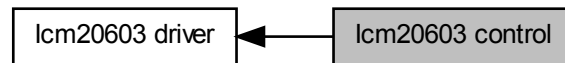
**Returns**

0 in case of success, -1 for any error

## 7.24 lcm20603 control

Low-level function to control a lcm20603 device.

Collaboration diagram for lcm20603 control:



### Classes

- struct [inv\\_icm20603\\_fifo\\_states](#)  
*Check and retrieve for new data.*

### Enumerations

### Functions

- int INV\_EXPORT [inv\\_icm20603\\_enable\\_mems](#) (struct [inv\\_icm20603](#) \*s, int bit\_mask, uint16\_t smplr\_t ← divider)  
*Enables accel and/or gyro and/or pressure if integrated with gyro and accel.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_sensor\\_period](#) (struct [inv\\_icm20603](#) \*s, enum [inv\\_icm20603\\_sensor](#) sensor, uint16\_t delayInMs)  
*Sets the odr for a sensor.*
- int INV\_EXPORT [inv\\_icm20603\\_is\\_sensor\\_enabled](#) (struct [inv\\_icm20603](#) \*s, enum [inv\\_icm20603\\_sensor](#) sensor)  
*Knows if the sensor is enabled or disabled.*
- int INV\_EXPORT [inv\\_icm20603\\_enable\\_sensor](#) (struct [inv\\_icm20603](#) \*s, enum [inv\\_icm20603\\_sensor](#) sensor, uint8\_t enable)  
*Enables / disables a sensor.*
- int INV\_EXPORT [inv\\_icm20603\\_configure\\_accel\\_wom](#) (struct [inv\\_icm20603](#) \*s, uint8\_t wom\_threshold)  
*Configures accel WOM.*
- int INV\_EXPORT [inv\\_icm20603\\_poll\\_sensor\\_data\\_reg](#) (struct [inv\\_icm20603](#) \*s, int16\_t acc\_data[3], int16\_t ← \*temp\_data, int16\_t gyro\_data[3])  
*Check and retrieve for new data.*
- int INV\_EXPORT [inv\\_icm20603\\_poll\\_fifo\\_data\\_setup](#) (struct [inv\\_icm20603](#) \*s, struct [inv\\_icm20603\\_fifo\\_ ← states](#) \*states, uint8\_t int\_status)  
*Initializes states for FIFO reading and parsing.*
- int INV\_EXPORT [inv\\_icm20603\\_poll\\_fifo\\_data](#) (struct [inv\\_icm20603](#) \*s, struct [inv\\_icm20603\\_fifo\\_states](#) \*states, int16\_t acc\_data[3], int16\_t \*temp\_data, int16\_t gyro\_data[3], int16\_t compass\_data[3])  
*Read one packet from the FIFO (packet corresponding to data for all current active sensor)*
- int INV\_EXPORT [inv\\_icm20603\\_has\\_data\\_ready](#) (struct [inv\\_icm20603](#) \*s)  
*Read interrupt status and check for DDRY flag.*
- int INV\_EXPORT [inv\\_icm20603\\_get\\_int\\_status](#) (struct [inv\\_icm20603](#) \*s, uint8\_t \*int\_status)

- Read interrupt status and return its value.*
- int INV\_EXPORT [inv\\_icm20603\\_check\\_drdy](#) (struct [inv\\_icm20603](#) \*s, uint8\_t int\_status)  
*Check for DRDY flag in INT register value.*
- int INV\_EXPORT [inv\\_icm20603\\_check\\_wom\\_status](#) (struct [inv\\_icm20603](#) \*s, uint8\_t int\_status)  
*Check for WOM bits in INT register value.*
- int INV\_EXPORT [inv\\_icm20603\\_disable\\_fifo](#) (struct [inv\\_icm20603](#) \*s)  
*Disable FIFO.*
- int INV\_EXPORT [inv\\_icm20603\\_enable\\_fifo](#) (struct [inv\\_icm20603](#) \*s, int bit\_mask)  
*Enable FIFO.*
- int INV\_EXPORT [inv\\_icm20603\\_reset\\_fifo](#) (struct [inv\\_icm20603](#) \*s)  
*Reset FIFO.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_slave\\_compass\\_id](#) (struct [inv\\_icm20603](#) \*s)  
*Initialize AUX I2C and Compass.*
- int INV\_EXPORT [inv\\_icm20603\\_poll\\_ois\\_gyro\\_data](#) (struct [inv\\_icm20603](#) \*s, int16\_t gyro\_data[3])  
*Poll OIS gyro data.*
- int INV\_EXPORT [inv\\_icm20603\\_poll\\_delay\\_count](#) (struct [inv\\_icm20603](#) \*s, int16\_t \*delay\_count)  
*Poll EIS flag and delay counter.*
- int INV\_EXPORT [inv\\_icm20603\\_all\\_sensors\\_off](#) (struct [inv\\_icm20603](#) \*s)  
*test if all sensors are off*
- int [inv\\_icm20603\\_run\\_selftest](#) (struct [inv\\_icm20603](#) \*s)  
*Perform hardware self-test for Accel, Gyro and Compass.*
- void INV\_EXPORT [inv\\_icm20603\\_get\\_st\\_bias](#) (struct [inv\\_icm20603](#) \*s, int \*st\_bias)  
*Retrieve bias collected by self-test.*
- void [inv\\_icm20603\\_set\\_st\\_bias](#) (struct [inv\\_icm20603](#) \*s, int \*st\_bias)  
*Apply bias.*

### 7.24.1 Detailed Description

Low-level function to control a Icm20603 device.

### 7.24.2 Function Documentation

#### 7.24.2.1 int INV\_EXPORT inv\_icm20603\_all\_sensors\_off ( struct inv\_icm20603 \* s )

test if all sensors are off

#### Returns

true if all sensors are off, false otherwise

#### 7.24.2.2 int INV\_EXPORT inv\_icm20603\_check\_drdy ( struct inv\_icm20603 \* s, uint8\_t int\_status )

Check for DRDY flag in INT register value.

#### Parameters

in	<i>int_status</i>	INT status register value
----	-------------------	---------------------------

**Returns**

1 if DDRDY is set, 0 otherwise

7.24.2.3 `int INV_EXPORT inv_icm20603_check_wom_status ( struct inv_icm20603 * s, uint8_t int_status )`

Check for WOM bits in INT register value.

**Parameters**

in	<i>int_status</i>	INT status register value
----	-------------------	---------------------------

**Returns**

bit mask corresponding to x, y, z axis that caused WOM (0x01: x axis, 0x02: y axis, 0x01: z axis)

7.24.2.4 `int INV_EXPORT inv_icm20603_configure_accel_wom ( struct inv_icm20603 * s, uint8_t wom_threshold )`

Configure accel WOM.

**Parameters**

in	<i>wom_threshold</i>	threshold value for X,Y,Z axis that should trigger a WOM interrupt (0 to disable WOM)
----	----------------------	---

**Returns**

0 on success, negative value on error.

7.24.2.5 `int INV_EXPORT inv_icm20603_disable_fifo ( struct inv_icm20603 * s )`

Disable FIFO.

**Returns**

0 on success, negative value on error.

7.24.2.6 `int INV_EXPORT inv_icm20603_enable_fifo ( struct inv_icm20603 * s, int bit_mask )`

Enable FIFO.

**Parameters**

in	<i>bit_mask</i>	A mask of sensor to push to FIFO.
----	-----------------	-----------------------------------

**Returns**

0 on success, negative value on error.

7.24.2.7 `int INV_EXPORT inv_icm20603_enable_mems ( struct inv_icm20603 * s, int bit_mask, uint16_t smplr_t_divider )`

Enables accel and/or gyro and/or pressure if integrated with gyro and accel.

**Parameters**

in	<i>bit_mask</i>	A mask where 2 means turn on accel, 1 means turn on gyro, 4 is for pressure. By default, this only turns on a sensor if all sensors are off otherwise the DMP controls this register including turning off a sensor. To override this behavior add in a mask of 128.
in	<i>smplr_t_divider</i>	The divider which was applied to internal sample rate based on field <i>sample_rate</i> from <i>base_driver_t</i> to get minimum ODR for accel and gyro

**Returns**

0 on success, negative value on error

7.24.2.8 `int INV_EXPORT inv_icm20603_enable_sensor ( struct inv_icm20603 * s, enum inv_icm20603_sensor sensor, uint8_t enable )`

Enables / disables a sensor.

**Parameters**

in	<i>androidSensor</i>	Sensor Identity
in	<i>enable</i>	0=off, 1=on

**Returns**

0 in case of success, negative value on error

7.24.2.9 `int INV_EXPORT inv_icm20603_get_int_status ( struct inv_icm20603 * s, uint8_t * int_status )`

Read interrupt status and return its value.

**Parameters**

out	<i>int_status</i>	INT status register value
-----	-------------------	---------------------------

**Returns**

0 on success, negative value on error



7.24.2.10 void INV\_EXPORT inv\_icm20603\_get\_st\_bias ( struct inv\_icm20603 \* s, int \* st\_bias )

Retrieve bias collected by self-test.

#### Parameters

out	st_bias	bias scaled by 2 <sup>16</sup> , accel is gee and gyro is dps. The buffer will be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z
-----	---------	--

7.24.2.11 int INV\_EXPORT inv\_icm20603\_has\_data\_ready ( struct inv\_icm20603 \* s )

Read interrupt status and check for DDRY flag.

#### Returns

1 if DDRY inteript is set, 0 otherwise and negative value on error

7.24.2.12 int INV\_EXPORT inv\_icm20603\_is\_sensor\_enabled ( struct inv\_icm20603 \* s, enum inv\_icm20603\_sensor sensor )

Knows if the sensor is enabled or disbaled.

#### Parameters

in	sensor	the sensor is enabled or not
----	--------	------------------------------

#### Returns

1 if active, 0 otherwise

7.24.2.13 int INV\_EXPORT inv\_icm20603\_poll\_delay\_count ( struct inv\_icm20603 \* s, int16\_t \* delay\_count )

Poll EIS flag and delay counter.

#### Parameters

out	delay_count	delay time in us between the FSYNC event (before the gyro data event) and the gyro data event
-----	-------------	---

#### Returns

0x8 if FSYNC event reported, 0 otherwise, negative value on error.

7.24.2.14 `int INV_EXPORT inv_icm20603_poll_fifo_data ( struct inv_icm20603 * s, struct inv_icm20603_fifo_states * states, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3], int16_t compass_data[3] )`

Read one packet from the FIFO (packet corresponding to data for all current active sensor)

#### Parameters

in	<i>states</i>	placeholder to FIFO states
out	<i>acc_data</i>	raw gyro data
out	<i>temp_data</i>	raw temp data
out	<i>gyro_data</i>	raw gyro data
out	<i>compass_data</i>	raw compass data

#### Returns

0 : FIFO empty bit0 : gyro data output set bit1 : acc data output set bit2 : compass output set < 0 : error

7.24.2.15 `int INV_EXPORT inv_icm20603_poll_fifo_data_setup ( struct inv_icm20603 * s, struct inv_icm20603_fifo_states * states, uint8_t int_status )`

Initializes states for FIFO reading and parsing.

#### Parameters

in	<i>states</i>	placeholder to FIFO states
in	<i>int_status</i>	int status register value (used to check for overflow)

#### Returns

0 for success, 1 for overflow detected, negative value for other errors

7.24.2.16 `int INV_EXPORT inv_icm20603_poll_ois_gyro_data ( struct inv_icm20603 * s, int16_t gyro_data[3] )`

Poll OIS gyro data.

#### Returns

0x10 if OIS data reported, 0 otherwise, negative value on error.

7.24.2.17 `int INV_EXPORT inv_icm20603_poll_sensor_data_reg ( struct inv_icm20603 * s, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3] )`

Check and retrieve for new data.

## Parameters

out	<i>acc_data</i>	raw gyro data
out	<i>temp_data</i>	raw temp data
out	<i>gyro_data</i>	raw gyro data

## Returns

0 : not data ready 0x01 : gyro data output set 0x02 : acc data output set 0x03 : gyro and acc data output set  
 < 0 : error

7.24.2.18 int INV\_EXPORT inv\_icm20603\_reset\_fifo ( struct inv\_icm20603 \* s )

Reset FIFO.

## Returns

0 on success, negative value on error.

7.24.2.19 int inv\_icm20603\_run\_selftest ( struct inv\_icm20603 \* s )

Perform hardware self-test for Accel, Gyro and Compass.

## Parameters

in	<i>None</i>	
----	-------------	--

## Returns

COMPASS\_SUCCESS<<2 | ACCEL\_SUCCESS<<1 | GYRO\_SUCCESS so 3

7.24.2.20 int INV\_EXPORT inv\_icm20603\_set\_sensor\_period ( struct inv\_icm20603 \* s, enum inv\_icm20603\_sensor sensor, uint16\_t delayInMs )

Sets the odr for a sensor.

## Parameters

in	<i>sensor</i>	Sensor Identity
in	<i>delayInMs</i>	the delay between two values in ms

## Returns

0 in case of success, -1 for any error

7.24.2.21 `int INV_EXPORT inv_icm20603_set_slave_compass_id ( struct inv_icm20603 * s )`

Initialize AUX I2C and Compass.

#### Returns

0 on success, negative value on error.

7.24.2.22 `void inv_icm20603_set_st_bias ( struct inv_icm20603 * s, int * st_bias )`

Apply bias.

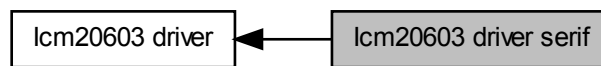
#### Parameters

in	<i>st_bias</i>	bias scaled by $2^{16}$ , accel is gee and gyro is dps. The buffer should be be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z
----	----------------	--

## 7.25 lcm20603 driver serif

Interface for low-level serial (I2C/SPI) access.

Collaboration diagram for lcm20603 driver serif:



### Classes

- struct [inv\\_lcm20603\\_serif](#)  
*basesensor serial interface*

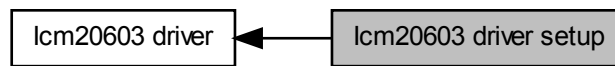
### 7.25.1 Detailed Description

Interface for low-level serial (I2C/SPI) access.

## 7.26 Icm20603 driver setup

Low-level function to setup an Icm20603 device.

Collaboration diagram for Icm20603 driver setup:



### Functions

- int INV\_EXPORT [inv\\_icm20603\\_get\\_whoami](#) (struct [inv\\_icm20603](#) \*s, uint8\_t \*whoami)  
*return WHOAMI value*
- int INV\_EXPORT [inv\\_icm20603\\_get\\_chip\\_info](#) (struct [inv\\_icm20603](#) \*s, uint8\_t chip\_info[3])  
*return WHOAMI value*
- int INV\_EXPORT [inv\\_icm20603\\_initialize](#) (struct [inv\\_icm20603](#) \*s)  
*Initialize the device.*
- int INV\_EXPORT [inv\\_icm20603\\_soft\\_reset](#) (struct [inv\\_icm20603](#) \*s)  
*Perform a soft reset of the device.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_divider](#) (struct [inv\\_icm20603](#) \*s, uint8\_t idiv)  
*Set the mpu sample rate.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_chip\\_power\\_state](#) (struct [inv\\_icm20603](#) \*s, uint8\_t func, uint8\_t on\_off)  
*Sets the power state of the Ivory chip loop.*
- uint8\_t INV\_EXPORT [inv\\_icm20603\\_get\\_chip\\_power\\_state](#) (struct [inv\\_icm20603](#) \*s)  
*Current wake status of the Mems chip.*
- uint16\_t INV\_EXPORT [inv\\_icm20603\\_get\\_chip\\_base\\_sample\\_rate](#) (struct [inv\\_icm20603](#) \*s)  
*Get internal sample rate currently configured.*
- int INV\_EXPORT [inv\\_icm20603\\_accel\\_fsr\\_2\\_reg](#) (int32\_t fsr)  
*Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.*
- int INV\_EXPORT [inv\\_icm20603\\_reg\\_2\\_accel\\_fsr](#) (uint8\_t reg)  
*Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_accel\\_fullscale](#) (struct [inv\\_icm20603](#) \*s, int level)  
*Sets fullscale range of accel in hardware.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_accel\\_ois\\_fullscale](#) (struct [inv\\_icm20603](#) \*s, int level)  
*Sets fullscale range of OIS accel in hardware.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_accel\\_bandwidth](#) (struct [inv\\_icm20603](#) \*s, int level)  
*Sets bandwidth range of accel in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20603\\_get\\_accel\\_fullscale](#) (struct [inv\\_icm20603](#) \*s)  
*Returns fullscale range of accelerometer in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20603\\_get\\_accel\\_ois\\_fullscale](#) (struct [inv\\_icm20603](#) \*s)  
*Returns fullscale range of OIS accelerometer in hardware.*
- uint16\_t INV\_EXPORT [inv\\_icm20603\\_get\\_accel\\_bandwidth](#) (struct [inv\\_icm20603](#) \*s)

- Returns bandwidth range of accelerometer in hardware.*
- int INV\_EXPORT [inv\\_icm20603\\_gyro\\_fsr\\_2\\_reg](#) (int32\_t fsr)  
*Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)*
- int INV\_EXPORT [inv\\_icm20603\\_reg\\_2\\_gyro\\_fsr](#) (uint8\_t reg)  
*Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_gyro\\_fullscale](#) (struct [inv\\_icm20603](#) \*s, int level)  
*Sets fullscale range of gyro in hardware.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_gyro\\_ois\\_fullscale](#) (struct [inv\\_icm20603](#) \*s, int level)  
*Sets fullscale range of OIS gyro in hardware.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_gyro\\_bandwidth](#) (struct [inv\\_icm20603](#) \*s, int level)  
*Sets bandwidth range of gyro in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20603\\_get\\_gyro\\_fullscale](#) (struct [inv\\_icm20603](#) \*s)  
*Returns fullscale range of gyroscope in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20603\\_get\\_gyro\\_ois\\_fullscale](#) (struct [inv\\_icm20603](#) \*s)  
*Returns fullscale range of OIS gyroscope in hardware.*
- uint16\_t INV\_EXPORT [inv\\_icm20603\\_get\\_gyro\\_bandwidth](#) (struct [inv\\_icm20603](#) \*s)  
*Returns bandwidth range of gyroscope in hardware.*
- int INV\_EXPORT [inv\\_icm20603\\_is\\_advanced\\_features\\_supported](#) (void)  
*Get the available features according the base sensor device.*
- int INV\_EXPORT [inv\\_icm20603\\_set\\_fsycn\\_bit\\_location](#) (struct [inv\\_icm20603](#) \*s, int bit\_location)  
*Set FSYN bit location.*

## 7.26.1 Detailed Description

Low-level function to setup an Icm20603 device.

## 7.26.2 Function Documentation

### 7.26.2.1 int INV\_EXPORT inv\_icm20603\_accel\_fsr\_2\_reg ( int32\_t fsr )

Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.

#### Returns

internal register value for FSR configuration

### 7.26.2.2 uint16\_t INV\_EXPORT inv\_icm20603\_get\_accel\_bandwidth ( struct inv\_icm20603 \* s )

Returns bandwidth range of accelerometer in hardware.

#### Returns

the bandwidth range

7.26.2.3 `uint8_t INV_EXPORT inv_icm20603_get_accel_fullscale ( struct inv_icm20603 * s )`

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.26.2.4 `uint8_t INV_EXPORT inv_icm20603_get_accel_ois_fullscale ( struct inv_icm20603 * s )`

Returns fullscale range of OIS accelerometer in hardware.

Returns

the fullscale range

7.26.2.5 `uint16_t INV_EXPORT inv_icm20603_get_chip_base_sample_rate ( struct inv_icm20603 * s )`

Get internal sample rate currently configured.

Returns

Internal sample rate currently configured in MEMS registers in Hz

7.26.2.6 `int INV_EXPORT inv_icm20603_get_chip_info ( struct inv_icm20603 * s, uint8_t chip_info[3] )`

return WHOAMI value

Parameters

out	<code>chip_info[0]</code>	WHOAMI for device
out	<code>chip_info[1]</code>	MANUFACTURER_ID for device
out	<code>chip_info[2]</code>	CHIP_ID for device

Returns

0 on success, negative value on error

7.26.2.7 `uint8_t INV_EXPORT inv_icm20603_get_chip_power_state ( struct inv_icm20603 * s )`

Current wake status of the Mems chip.

Returns

the wake status



7.26.2.8 `uint16_t INV_EXPORT inv_icm20603_get_gyro_bandwidth ( struct inv_icm20603 * s )`

Returns bandwidth range of gyroscope in hardware.

Returns

the bandwidth range

7.26.2.9 `uint8_t INV_EXPORT inv_icm20603_get_gyro_fullscale ( struct inv_icm20603 * s )`

Returns fullscale range of gyroscope in hardware.

Returns

the fullscale range

7.26.2.10 `uint8_t INV_EXPORT inv_icm20603_get_gyro_ois_fullscale ( struct inv_icm20603 * s )`

Returns fullscale range of OIS gyroscope in hardware.

Returns

the fullscale range

7.26.2.11 `int INV_EXPORT inv_icm20603_get_whoami ( struct inv_icm20603 * s, uint8_t * whoami )`

return WHOAMI value

Parameters

out	whoami	WHOAMI for device
-----	--------	-------------------

Returns

0 on success, negative value on error

7.26.2.12 `int INV_EXPORT inv_icm20603_gyro_fsr_2_reg ( int32_t fsr )`

Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)

Returns

internal register value for FSR configuration

7.26.2.13 `int INV_EXPORT inv_icm20603_initialize ( struct inv_icm20603 * s )`

Initialize the device.

#### Returns

0 on success, negative value on error.

7.26.2.14 `int INV_EXPORT inv_icm20603_is_advanced_features_supported ( void )`

Get the available features according the base sensor device.

#### Parameters

<i>in</i>	<i>in</i>	0: No advanced features supported 1: OIS sensor and FSYNC behavior supported (for icm20690 only)
-----------	-----------	--

7.26.2.15 `int INV_EXPORT inv_icm20603_reg_2_accel_fsr ( uint8_t reg )`

Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),..

#### Returns

FSR value in mg

7.26.2.16 `int INV_EXPORT inv_icm20603_reg_2_gyro_fsr ( uint8_t reg )`

Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)

#### Returns

FSR value in dps

7.26.2.17 `int INV_EXPORT inv_icm20603_set_accel_bandwidth ( struct inv_icm20603 * s, int level )`

Sets bandwidth range of accel in hardware.

#### Parameters

<i>in</i>	<i>level</i>	See mpu_accel_bw.
-----------	--------------	-------------------

**Returns**

0 on success, negative value on error.

**7.26.2.18** `int INV_EXPORT inv_icm20603_set_accel_fullscale ( struct inv_icm20603 * s, int level )`

Sets fullscale range of accel in hardware.

**Parameters**

in	<i>level</i>	See mpu_accel_fs.
----	--------------	-------------------

**Returns**

0 on success, negative value on error.

**7.26.2.19** `int INV_EXPORT inv_icm20603_set_accel_ois_fullscale ( struct inv_icm20603 * s, int level )`

Sets fullscale range of OIS accel in hardware.

**Parameters**

in	<i>level</i>	See mpu_accel_ois_fs.
----	--------------	-----------------------

**Returns**

0 on success, negative value on error.

**7.26.2.20** `int INV_EXPORT inv_icm20603_set_chip_power_state ( struct inv_icm20603 * s, uint8_t func, uint8_t on_off )`

Sets the power state of the Ivory chip loop.

**Parameters**

in	<i>func</i>	CHIP_AWAKE, CHIP_LP_ENABLE
in	<i>on_off</i>	The functions are enabled if previously disabled and disabled if previously enabled based on the value of On/Off.

**Returns**

0 on success, negative value on error.

**7.26.2.21** `int INV_EXPORT inv_icm20603_set_divider ( struct inv_icm20603 * s, uint8_t idiv )`

Set the mpu sample rate.

**Returns**

Value written to MPUREG\_SMPLRT\_DIV register.

**7.26.2.22** `int INV_EXPORT inv_icm20603_set_fsycn_bit_location ( struct inv_icm20603 * s, int bit_location )`

Set FSYNC bit location.

**Parameters**

<code>in</code>	<code>in</code>	0: Disable FSYNC pin data to be sampled 1: Enable FSYNC pin data to be sampled at TEMP_OUT_L[0]
-----------------	-----------------	---

**7.26.2.23** `int INV_EXPORT inv_icm20603_set_gyro_bandwidth ( struct inv_icm20603 * s, int level )`

Sets bandwidth range of gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_bw.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

**7.26.2.24** `int INV_EXPORT inv_icm20603_set_gyro_fullscale ( struct inv_icm20603 * s, int level )`

Sets fullscale range of gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_fs.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

**7.26.2.25** `int INV_EXPORT inv_icm20603_set_gyro_ois_fullscale ( struct inv_icm20603 * s, int level )`

Sets fullscale range of OIS gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_fs.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

**7.26.2.26** `int INV_EXPORT inv_icm20603_soft_reset ( struct inv_icm20603 * s )`

Perform a soft reset of the device.

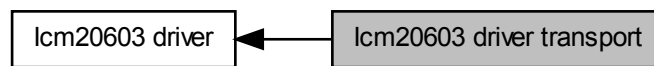
**Returns**

0 on success, negative value on error.

## 7.27 Icm20603 driver transport

Low-level Icm20603 register access.

Collaboration diagram for Icm20603 driver transport:



### Functions

- int INV\_EXPORT [inv\\_icm20603\\_mems\\_write\\_reg](#) (struct [inv\\_icm20603](#) \*s, uint16\_t reg, uint32\_t length, const uint8\_t \*data)  
*Writes data from a register on mems.*
- int INV\_EXPORT [inv\\_icm20603\\_mems\\_write\\_reg\\_one](#) (struct [inv\\_icm20603](#) \*s, uint16\_t reg, uint8\_t data)  
*Writes a single byte of data from a register on mems with no power control.*
- int INV\_EXPORT [inv\\_icm20603\\_mems\\_read\\_reg](#) (struct [inv\\_icm20603](#) \*s, uint16\_t reg, uint32\_t length, uint8\_t \*data)  
*Reads data from a register on mems.*

### 7.27.1 Detailed Description

Low-level Icm20603 register access.

### 7.27.2 Function Documentation

7.27.2.1 int INV\_EXPORT [inv\\_icm20603\\_mems\\_read\\_reg](#) ( struct [inv\\_icm20603](#) \* s, uint16\_t *reg*, uint32\_t *length*, uint8\_t \* *data* )

Reads data from a register on mems.

#### Parameters

in	<i>reg</i>	register address
in	<i>length</i>	length of data
out	<i>data</i>	output data from the register

#### Returns

0 in case of success, -1 for any error

7.27.2.2 `int INV_EXPORT inv_icm20603_mems_write_reg ( struct inv_icm20603 * s, uint16_t reg, uint32_t length, const uint8_t * data )`

Writes data from a register on mems.

**Parameters**

in	<i>length</i>	number of byte to be written
out	<i>data</i>	output data from the register

**Returns**

0 in case of success, -1 for any error

7.27.2.3 `int INV_EXPORT inv_icm20603_mems_write_reg_one ( struct inv_icm20603 * s, uint16_t reg, uint8_t data )`

Writes a single byte of data from a register on mems with no power control.

**Parameters**

in	<i>reg</i>	Register address
out	<i>data</i>	Data to be written

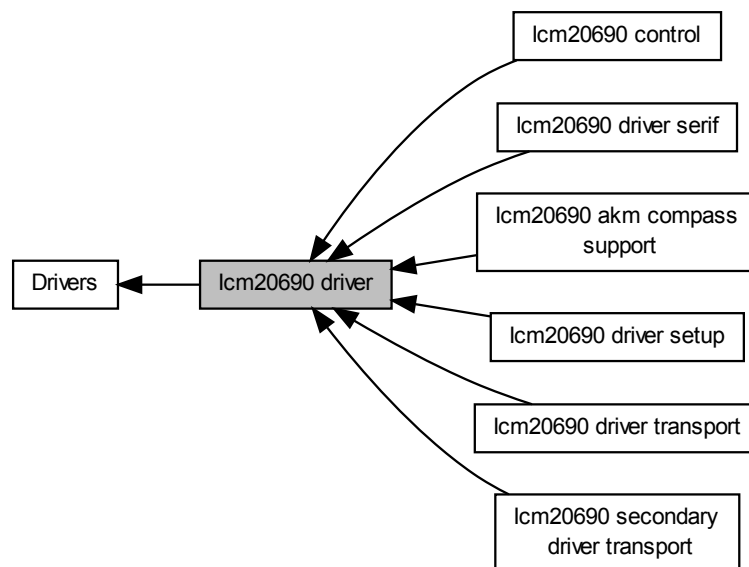
**Returns**

0 in case of success, -1 for any error

## 7.28 lcm20690 driver

Low-level driver for lcm20690 devices.

Collaboration diagram for lcm20690 driver:



### Modules

- [lcm20690 akm compass support](#)  
*Low-level lcm20690 aux sensor access.*
- [lcm20690 secondary driver transport](#)  
*Low-level lcm20690 secondary interface access.*
- [lcm20690 control](#)  
*Low-level function to control a lcm20690 device.*
- [lcm20690 driver serif](#)  
*Interface for low-level serial (I2C/SPI) access.*
- [lcm20690 driver setup](#)  
*Low-level function to setup an lcm20690 device.*
- [lcm20690 driver transport](#)  
*Low-level lcm20690 register access.*

### Classes

- struct [inv\\_lcm20690](#)  
*lcm20690 driver states definition.*



## Typedefs

- typedef struct `inv_icm20690` `inv_icm20690_t`  
*Icm20690 driver states definition.*

## Functions

- void `inv_icm20690_sleep` (int ms)  
*Hook for low-level system sleep() function to be implemented by upper layer.*
- void `inv_icm20690_sleep_us` (int us)  
*Hook for low-level high res system sleep() function to be implemented by upper layer ~ 100us resolution is sufficient.*
- uint64\_t `inv_icm20690_get_time_us` (void)  
*Hook for low-level system time() function to be implemented by upper layer.*
- uint64\_t `inv_icm20690_get_dataready_interrupt_time_us` (void)  
*Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get\_time\_us()*
- static void `inv_icm20690_reset_states` (struct `inv_icm20690` \*s, const struct `inv_icm20690_serif` \*serif)  
*Reset and initialize driver states.*
- static void `inv_icm20690_reset_states_serif_ois` (struct `inv_icm20690` \*s, const struct `inv_icm20690_serif` \*serif\_ois)  
*Register secondary SERIF object for OIS access Must be called after inv\_icm20690\_reset\_states()*

### 7.28.1 Detailed Description

Low-level driver for Icm20690 devices.

### 7.28.2 Function Documentation

#### 7.28.2.1 uint64\_t inv\_icm20690\_get\_dataready\_interrupt\_time\_us ( void )

Hook to get interrupt data ready timestamp to be implemented by upper layer Using this hook in embedded firmware, the timestamping could be done in ISR and allowed a better accuracy than getting the current time in the polling function But in host application, this function will have the same implementation than get\_time\_us()

#### Returns

data ready interrupt timestamp in us

#### 7.28.2.2 uint64\_t inv\_icm20690\_get\_time\_us ( void )

Hook for low-level system time() function to be implemented by upper layer.

#### Returns

monotonic timestamp in us

#### 7.28.2.3 static void inv\_icm20690\_reset\_states ( struct inv\_icm20690 \* s, const struct inv\_icm20690\_serif \* serif ) [inline],[static]

Reset and initialize driver states.

**Parameters**

in	<i>s</i>	handle to driver states structure
in	<i>serif</i>	handle to SERIF object for underlying register access

**7.28.2.4** `static void inv_icm20690_reset_states_serif_ois ( struct inv_icm20690 * s, const struct inv_icm20690_serif * serif_ois )` `[inline],[static]`

Register secondary SERIF object for OIS access Must be called after [inv\\_icm20690\\_reset\\_states\(\)](#)

**Parameters**

in	<i>s</i>	handle to driver states structure
in	<i>serif</i>	handle to SERIF object for underlying register access to OIS

**7.28.2.5** `void inv_icm20690_sleep ( int ms )`

Hook for low-level system sleep() function to be implemented by upper layer.

**Parameters**

in	<i>ms</i>	number of millisecond the calling thread should sleep
----	-----------	---

**7.28.2.6** `void inv_icm20690_sleep_us ( int us )`

Hook for low-level high res system sleep() function to be implemented by upper layer ~100us resolution is sufficient.

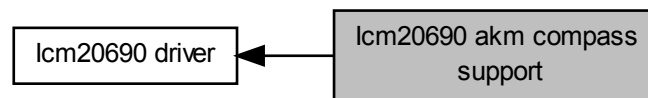
**Parameters**

in	<i>us</i>	number of us the calling thread should sleep
----	-----------	--

## 7.29 lcm20690 akm compass support

Low-level lcm20690 aux sensor access.

Collaboration diagram for lcm20690 akm compass support:



### Enumerations

### Functions

- void INV\_EXPORT [inv\\_icm20690\\_register\\_aux\\_compass](#) (struct [inv\\_icm20690](#) \*s, enum [inv\\_icm20690\\_compass\\_id](#) compass\_id, uint8\_t compass\_i2c\_addr)  
*Register AUX compass.*
- int INV\_EXPORT [inv\\_icm20690\\_is\\_compass\\_registered](#) (struct [inv\\_icm20690](#) \*s)  
*Return non-zero value is AUX compass was regitered, 0 if not If compass was registered but setup failed, will also return 0.*
- int INV\_EXPORT [inv\\_icm20690\\_setup\\_compass\\_akm](#) (struct [inv\\_icm20690](#) \*s)  
*Initializes the compass.*
- int INV\_EXPORT [inv\\_icm20690\\_check\\_akm\\_self\\_test](#) (struct [inv\\_icm20690](#) \*s)  
*Self test for the compass.*
- int INV\_EXPORT [inv\\_icm20690\\_read\\_akm\\_scale](#) (struct [inv\\_icm20690](#) \*s, int \*scale)  
*Reads the scale of the compass.*
- int INV\_EXPORT [inv\\_icm20690\\_suspend\\_akm](#) (struct [inv\\_icm20690](#) \*s)  
*Stops the compass.*
- int INV\_EXPORT [inv\\_icm20690\\_resume\\_akm](#) (struct [inv\\_icm20690](#) \*s)  
*Starts the compass.*
- char INV\_EXPORT [inv\\_icm20690\\_compass\\_getstate](#) (struct [inv\\_icm20690](#) \*s)  
*Get compass power status.*
- int INV\_EXPORT [inv\\_icm20690\\_get\\_compass\\_data](#) (struct [inv\\_icm20690](#) \*s, const unsigned char \*packet, unsigned char \*raw\_compass)  
*Parse compass data packet.*
- int INV\_EXPORT [inv\\_icm20690\\_get\\_compass\\_bytes](#) (struct [inv\\_icm20690](#) \*s)  
*Get data packet size according to comass type.*

### 7.29.1 Detailed Description

Low-level lcm20690 aux sensor access.

## 7.29.2 Enumeration Type Documentation

### 7.29.2.1 enum inv\_icm20690\_compass\_id

Supported auxiliary compass identifier.

Enumerator

**INV\_ICM20690\_COMPASS\_ID\_NONE** no compass  
**INV\_ICM20690\_COMPASS\_ID\_AK09911** AKM AK09911.  
**INV\_ICM20690\_COMPASS\_ID\_AK09912** AKM AK09912.

## 7.29.3 Function Documentation

### 7.29.3.1 int INV\_EXPORT inv\_icm20690\_check\_akm\_self\_test ( struct inv\_icm20690 \* s )

Self test for the compass.

Returns

0 in case of success, -1 for any error

### 7.29.3.2 char INV\_EXPORT inv\_icm20690\_compass\_getstate ( struct inv\_icm20690 \* s )

Get compass power status.

Returns

1 in case compass is enabled, 0 if not started

### 7.29.3.3 int INV\_EXPORT inv\_icm20690\_get\_compass\_bytes ( struct inv\_icm20690 \* s )

Get data packet size according to compass type.

Returns

size in bytes

### 7.29.3.4 int INV\_EXPORT inv\_icm20690\_get\_compass\_data ( struct inv\_icm20690 \* s, const unsigned char \* packet, unsigned char \* raw\_compass )

Parse compass data packet.

## Parameters

in	<i>data</i>	packet pointer to compass data packet
in	<i>raw_compass</i>	pointer to raw compass. big endian 2 bytes for each axis.

## Returns

0 in case of success, -1 for any error

7.29.3.5 `int INV_EXPORT inv_icm20690_read_akm_scale ( struct inv_icm20690 * s, int * scale )`

Reads the scale of the compass.

## Parameters

out	<i>scale</i>	pointer to recuperate the scale
-----	--------------	---------------------------------

## Returns

0 in case of success, -1 for any error

7.29.3.6 `void INV_EXPORT inv_icm20690_register_aux_compass ( struct inv_icm20690 * s, enum inv_icm20690_compass_id compass_id, uint8_t compass_i2c_addr )`

Register AUX compass.

Will only set internal states and won't perform any transaction on the bus. Must be called before [inv\\_icm20690\\_initialize\(\)](#).

## Parameters

in	<i>compass_id</i>	Compass ID
in	<i>compass_i2c_addr</i>	Compass I2C address

## Returns

0 on success, negative value on error

7.29.3.7 `int INV_EXPORT inv_icm20690_resume_akm ( struct inv_icm20690 * s )`

Starts the compass.

## Returns

0 in case of success, -1 for any error

7.29.3.8 `int INV_EXPORT inv_icm20690_setup_compass_akm ( struct inv_icm20690 * s )`

Initializes the compass.

**Returns**

0 in case of success, -1 for any error

7.29.3.9 `int INV_EXPORT inv_icm20690_suspend_akm ( struct inv_icm20690 * s )`

Stops the compass.

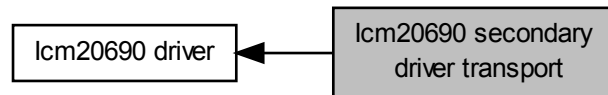
**Returns**

0 in case of success, -1 for any error

## 7.30 lcm20690 secondary driver transport

Low-level lcm20690 secondary interface access.

Collaboration diagram for lcm20690 secondary driver transport:



### Macros

- #define `COMPASS_I2C_SLV_READ` 0  
Secondary I2C channel usage :

### Functions

- void `inv_icm20690_init_secondary` (struct `inv_icm20690` \*s)  
*Initializes the register for the i2c communication.*
- int `inv_icm20690_read_secondary` (struct `inv_icm20690` \*s, int index, unsigned char addr, unsigned char reg, char len)  
*Reads data in i2c a secondary device.*
- int `inv_icm20690_execute_read_secondary` (struct `inv_icm20690` \*s, int index, unsigned char addr, int reg, int len, unsigned char \*d)  
*Reads data in i2c a secondary device directly.*
- int `inv_icm20690_write_secondary` (struct `inv_icm20690` \*s, int index, unsigned char addr, unsigned char reg, char v)  
*Writes data in i2c a secondary device.*
- int `inv_icm20690_execute_write_secondary` (struct `inv_icm20690` \*s, int index, unsigned char addr, int reg, unsigned char v)  
*Writes data in i2c a secondary device directly.*
- int `inv_icm20690_secondary_stop_channel` (struct `inv_icm20690` \*s, int index)  
*Stop one secondary I2C channel by writing 0 in its control register.*
- int `inv_icm20690_secondary_enable_i2c` (struct `inv_icm20690` \*s)  
*Enable secondary I2C interface.*
- int `inv_icm20690_secondary_disable_i2c` (struct `inv_icm20690` \*s)  
*Stop secondary I2C interface.*

#### 7.30.1 Detailed Description

Low-level lcm20690 secondary interface access.

## 7.30.2 Macro Definition Documentation

### 7.30.2.1 `#define COMPASS_I2C_SLV_READ 0`

Secondary I2C channel usage :

- channel 0 is reserved for compass reading data
- channel 1 is reserved for compass writing one-shot acquisition register

## 7.30.3 Function Documentation

### 7.30.3.1 `int inv_icm20690_execute_read_secondary ( struct inv_icm20690 * s, int index, unsigned char addr, int reg, int len, unsigned char * d )`

Reads data in i2c a secondary device directly.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be read on the secondary device
in	<i>len</i>	Size of data to be read
out	<i>d</i>	pointer to the data to be read

#### Returns

0 in case of success, -1 for any error

### 7.30.3.2 `int inv_icm20690_execute_write_secondary ( struct inv_icm20690 * s, int index, unsigned char addr, int reg, unsigned char v )`

Writes data in i2c a secondary device directly.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be write on the secondary device
in	<i>v</i>	the data to be written

#### Returns

0 in case of success, -1 for any error



7.30.3.3 `int inv_icm20690_read_secondary ( struct inv_icm20690 * s, int index, unsigned char addr, unsigned char reg, char len )`

Reads data in i2c a secondary device.

#### Parameters

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	I2c address slave of the secondary slave
in	<i>reg</i>	the register to be read on the secondary device
in	<i>len</i>	Size of data to be read

#### Returns

0 in case of success, -1 for any error

7.30.3.4 `int inv_icm20690_secondary_disable_i2c ( struct inv_icm20690 * s )`

Stop secondary I2C interface.

#### Returns

0 in case of success, -1 for any error

#### Warning

It stops all I2C transactions, whatever the channel status

7.30.3.5 `int inv_icm20690_secondary_enable_i2c ( struct inv_icm20690 * s )`

Enable secondary I2C interface.

#### Returns

0 in case of success, -1 for any error

7.30.3.6 `int inv_icm20690_secondary_stop_channel ( struct inv_icm20690 * s, int index )`

Stop one secondary I2C channel by writing 0 in its control register.

#### Parameters

in	<i>index</i>	the channel id to be stopped
----	--------------	------------------------------

**Returns**

0 in case of success, -1 for any error

**Warning**

It does not stop I2C secondary interface, just one channel

**7.30.3.7** `int inv_icm20690_write_secondary ( struct inv_icm20690 * s, int index, unsigned char addr, unsigned char reg, char v )`

Writes data in i2c a secondary device.

**Parameters**

in	<i>index</i>	The i2c slave what you would use
in	<i>addr</i>	i2c address slave of the secondary slave
in	<i>reg</i>	the register to be write on the secondary device
in	<i>v</i>	the data to be written

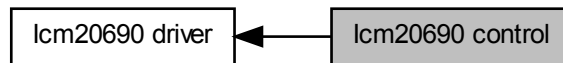
**Returns**

0 in case of success, -1 for any error

## 7.31 lcm20690 control

Low-level function to control a lcm20690 device.

Collaboration diagram for lcm20690 control:



### Classes

- struct [inv\\_icm20690\\_fifo\\_states](#)  
Check and retrieve for new data.

### Enumerations

### Functions

- int INV\_EXPORT [inv\\_icm20690\\_enable\\_mems](#) (struct [inv\\_icm20690](#) \*s, int bit\_mask, uint16\_t smplr\_t ← divider)  
Enables accel and/or gyro and/or pressure if integrated with gyro and accel.
- int INV\_EXPORT [inv\\_icm20690\\_set\\_sensor\\_period](#) (struct [inv\\_icm20690](#) \*s, enum [inv\\_icm20690\\_sensor](#) sensor, uint16\_t delayInMs)  
Sets the odr for a sensor.
- int INV\_EXPORT [inv\\_icm20690\\_is\\_sensor\\_enabled](#) (struct [inv\\_icm20690](#) \*s, enum [inv\\_icm20690\\_sensor](#) sensor)  
Knows if the sensor is enabled or disabled.
- int INV\_EXPORT [inv\\_icm20690\\_enable\\_sensor](#) (struct [inv\\_icm20690](#) \*s, enum [inv\\_icm20690\\_sensor](#) sensor, uint8\_t enable)  
Enables / disables a sensor.
- int INV\_EXPORT [inv\\_icm20690\\_configure\\_accel\\_wom](#) (struct [inv\\_icm20690](#) \*s, uint8\_t wom\_threshold)  
Configures accel WOM.
- int INV\_EXPORT [inv\\_icm20690\\_poll\\_sensor\\_data\\_reg](#) (struct [inv\\_icm20690](#) \*s, int16\_t acc\_data[3], int16\_t ← \*temp\_data, int16\_t gyro\_data[3])  
Check and retrieve for new data.
- int INV\_EXPORT [inv\\_icm20690\\_poll\\_fifo\\_data\\_setup](#) (struct [inv\\_icm20690](#) \*s, struct [inv\\_icm20690\\_fifo\\_ ← states](#) \*states, uint8\_t int\_status)  
Initializes states for FIFO reading and parsing.
- int INV\_EXPORT [inv\\_icm20690\\_poll\\_fifo\\_data](#) (struct [inv\\_icm20690](#) \*s, struct [inv\\_icm20690\\_fifo\\_states](#) \*states, int16\_t acc\_data[3], int16\_t \*temp\_data, int16\_t gyro\_data[3], int16\_t compass\_data[3])  
Read one packet from the FIFO (packet corresponding to data for all current active sensor)
- int INV\_EXPORT [inv\\_icm20690\\_has\\_data\\_ready](#) (struct [inv\\_icm20690](#) \*s)  
Read interrupt status and check for DDRY flag.
- int INV\_EXPORT [inv\\_icm20690\\_get\\_int\\_status](#) (struct [inv\\_icm20690](#) \*s, uint8\_t \*int\_status)

- Read interrupt status and return its value.*
- int INV\_EXPORT [inv\\_icm20690\\_check\\_drdy](#) (struct [inv\\_icm20690](#) \*s, uint8\_t int\_status)  
*Check for DRDY flag in INT register value.*
- int INV\_EXPORT [inv\\_icm20690\\_check\\_wom\\_status](#) (struct [inv\\_icm20690](#) \*s, uint8\_t int\_status)  
*Check for WOM bits in INT register value.*
- int INV\_EXPORT [inv\\_icm20690\\_disable\\_fifo](#) (struct [inv\\_icm20690](#) \*s)  
*Disable FIFO.*
- int INV\_EXPORT [inv\\_icm20690\\_enable\\_fifo](#) (struct [inv\\_icm20690](#) \*s, int bit\_mask)  
*Enable FIFO.*
- int INV\_EXPORT [inv\\_icm20690\\_reset\\_fifo](#) (struct [inv\\_icm20690](#) \*s)  
*Reset FIFO.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_slave\\_compass\\_id](#) (struct [inv\\_icm20690](#) \*s)  
*Initialize AUX I2C and Compass.*
- int INV\_EXPORT [inv\\_icm20690\\_poll\\_ois\\_gyro\\_data](#) (struct [inv\\_icm20690](#) \*s, int16\_t gyro\_data[3])  
*Poll OIS gyro data.*
- int INV\_EXPORT [inv\\_icm20690\\_poll\\_delay\\_count](#) (struct [inv\\_icm20690](#) \*s, int16\_t \*delay\_count)  
*Poll EIS flag and delay counter.*
- int INV\_EXPORT [inv\\_icm20690\\_all\\_sensors\\_off](#) (struct [inv\\_icm20690](#) \*s)  
*test if all sensors are off*
- int [inv\\_icm20690\\_run\\_selftest](#) (struct [inv\\_icm20690](#) \*s)  
*Perform hardware self-test for Accel, Gyro and Compass.*
- void INV\_EXPORT [inv\\_icm20690\\_get\\_st\\_bias](#) (struct [inv\\_icm20690](#) \*s, int \*st\_bias)  
*Retrieve bias collected by self-test.*
- void [inv\\_icm20690\\_set\\_st\\_bias](#) (struct [inv\\_icm20690](#) \*s, int \*st\_bias)  
*Apply bias.*

### 7.31.1 Detailed Description

Low-level function to control a Icm20690 device.

### 7.31.2 Function Documentation

#### 7.31.2.1 int INV\_EXPORT inv\_icm20690\_all\_sensors\_off ( struct inv\_icm20690 \* s )

test if all sensors are off

#### Returns

true if all sensors are off, false otherwise

#### 7.31.2.2 int INV\_EXPORT inv\_icm20690\_check\_drdy ( struct inv\_icm20690 \* s, uint8\_t int\_status )

Check for DRDY flag in INT register value.

#### Parameters

in	<i>int_status</i>	INT status register value
----	-------------------	---------------------------

**Returns**

1 if DDRDY is set, 0 otherwise

7.31.2.3 `int INV_EXPORT inv_icm20690_check_wom_status ( struct inv_icm20690 * s, uint8_t int_status )`

Check for WOM bits in INT register value.

**Parameters**

in	<i>int_status</i>	INT status register value
----	-------------------	---------------------------

**Returns**

bit mask corresponding to x, y, z axis that caused WOM (0x01: x axis, 0x02: y axis, 0x01: z axis)

7.31.2.4 `int INV_EXPORT inv_icm20690_configure_accel_wom ( struct inv_icm20690 * s, uint8_t wom_threshold )`

Configure accel WOM.

**Parameters**

in	<i>wom_threshold</i>	threshold value for X,Y,Z axis that should trigger a WOM interrupt (0 to disable WOM)
----	----------------------	---

**Returns**

0 on success, negative value on error.

7.31.2.5 `int INV_EXPORT inv_icm20690_disable_fifo ( struct inv_icm20690 * s )`

Disable FIFO.

**Returns**

0 on success, negative value on error.

7.31.2.6 `int INV_EXPORT inv_icm20690_enable_fifo ( struct inv_icm20690 * s, int bit_mask )`

Enable FIFO.

**Parameters**

in	<i>bit_mask</i>	A mask of sensor to push to FIFO.
----	-----------------	-----------------------------------

**Returns**

0 on success, negative value on error.

7.31.2.7 `int INV_EXPORT inv_icm20690_enable_mems ( struct inv_icm20690 * s, int bit_mask, uint16_t smplr_t_divider )`

Enables accel and/or gyro and/or pressure if integrated with gyro and accel.

**Parameters**

in	<i>bit_mask</i>	A mask where 2 means turn on accel, 1 means turn on gyro, 4 is for pressure. By default, this only turns on a sensor if all sensors are off otherwise the DMP controls this register including turning off a sensor. To override this behavior add in a mask of 128.
in	<i>smplr_t_divider</i>	The divider which was applied to internal sample rate based on field <i>sample_rate</i> from <i>base_driver_t</i> to get minimum ODR for accel and gyro

**Returns**

0 on success, negative value on error

7.31.2.8 `int INV_EXPORT inv_icm20690_enable_sensor ( struct inv_icm20690 * s, enum inv_icm20690_sensor sensor, uint8_t enable )`

Enables / disables a sensor.

**Parameters**

in	<i>androidSensor</i>	Sensor Identity
in	<i>enable</i>	0=off, 1=on

**Returns**

0 in case of success, negative value on error

7.31.2.9 `int INV_EXPORT inv_icm20690_get_int_status ( struct inv_icm20690 * s, uint8_t * int_status )`

Read interrupt status and return its value.

**Parameters**

out	<i>int_status</i>	INT status register value
-----	-------------------	---------------------------

**Returns**

0 on success, negative value on error

7.31.2.10 void INV\_EXPORT inv\_icm20690\_get\_st\_bias ( struct inv\_icm20690 \* s, int \* st\_bias )

Retrieve bias collected by self-test.

#### Parameters

out	st_bias	bias scaled by 2 <sup>16</sup> , accel is gee and gyro is dps. The buffer will be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z
-----	---------	--

7.31.2.11 int INV\_EXPORT inv\_icm20690\_has\_data\_ready ( struct inv\_icm20690 \* s )

Read interrupt status and check for DDRY flag.

#### Returns

1 if DDRY inteript is set, 0 otherwise and negative value on error

7.31.2.12 int INV\_EXPORT inv\_icm20690\_is\_sensor\_enabled ( struct inv\_icm20690 \* s, enum inv\_icm20690\_sensor sensor )

Knows if the sensor is enabled or disbaled.

#### Parameters

in	sensor	the sensor is enabled or not
----	--------	------------------------------

#### Returns

1 if active, 0 otherwise

7.31.2.13 int INV\_EXPORT inv\_icm20690\_poll\_delay\_count ( struct inv\_icm20690 \* s, int16\_t \* delay\_count )

Poll EIS flag and delay counter.

#### Parameters

out	delay_count	delay time in us between the FSYNC event (before the gyro data event) and the gyro data event
-----	-------------	---

#### Returns

0x8 if FSYNC event reported, 0 otherwise, negative value on error.

7.31.2.14 `int INV_EXPORT inv_icm20690_poll_fifo_data ( struct inv_icm20690 * s, struct inv_icm20690_fifo_states * states, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3], int16_t compass_data[3] )`

Read one packet from the FIFO (packet corresponding to data for all current active sensor)

#### Parameters

in	<i>states</i>	placeholder to FIFO states
out	<i>acc_data</i>	raw gyro data
out	<i>temp_data</i>	raw temp data
out	<i>gyro_data</i>	raw gyro data
out	<i>compass_data</i>	raw compass data

#### Returns

0 : FIFO empty bit0 : gyro data output set bit1 : acc data output set bit2 : compass output set < 0 : error

7.31.2.15 `int INV_EXPORT inv_icm20690_poll_fifo_data_setup ( struct inv_icm20690 * s, struct inv_icm20690_fifo_states * states, uint8_t int_status )`

Initializes states for FIFO reading and parsing.

#### Parameters

in	<i>states</i>	placeholder to FIFO states
in	<i>int_status</i>	int status register value (used to check for overflow)

#### Returns

0 for success, 1 for overflow detected, negative value for other errors

7.31.2.16 `int INV_EXPORT inv_icm20690_poll_ois_gyro_data ( struct inv_icm20690 * s, int16_t gyro_data[3] )`

Poll OIS gyro data.

#### Returns

0x10 if OIS data reported, 0 otherwise, negative value on error.

7.31.2.17 `int INV_EXPORT inv_icm20690_poll_sensor_data_reg ( struct inv_icm20690 * s, int16_t acc_data[3], int16_t * temp_data, int16_t gyro_data[3] )`

Check and retrieve for new data.



## Parameters

out	<i>acc_data</i>	raw gyro data
out	<i>temp_data</i>	raw temp data
out	<i>gyro_data</i>	raw gyro data

## Returns

0 : not data ready 0x01 : gyro data output set 0x02 : acc data output set 0x03 : gyro and acc data output set  
 < 0 : error

7.31.2.18 int INV\_EXPORT inv\_icm20690\_reset\_fifo ( struct inv\_icm20690 \* s )

Reset FIFO.

## Returns

0 on success, negative value on error.

7.31.2.19 int inv\_icm20690\_run\_selftest ( struct inv\_icm20690 \* s )

Perform hardware self-test for Accel, Gyro and Compass.

## Parameters

in	<i>None</i>	
----	-------------	--

## Returns

COMPASS\_SUCCESS<<2 | ACCEL\_SUCCESS<<1 | GYRO\_SUCCESS so 3

7.31.2.20 int INV\_EXPORT inv\_icm20690\_set\_sensor\_period ( struct inv\_icm20690 \* s, enum inv\_icm20690\_sensor sensor, uint16\_t delayInMs )

Sets the odr for a sensor.

## Parameters

in	<i>sensor</i>	Sensor Identity
in	<i>delayInMs</i>	the delay between two values in ms

## Returns

0 in case of success, -1 for any error

7.31.2.21 `int INV_EXPORT inv_icm20690_set_slave_compass_id ( struct inv_icm20690 * s )`

Initialize AUX I2C and Compass.

#### Returns

0 on success, negative value on error.

7.31.2.22 `void inv_icm20690_set_st_bias ( struct inv_icm20690 * s, int * st_bias )`

Apply bias.

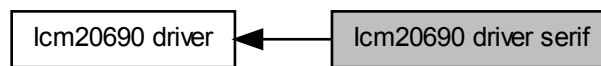
#### Parameters

in	<i>st_bias</i>	bias scaled by $2^{16}$ , accel is gee and gyro is dps. The buffer should be be stuffed in order as below. Gyro normal mode X,Y,Z Gyro LP mode X,Y,Z Accel normal mode X,Y,Z Accel LP mode X,Y,Z
----	----------------	--

## 7.32 lcm20690 driver serif

Interface for low-level serial (I2C/SPI) access.

Collaboration diagram for lcm20690 driver serif:



### Classes

- struct [inv\\_lcm20690\\_serif](#)  
*basesensor serial interface*

### 7.32.1 Detailed Description

Interface for low-level serial (I2C/SPI) access.

### 7.33 Icm20690 driver setup

Low-level function to setup an Icm20690 device.

Collaboration diagram for Icm20690 driver setup:



#### Functions

- int INV\_EXPORT [inv\\_icm20690\\_get\\_whoami](#) (struct [inv\\_icm20690](#) \*s, uint8\_t \*whoami)  
*return WHOAMI value*
- int INV\_EXPORT [inv\\_icm20690\\_get\\_chip\\_info](#) (struct [inv\\_icm20690](#) \*s, uint8\_t chip\_info[3])  
*return WHOAMI value*
- int INV\_EXPORT [inv\\_icm20690\\_initialize](#) (struct [inv\\_icm20690](#) \*s)  
*Initialize the device.*
- int INV\_EXPORT [inv\\_icm20690\\_soft\\_reset](#) (struct [inv\\_icm20690](#) \*s)  
*Perform a soft reset of the device.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_divider](#) (struct [inv\\_icm20690](#) \*s, uint8\_t idiv)  
*Set the mpu sample rate.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_chip\\_power\\_state](#) (struct [inv\\_icm20690](#) \*s, uint8\_t func, uint8\_t on\_off)  
*Sets the power state of the Ivory chip loop.*
- uint8\_t INV\_EXPORT [inv\\_icm20690\\_get\\_chip\\_power\\_state](#) (struct [inv\\_icm20690](#) \*s)  
*Current wake status of the Mems chip.*
- uint16\_t INV\_EXPORT [inv\\_icm20690\\_get\\_chip\\_base\\_sample\\_rate](#) (struct [inv\\_icm20690](#) \*s)  
*Get internal sample rate currently configured.*
- int INV\_EXPORT [inv\\_icm20690\\_accel\\_fsr\\_2\\_reg](#) (int32\_t fsr)  
*Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),.*
- int INV\_EXPORT [inv\\_icm20690\\_reg\\_2\\_accel\\_fsr](#) (uint8\_t reg)  
*Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_accel\\_fullscale](#) (struct [inv\\_icm20690](#) \*s, int level)  
*Sets fullscale range of accel in hardware.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_accel\\_ois\\_fullscale](#) (struct [inv\\_icm20690](#) \*s, int level)  
*Sets fullscale range of OIS accel in hardware.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_accel\\_bandwidth](#) (struct [inv\\_icm20690](#) \*s, int level)  
*Sets bandwidth range of accel in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20690\\_get\\_accel\\_fullscale](#) (struct [inv\\_icm20690](#) \*s)  
*Returns fullscale range of accelerometer in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20690\\_get\\_accel\\_ois\\_fullscale](#) (struct [inv\\_icm20690](#) \*s)  
*Returns fullscale range of OIS accelerometer in hardware.*
- uint16\_t INV\_EXPORT [inv\\_icm20690\\_get\\_accel\\_bandwidth](#) (struct [inv\\_icm20690](#) \*s)

- Returns bandwidth range of accelerometer in hardware.*
- int INV\_EXPORT [inv\\_icm20690\\_gyro\\_fsr\\_2\\_reg](#) (int32\_t fsr)  
*Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)*
- int INV\_EXPORT [inv\\_icm20690\\_reg\\_2\\_gyro\\_fsr](#) (uint8\_t reg)  
*Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_gyro\\_fullscale](#) (struct [inv\\_icm20690](#) \*s, int level)  
*Sets fullscale range of gyro in hardware.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_gyro\\_ois\\_fullscale](#) (struct [inv\\_icm20690](#) \*s, int level)  
*Sets fullscale range of OIS gyro in hardware.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_gyro\\_bandwidth](#) (struct [inv\\_icm20690](#) \*s, int level)  
*Sets bandwidth range of gyro in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20690\\_get\\_gyro\\_fullscale](#) (struct [inv\\_icm20690](#) \*s)  
*Returns fullscale range of gyroscope in hardware.*
- uint8\_t INV\_EXPORT [inv\\_icm20690\\_get\\_gyro\\_ois\\_fullscale](#) (struct [inv\\_icm20690](#) \*s)  
*Returns fullscale range of OIS gyroscope in hardware.*
- uint16\_t INV\_EXPORT [inv\\_icm20690\\_get\\_gyro\\_bandwidth](#) (struct [inv\\_icm20690](#) \*s)  
*Returns bandwidth range of gyroscope in hardware.*
- int INV\_EXPORT [inv\\_icm20690\\_is\\_advanced\\_features\\_supported](#) (void)  
*Get the available features according the base sensor device.*
- int INV\_EXPORT [inv\\_icm20690\\_set\\_fsycn\\_bit\\_location](#) (struct [inv\\_icm20690](#) \*s, int bit\_location)  
*Set FSYCN bit location.*

### 7.33.1 Detailed Description

Low-level function to setup an Icm20690 device.

### 7.33.2 Function Documentation

#### 7.33.2.1 int INV\_EXPORT inv\_icm20690\_accel\_fsr\_2\_reg ( int32\_t fsr )

Get internal register value for given FSR in mg for Accelerometer Allowed value are: 2000 (+/-2g), 4000 (+/-4g), 8000 (+/-8g), 16000 (+/-16g),..

#### Returns

internal register value for FSR configuration

#### 7.33.2.2 uint16\_t INV\_EXPORT inv\_icm20690\_get\_accel\_bandwidth ( struct inv\_icm20690 \* s )

Returns bandwidth range of accelerometer in hardware.

#### Returns

the bandwidth range

7.33.2.3 `uint8_t INV_EXPORT inv_icm20690_get_accel_fullscale ( struct inv_icm20690 * s )`

Returns fullscale range of accelerometer in hardware.

Returns

the fullscale range

7.33.2.4 `uint8_t INV_EXPORT inv_icm20690_get_accel_ois_fullscale ( struct inv_icm20690 * s )`

Returns fullscale range of OIS accelerometer in hardware.

Returns

the fullscale range

7.33.2.5 `uint16_t INV_EXPORT inv_icm20690_get_chip_base_sample_rate ( struct inv_icm20690 * s )`

Get internal sample rate currently configured.

Returns

Internal sample rate currently configured in MEMS registers in Hz

7.33.2.6 `int INV_EXPORT inv_icm20690_get_chip_info ( struct inv_icm20690 * s, uint8_t chip_info[3] )`

return WHOAMI value

Parameters

out	<code>chip_info[0]</code>	WHOAMI for device
out	<code>chip_info[1]</code>	MANUFACTURER_ID for device
out	<code>chip_info[2]</code>	CHIP_ID for device

Returns

0 on success, negative value on error

7.33.2.7 `uint8_t INV_EXPORT inv_icm20690_get_chip_power_state ( struct inv_icm20690 * s )`

Current wake status of the Mems chip.

Returns

the wake status

7.33.2.8 `uint16_t INV_EXPORT inv_icm20690_get_gyro_bandwidth ( struct inv_icm20690 * s )`

Returns bandwidth range of gyroscope in hardware.

Returns

the bandwidth range

7.33.2.9 `uint8_t INV_EXPORT inv_icm20690_get_gyro_fullscale ( struct inv_icm20690 * s )`

Returns fullscale range of gyroscope in hardware.

Returns

the fullscale range

7.33.2.10 `uint8_t INV_EXPORT inv_icm20690_get_gyro_ois_fullscale ( struct inv_icm20690 * s )`

Returns fullscale range of OIS gyroscope in hardware.

Returns

the fullscale range

7.33.2.11 `int INV_EXPORT inv_icm20690_get_whoami ( struct inv_icm20690 * s, uint8_t * whoami )`

return WHOAMI value

Parameters

out	whoami	WHOAMI for device
-----	--------	-------------------

Returns

0 on success, negative value on error

7.33.2.12 `int INV_EXPORT inv_icm20690_gyro_fsr_2_reg ( int32_t fsr )`

Get internal register value for given FSR in dps for Gyroscope Allowed value are: 250 (+/-250dps), 500 (+/-500dps), 1000 (+/-1000dps), 2000 (+/-2000dps), 31 and 32 (+/-31.25dps), 62 and 63 (+/-62.5dps), 125 (+/-125dps)

Returns

internal register value for FSR configuration

7.33.2.13 `int INV_EXPORT inv_icm20690_initialize ( struct inv_icm20690 * s )`

Initialize the device.

#### Returns

0 on success, negative value on error.

7.33.2.14 `int INV_EXPORT inv_icm20690_is_advanced_features_supported ( void )`

Get the available features according the base sensor device.

#### Parameters

<i>in</i>	<i>in</i>	0: No advanced features supported 1: OIS sensor and FSYNC behavior supported (for icm20690 only)
-----------	-----------	--

7.33.2.15 `int INV_EXPORT inv_icm20690_reg_2_accel_fsr ( uint8_t reg )`

Get FSR value in mg corresponding to internal register value for Accelerometer Allowed value are: 0 (+/-2g), 1 (+/-4g), 2 (+/-8g), 3 (+/-16g),..

#### Returns

FSR value in mg

7.33.2.16 `int INV_EXPORT inv_icm20690_reg_2_gyro_fsr ( uint8_t reg )`

Get FSR value in dps corresponding to internal register value for Gyroscope Allowed value are: 0 (+/-250dps), 1 (+/-500dps), 2 (+/-1000dps), 3 (+/-2000dps), 5 (+/-31.25dps), 6 (+/-62.5dps), 7 (+/-125dps)

#### Returns

FSR value in dps

7.33.2.17 `int INV_EXPORT inv_icm20690_set_accel_bandwidth ( struct inv_icm20690 * s, int level )`

Sets bandwidth range of accel in hardware.

#### Parameters

<i>in</i>	<i>level</i>	See mpu_accel_bw.
-----------	--------------	-------------------



**Returns**

0 on success, negative value on error.

7.33.2.18 `int INV_EXPORT inv_icm20690_set_accel_fullscale ( struct inv_icm20690 * s, int level )`

Sets fullscale range of accel in hardware.

**Parameters**

in	<i>level</i>	See mpu_accel_fs.
----	--------------	-------------------

**Returns**

0 on success, negative value on error.

7.33.2.19 `int INV_EXPORT inv_icm20690_set_accel_ois_fullscale ( struct inv_icm20690 * s, int level )`

Sets fullscale range of OIS accel in hardware.

**Parameters**

in	<i>level</i>	See mpu_accel_ois_fs.
----	--------------	-----------------------

**Returns**

0 on success, negative value on error.

7.33.2.20 `int INV_EXPORT inv_icm20690_set_chip_power_state ( struct inv_icm20690 * s, uint8_t func, uint8_t on_off )`

Sets the power state of the Ivory chip loop.

**Parameters**

in	<i>func</i>	CHIP_AWAKE, CHIP_LP_ENABLE
in	<i>on_off</i>	The functions are enabled if previously disabled and disabled if previously enabled based on the value of On/Off.

**Returns**

0 on success, negative value on error.

7.33.2.21 `int INV_EXPORT inv_icm20690_set_divider ( struct inv_icm20690 * s, uint8_t idiv )`

Set the mpu sample rate.

**Returns**

Value written to MPUREG\_SMPLRT\_DIV register.

**7.33.2.22** `int INV_EXPORT inv_icm20690_set_fsycn_bit_location ( struct inv_icm20690 * s, int bit_location )`

Set FSYNC bit location.

**Parameters**

<code>in</code>	<code>in</code>	0: Disable FSYNC pin data to be sampled 1: Enable FSYNC pin data to be sampled at TEMP_OUT_L[0]
-----------------	-----------------	---

**7.33.2.23** `int INV_EXPORT inv_icm20690_set_gyro_bandwidth ( struct inv_icm20690 * s, int level )`

Sets bandwidth range of gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_bw.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

**7.33.2.24** `int INV_EXPORT inv_icm20690_set_gyro_fullscale ( struct inv_icm20690 * s, int level )`

Sets fullscale range of gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_fs.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

**7.33.2.25** `int INV_EXPORT inv_icm20690_set_gyro_ois_fullscale ( struct inv_icm20690 * s, int level )`

Sets fullscale range of OIS gyro in hardware.

**Parameters**

<code>in</code>	<code>level</code>	See mpu_gyro_fs.
-----------------	--------------------	------------------

**Returns**

0 on success, negative value on error.

7.33.2.26 `int INV_EXPORT inv_icm20690_soft_reset ( struct inv_icm20690 * s )`

Perform a soft reset of the device.

**Returns**

0 on success, negative value on error.

## 7.34 Icm20690 driver transport

Low-level Icm20690 register access.

Collaboration diagram for Icm20690 driver transport:



### Functions

- int INV\_EXPORT [inv\\_icm20690\\_mems\\_write\\_reg](#) (struct [inv\\_icm20690](#) \*s, uint16\_t reg, uint32\_t length, const uint8\_t \*data)  
*Writes data from a register on mems.*
- int INV\_EXPORT [inv\\_icm20690\\_mems\\_write\\_reg\\_one](#) (struct [inv\\_icm20690](#) \*s, uint16\_t reg, uint8\_t data)  
*Writes a single byte of data from a register on mems with no power control.*
- int INV\_EXPORT [inv\\_icm20690\\_mems\\_read\\_reg](#) (struct [inv\\_icm20690](#) \*s, uint16\_t reg, uint32\_t length, uint8\_t \*data)  
*Reads data from a register on mems.*

### 7.34.1 Detailed Description

Low-level Icm20690 register access.

### 7.34.2 Function Documentation

7.34.2.1 int INV\_EXPORT [inv\\_icm20690\\_mems\\_read\\_reg](#) ( struct [inv\\_icm20690](#) \* s, uint16\_t *reg*, uint32\_t *length*, uint8\_t \* *data* )

Reads data from a register on mems.

#### Parameters

in	<i>reg</i>	register address
in	<i>length</i>	length of data
out	<i>data</i>	output data from the register

#### Returns

0 in case of success, -1 for any error

7.34.2.2 `int INV_EXPORT inv_icm20690_mems_write_reg ( struct inv_icm20690 * s, uint16_t reg, uint32_t length, const uint8_t * data )`

Writes data from a register on mems.

#### Parameters

in	<i>length</i>	number of byte to be written
out	<i>data</i>	output data from the register

#### Returns

0 in case of success, -1 for any error

7.34.2.3 `int INV_EXPORT inv_icm20690_mems_write_reg_one ( struct inv_icm20690 * s, uint16_t reg, uint8_t data )`

Writes a single byte of data from a register on mems with no power control.

#### Parameters

in	<i>reg</i>	Register address
out	<i>data</i>	Data to be written

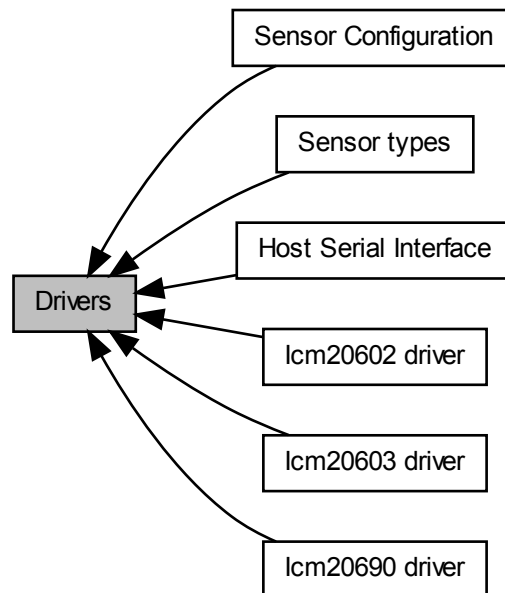
#### Returns

0 in case of success, -1 for any error

## 7.35 Drivers

Low-level drivers for InvenSense devices.

Collaboration diagram for Drivers:



### Modules

- [Sensor types](#)  
*Sensor related types definitions.*
- [Sensor Configuration](#)  
*General sensor configuration types definitions.*
- [Host Serial Interface](#)  
*Virtual abstraction of host adapter for serial interface.*
- [lcm20602 driver](#)  
*Low-level driver for lcm20602 devices.*
- [lcm20603 driver](#)  
*Low-level driver for lcm20603 devices.*
- [lcm20690 driver](#)  
*Low-level driver for lcm20690 devices.*

### 7.35.1 Detailed Description

Low-level drivers for InvenSense devices.

Those drivers are intended to be portable and used in embedded context. They can be used directly but this is not advised as they may not be user-friendly. The proper way to access a device from the application is through the [Device API](#).

## 7.36 Utils

Utility functions.

Utility functions.





## Chapter 8

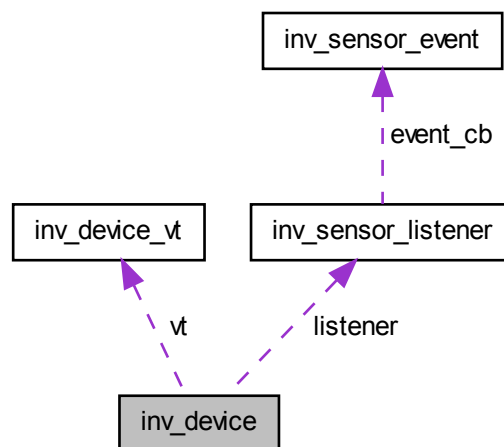
# Class Documentation

### 8.1 inv\_device Struct Reference

Abstract device object definition.

```
#include <Device.h>
```

Collaboration diagram for inv\_device:



#### Public Attributes

- void \* [instance](#)  
*pointer to object instance*
- const struct [inv\\_device\\_vt](#) \* [vt](#)  
*pointer to object virtual table*
- const [inv\\_sensor\\_listener\\_t](#) \* [listener](#)  
*pointer to listener instance*

### 8.1.1 Detailed Description

Abstract device object definition.

Examples:

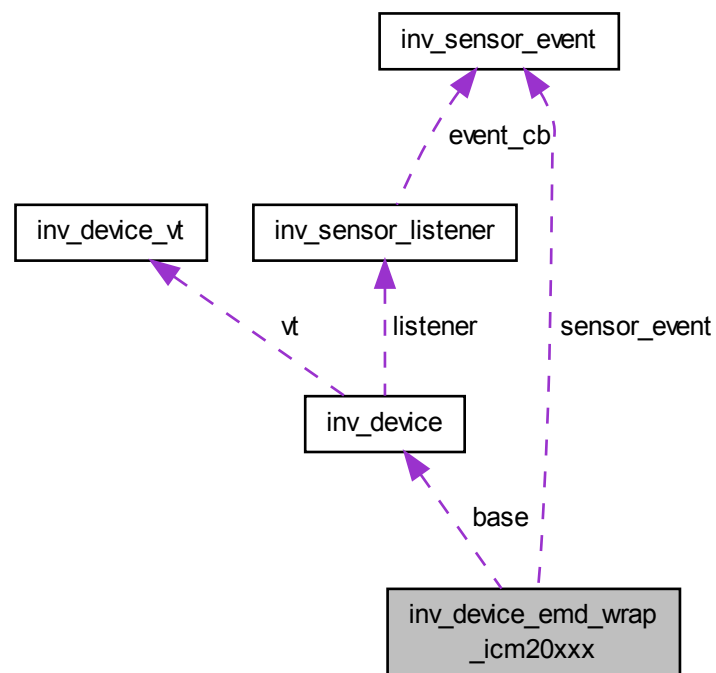
[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

The documentation for this struct was generated from the following file:

- Device.h

## 8.2 inv\_device\_emd\_wrap\_icm20xxx Struct Reference

Collaboration diagram for inv\_device\_emd\_wrap\_icm20xxx:



The documentation for this struct was generated from the following file:

- DeviceEmdWrapIcm20xxx.h

### 8.3 inv\_device\_emd\_wrap\_icm20xxx\_serial Struct Reference

The documentation for this struct was generated from the following file:

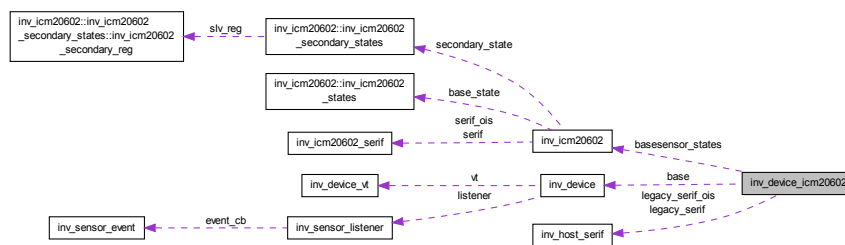
- DeviceEmdWrapIcm20xxx.h

### 8.4 inv\_device\_icm20602 Struct Reference

States for Icm20602 device.

```
#include <DeviceIcm20602.h>
```

Collaboration diagram for inv\_device\_icm20602:



#### 8.4.1 Detailed Description

States for Icm20602 device.

Examples:

[ExampleDeviceIcm20602EMD.c](#).

The documentation for this struct was generated from the following file:

- DeviceIcm20602.h

### 8.5 inv\_device\_icm20602\_config\_bias\_st Struct Reference

Bias collected during self-test (config INV\_DEVICE\_ICM20602\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.

```
#include <DeviceIcm20602.h>
```

## Public Attributes

- `int32_t gyr_bias_lp [3]`  
*Gyro LP mode X,Y,Z.*
- `int32_t gyr_bias_nl [3]`  
*Gyro normal mode X,Y,Z.*
- `int32_t acc_bias_lp [3]`  
*Accel LP mode X,Y,Z.*
- `int32_t acc_bias_nl [3]`  
*Accel normal mode X,Y,Z.*

### 8.5.1 Detailed Description

Bias collected during self-test (config INV\_DEVICE\_ICM20602\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.

The documentation for this struct was generated from the following file:

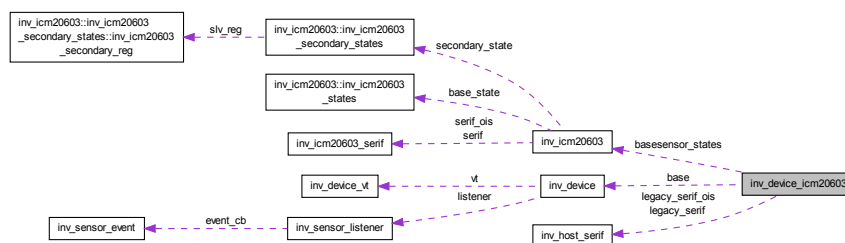
- DeviceIcm20602.h

## 8.6 inv\_device\_icm20603 Struct Reference

States for Icm20603 device.

```
#include <DeviceIcm20603.h>
```

Collaboration diagram for `inv_device_icm20603`:



### 8.6.1 Detailed Description

States for Icm20603 device.

Examples:

[ExampleDeviceIcm20603EMD.c](#).

The documentation for this struct was generated from the following file:

- DeviceIcm20603.h

## 8.7 inv\_device\_icm20603\_config\_bias\_st Struct Reference

Bias collected during self-test (config INV\_DEVICE\_ICM20603\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.

```
#include <DeviceIcm20603.h>
```

## Public Attributes

- `int32_t gyr_bias_lp [3]`  
*Gyro LP mode X, Y, Z.*
- `int32_t gyr_bias_nl [3]`  
*Gyro normal mode X, Y, Z.*
- `int32_t acc_bias_lp [3]`  
*Accel LP mode X, Y, Z.*
- `int32_t acc_bias_nl [3]`  
*Accel normal mode X, Y, Z.*

### 8.7.1 Detailed Description

Bias collected during self-test (config INV\_DEVICE\_ICM20603\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.

The documentation for this struct was generated from the following file:

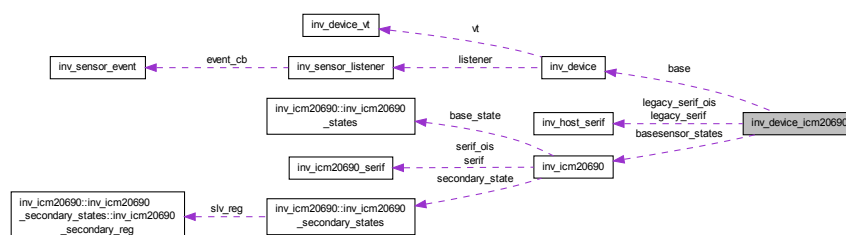
- DeviceIcm20603.h

## 8.8 inv\_device\_icm20690 Struct Reference

States for lcm20690 device.

```
#include <DeviceIcm20690.h>
```

Collaboration diagram for inv\_device\_icm20690:



### 8.8.1 Detailed Description

States for Icm20690 device.

Examples:

[ExampleDeviceIcm20690EMD.c](#).

The documentation for this struct was generated from the following file:

- DeviceIcm20690.h

## 8.9 inv\_device\_icm20690\_config\_bias\_st Struct Reference

Bias collected during self-test (config INV\_DEVICE\_ICM20690\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.

```
#include <DeviceIcm20690.h>
```

### Public Attributes

- `int32_t` [gyr\\_bias\\_lp](#) [3]  
*Gyro LP mode X,Y,Z.*
- `int32_t` [gyr\\_bias\\_nl](#) [3]  
*Gyro normal mode X,Y,Z.*
- `int32_t` [acc\\_bias\\_lp](#) [3]  
*Accel LP mode X,Y,Z.*
- `int32_t` [acc\\_bias\\_nl](#) [3]  
*Accel normal mode X,Y,Z.*

### 8.9.1 Detailed Description

Bias collected during self-test (config INV\_DEVICE\_ICM20690\_CONFIG\_BIAS\_ST) Value is scaled by  $2^{16}$ , accel is gee and gyro is dps.

The documentation for this struct was generated from the following file:

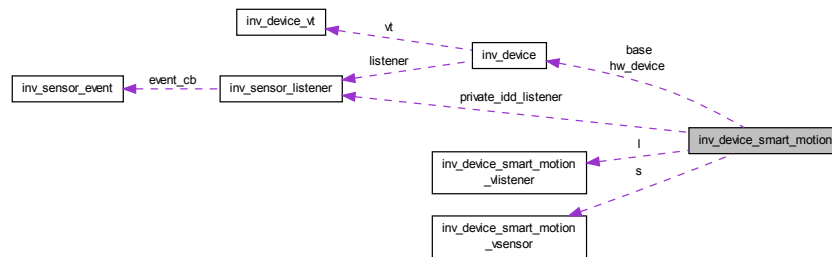
- DeviceIcm20690.h

## 8.10 inv\_device\_smart\_motion Struct Reference

States for SmartMotion device implementation.

```
#include <DeviceSmartMotion.h>
```

Collaboration diagram for inv\_device\_smart\_motion:



### Public Attributes

- [inv\\_device\\_t](#) [base](#)  
base [inv\\_device\\_t](#) states
- [inv\\_device\\_t](#) \* [hw\\_device](#)  
reference to [inv\\_device\\_t](#) that will provide sensor data for base sensor
- [uint8\\_t](#) [base\\_chip\\_info](#) [3]  
parameter given at init time to identify underlying ICM
- [inv\\_sensor\\_listener\\_t](#) [private\\_idd\\_listener](#)  
internal IDD listener to catch events from HW device
- [InvList](#) [leaves\\_list](#)  
list of graph leaves (corresponds to sensors not present at hardware level and that are emulated)
- struct {  
  struct [inv\\_device\\_smart\\_motion\\_vsensor](#) [inv\\_device\\_smart\\_motion::s](#)  
    placeholder for roots VSensor  
  struct [inv\\_device\\_smart\\_motion\\_vlistener](#) [inv\\_device\\_smart\\_motion::l](#)  
    placeholder for leaves corresponding to root VSensor  
} [graph\\_roots](#) [5]  
  
placeholder holder for graph roots objects
- struct {  
  struct [inv\\_device\\_smart\\_motion\\_vlistener](#) [inv\\_device\\_smart\\_motion::l](#)  
    placeholder for leaves VSensorListener  
} [graph\\_leaves](#) [24]  
  
placeholder for leaves corresponding to emulated sensors
- [uint64\\_t](#) [hw\\_sensor\\_available\\_mask](#)  
mask to keep track of HW available sensor from underlying device

### 8.10.1 Detailed Description

States for SmartMotion device implementation.

The documentation for this struct was generated from the following file:

- DeviceSmartMotion.h

## 8.11 inv\_device\_smart\_motion\_vlistener Struct Reference

Internal definition for graph leaves (based on VSensorListener)

```
#include <DeviceSmartMotion.h>
```

### Public Attributes

- VSensorListener [vlistener](#)  
*base VSensorListener states*
- void(\* [build\\_event](#))(const void \*, [inv\\_sensor\\_event\\_t](#) \*)  
*callback to convert VSensorData to IDD sensor event*
- uint8\_t [idd\\_type](#)  
*corresponding IDD sensor type*
- InvList [node](#)  
*pointer to next element when stored in a list*

### 8.11.1 Detailed Description

Internal definition for graph leaves (based on VSensorListener)

The documentation for this struct was generated from the following file:

- DeviceSmartMotion.h

## 8.12 inv\_device\_smart\_motion\_vsensor Struct Reference

Internal definition for graph roots (based on VSensor)

```
#include <DeviceSmartMotion.h>
```

### Public Attributes

- VSensor [vsensor](#)  
*base VSensor states*
- void(\* [build\\_vsensor\\_data](#))(const [inv\\_sensor\\_event\\_t](#) \*, void \*)  
*callback to convert IDD sensor event to VSensorData*
- uint8\_t [idd\\_type](#)  
*corresponding IDD sensor type*



### 8.12.1 Detailed Description

Internal definition for graph roots (based on VSensor)

The documentation for this struct was generated from the following file:

- DeviceSmartMotion.h

## 8.13 inv\_device\_vt Struct Reference

Device virtual table definition.

```
#include <Device.h>
```

### 8.13.1 Detailed Description

Device virtual table definition.

The documentation for this struct was generated from the following file:

- Device.h

## 8.14 inv\_fw\_version Struct Reference

FW version structure definition.

```
#include <Device.h>
```

### Public Attributes

- uint8\_t [patch](#)  
*major, minor, patch version number*
- char [suffix](#) [16]  
*version suffix string (always terminated by '\0')*
- uint32\_t [crc](#)  
*FW checksum.*

### 8.14.1 Detailed Description

FW version structure definition.

The documentation for this struct was generated from the following file:

- Device.h

## 8.15 inv\_host\_serif Struct Reference

Serial Interface interface definition.

```
#include <HostSerif.h>
```

### Public Attributes

- `int(* open)(void)`  
*Open connection to and initialize Serial Interface adapter.*
- `int(* close)(void)`  
*Close connection to Serial Interface adapter.*
- `int(* read_reg)(uint8_t reg, uint8_t *data, uint32_t len)`  
*Perform a read register transaction over the serial interface.*
- `int(* write_reg)(uint8_t reg, const uint8_t *data, uint32_t len)`  
*Perform a write register transaction over the serial interface.*
- `int(* register_interrupt_callback)(void(*interrupt_cb)(void *context, int int_num), void *context)`  
*Register a callback to the adapter.*
- `uint32_t max_read_size`  
*Maximum number of bytes allowed per serial read.*
- `uint32_t max_write_size`  
*Maximum number of bytes allowed per serial write.*
- `int serif_type`  
*Type of underlying serial interface.*

### 8.15.1 Detailed Description

Serial Interface interface definition.

### 8.15.2 Member Data Documentation

#### 8.15.2.1 `int(* inv_host_serif::close)(void)`

Close connection to Serial Interface adapter.

Returns

0 on sucess, negative value on error

#### 8.15.2.2 `int(* inv_host_serif::open)(void)`

Open connection to and initialize Serial Interface adapter.

Returns

0 on sucess, negative value on error

#### 8.15.2.3 `int(* inv_host_serif::read_reg)(uint8_t reg, uint8_t *data, uint32_t len)`

Perform a read register transaction over the serial interface.

## Parameters

in	<i>reg</i>	register
out	<i>data</i>	pointer to output buffer
in	<i>len</i>	number of byte to read (should not exceed MAX_TRANSACTION_SIZE)

## Returns

0 on sucess, negative value on error

**8.15.2.4** `int(* inv_host_serif::register_interrupt_callback)(void(*interrupt_cb)(void *context, int int_num), void *context)`

Register a callback to the adapter.

## Parameters

in	<i>interrupt_cb</i>	callback to call on interrupt
in	<i>context</i>	context passed to callback

## Returns

0 on sucess, negative value on error

**8.15.2.5** `int(* inv_host_serif::write_reg)(uint8_t reg, const uint8_t *data, uint32_t len)`

Perform a write register transaction over the serial interface.

## Parameters

in	<i>reg</i>	register
out	<i>data</i>	pointer to output buffer
in	<i>len</i>	number of byte to read (should not exceed MAX_TRANSACTION_SIZE)

## Returns

0 on sucess, negative value on error

The documentation for this struct was generated from the following file:

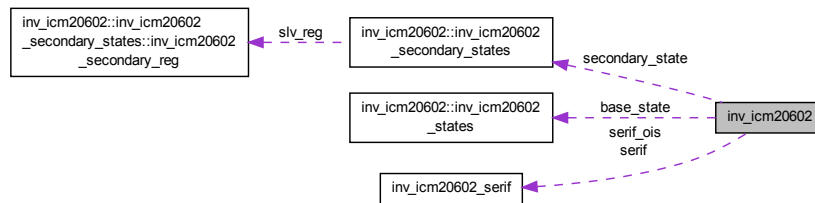
- HostSerif.h

## 8.16 inv\_icm20602 Struct Reference

Icm20602 driver states definition.

```
#include <Icm20602.h>
```

Collaboration diagram for inv\_icm20602:



## Classes

- struct [inv\\_icm20602\\_secondary\\_states](#)  
*secondary device support*
- struct [inv\\_icm20602\\_states](#)  
*icm20602 internal state structure*

### 8.16.1 Detailed Description

Icm20602 driver states definition.

The documentation for this struct was generated from the following file:

- Icm20602.h

## 8.17 inv\_icm20602\_fifo\_states Struct Reference

Check and retrieve for new data.

```
#include <Icm20602Ctrl.h>
```

### Public Attributes

- `uint8_t overflow:1`  
*indicates FIFO overflow: user should restart all sensors if this occurs*

### 8.17.1 Detailed Description

Check and retrieve for new data.

The documentation for this struct was generated from the following file:

- Icm20602Ctrl.h

## 8.18 inv\_icm20602::inv\_icm20602\_secondary\_states::inv\_icm20602\_secondary\_reg Struct Reference

secondary register mapping

```
#include <Icm20602.h>
```

### 8.18.1 Detailed Description

secondary register mapping

The documentation for this struct was generated from the following file:

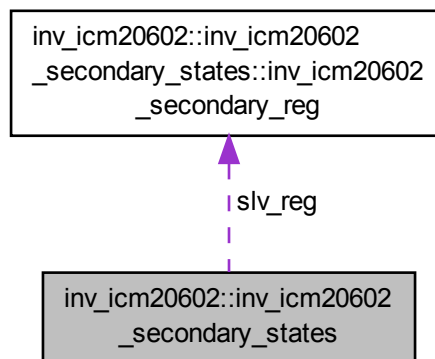
- Icm20602.h

## 8.19 inv\_icm20602::inv\_icm20602\_secondary\_states Struct Reference

secondary device support

```
#include <Icm20602.h>
```

Collaboration diagram for inv\_icm20602::inv\_icm20602\_secondary\_states:



### Classes

- struct [inv\\_icm20602\\_secondary\\_reg](#)  
secondary register mapping

### 8.19.1 Detailed Description

secondary device support

The documentation for this struct was generated from the following file:

- Icm20602.h

## 8.20 inv\_icm20602\_serif Struct Reference

basesensor serial interface

```
#include <Icm20602Serif.h>
```

### 8.20.1 Detailed Description

basesensor serial interface

The documentation for this struct was generated from the following file:

- Icm20602Serif.h

## 8.21 inv\_icm20602::inv\_icm20602\_states Struct Reference

icm20602 internal state structure

```
#include <Icm20602.h>
```

### 8.21.1 Detailed Description

icm20602 internal state structure

The documentation for this struct was generated from the following file:

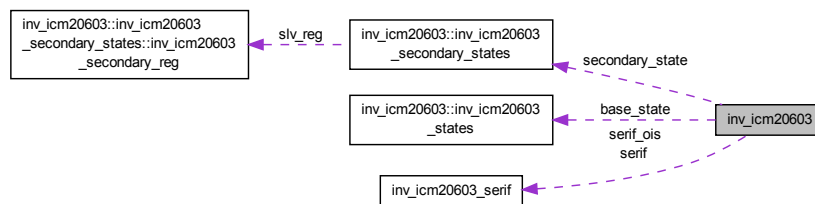
- Icm20602.h

## 8.22 inv\_icm20603 Struct Reference

lcm20603 driver states definition.

```
#include <Icm20603.h>
```

Collaboration diagram for inv\_icm20603:



### Classes

- struct [inv\\_icm20603\\_secondary\\_states](#)  
*secondary device support*
- struct [inv\\_icm20603\\_states](#)  
*icm20603 internal state structure*

### 8.22.1 Detailed Description

lcm20603 driver states definition.

The documentation for this struct was generated from the following file:

- lcm20603.h

## 8.23 inv\_icm20603\_fifo\_states Struct Reference

Check and retrieve for new data.

```
#include <Icm20603Ctrl.h>
```

### Public Attributes

- `uint8_t overflow:1`  
*indicates FIFO overflow: user should restart all sensors if this occurs*

### 8.23.1 Detailed Description

Check and retrieve for new data.

The documentation for this struct was generated from the following file:

- Icm20603Ctrl.h

## 8.24 inv\_icm20603::inv\_icm20603\_secondary\_states::inv\_icm20603\_secondary\_reg Struct Reference

secondary register mapping

```
#include <Icm20603.h>
```

### 8.24.1 Detailed Description

secondary register mapping

The documentation for this struct was generated from the following file:

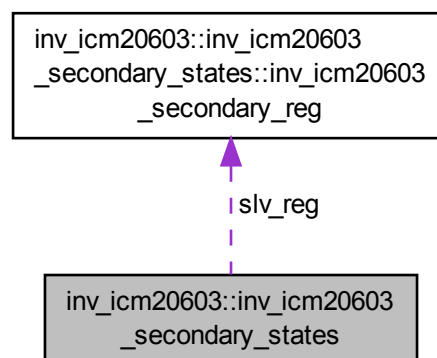
- Icm20603.h

## 8.25 inv\_icm20603::inv\_icm20603\_secondary\_states Struct Reference

secondary device support

```
#include <Icm20603.h>
```

Collaboration diagram for inv\_icm20603::inv\_icm20603\_secondary\_states:





## Classes

- struct [inv\\_icm20603\\_secondary\\_reg](#)  
*secondary register mapping*

### 8.25.1 Detailed Description

secondary device support

The documentation for this struct was generated from the following file:

- Icm20603.h

## 8.26 inv\_icm20603\_serif Struct Reference

basesensor serial interface

```
#include <Icm20603Serif.h>
```

### 8.26.1 Detailed Description

basesensor serial interface

The documentation for this struct was generated from the following file:

- Icm20603Serif.h

## 8.27 inv\_icm20603::inv\_icm20603\_states Struct Reference

icm20603 internal state structure

```
#include <Icm20603.h>
```

### 8.27.1 Detailed Description

icm20603 internal state structure

The documentation for this struct was generated from the following file:

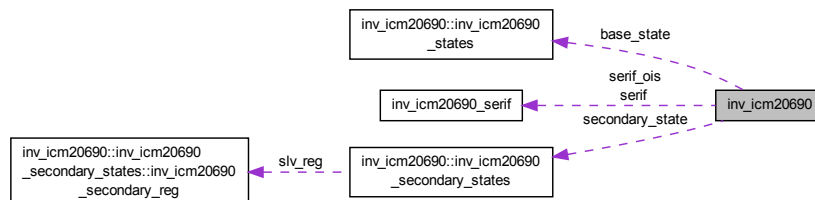
- Icm20603.h

## 8.28 inv\_icm20690 Struct Reference

Icm20690 driver states definition.

```
#include <Icm20690.h>
```

Collaboration diagram for inv\_icm20690:



### Classes

- struct [inv\\_icm20690\\_secondary\\_states](#)  
*secondary device support*
- struct [inv\\_icm20690\\_states](#)  
*icm20690 internal state structure*

### 8.28.1 Detailed Description

Icm20690 driver states definition.

The documentation for this struct was generated from the following file:

- Icm20690.h

## 8.29 inv\_icm20690\_fifo\_states Struct Reference

Check and retrieve for new data.

```
#include <Icm20690Ctrl.h>
```

### Public Attributes

- `uint8_t overflow:1`  
*indicates FIFO overflow: user should restart all sensors if this occurs*

### 8.29.1 Detailed Description

Check and retrieve for new data.

The documentation for this struct was generated from the following file:

- Icm20690Ctrl.h

## 8.30 inv\_icm20690::inv\_icm20690\_secondary\_states::inv\_icm20690\_secondary\_reg Struct Reference

secondary register mapping

```
#include <Icm20690.h>
```

### 8.30.1 Detailed Description

secondary register mapping

The documentation for this struct was generated from the following file:

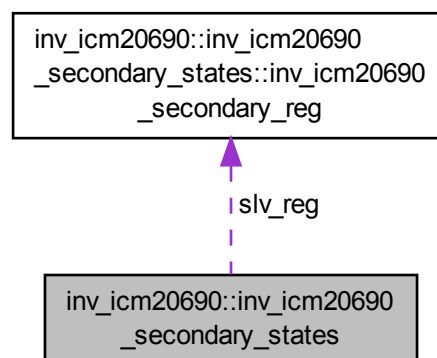
- Icm20690.h

## 8.31 inv\_icm20690::inv\_icm20690\_secondary\_states Struct Reference

secondary device support

```
#include <Icm20690.h>
```

Collaboration diagram for inv\_icm20690::inv\_icm20690\_secondary\_states:



## Classes

- struct [inv\\_icm20690\\_secondary\\_reg](#)  
*secondary register mapping*

### 8.31.1 Detailed Description

secondary device support

The documentation for this struct was generated from the following file:

- Icm20690.h

## 8.32 inv\_icm20690\_serif Struct Reference

basesensor serial interface

```
#include <Icm20690Serif.h>
```

### 8.32.1 Detailed Description

basesensor serial interface

The documentation for this struct was generated from the following file:

- Icm20690Serif.h

## 8.33 inv\_icm20690::inv\_icm20690\_states Struct Reference

icm20690 internal state structure

```
#include <Icm20690.h>
```

### 8.33.1 Detailed Description

icm20690 internal state structure

The documentation for this struct was generated from the following file:

- Icm20690.h

## 8.34 inv\_sensor\_config\_bac Struct Reference

Define the configuration for BAC.

```
#include <SensorConfig.h>
```

### 8.34.1 Detailed Description

Define the configuration for BAC.

## Parameters

<i>enableNotify</i>	enable disable notify
---------------------	-----------------------

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.35 inv\_sensor\_config\_BSCD Struct Reference

Define the configuration for the BSCD virtual sensor.

```
#include <SensorConfig.h>
```

### 8.35.1 Detailed Description

Define the configuration for the BSCD virtual sensor.

## Parameters

<i>Age</i>	age in year; Range is (0;100). Default is 35.
<i>Gender</i>	gender is 0 for men, 1 for female. Default is 0
<i>Height</i>	height in centimeter; Range is (50;250). Default is 175.
<i>Weight</i>	weight in kg; Range is (3;300). Default is 75
<i>enableNotify</i>	bitmask to enable/disable notify on a a specific sensor event bit 0 (1): enable/disable notify on BAC event bit 1 (2): enable/disable notify on step counter event bit 2 (4): enable/disable notify on energy expenditure event bit 3 (8): enable/disable notify on distance event

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.36 inv\_sensor\_config\_context Struct Reference

Define configuration context value (associated with INV\_SENSOR\_CONFIG\_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implemetation.

```
#include <SensorConfig.h>
```

### 8.36.1 Detailed Description

Define configuration context value (associated with INV\_SENSOR\_CONFIG\_CONTEXT config ID) Context is an arbitrary buffer specific to the sensor and device implemetation.

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.37 `inv_sensor_config_distance` Struct Reference

Define the configuration for the distance's algorithm.

```
#include <SensorConfig.h>
```

### 8.37.1 Detailed Description

Define the configuration for the distance's algorithm.

#### Parameters

<i>user_height</i>	height of the user in cm
<i>enableNotify</i>	enable disable notify

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.38 `inv_sensor_config_double_tap` Struct Reference

Define the configuration for the double tap's algorithm.

```
#include <SensorConfig.h>
```

### 8.38.1 Detailed Description

Define the configuration for the double tap's algorithm.

#### Parameters

<i>minimum_threshold</i>	This parameter sets the minimum threshold to reach in order to start a Tap detection. Default value is 2000, recommended range [500 ; 2500]
<i>t_max</i>	This parameter sets the maximum time after a Tap event in [sample]. Default value is 100, recommended range [30 ; 200].

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.39 `inv_sensor_config_energy_expenditure` Struct Reference

Define the configuration for the energy expenditure's algorithm.

```
#include <SensorConfig.h>
```

### 8.39.1 Detailed Description

Define the configuration for the energy expenditure's algorithm.

#### Parameters

<i>age</i>	age in year; Range is (0;100).
<i>gender</i>	gender is 0 for men, 1 for female.
<i>height</i>	height in centimeter; Range is (50;250)
<i>weight</i>	weight in kg; Range is (3;300)
<i>enableNotify</i>	enable disable notify

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.40 inv\_sensor\_config\_fsr Struct Reference

Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV\_SENSOR\_CONFIG\_FSR config ID) Value is expetcted to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.

```
#include <SensorConfig.h>
```

### 8.40.1 Detailed Description

Define full-scale range value for accelero, gyro or mangetometer based sensor (associated with INV\_SENSOR\_CONFIG\_FSR config ID) Value is expetcted to be expressed in mg, dps and uT for accelero, gyro or mangetometer eg: +/-2g = 2000 +/-250 dps = 250 +/-2000 uT = 2000.

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.41 inv\_sensor\_config\_gain Struct Reference

Define gain matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.

```
#include <SensorConfig.h>
```

### 8.41.1 Detailed Description

Define gain matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_GAIN config ID) Gain matrix value can be set (is supported by device implementation) to correct for cross-axis defect.

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.42 inv\_sensor\_config\_mounting\_mtx Struct Reference

Define mounting matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.

```
#include <SensorConfig.h>
```

### 8.42.1 Detailed Description

Define mounting matrix value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX config ID) Mounting matrix value can be set (is supported by device implementation) to convert from sensor reference to system reference.

Value is expected to be a rotation matrix.

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.43 inv\_sensor\_config\_offset Struct Reference

Define offset vector value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.

```
#include <SensorConfig.h>
```

### 8.43.1 Detailed Description

Define offset vector value for 3-axis sensors (associated with INV\_SENSOR\_CONFIG\_OFFSET config ID) Offset value can be set (is supported by device implementation) to correct for bias defect.

If applied to RAW sensor, value is expected to be in lsb. If applied to other sensor, value is expected to be in sensor unit (g, uT or dps).

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.44 inv\_sensor\_config\_powermode Struct Reference

Define chip power mode (associated with INV\_SENSOR\_CONFIG\_POWER\_MODE config ID) Value is expected to be 0 for low power or 1 for low noise.

```
#include <SensorConfig.h>
```



### 8.44.1 Detailed Description

Define chip power mode (associated with INV\_SENSOR\_CONFIG\_POWER\_MODE config ID) Value is expected to be 0 for low power or 1 for low noise.

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.45 inv\_sensor\_config\_shake\_wrist Struct Reference

Define the configuration for the shake wrist's algorithm.

```
#include <SensorConfig.h>
```

### 8.45.1 Detailed Description

Define the configuration for the shake wrist's algorithm.

#### Parameters

<i>max_period</i>	This parameter sets the maximal duration for half oscillation to detect a Shake wrist. The default value is 20, recommend range [15 ; 40], 15 for the lower sensitivity and 40 for the higher sensitivity. Notice that increasing the sensitivity will increase the number of false detection, and also slightly increase response time.
<i>dummy_padding</i>	Dummy byte for padding. Set it to 0.

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.46 inv\_sensor\_config\_stepc Struct Reference

Define the configuration for steps counter.

```
#include <SensorConfig.h>
```

### 8.46.1 Detailed Description

Define the configuration for steps counter.

#### Parameters

<i>enableNotify</i>	enable disable notify
---------------------	-----------------------

The documentation for this struct was generated from the following file:

- SensorConfig.h

## 8.47 inv\_sensor\_event Struct Reference

Sensor event definition.

```
#include <SensorTypes.h>
```

### Public Attributes

- unsigned int [sensor](#)  
*sensor type*
- int [status](#)  
*sensor data status as of enum inv\_sensor\_status*
- uint64\_t [timestamp](#)  
*sensor data timestamp in us*
- union {
  - struct {
    - float [vect](#) [3]  
*x,y,z vector data*
    - float [bias](#) [3]  
*x,y,z bias vector data*
    - uint8\_t [accuracy\\_flag](#)  
*accuracy flag*
  - [acc](#)  
*3d accelerometer data in g*
  - struct {
    - float [vect](#) [3]  
*x,y,z vector data*
    - float [bias](#) [3]  
*x,y,z bias vector data (for uncal sensor variant)*
    - uint8\_t [accuracy\\_flag](#)  
*accuracy flag*
  - [mag](#)  
*3d magnetometer data in uT*
  - struct {
    - float [vect](#) [3]  
*x,y,z vector data*
    - float [bias](#) [3]  
*x,y,z bias vector data (for uncal sensor variant)*
    - uint8\_t [accuracy\\_flag](#)  
*accuracy flag*
  - [gyr](#)  
*3d gyroscope data in deg/s*
  - struct {
    - float [quat](#) [4]  
*w,x,y,z quaternion data*
    - float [accuracy](#)  
*heading accuracy in deg*
    - uint8\_t [accuracy\\_flag](#)  
*accuracy flag specific for GRV*

```

} quaternion
    quaternion data
struct {
    float z
        x,y,z angles in deg as defined by Google Orientation sensor
    uint8_t accuracy_flag
        heading accuracy in deg
} orientation
    orientation data
struct {
    float bpm
        beat per minute
    uint8_t confidence
        confidence level
    uint8_t sqi
        signal quality as seen by the the HRM engine
} hrm
    heart rate monitor data
struct {
    int32_t acc [3]
        accel data used by hrm algorithm
    int32_t gyr [3]
        gyro data used by hrm algorithm
    uint32_t ppg_value
        ppg value read from HRM sensor
    float ppm
        beat per minute
    uint8_t confidence
        confidence level
    uint8_t sqi
        signal quality as seen by the the HRM engine
    uint8_t touch_status
        touch status, detected or not by the PPG
    uint8_t gyrEnable
        1 gyro is enable else 0
} hrmlogger
    heart rate monitor logger data
struct {
    int16_t rr_interval [4]
        beat-to-beat(RR) interval
} hrv
    heart rate variability data
struct {
    uint32_t ppg_value
        ppg value read from HRM sensor
    uint8_t touch_status
        touch status, detected or not
} rawppg
    raw heart rate monitor data
struct {
    uint8_t sleep_phase
        state of sleep phases: 0 not defined, 1 restless sleep, 2 light sleep, 3 deep sleep
    uint32_t timestamp
        time stamp of the sleep phase transition (seconds)
    int32_t sleep_onset
        time until first period of 20 min sleep without more than 1 min wake
    int32_t sleep_latency
        time until first sleep phase

```

```

uint32_t time_in_bed
    time in bed (seconds)
uint32_t total_sleep_time
    total sleep time (seconds)
uint8_t sleep_efficiency
    ratio between total sleep time and time in bed
} sleepanalysis
    sleep analysis data
struct {
    int event
        BAC extended data begin/end event as of enum inv_sensor_bac_ext_event.
} bacext
    activity classifier (BAC) extended data
struct {
    uint32_t durationWalk
        ms
    uint32_t durationRun
        ms
    uint32_t durationTransportSit
        ms
    uint32_t durationTransportStand
        ms
    uint32_t durationBiking
        ms
    uint32_t durationStillSit
        ms
    uint32_t durationStillStand
        ms
    uint32_t durationTotalSit
        Still-Sit + Transport-Sit + Biking (ms)
    uint32_t durationTotalStand
        Still-Stand + Transport-Stand (ms)
    uint32_t stepWalk
        walk step count
    uint32_t stepRun
        run step count
} bacstat
    activity classifier (BAC) statistics data
struct {
    int32_t floorsUp
        number of floors climbed Up on foot by user.
    int32_t floorsDown
        number of floors climbed Down on foot by user.
} floorclimb
    floor climbed data
struct {
    int32_t instantEEkcal
        energy expenditure in kilocalorie/min since last output.
    int32_t instantEEmets
        energy expenditure in METs(Metabolic Equivalent of Task) since last output.
    int32_t cumulativeEEkcal
        cumulative energy expenditure since the last reset in kilocalorie.
    int32_t cumulativeEEmets
        cumulative energy expenditure since the last reset in METs (Metabolic Equivalent of Task).
} energyexp
    energy expenditure data
struct {
    int32_t distanceWalk

```

```

    distance in meters
    int32_t distanceRun
    distance in meters
} distance
    distance data
struct {
    float tmp
    temperature in deg celcius
} temperature
    temperature data
struct {
    float percent
    relative humidity in %
} humidity
    humidity data
struct {
    uint64_t count
    number of steps
} step
    step-counter data
struct {
    uint32_t level
    light level in lux
} light
    light data
struct {
    uint32_t distance
    distance in mm
} proximity
    proximity data
struct {
    uint32_t pressure
    pressure in Pa
} pressure
    pressure data
struct {
    int event
    BAC data begin/end event as of enum inv_sensor_bac_event.
} bac
    BAC data.
struct {
    uint32_t fxdata [12]
    PDR data in fixpoint.
} pdr
    PDR data.
struct {
    float vect [3]
    x,y,z vector data
    float bias [3]
    x,y,z bias vector data (for uncal sensor variant)
    int16_t delta_ts
    timestamp delta between standard gyro and EIS gyro
} eis
    EIS data.
struct {
    int32_t vect [3]
    x,y,z vector data
    uint32_t fsr

```

```

    full scale range
} raw3d
    3d raw acc, mag or gyr
struct {
    int32_t raw
        raw temperature value
} rawtemp
    Raw temperature data.
struct {
    uint8_t status [6]
        raw temperature value
} tsimu_status
    TSIMU status data.
inv_bool_t event
    event state for gesture-like sensor (SMD, B2S, Step-detector, Tilt-detector, Wake, Glance, Pick-Up, Shake, Double-tap,
struct {
} wom
    < FSYNC tag (EIS sensor)
struct {
    uint8_t * buffer
        pointer to buffer
    uint32_t size
        current buffer size
} audio_buffer
    Wake-up on motion data.
struct {
    struct {
        int event
            BAC data begin/end event as of enum inv_sensor_bac_event.
    } bac
        BAC data.
    struct {
        uint64_t count
            number of steps
    } step
        step-counter data
    int32_t cumulativeEEkcal
        cumulative energy expenditure since the last reset in kilocalorie.
    int32_t distance
        sum of walk and run distance in meters
} bscd
    buffer of custom BSCD
struct {
    int32_t raw_pressure
        raw pressure
    float pressure
        pressure in Pa
    int32_t raw_temperature
        raw temperature
    float temperature
        temperature in deg C
} custom_pressure
    pressure data
uint8_t reserved [INV_SENSOR_EVENT_DATA_SIZE]
    reserved sensor data for future sensor
} data

```

sensor data

- `int32_t table` [7]  
*data encrypted table*
- `int16_t delay_count`  
*delay counter in us between FSYNC tag and previous gyro data*

### 8.47.1 Detailed Description

Sensor event definition.

Examples:

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

### 8.47.2 Member Data Documentation

#### 8.47.2.1 `uint8_t inv_sensor_event::accuracy_flag`

accuracy flag

heading accuracy in deg

accuracy flag specific for GRV

#### 8.47.2.2 `struct { ... } inv_sensor_event::audio_buffer`

Wake-up on motion data.

buffer of audio data

#### 8.47.2.3 `float inv_sensor_event::bias[3]`

x,y,z bias vector data

x,y,z bias vector data (for uncal sensor variant)

#### 8.47.2.4 `int32_t inv_sensor_event::cumulativeEEkcal`

cumulative energy expenditure since the last reset in kilocalorie.

Format is q0: 1 = 1 kcal

#### 8.47.2.5 `int32_t inv_sensor_event::cumulativeEEmets`

cumulative energy expenditure since the last reset in METs (Metabolic Equivalent of Task).

Format is q0: 1 = 1 METs

#### 8.47.2.6 struct { ... } inv\_sensor\_event::eis

EIS data.

#### Warning

experimental: structure is likely to change in near future

#### 8.47.2.7 int inv\_sensor\_event::event

BAC extended data begin/end event as of enum inv\_sensor\_bac\_ext\_event.

BAC data begin/end event as of enum inv\_sensor\_bac\_event.

#### 8.47.2.8 int32\_t inv\_sensor\_event::floorsDown

number of floors climbed Down on foot by user.

#### 8.47.2.9 int32\_t inv\_sensor\_event::floorsUp

number of floors climbed Up on foot by user.

#### 8.47.2.10 struct { ... } inv\_sensor\_event::hrm

heart rate monitor data

heart rate monitor data

#### 8.47.2.11 int32\_t inv\_sensor\_event::instantEEkcal

energy expenditure in kilocalorie/min since last output.

Format is q15:  $2^{15} = 1$  kcal/min

#### 8.47.2.12 int32\_t inv\_sensor\_event::instantEEmets

energy expenditure in METs(Metabolic Equivalent of Task) since last output.

Format is q15:  $2^{15} = 1$  METs



## 8.47.2.13 uint8\_t inv\_sensor\_event::touch\_status

touch status, detected or not by the PPG

touch status, detected or not

The documentation for this struct was generated from the following file:

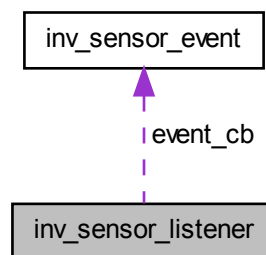
- SensorTypes.h

## 8.48 inv\_sensor\_listener Struct Reference

Sensor event listener definition.

```
#include <SensorTypes.h>
```

Collaboration diagram for inv\_sensor\_listener:



### Public Attributes

- [inv\\_sensor\\_listener\\_event\\_cb\\_t event\\_cb](#)  
*sensor event callback*
- void \* [context](#)  
*listener context*

### 8.48.1 Detailed Description

Sensor event listener definition.

Examples:

[ExampleDeviceIcm20602EMD.c](#), [ExampleDeviceIcm20603EMD.c](#), and [ExampleDeviceIcm20690EMD.c](#).

The documentation for this struct was generated from the following file:

- SensorTypes.h



## Chapter 9

# Example Documentation

### 9.1 ExampleDeviceIcm20602EMD.c

This example shows how to use the Device API for ICM20602 device in embedded context

```
/*
 *
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 * This software, related documentation and any modifications thereto (collectively "Software") is subject
 * to InvenSense and its licensors' intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
 * Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
 * agreement
 * from InvenSense is strictly prohibited.
 *
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
 * PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
 * INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
 * NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THE SOFTWARE.
 */

#include "Invn/Devices/DeviceIcm20602.h"

/*
 * Sleep implementation for Icm20602.
 */
void inv_icm20602_sleep(int ms)
{
    /*
     * You may provide a sleep function that blocks the current programm
     * execution for the specified amount of ms
     */
    (void)ms;
}

/*
 * High resolution sleep implementation for Icm20602.
 * Used at initialization stage. ~100us is sufficient.
 */
void inv_icm20602_sleep_us(int us)
{
    /*
     * You may provide a sleep function that blocks the current programm
     * execution for the specified amount of us
     */
    (void)us;
}

/*
```

```

    * Time implementation for Icm20602.
    */
uint64_t inv_icm20602_get_time_us(void)
{
    /*
     * You may provide a time function that return a monotonic timestamp in us
     */
    return 0;
}

/*
 * Callback called upon sensor event reception
 * This function is called in the same function than inv_device_poll()
 */
static void sensor_event_cb(const inv_sensor_event_t * event, void * arg)
{
    /* arg will contained the value provided at init time */
    (void)arg;

    (void)event;
    /* ... do something with event */
}

/*
 * A listener object will handle sensor events
 */
static inv_sensor_listener_t sensor_listener = {
    sensor_event_cb, /* callback that will receive sensor events */
    (void *)0xDEAD /* some pointer passed to the callback */
};

/*
 * States for icm20602 device object
 */
static inv_device_icm20602_t device_icm20602;

/*
 * serif_hal object that abstract low level serial interface between host and device
 */
extern int my_serif_read_reg(void * context, uint8_t reg, uint8_t * data, uint32_t len);
extern int my_serif_write_reg(void * context, uint8_t reg, const uint8_t * data, uint32_t len);

const inv_serif_hal_t serif_instance = {
    my_serif_read_reg, /* user read_register() function that reads a register over the serial
                       interface */
    my_serif_write_reg, /* user write_register() function that writes a register over the serial
                       interface */
    128, /* maximum number of bytes allowed per read transaction */
    128, /* maximum number of bytes allowed per write transaction */
    INV_SERIF_HAL_TYPE_SPI, /* type of the serial interface used between SPI or I2C */
    (void *)0xDEADBEEF /* some context pointer passed to read/write callbacks */
};

/*
 * Example main function
 */
int example(void);
int example(void)
{
    int rc = 0;

    inv_device_t * device; /* just a handy variable to keep the handle to device object */
    uint8_t whoami;

    /*
     * Open serial interface (SPI or I2C) before playing with the device
     */
    /* call low level drive initialization here... */

    /*
     * Create ICM20602 Device
     * Pass to the driver:
     * - reference to serial interface object,
     * - reference to listener that will catch sensor events,
     */
    inv_device_icm20602_init2(&device_icm20602, &serif_instance, &
sensor_listener);

    /*
     * Simply get generic device handle from Icm20602 Device
     */
    device = inv_device_icm20602_get_base(&device_icm20602);

    /*
     * Just get the whoami

```

```

    */
    rc += inv_device_whoami(device, &whoami);
    /* ... do something with whoami */

    /*
     * Configure and initialize the Icm20602 device
     */
    rc += inv_device_setup(device);

    /*
     * Now that device is ready, you must call inv_device_poll() function
     * periodically or upon interrupt.
     * The poll function will check for sensor events, and notify, if any,
     * by means of the callback from the listener that was provided on device init.
     */
    rc += inv_device_poll(device);

    /*
     * Check if Accelerometer is available sensor
     * if rc value is 0, it means sensor is available,
     * if rc value is INV_ERROR or INV_ERROR_BAD_ARG, sensor is NA
     */
    rc += inv_device_ping_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_ping_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /* ... */

    /*
     * Start RAW accelerometer and gyroscope sensor at 20 Hz
     */
    rc += inv_device_set_sensor_period(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER, 50);
    rc += inv_device_start_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_set_sensor_period(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE, 50);
    rc += inv_device_start_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /* ... */

    /*
     * Stop accelerometer sensor
     */
    rc += inv_device_stop_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_stop_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /*
     * Shutdown everything.
     */
    rc += inv_device_cleanup(device);

    /*
     * Close serial interface link
     */
    /* call low level drive de-initialization here... */

    return rc;
}

```

## 9.2 ExampleDeviceIcm20603EMD.c

This example shows how to use the Device API for ICM20603 device in embedded context

```

/*
 * _____
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 * This software, related documentation and any modifications thereto (collectively "Software") is subject
 * to InvenSense and its licensors' intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
 * Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
 * agreement

```

```

* from InvenSense is strictly prohibited.
*
* EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
* PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
* EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
* INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
* DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
* NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
* OF THE SOFTWARE.
*/


---


#include "Invn/Devices/DeviceIcm20603.h"

/*
 * Sleep implementation for Icm20603.
 */
void inv_icm20603_sleep(int ms)
{
    /*
     * You may provide a sleep function that blocks the current programm
     * execution for the specified amount of ms
     */
    (void)ms;
}

/*
 * High resolution sleep implementation for Icm20603.
 * Used at initialization stage. ~100us is sufficient.
 */
void inv_icm20603_sleep_us(int us)
{
    /*
     * You may provide a sleep function that blocks the current programm
     * execution for the specified amount of us
     */
    (void)us;
}

/*
 * Time implementation for Icm20603.
 */
uint64_t inv_icm20603_get_time_us(void)
{
    /*
     * You may provide a time function that return a monotonic timestamp in us
     */
    return 0;
}

/*
 * Callback called upon sensor event reception
 * This function is called in the same function than inv_device_poll()
 */
static void sensor_event_cb(const inv_sensor_event_t * event, void * arg)
{
    /* arg will contained the value provided at init time */
    (void)arg;

    (void)event;
    /* ... do something with event */
}

/*
 * A listener object will handle sensor events
 */
static inv_sensor_listener_t sensor_listener = {
    sensor_event_cb, /* callback that will receive sensor events */
    (void *)0xDEAD /* some pointer passed to the callback */
};

/*
 * States for icm20603 device object
 */
static inv_device_icm20603_t device_icm20603;

/*
 * serif_hal object that abstract low level serial interface between host and device
 */
extern int my_serif_read_reg(void * context, uint8_t reg, uint8_t * data, uint32_t len);
extern int my_serif_write_reg(void * context, uint8_t reg, const uint8_t * data, uint32_t len);

const inv_serif_hal_t serif_instance = {
    my_serif_read_reg, /* user read_register() function that reads a register over the serial

```

```

    interface */
    my_serif_write_reg,      /* user write_register() function that writes a register over the serial
    interface */
    128,                     /* maximum number of bytes allowed per read transaction */
    128,                     /* maximum number of bytes allowed per write transaction */
    INV_SERIF_HAL_TYPE_SPI, /* type of the serial interface used between SPI or I2C */
    (void *)0xDEADBEEF      /* some context pointer passed to read/write callbacks */
};

/*
 * Example main function
 */
int example(void);
int example(void)
{
    int rc = 0;

    inv_device_t * device; /* just a handy variable to keep the handle to device object */
    uint8_t whoami;

    /*
     * Open serial interface (SPI or I2C) before playing with the device
     */
    /* call low level drive initialization here... */

    /*
     * Create ICM20603 Device
     * Pass to the driver:
     * - reference to serial interface object,
     * - reference to listener that will catch sensor events,
     */
    inv_device_icm20603_init2(&device_icm20603, &serif_instance, &
    sensor_listener);

    /*
     * Simply get generic device handle from Icm20603 Device
     */
    device = inv_device_icm20603_get_base(&device_icm20603);

    /*
     * Just get the whoami
     */
    rc += inv_device_whoami(device, &whoami);
    /* ... do something with whoami */

    /*
     * Configure and initialize the Icm20603 device
     */
    rc += inv_device_setup(device);

    /*
     * Now that device is ready, you must call inv_device_poll() function
     * periodically or upon interrupt.
     * The poll function will check for sensor events, and notify, if any,
     * by means of the callback from the listener that was provided on device init.
     */
    rc += inv_device_poll(device);

    /*
     * Check if Accelerometer is available sensor
     * if rc value is 0, it means sensor is available,
     * if rc value is INV_ERROR or INV_ERROR_BAD_ARG, sensor is NA
     */
    rc += inv_device_ping_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_ping_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /* ... */

    /*
     * Start RAW accelerometer and gyroscope sensor at 20 Hz
     */
    rc += inv_device_set_sensor_period(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER, 50);
    rc += inv_device_start_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_set_sensor_period(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE, 50);
    rc += inv_device_start_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /* ... */

    /*
     * Stop accelerometer sensor

```

```

    */
    rc += inv_device_stop_sensor(device,
INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_stop_sensor(device,
INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /*
    * Shutdown everything.
    */
    rc += inv_device_cleanup(device);

    /*
    * Close serial interface link
    */
    /* call low level drive de-initialization here... */

    return rc;
}

```

### 9.3 ExampleDeviceIcm20690EMD.c

This example shows how to use the Device API for ICM20690 device in embedded context

```

/*
 * _____
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 * This software, related documentation and any modifications thereto (collectively "Software") is subject
 * to InvenSense and its licensors' intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
 * Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
 * agreement
 * from InvenSense is strictly prohibited.
 *
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
 * PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
 * INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
 * NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
 * OF THE SOFTWARE.
 * _____
 */

#include "Invn/Devices/DeviceIcm20690.h"

/*
 * Sleep implementation for Icm20690.
 */
void inv_icm20690_sleep(int ms)
{
    /*
    * You may provide a sleep function that blocks the current program
    * execution for the specified amount of ms
    */
    (void)ms;
}

/*
 * High resolution sleep implementation for Icm20690.
 * Used at initialization stage. ~100us is sufficient.
 */
void inv_icm20690_sleep_us(int us)
{
    /*
    * You may provide a sleep function that blocks the current program
    * execution for the specified amount of us
    */
    (void)us;
}

/*
 * Time implementation for Icm20690.
 */
uint64_t inv_icm20690_get_time_us(void)
{
    /*

```



```

        * You may provide a time function that return a monotonic timestamp in us
        */
        return 0;
    }

/*
 * Callback called upon sensor event reception
 * This function is called in the same function than inv_device_poll()
 */
static void sensor_event_cb(const inv_sensor_event_t * event, void * arg)
{
    /* arg will contained the value provided at init time */
    (void)arg;

    (void)event;
    /* ... do something with event */
}

/*
 * A listener onject will handle sensor events
 */
static inv_sensor_listener_t sensor_listener = {
    sensor_event_cb, /* callback that will receive sensor events */
    (void *)0xDEAD /* some pointer passed to the callback */
};

/*
 * States for icm20690 device object
 */
static inv_device_icm20690_t device_icm20690;

/*
 * serif_hal object that abstract low level serial interface between host and device
 */
extern int my_serif_read_reg(void * context, uint8_t reg, uint8_t * data, uint32_t len);
extern int my_serif_write_reg(void * context, uint8_t reg, const uint8_t * data, uint32_t len);

const inv_serif_hal_t serif_instance = {
    my_serif_read_reg, /* user read_register() function that reads a register over the serial
    interface */
    my_serif_write_reg, /* user write_register() function that writes a register over the serial
    interface */
    128, /* maximum number of bytes allowed per read transaction */
    128, /* maximum number of bytes allowed per write transaction */
    INV_SERIF_HAL_TYPE_SPI, /* type of the serial interface used between SPI or I2C */
    (void *)0xDEADBEEF /* some context pointer passed to read/write callbacks */
};

/*
 * Example main function
 */
int example(void);
int example(void)
{
    int rc = 0;

    inv_device_t * device; /* just a handy variable to keep the handle to device object */
    uint8_t whoami;

    /*
     * Open serial interface (SPI or I2C) before playing with the device
     */
    /* call low level drive initialization here... */

    /*
     * Create ICM20690 Device
     * Pass to the driver:
     * - reference to serial interface object,
     * - reference to listener that will catch sensor events,
     */
    inv_device_icm20690_init2(&device_icm20690, &serif_instance, &
    sensor_listener);

    /*
     * Simply get generic device handle from Icm20690 Device
     */
    device = inv_device_icm20690_get_base(&device_icm20690);

    /*
     * Just get the whoami
     */
    rc += inv_device_whoami(device, &whoami);
    /* ... do something with whoami */

    /*

```

```

    * Configure and initialize the Icm20690 device
    */
    rc += inv_device_setup(device);

    /*
    * Now that device is ready, you must call inv_device_poll() function
    * periodically or upon interrupt.
    * The poll function will check for sensor events, and notify, if any,
    * by means of the callback from the listener that was provided on device init.
    */
    rc += inv_device_poll(device);

    /*
    * Check if Accelerometer is available sensor
    * if rc value is 0, it means sensor is available,
    * if rc value is INV_ERROR or INV_ERROR_BAD_ARG, sensor is NA
    */
    rc += inv_device_ping_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_ping_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /* ... */

    /*
    * Start RAW accelerometer and gyroscope sensor at 20 Hz
    */
    rc += inv_device_set_sensor_period(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER, 50);
    rc += inv_device_start_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_set_sensor_period(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE, 50);
    rc += inv_device_start_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /* ... */

    /*
    * Stop accelerometer sensor
    */
    rc += inv_device_stop_sensor(device,
    INV_SENSOR_TYPE_RAW_ACCELEROMETER);
    rc += inv_device_stop_sensor(device,
    INV_SENSOR_TYPE_RAW_GYROSCOPE);

    /*
    * Shutdown everything.
    */
    rc += inv_device_cleanup(device);

    /*
    * Close serial interface link
    */
    /* call low level drive de-initialization here... */

    return rc;
}

```

## 9.4 ExampleSerifHal.c

Basic template for SerifHal implementation.

```

/*
 * _____
 * Copyright (c) 2015-2015 InvenSense Inc. All rights reserved.
 *
 * This software, related documentation and any modifications thereto (collectively "Software") is subject
 * to InvenSense and its licensors' intellectual property rights under U.S. and international copyright
 * and other intellectual property rights laws.
 *
 * InvenSense and its licensors retain all intellectual property and proprietary rights in and to the
 * Software
 * and any use, reproduction, disclosure or distribution of the Software without an express license
 * agreement
 * from InvenSense is strictly prohibited.
 *
 * EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, THE SOFTWARE IS
 * PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED

```

```

* TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
* EXCEPT AS OTHERWISE PROVIDED IN A LICENSE AGREEMENT BETWEEN THE PARTIES, IN NO EVENT SHALL
* INVENSENSE BE LIABLE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
* DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
* NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
* OF THE SOFTWARE.
*/


---


#include "Invn/Devices/SerifHal.h"

// include to low level system driver
// #include "MyTarget/SPI.h"

// forward declarations
int my_serif_open_read_reg(void * context, uint8_t reg, uint8_t * rbuffer, uint32_t rlen);
int my_serif_open_write_reg(void * context, uint8_t reg, const uint8_t * wbuffer, uint32_t wlen);

// Exported instance of SerifHal object:
// A pointer to this structure shall be passed to the Device API,
// for the device driver to access to the SPI/I2C bus.
// The device will not modify the object, so it can be declared const
// The underlying HW serial interface must be up and running before calling any
// device methods
const inv_serif_hal_t my_serif_instance = {
    my_serif_open_read_reg, /* callback to read_reg low level method */
    my_serif_open_write_reg, /* callback to read_reg low level method */
    128, /* maximum number of bytes allowed per read transaction,
          (limitation can come from internal buffer in the system driver) */
    128, /* maximum number of bytes allowed per write transaction,
          (limitation can come from internal buffer in the system driver) */
    INV_SERIF_HAL_TYPE_SPI, /* type of the serial interface (between SPI or I2C) */
    (void *)0xDEAD /* some context pointer passed to read_reg/write_reg callbacks */
};

int my_serif_open_read_reg(void * context, uint8_t reg, uint8_t * rbuffer, uint32_t rlen)
{
    (void)context, (void)reg, (void)rbuffer, (void)rlen;
    // MyTarget_SPI_do_read_reg(&reg, 1, rbuffer, rlen);
    return 0; // shall return a negative value on error
}

int my_serif_open_write_reg(void * context, uint8_t reg, const uint8_t * wbuffer, uint32_t wlen)
{
    (void)context, (void)reg, (void)wbuffer, (void)wlen;
    // MyTarget_SPI_do_write_reg(&reg, 1, wbuffer, wlen);
    return 0; // shall return a negative value on error
}

```



# Index

- accuracy\_flag
  - inv\_sensor\_event, [191](#)
- audio\_buffer
  - inv\_sensor\_event, [191](#)
- bias
  - inv\_sensor\_event, [191](#)
- COMPASS\_I2C\_SLV\_READ
  - lcm20602 secondary driver transport, [76](#)
  - lcm20603 secondary driver transport, [106](#)
  - lcm20690 secondary driver transport, [136](#)
- close
  - inv\_host\_serif, [170](#)
- cumulativeEEkcal
  - inv\_sensor\_event, [191](#)
- cumulativeEEemets
  - inv\_sensor\_event, [191](#)
- Data Converter, [61](#)
  - inv\_dc\_float\_to\_sf32, [61](#)
  - inv\_dc\_sf32\_to\_float, [61](#)
- Device, [27](#)
  - inv\_device\_cleanup, [29](#)
  - inv\_device\_enable, [29](#)
  - inv\_device\_enable\_sensor, [30](#)
  - inv\_device\_flush\_sensor, [30](#)
  - inv\_device\_get\_fw\_info, [31](#)
  - inv\_device\_get\_sensor\_bias, [31](#)
  - inv\_device\_get\_sensor\_config, [32](#)
  - inv\_device\_get\_sensor\_data, [32](#)
  - inv\_device\_load, [33](#)
  - inv\_device\_ping\_sensor, [33](#)
  - inv\_device\_poll, [34](#)
  - inv\_device\_read\_mems\_register, [34](#)
  - inv\_device\_reset, [35](#)
  - inv\_device\_self\_test, [35](#)
  - inv\_device\_set\_running\_state, [36](#)
  - inv\_device\_set\_sensor\_bias, [36](#)
  - inv\_device\_set\_sensor\_config, [37](#)
  - inv\_device\_set\_sensor\_mounting\_matrix, [38](#)
  - inv\_device\_set\_sensor\_period, [38](#)
  - inv\_device\_set\_sensor\_period\_us, [39](#)
  - inv\_device\_set\_sensor\_timeout, [39](#)
  - inv\_device\_set\_sensor\_timeout\_us, [40](#)
  - inv\_device\_setup, [41](#)
  - inv\_device\_start\_sensor, [41](#)
  - inv\_device\_stop\_sensor, [42](#)
  - inv\_device\_whoami, [42](#)
  - inv\_device\_write\_mems\_register, [43](#)
- DeviceEmdWrapper, [49](#)
- DeviceIcm20602, [50](#)
  - inv\_device\_icm20602\_init, [51](#)
  - inv\_device\_icm20602\_init2, [51](#)
  - inv\_device\_icm20602\_init\_aux\_compass, [51](#)
  - inv\_device\_icm20602\_init\_serif\_ois, [52](#)
  - inv\_device\_icm20602\_init\_serif\_ois2, [52](#)
- DeviceIcm20603, [53](#)
  - inv\_device\_icm20603\_init, [54](#)
  - inv\_device\_icm20603\_init2, [54](#)
  - inv\_device\_icm20603\_init\_aux\_compass, [54](#)
  - inv\_device\_icm20603\_init\_serif\_ois, [55](#)
  - inv\_device\_icm20603\_init\_serif\_ois2, [55](#)
- DeviceIcm20690, [56](#)
  - inv\_device\_icm20690\_init, [57](#)
  - inv\_device\_icm20690\_init2, [57](#)
  - inv\_device\_icm20690\_init\_aux\_compass, [57](#)
  - inv\_device\_icm20690\_init\_serif\_ois, [58](#)
  - inv\_device\_icm20690\_init\_serif\_ois2, [58](#)
- DeviceSmartMotion, [44](#)
  - INV\_SMARTMOTION\_CONFIG\_SENSITIVITY, [47](#)
  - inv\_device\_smart\_motion\_init, [47](#)
  - inv\_device\_smart\_motion\_init2, [47](#)
  - inv\_device\_smart\_motion\_setup, [48](#)
  - inv\_smartmotion\_config\_setting, [47](#)
- I, [48](#)
- Drivers, [158](#)
- eis
  - inv\_sensor\_event, [191](#)
- Error code, [17](#)
  - INV\_ERROR\_BAD\_ARG, [17](#)
  - INV\_ERROR\_FILE, [17](#)
  - INV\_ERROR\_HW, [17](#)
  - INV\_ERROR\_IMAGE\_TYPE, [17](#)
  - INV\_ERROR\_IO, [17](#)
  - INV\_ERROR\_MEM, [17](#)
  - INV\_ERROR\_NIMPL, [17](#)
  - INV\_ERROR\_OS, [17](#)
  - INV\_ERROR\_PATH, [17](#)
  - INV\_ERROR\_SIZE, [17](#)
  - INV\_ERROR\_SUCCESS, [17](#)
  - INV\_ERROR\_TIMEOUT, [17](#)
  - INV\_ERROR\_TRANSPORT, [17](#)
  - INV\_ERROR\_UNEXPECTED, [17](#)
  - INV\_ERROR\_WATCHDOG, [17](#)
  - INV\_ERROR, [17](#)
  - inv\_error, [17](#)
- Error Helper, [63](#)
  - inv\_error\_str, [63](#)

- event
  - inv\_sensor\_event, [192](#)
- floorsDown
  - inv\_sensor\_event, [192](#)
- floorsUp
  - inv\_sensor\_event, [192](#)
- Host Serial Interface, [59](#)
- hrm
  - inv\_sensor\_event, [192](#)
- INV\_ERROR\_BAD\_ARG
  - Error code, [17](#)
- INV\_ERROR\_FILE
  - Error code, [17](#)
- INV\_ERROR\_HW
  - Error code, [17](#)
- INV\_ERROR\_IMAGE\_TYPE
  - Error code, [17](#)
- INV\_ERROR\_IO
  - Error code, [17](#)
- INV\_ERROR\_MEM
  - Error code, [17](#)
- INV\_ERROR\_NIMPL
  - Error code, [17](#)
- INV\_ERROR\_OS
  - Error code, [17](#)
- INV\_ERROR\_PATH
  - Error code, [17](#)
- INV\_ERROR\_SIZE
  - Error code, [17](#)
- INV\_ERROR\_SUCCESS
  - Error code, [17](#)
- INV\_ERROR\_TIMEOUT
  - Error code, [17](#)
- INV\_ERROR\_TRANSPORT
  - Error code, [17](#)
- INV\_ERROR\_UNEXPECTED
  - Error code, [17](#)
- INV\_ERROR\_WATCHDOG
  - Error code, [17](#)
- INV\_ERROR
  - Error code, [17](#)
- INV\_ICM20602\_COMPASS\_ID\_AK09911
  - lcm20602 akm compass support, [72](#)
- INV\_ICM20602\_COMPASS\_ID\_AK09912
  - lcm20602 akm compass support, [72](#)
- INV\_ICM20602\_COMPASS\_ID\_NONE
  - lcm20602 akm compass support, [72](#)
- INV\_ICM20603\_COMPASS\_ID\_AK09911
  - lcm20603 akm compass support, [102](#)
- INV\_ICM20603\_COMPASS\_ID\_AK09912
  - lcm20603 akm compass support, [102](#)
- INV\_ICM20603\_COMPASS\_ID\_NONE
  - lcm20603 akm compass support, [102](#)
- INV\_ICM20690\_COMPASS\_ID\_AK09911
  - lcm20690 akm compass support, [132](#)
- INV\_ICM20690\_COMPASS\_ID\_AK09912
  - lcm20690 akm compass support, [132](#)
- INV\_ICM20690\_COMPASS\_ID\_NONE
  - lcm20690 akm compass support, [132](#)
- INV\_MSG\_LEVEL
  - Message, [65](#)
- INV\_MSG
  - Message, [65](#)
- INV\_SENSOR\_CONFIG\_CONTEXT
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_CUSTOM
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_FSR
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_GAIN
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_MAX
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_OFFSET
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_POWER\_MODE
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_RESERVED
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_CONFIG\_RESET
  - Sensor Configuration, [26](#)
- INV\_SENSOR\_STATUS\_DATA\_UPDATED
  - Sensor types, [20](#)
- INV\_SENSOR\_STATUS\_FLUSH\_COMPLETE
  - Sensor types, [20](#)
- INV\_SENSOR\_STATUS\_POLLED\_DATA
  - Sensor types, [20](#)
- INV\_SENSOR\_STATUS\_STATE\_CHANGED
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_3AXIS
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_ACCELEROMETER
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_AMBIENT\_TEMPERATURE
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_B2S
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_BAC\_EXTENDED
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_BAC\_STATISTICS
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_BAC
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM0
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM1
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM2
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM3
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM4

- Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM5
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM6
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM7
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM\_BSCD
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_CUSTOM\_PRESSURE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_DATA\_ENCRYPTION
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_DISTANCE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_DOUBLE\_TAP
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_EIS
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_ENERGY\_EXPENDITURE
  - Sensor types, [19](#)
- INV\_SENSOR\_TYPE\_ENERGY\_EXPENDITURE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_FLOOR\_CLIMB\_COUNTER
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_FSYNC\_EVENT
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_GAME\_ROTATION\_VECTOR
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_GEOMAG\_ROTATION\_VECTOR↔
  - OR
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_GLANCE\_GESTURE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_GRAVITY
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_GYROMETER
  - Sensor types, [19](#)
- INV\_SENSOR\_TYPE\_GYROSCOPE
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_HEART\_RATE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_HIGH\_RATE\_GYRO
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_HRM\_LOGGER
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_HRV
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_HUMIDITY
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_LIGHT
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_LINEAR\_ACCELERATION
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_MAGNETOMETER
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_MAX
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_META\_DATA
  - Sensor types, [19](#)
- INV\_SENSOR\_TYPE\_MIC
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_OIS
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_ORIENTATION
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_PDR
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_PICK\_UP\_GESTURE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_PREDICTIVE\_QUATERNION
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_PRESSURE
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_PROXIMITY
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_RAW\_ACCELEROMETER
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_RAW\_GYROSCOPE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_RAW\_MAGNETOMETER
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_RAW\_PPG
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_RAW\_TEMPERATURE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_RESERVED
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_ROTATION\_VECTOR
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_SEDENTARY\_REMIND
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_SHAKE
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_SLEEP\_ANALYSIS
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_SMD
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_STEP\_COUNTER
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_STEP\_DETECTOR
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_TEMPERATURE
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_TILT\_DETECTOR
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_TSIMU
  - Sensor types, [21](#)
- INV\_SENSOR\_TYPE\_UNCAL\_GYROMETER
  - Sensor types, [19](#)
- INV\_SENSOR\_TYPE\_UNCAL\_GYROSCOPE
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_UNCAL\_MAGNETOMETER
  - Sensor types, [20](#)
- INV\_SENSOR\_TYPE\_WAKE\_GESTURE
  - Sensor types, [21](#)

- INV\_SENSOR\_TYPE\_WOM
  - Sensor types, 21
- INV\_SMARTMOTION\_CONFIG\_SENSITIVITY
  - DeviceSmartMotion, 47
- Icm20602 akm compass support, 71
  - INV\_ICM20602\_COMPASS\_ID\_AK09911, 72
  - INV\_ICM20602\_COMPASS\_ID\_AK09912, 72
  - INV\_ICM20602\_COMPASS\_ID\_NONE, 72
  - inv\_icm20602\_check\_akm\_self\_test, 72
  - inv\_icm20602\_compass\_getstate, 72
  - inv\_icm20602\_compass\_id, 72
  - inv\_icm20602\_get\_compass\_bytes, 72
  - inv\_icm20602\_get\_compass\_data, 72
  - inv\_icm20602\_read\_akm\_scale, 73
  - inv\_icm20602\_register\_aux\_compass, 73
  - inv\_icm20602\_resume\_akm, 73
  - inv\_icm20602\_setup\_compass\_akm, 73
  - inv\_icm20602\_suspend\_akm, 74
- Icm20602 control, 79
  - inv\_icm20602\_all\_sensors\_off, 80
  - inv\_icm20602\_check\_drdy, 80
  - inv\_icm20602\_check\_wom\_status, 81
  - inv\_icm20602\_configure\_accel\_wom, 81
  - inv\_icm20602\_disable\_fifo, 81
  - inv\_icm20602\_enable\_fifo, 81
  - inv\_icm20602\_enable\_mems, 82
  - inv\_icm20602\_enable\_sensor, 82
  - inv\_icm20602\_get\_int\_status, 82
  - inv\_icm20602\_get\_st\_bias, 82
  - inv\_icm20602\_has\_data\_ready, 83
  - inv\_icm20602\_is\_sensor\_enabled, 83
  - inv\_icm20602\_poll\_delay\_count, 83
  - inv\_icm20602\_poll\_fifo\_data, 83
  - inv\_icm20602\_poll\_fifo\_data\_setup, 84
  - inv\_icm20602\_poll\_ois\_gyro\_data, 84
  - inv\_icm20602\_poll\_sensor\_data\_reg, 84
  - inv\_icm20602\_reset\_fifo, 85
  - inv\_icm20602\_run\_selftest, 85
  - inv\_icm20602\_set\_sensor\_period, 85
  - inv\_icm20602\_set\_slave\_compass\_id, 85
  - inv\_icm20602\_set\_st\_bias, 86
- Icm20602 driver, 68
  - inv\_icm20602\_get\_dataready\_interrupt\_time\_us, 69
  - inv\_icm20602\_get\_time\_us, 69
  - inv\_icm20602\_reset\_states, 69
  - inv\_icm20602\_reset\_states\_serif\_ois, 70
  - inv\_icm20602\_sleep, 70
  - inv\_icm20602\_sleep\_us, 70
- Icm20602 driver serif, 87
- Icm20602 driver setup, 88
  - inv\_icm20602\_accel\_fsr\_2\_reg, 89
  - inv\_icm20602\_get\_accel\_bandwidth, 89
  - inv\_icm20602\_get\_accel\_fullscale, 89
  - inv\_icm20602\_get\_accel\_ois\_fullscale, 90
  - inv\_icm20602\_get\_chip\_base\_sample\_rate, 90
  - inv\_icm20602\_get\_chip\_info, 90
  - inv\_icm20602\_get\_chip\_power\_state, 90
  - inv\_icm20602\_get\_gyro\_bandwidth, 90
  - inv\_icm20602\_get\_gyro\_fullscale, 91
  - inv\_icm20602\_get\_gyro\_ois\_fullscale, 91
  - inv\_icm20602\_get\_whoami, 91
  - inv\_icm20602\_gyro\_fsr\_2\_reg, 91
  - inv\_icm20602\_initialize, 91
  - inv\_icm20602\_is\_advanced\_features\_supported, 92
  - inv\_icm20602\_reg\_2\_accel\_fsr, 92
  - inv\_icm20602\_reg\_2\_gyro\_fsr, 92
  - inv\_icm20602\_set\_accel\_bandwidth, 92
  - inv\_icm20602\_set\_accel\_fullscale, 93
  - inv\_icm20602\_set\_accel\_ois\_fullscale, 93
  - inv\_icm20602\_set\_chip\_power\_state, 93
  - inv\_icm20602\_set\_divider, 93
  - inv\_icm20602\_set\_fsyc\_bit\_location, 94
  - inv\_icm20602\_set\_gyro\_bandwidth, 94
  - inv\_icm20602\_set\_gyro\_fullscale, 94
  - inv\_icm20602\_set\_gyro\_ois\_fullscale, 94
  - inv\_icm20602\_soft\_reset, 95
- Icm20602 driver transport, 96
  - inv\_icm20602\_mems\_read\_reg, 96
  - inv\_icm20602\_mems\_write\_reg, 96
  - inv\_icm20602\_mems\_write\_reg\_one, 97
- Icm20602 secondary driver transport, 75
  - COMPASS\_I2C\_SLV\_READ, 76
  - inv\_icm20602\_execute\_read\_secondary, 76
  - inv\_icm20602\_execute\_write\_secondary, 76
  - inv\_icm20602\_read\_secondary, 76
  - inv\_icm20602\_secondary\_disable\_i2c, 77
  - inv\_icm20602\_secondary\_enable\_i2c, 77
  - inv\_icm20602\_secondary\_stop\_channel, 77
  - inv\_icm20602\_write\_secondary, 78
- Icm20603 akm compass support, 101
  - INV\_ICM20603\_COMPASS\_ID\_AK09911, 102
  - INV\_ICM20603\_COMPASS\_ID\_AK09912, 102
  - INV\_ICM20603\_COMPASS\_ID\_NONE, 102
  - inv\_icm20603\_check\_akm\_self\_test, 102
  - inv\_icm20603\_compass\_getstate, 102
  - inv\_icm20603\_compass\_id, 102
  - inv\_icm20603\_get\_compass\_bytes, 102
  - inv\_icm20603\_get\_compass\_data, 102
  - inv\_icm20603\_read\_akm\_scale, 103
  - inv\_icm20603\_register\_aux\_compass, 103
  - inv\_icm20603\_resume\_akm, 103
  - inv\_icm20603\_setup\_compass\_akm, 103
  - inv\_icm20603\_suspend\_akm, 104
- Icm20603 control, 109
  - inv\_icm20603\_all\_sensors\_off, 110
  - inv\_icm20603\_check\_drdy, 110
  - inv\_icm20603\_check\_wom\_status, 111
  - inv\_icm20603\_configure\_accel\_wom, 111
  - inv\_icm20603\_disable\_fifo, 111
  - inv\_icm20603\_enable\_fifo, 111
  - inv\_icm20603\_enable\_mems, 112
  - inv\_icm20603\_enable\_sensor, 112
  - inv\_icm20603\_get\_int\_status, 112
  - inv\_icm20603\_get\_st\_bias, 112



- inv\_icm20603\_has\_data\_ready, 113
- inv\_icm20603\_is\_sensor\_enabled, 113
- inv\_icm20603\_poll\_delay\_count, 113
- inv\_icm20603\_poll\_fifo\_data, 113
- inv\_icm20603\_poll\_fifo\_data\_setup, 114
- inv\_icm20603\_poll\_ois\_gyro\_data, 114
- inv\_icm20603\_poll\_sensor\_data\_reg, 114
- inv\_icm20603\_reset\_fifo, 115
- inv\_icm20603\_run\_selftest, 115
- inv\_icm20603\_set\_sensor\_period, 115
- inv\_icm20603\_set\_slave\_compass\_id, 115
- inv\_icm20603\_set\_st\_bias, 116
- lcm20603 driver, 98
  - inv\_icm20603\_get\_dataready\_interrupt\_time\_us, 99
  - inv\_icm20603\_get\_time\_us, 99
  - inv\_icm20603\_reset\_states, 99
  - inv\_icm20603\_reset\_states\_serif\_ois, 100
  - inv\_icm20603\_sleep, 100
  - inv\_icm20603\_sleep\_us, 100
- lcm20603 driver serif, 117
- lcm20603 driver setup, 118
  - inv\_icm20603\_accel\_fsr\_2\_reg, 119
  - inv\_icm20603\_get\_accel\_bandwidth, 119
  - inv\_icm20603\_get\_accel\_fullscale, 119
  - inv\_icm20603\_get\_accel\_ois\_fullscale, 120
  - inv\_icm20603\_get\_chip\_base\_sample\_rate, 120
  - inv\_icm20603\_get\_chip\_info, 120
  - inv\_icm20603\_get\_chip\_power\_state, 120
  - inv\_icm20603\_get\_gyro\_bandwidth, 120
  - inv\_icm20603\_get\_gyro\_fullscale, 121
  - inv\_icm20603\_get\_gyro\_ois\_fullscale, 121
  - inv\_icm20603\_get\_whoami, 121
  - inv\_icm20603\_gyro\_fsr\_2\_reg, 121
  - inv\_icm20603\_initialize, 121
  - inv\_icm20603\_is\_advanced\_features\_supported, 122
  - inv\_icm20603\_reg\_2\_accel\_fsr, 122
  - inv\_icm20603\_reg\_2\_gyro\_fsr, 122
  - inv\_icm20603\_set\_accel\_bandwidth, 122
  - inv\_icm20603\_set\_accel\_fullscale, 123
  - inv\_icm20603\_set\_accel\_ois\_fullscale, 123
  - inv\_icm20603\_set\_chip\_power\_state, 123
  - inv\_icm20603\_set\_divider, 123
  - inv\_icm20603\_set\_fsycn\_bit\_location, 124
  - inv\_icm20603\_set\_gyro\_bandwidth, 124
  - inv\_icm20603\_set\_gyro\_fullscale, 124
  - inv\_icm20603\_set\_gyro\_ois\_fullscale, 124
  - inv\_icm20603\_soft\_reset, 125
- lcm20603 driver transport, 126
  - inv\_icm20603\_mems\_read\_reg, 126
  - inv\_icm20603\_mems\_write\_reg, 126
  - inv\_icm20603\_mems\_write\_reg\_one, 127
- lcm20603 secondary driver transport, 105
  - COMPASS\_I2C\_SLV\_READ, 106
  - inv\_icm20603\_execute\_read\_secondary, 106
  - inv\_icm20603\_execute\_write\_secondary, 106
  - inv\_icm20603\_read\_secondary, 106
  - inv\_icm20603\_secondary\_disable\_i2c, 107
  - inv\_icm20603\_secondary\_enable\_i2c, 107
  - inv\_icm20603\_secondary\_stop\_channel, 107
  - inv\_icm20603\_write\_secondary, 108
- lcm20690 akm compass support, 131
  - INV\_ICM20690\_COMPASS\_ID\_AK09911, 132
  - INV\_ICM20690\_COMPASS\_ID\_AK09912, 132
  - INV\_ICM20690\_COMPASS\_ID\_NONE, 132
  - inv\_icm20690\_check\_akm\_self\_test, 132
  - inv\_icm20690\_compass\_getstate, 132
  - inv\_icm20690\_compass\_id, 132
  - inv\_icm20690\_get\_compass\_bytes, 132
  - inv\_icm20690\_get\_compass\_data, 132
  - inv\_icm20690\_read\_akm\_scale, 133
  - inv\_icm20690\_register\_aux\_compass, 133
  - inv\_icm20690\_resume\_akm, 133
  - inv\_icm20690\_setup\_compass\_akm, 133
  - inv\_icm20690\_suspend\_akm, 134
- lcm20690 control, 139
  - inv\_icm20690\_all\_sensors\_off, 140
  - inv\_icm20690\_check\_drdy, 140
  - inv\_icm20690\_check\_wom\_status, 141
  - inv\_icm20690\_configure\_accel\_wom, 141
  - inv\_icm20690\_disable\_fifo, 141
  - inv\_icm20690\_enable\_fifo, 141
  - inv\_icm20690\_enable\_mems, 142
  - inv\_icm20690\_enable\_sensor, 142
  - inv\_icm20690\_get\_int\_status, 142
  - inv\_icm20690\_get\_st\_bias, 142
  - inv\_icm20690\_has\_data\_ready, 143
  - inv\_icm20690\_is\_sensor\_enabled, 143
  - inv\_icm20690\_poll\_delay\_count, 143
  - inv\_icm20690\_poll\_fifo\_data, 143
  - inv\_icm20690\_poll\_fifo\_data\_setup, 144
  - inv\_icm20690\_poll\_ois\_gyro\_data, 144
  - inv\_icm20690\_poll\_sensor\_data\_reg, 144
  - inv\_icm20690\_reset\_fifo, 145
  - inv\_icm20690\_run\_selftest, 145
  - inv\_icm20690\_set\_sensor\_period, 145
  - inv\_icm20690\_set\_slave\_compass\_id, 145
  - inv\_icm20690\_set\_st\_bias, 146
- lcm20690 driver, 128
  - inv\_icm20690\_get\_dataready\_interrupt\_time\_us, 129
  - inv\_icm20690\_get\_time\_us, 129
  - inv\_icm20690\_reset\_states, 129
  - inv\_icm20690\_reset\_states\_serif\_ois, 130
  - inv\_icm20690\_sleep, 130
  - inv\_icm20690\_sleep\_us, 130
- lcm20690 driver serif, 147
- lcm20690 driver setup, 148
  - inv\_icm20690\_accel\_fsr\_2\_reg, 149
  - inv\_icm20690\_get\_accel\_bandwidth, 149
  - inv\_icm20690\_get\_accel\_fullscale, 149
  - inv\_icm20690\_get\_accel\_ois\_fullscale, 150
  - inv\_icm20690\_get\_chip\_base\_sample\_rate, 150
  - inv\_icm20690\_get\_chip\_info, 150
  - inv\_icm20690\_get\_chip\_power\_state, 150

- inv\_icm20690\_get\_gyro\_bandwidth, [150](#)
- inv\_icm20690\_get\_gyro\_fullscale, [151](#)
- inv\_icm20690\_get\_gyro\_ois\_fullscale, [151](#)
- inv\_icm20690\_get\_whoami, [151](#)
- inv\_icm20690\_gyro\_fsr\_2\_reg, [151](#)
- inv\_icm20690\_initialize, [151](#)
- inv\_icm20690\_is\_advanced\_features\_supported, [152](#)
- inv\_icm20690\_reg\_2\_accel\_fsr, [152](#)
- inv\_icm20690\_reg\_2\_gyro\_fsr, [152](#)
- inv\_icm20690\_set\_accel\_bandwidth, [152](#)
- inv\_icm20690\_set\_accel\_fullscale, [153](#)
- inv\_icm20690\_set\_accel\_ois\_fullscale, [153](#)
- inv\_icm20690\_set\_chip\_power\_state, [153](#)
- inv\_icm20690\_set\_divider, [153](#)
- inv\_icm20690\_set\_fsnc\_bit\_location, [154](#)
- inv\_icm20690\_set\_gyro\_bandwidth, [154](#)
- inv\_icm20690\_set\_gyro\_fullscale, [154](#)
- inv\_icm20690\_set\_gyro\_ois\_fullscale, [154](#)
- inv\_icm20690\_soft\_reset, [155](#)
- lcm20690 driver transport, [156](#)
  - inv\_icm20690\_mems\_read\_reg, [156](#)
  - inv\_icm20690\_mems\_write\_reg, [156](#)
  - inv\_icm20690\_mems\_write\_reg\_one, [157](#)
- lcm20690 secondary driver transport, [135](#)
  - COMPASS\_I2C\_SLV\_READ, [136](#)
  - inv\_icm20690\_execute\_read\_secondary, [136](#)
  - inv\_icm20690\_execute\_write\_secondary, [136](#)
  - inv\_icm20690\_read\_secondary, [136](#)
  - inv\_icm20690\_secondary\_disable\_i2c, [137](#)
  - inv\_icm20690\_secondary\_enable\_i2c, [137](#)
  - inv\_icm20690\_secondary\_stop\_channel, [137](#)
  - inv\_icm20690\_write\_secondary, [138](#)
- instantEEkcal
  - inv\_sensor\_event, [192](#)
- instantEEmets
  - inv\_sensor\_event, [192](#)
- inv\_dc\_float\_to\_sfix32
  - Data Converter, [61](#)
- inv\_dc\_sfix32\_to\_float
  - Data Converter, [61](#)
- inv\_device, [161](#)
- inv\_device\_cleanup
  - Device, [29](#)
- inv\_device\_emd\_wrap\_icm20xxx, [162](#)
- inv\_device\_emd\_wrap\_icm20xxx\_serial, [163](#)
- inv\_device\_enable
  - Device, [29](#)
- inv\_device\_enable\_sensor
  - Device, [30](#)
- inv\_device\_flush\_sensor
  - Device, [30](#)
- inv\_device\_get\_fw\_info
  - Device, [31](#)
- inv\_device\_get\_sensor\_bias
  - Device, [31](#)
- inv\_device\_get\_sensor\_config
  - Device, [32](#)
- inv\_device\_get\_sensor\_data
  - Device, [32](#)
- inv\_device\_icm20602, [163](#)
- inv\_device\_icm20602\_config\_bias\_st, [163](#)
- inv\_device\_icm20602\_init
  - DeviceIcm20602, [51](#)
- inv\_device\_icm20602\_init2
  - DeviceIcm20602, [51](#)
- inv\_device\_icm20602\_init\_aux\_compass
  - DeviceIcm20602, [51](#)
- inv\_device\_icm20602\_init\_serif\_ois
  - DeviceIcm20602, [52](#)
- inv\_device\_icm20602\_init\_serif\_ois2
  - DeviceIcm20602, [52](#)
- inv\_device\_icm20603, [164](#)
- inv\_device\_icm20603\_config\_bias\_st, [165](#)
- inv\_device\_icm20603\_init
  - DeviceIcm20603, [54](#)
- inv\_device\_icm20603\_init2
  - DeviceIcm20603, [54](#)
- inv\_device\_icm20603\_init\_aux\_compass
  - DeviceIcm20603, [54](#)
- inv\_device\_icm20603\_init\_serif\_ois
  - DeviceIcm20603, [55](#)
- inv\_device\_icm20603\_init\_serif\_ois2
  - DeviceIcm20603, [55](#)
- inv\_device\_icm20690, [165](#)
- inv\_device\_icm20690\_config\_bias\_st, [166](#)
- inv\_device\_icm20690\_init
  - DeviceIcm20690, [57](#)
- inv\_device\_icm20690\_init2
  - DeviceIcm20690, [57](#)
- inv\_device\_icm20690\_init\_aux\_compass
  - DeviceIcm20690, [57](#)
- inv\_device\_icm20690\_init\_serif\_ois
  - DeviceIcm20690, [58](#)
- inv\_device\_icm20690\_init\_serif\_ois2
  - DeviceIcm20690, [58](#)
- inv\_device\_load
  - Device, [33](#)
- inv\_device\_ping\_sensor
  - Device, [33](#)
- inv\_device\_poll
  - Device, [34](#)
- inv\_device\_read\_mems\_register
  - Device, [34](#)
- inv\_device\_reset
  - Device, [35](#)
- inv\_device\_self\_test
  - Device, [35](#)
- inv\_device\_set\_running\_state
  - Device, [36](#)
- inv\_device\_set\_sensor\_bias
  - Device, [36](#)
- inv\_device\_set\_sensor\_config
  - Device, [37](#)
- inv\_device\_set\_sensor\_mounting\_matrix
  - Device, [38](#)

- inv\_device\_set\_sensor\_period
  - Device, [38](#)
- inv\_device\_set\_sensor\_period\_us
  - Device, [39](#)
- inv\_device\_set\_sensor\_timeout
  - Device, [39](#)
- inv\_device\_set\_sensor\_timeout\_us
  - Device, [40](#)
- inv\_device\_setup
  - Device, [41](#)
- inv\_device\_smart\_motion, [167](#)
- inv\_device\_smart\_motion\_init
  - DeviceSmartMotion, [47](#)
- inv\_device\_smart\_motion\_init2
  - DeviceSmartMotion, [47](#)
- inv\_device\_smart\_motion\_setup
  - DeviceSmartMotion, [48](#)
- inv\_device\_smart\_motion\_vlistener, [168](#)
- inv\_device\_smart\_motion\_vsensor, [168](#)
- inv\_device\_start\_sensor
  - Device, [41](#)
- inv\_device\_stop\_sensor
  - Device, [42](#)
- inv\_device\_vt, [169](#)
- inv\_device\_whoami
  - Device, [42](#)
- inv\_device\_write\_mems\_register
  - Device, [43](#)
- inv\_error
  - Error code, [17](#)
- inv\_error\_str
  - Error Helper, [63](#)
- inv\_fw\_version, [169](#)
- inv\_host\_serif, [170](#)
  - close, [170](#)
  - open, [170](#)
  - read\_reg, [170](#)
  - register\_interrupt\_callback, [171](#)
  - write\_reg, [171](#)
- inv\_icm20602, [171](#)
- inv\_icm20602::inv\_icm20602\_secondary\_states, [173](#)
- inv\_icm20602::inv\_icm20602\_secondary\_states::inv\_icm20602\_secondary\_reg, [173](#)
- inv\_icm20602::inv\_icm20602\_states, [174](#)
- inv\_icm20602\_accel\_fsr\_2\_reg
  - lcm20602 driver setup, [89](#)
- inv\_icm20602\_all\_sensors\_off
  - lcm20602 control, [80](#)
- inv\_icm20602\_check\_akm\_self\_test
  - lcm20602 akm compass support, [72](#)
- inv\_icm20602\_check\_drdy
  - lcm20602 control, [80](#)
- inv\_icm20602\_check\_wom\_status
  - lcm20602 control, [81](#)
- inv\_icm20602\_compass\_getstate
  - lcm20602 akm compass support, [72](#)
- inv\_icm20602\_compass\_id
  - lcm20602 akm compass support, [72](#)
- inv\_icm20602\_configure\_accel\_wom
  - lcm20602 control, [81](#)
- inv\_icm20602\_disable\_fifo
  - lcm20602 control, [81](#)
- inv\_icm20602\_enable\_fifo
  - lcm20602 control, [81](#)
- inv\_icm20602\_enable\_mems
  - lcm20602 control, [82](#)
- inv\_icm20602\_enable\_sensor
  - lcm20602 control, [82](#)
- inv\_icm20602\_execute\_read\_secondary
  - lcm20602 secondary driver transport, [76](#)
- inv\_icm20602\_execute\_write\_secondary
  - lcm20602 secondary driver transport, [76](#)
- inv\_icm20602\_fifo\_states, [172](#)
- inv\_icm20602\_get\_accel\_bandwidth
  - lcm20602 driver setup, [89](#)
- inv\_icm20602\_get\_accel\_fullscale
  - lcm20602 driver setup, [89](#)
- inv\_icm20602\_get\_accel\_ois\_fullscale
  - lcm20602 driver setup, [90](#)
- inv\_icm20602\_get\_chip\_base\_sample\_rate
  - lcm20602 driver setup, [90](#)
- inv\_icm20602\_get\_chip\_info
  - lcm20602 driver setup, [90](#)
- inv\_icm20602\_get\_chip\_power\_state
  - lcm20602 driver setup, [90](#)
- inv\_icm20602\_get\_compass\_bytes
  - lcm20602 akm compass support, [72](#)
- inv\_icm20602\_get\_compass\_data
  - lcm20602 akm compass support, [72](#)
- inv\_icm20602\_get\_dataready\_interrupt\_time\_us
  - lcm20602 driver, [69](#)
- inv\_icm20602\_get\_gyro\_bandwidth
  - lcm20602 driver setup, [90](#)
- inv\_icm20602\_get\_gyro\_fullscale
  - lcm20602 driver setup, [91](#)
- inv\_icm20602\_get\_gyro\_ois\_fullscale
  - lcm20602 driver setup, [91](#)
- inv\_icm20602\_get\_int\_status
  - lcm20602 control, [82](#)
- inv\_icm20602\_get\_st\_bias
  - lcm20602 control, [82](#)
- inv\_icm20602\_get\_time\_us
  - lcm20602 driver, [69](#)
- inv\_icm20602\_get\_whoami
  - lcm20602 driver setup, [91](#)
- inv\_icm20602\_gyro\_fsr\_2\_reg
  - lcm20602 driver setup, [91](#)
- inv\_icm20602\_has\_data\_ready
  - lcm20602 control, [83](#)
- inv\_icm20602\_initialize
  - lcm20602 driver setup, [91](#)
- inv\_icm20602\_is\_advanced\_features\_supported
  - lcm20602 driver setup, [92](#)
- inv\_icm20602\_is\_sensor\_enabled
  - lcm20602 control, [83](#)
- inv\_icm20602\_mems\_read\_reg

- lcm20602 driver transport, 96
- inv\_icm20602\_mems\_write\_reg
  - lcm20602 driver transport, 96
- inv\_icm20602\_mems\_write\_reg\_one
  - lcm20602 driver transport, 97
- inv\_icm20602\_poll\_delay\_count
  - lcm20602 control, 83
- inv\_icm20602\_poll\_fifo\_data
  - lcm20602 control, 83
- inv\_icm20602\_poll\_fifo\_data\_setup
  - lcm20602 control, 84
- inv\_icm20602\_poll\_ois\_gyro\_data
  - lcm20602 control, 84
- inv\_icm20602\_poll\_sensor\_data\_reg
  - lcm20602 control, 84
- inv\_icm20602\_read\_akm\_scale
  - lcm20602 akm compass support, 73
- inv\_icm20602\_read\_secondary
  - lcm20602 secondary driver transport, 76
- inv\_icm20602\_reg\_2\_accel\_fsr
  - lcm20602 driver setup, 92
- inv\_icm20602\_reg\_2\_gyro\_fsr
  - lcm20602 driver setup, 92
- inv\_icm20602\_register\_aux\_compass
  - lcm20602 akm compass support, 73
- inv\_icm20602\_reset\_fifo
  - lcm20602 control, 85
- inv\_icm20602\_reset\_states
  - lcm20602 driver, 69
- inv\_icm20602\_reset\_states\_serif\_ois
  - lcm20602 driver, 70
- inv\_icm20602\_resume\_akm
  - lcm20602 akm compass support, 73
- inv\_icm20602\_run\_selftest
  - lcm20602 control, 85
- inv\_icm20602\_secondary\_disable\_i2c
  - lcm20602 secondary driver transport, 77
- inv\_icm20602\_secondary\_enable\_i2c
  - lcm20602 secondary driver transport, 77
- inv\_icm20602\_secondary\_stop\_channel
  - lcm20602 secondary driver transport, 77
- inv\_icm20602\_serif, 174
- inv\_icm20602\_set\_accel\_bandwidth
  - lcm20602 driver setup, 92
- inv\_icm20602\_set\_accel\_fullscale
  - lcm20602 driver setup, 93
- inv\_icm20602\_set\_accel\_ois\_fullscale
  - lcm20602 driver setup, 93
- inv\_icm20602\_set\_chip\_power\_state
  - lcm20602 driver setup, 93
- inv\_icm20602\_set\_divider
  - lcm20602 driver setup, 93
- inv\_icm20602\_set\_fsync\_bit\_location
  - lcm20602 driver setup, 94
- inv\_icm20602\_set\_gyro\_bandwidth
  - lcm20602 driver setup, 94
- inv\_icm20602\_set\_gyro\_fullscale
  - lcm20602 driver setup, 94
- inv\_icm20602\_set\_gyro\_ois\_fullscale
  - lcm20602 driver setup, 94
- inv\_icm20602\_set\_sensor\_period
  - lcm20602 control, 85
- inv\_icm20602\_set\_slave\_compass\_id
  - lcm20602 control, 85
- inv\_icm20602\_set\_st\_bias
  - lcm20602 control, 86
- inv\_icm20602\_setup\_compass\_akm
  - lcm20602 akm compass support, 73
- inv\_icm20602\_sleep
  - lcm20602 driver, 70
- inv\_icm20602\_sleep\_us
  - lcm20602 driver, 70
- inv\_icm20602\_soft\_reset
  - lcm20602 driver setup, 95
- inv\_icm20602\_suspend\_akm
  - lcm20602 akm compass support, 74
- inv\_icm20602\_write\_secondary
  - lcm20602 secondary driver transport, 78
- inv\_icm20603, 175
- inv\_icm20603::inv\_icm20603\_secondary\_states, 176
- inv\_icm20603::inv\_icm20603\_secondary\_states::inv\_↔
  - icm20603\_secondary\_reg, 176
- inv\_icm20603::inv\_icm20603\_states, 177
- inv\_icm20603\_accel\_fsr\_2\_reg
  - lcm20603 driver setup, 119
- inv\_icm20603\_all\_sensors\_off
  - lcm20603 control, 110
- inv\_icm20603\_check\_akm\_self\_test
  - lcm20603 akm compass support, 102
- inv\_icm20603\_check\_drdy
  - lcm20603 control, 110
- inv\_icm20603\_check\_wom\_status
  - lcm20603 control, 111
- inv\_icm20603\_compass\_getstate
  - lcm20603 akm compass support, 102
- inv\_icm20603\_compass\_id
  - lcm20603 akm compass support, 102
- inv\_icm20603\_configure\_accel\_wom
  - lcm20603 control, 111
- inv\_icm20603\_disable\_fifo
  - lcm20603 control, 111
- inv\_icm20603\_enable\_fifo
  - lcm20603 control, 111
- inv\_icm20603\_enable\_mems
  - lcm20603 control, 112
- inv\_icm20603\_enable\_sensor
  - lcm20603 control, 112
- inv\_icm20603\_execute\_read\_secondary
  - lcm20603 secondary driver transport, 106
- inv\_icm20603\_execute\_write\_secondary
  - lcm20603 secondary driver transport, 106
- inv\_icm20603\_fifo\_states, 175
- inv\_icm20603\_get\_accel\_bandwidth
  - lcm20603 driver setup, 119
- inv\_icm20603\_get\_accel\_fullscale
  - lcm20603 driver setup, 119

- inv\_icm20603\_get\_accel\_ois\_fullscale
  - lcm20603 driver setup, [120](#)
- inv\_icm20603\_get\_chip\_base\_sample\_rate
  - lcm20603 driver setup, [120](#)
- inv\_icm20603\_get\_chip\_info
  - lcm20603 driver setup, [120](#)
- inv\_icm20603\_get\_chip\_power\_state
  - lcm20603 driver setup, [120](#)
- inv\_icm20603\_get\_compass\_bytes
  - lcm20603 akm compass support, [102](#)
- inv\_icm20603\_get\_compass\_data
  - lcm20603 akm compass support, [102](#)
- inv\_icm20603\_get\_dataready\_interrupt\_time\_us
  - lcm20603 driver, [99](#)
- inv\_icm20603\_get\_gyro\_bandwidth
  - lcm20603 driver setup, [120](#)
- inv\_icm20603\_get\_gyro\_fullscale
  - lcm20603 driver setup, [121](#)
- inv\_icm20603\_get\_gyro\_ois\_fullscale
  - lcm20603 driver setup, [121](#)
- inv\_icm20603\_get\_int\_status
  - lcm20603 control, [112](#)
- inv\_icm20603\_get\_st\_bias
  - lcm20603 control, [112](#)
- inv\_icm20603\_get\_time\_us
  - lcm20603 driver, [99](#)
- inv\_icm20603\_get\_whoami
  - lcm20603 driver setup, [121](#)
- inv\_icm20603\_gyro\_fsr\_2\_reg
  - lcm20603 driver setup, [121](#)
- inv\_icm20603\_has\_data\_ready
  - lcm20603 control, [113](#)
- inv\_icm20603\_initialize
  - lcm20603 driver setup, [121](#)
- inv\_icm20603\_is\_advanced\_features\_supported
  - lcm20603 driver setup, [122](#)
- inv\_icm20603\_is\_sensor\_enabled
  - lcm20603 control, [113](#)
- inv\_icm20603\_mems\_read\_reg
  - lcm20603 driver transport, [126](#)
- inv\_icm20603\_mems\_write\_reg
  - lcm20603 driver transport, [126](#)
- inv\_icm20603\_mems\_write\_reg\_one
  - lcm20603 driver transport, [127](#)
- inv\_icm20603\_poll\_delay\_count
  - lcm20603 control, [113](#)
- inv\_icm20603\_poll\_fifo\_data
  - lcm20603 control, [113](#)
- inv\_icm20603\_poll\_fifo\_data\_setup
  - lcm20603 control, [114](#)
- inv\_icm20603\_poll\_ois\_gyro\_data
  - lcm20603 control, [114](#)
- inv\_icm20603\_poll\_sensor\_data\_reg
  - lcm20603 control, [114](#)
- inv\_icm20603\_read\_akm\_scale
  - lcm20603 akm compass support, [103](#)
- inv\_icm20603\_read\_secondary
  - lcm20603 secondary driver transport, [106](#)
- inv\_icm20603\_reg\_2\_accel\_fsr
  - lcm20603 driver setup, [122](#)
- inv\_icm20603\_reg\_2\_gyro\_fsr
  - lcm20603 driver setup, [122](#)
- inv\_icm20603\_register\_aux\_compass
  - lcm20603 akm compass support, [103](#)
- inv\_icm20603\_reset\_fifo
  - lcm20603 control, [115](#)
- inv\_icm20603\_reset\_states
  - lcm20603 driver, [99](#)
- inv\_icm20603\_reset\_states\_serif\_ois
  - lcm20603 driver, [100](#)
- inv\_icm20603\_resume\_akm
  - lcm20603 akm compass support, [103](#)
- inv\_icm20603\_run\_selftest
  - lcm20603 control, [115](#)
- inv\_icm20603\_secondary\_disable\_i2c
  - lcm20603 secondary driver transport, [107](#)
- inv\_icm20603\_secondary\_enable\_i2c
  - lcm20603 secondary driver transport, [107](#)
- inv\_icm20603\_secondary\_stop\_channel
  - lcm20603 secondary driver transport, [107](#)
- inv\_icm20603\_serif, [177](#)
- inv\_icm20603\_set\_accel\_bandwidth
  - lcm20603 driver setup, [122](#)
- inv\_icm20603\_set\_accel\_fullscale
  - lcm20603 driver setup, [123](#)
- inv\_icm20603\_set\_accel\_ois\_fullscale
  - lcm20603 driver setup, [123](#)
- inv\_icm20603\_set\_chip\_power\_state
  - lcm20603 driver setup, [123](#)
- inv\_icm20603\_set\_divider
  - lcm20603 driver setup, [123](#)
- inv\_icm20603\_set\_fsync\_bit\_location
  - lcm20603 driver setup, [124](#)
- inv\_icm20603\_set\_gyro\_bandwidth
  - lcm20603 driver setup, [124](#)
- inv\_icm20603\_set\_gyro\_fullscale
  - lcm20603 driver setup, [124](#)
- inv\_icm20603\_set\_gyro\_ois\_fullscale
  - lcm20603 driver setup, [124](#)
- inv\_icm20603\_set\_sensor\_period
  - lcm20603 control, [115](#)
- inv\_icm20603\_set\_slave\_compass\_id
  - lcm20603 control, [115](#)
- inv\_icm20603\_set\_st\_bias
  - lcm20603 control, [116](#)
- inv\_icm20603\_setup\_compass\_akm
  - lcm20603 akm compass support, [103](#)
- inv\_icm20603\_sleep
  - lcm20603 driver, [100](#)
- inv\_icm20603\_sleep\_us
  - lcm20603 driver, [100](#)
- inv\_icm20603\_soft\_reset
  - lcm20603 driver setup, [125](#)
- inv\_icm20603\_suspend\_akm
  - lcm20603 akm compass support, [104](#)
- inv\_icm20603\_write\_secondary



- lcm20603 secondary driver transport, 108
- inv\_icm20690, 178
- inv\_icm20690::inv\_icm20690\_secondary\_states, 179
- inv\_icm20690::inv\_icm20690\_secondary\_states::inv\_icm20690\_secondary\_reg, 179
- inv\_icm20690::inv\_icm20690\_states, 180
- inv\_icm20690\_accel\_fsr\_2\_reg
  - lcm20690 driver setup, 149
- inv\_icm20690\_all\_sensors\_off
  - lcm20690 control, 140
- inv\_icm20690\_check\_akm\_self\_test
  - lcm20690 akm compass support, 132
- inv\_icm20690\_check\_drdy
  - lcm20690 control, 140
- inv\_icm20690\_check\_wom\_status
  - lcm20690 control, 141
- inv\_icm20690\_compass\_getstate
  - lcm20690 akm compass support, 132
- inv\_icm20690\_compass\_id
  - lcm20690 akm compass support, 132
- inv\_icm20690\_configure\_accel\_wom
  - lcm20690 control, 141
- inv\_icm20690\_disable\_fifo
  - lcm20690 control, 141
- inv\_icm20690\_enable\_fifo
  - lcm20690 control, 141
- inv\_icm20690\_enable\_mems
  - lcm20690 control, 142
- inv\_icm20690\_enable\_sensor
  - lcm20690 control, 142
- inv\_icm20690\_execute\_read\_secondary
  - lcm20690 secondary driver transport, 136
- inv\_icm20690\_execute\_write\_secondary
  - lcm20690 secondary driver transport, 136
- inv\_icm20690\_fifo\_states, 178
- inv\_icm20690\_get\_accel\_bandwidth
  - lcm20690 driver setup, 149
- inv\_icm20690\_get\_accel\_fullscale
  - lcm20690 driver setup, 149
- inv\_icm20690\_get\_accel\_ois\_fullscale
  - lcm20690 driver setup, 150
- inv\_icm20690\_get\_chip\_base\_sample\_rate
  - lcm20690 driver setup, 150
- inv\_icm20690\_get\_chip\_info
  - lcm20690 driver setup, 150
- inv\_icm20690\_get\_chip\_power\_state
  - lcm20690 driver setup, 150
- inv\_icm20690\_get\_compass\_bytes
  - lcm20690 akm compass support, 132
- inv\_icm20690\_get\_compass\_data
  - lcm20690 akm compass support, 132
- inv\_icm20690\_get\_dataready\_interrupt\_time\_us
  - lcm20690 driver, 129
- inv\_icm20690\_get\_gyro\_bandwidth
  - lcm20690 driver setup, 150
- inv\_icm20690\_get\_gyro\_fullscale
  - lcm20690 driver setup, 151
- inv\_icm20690\_get\_gyro\_ois\_fullscale
  - lcm20690 driver setup, 151
- inv\_icm20690\_get\_int\_status
  - lcm20690 control, 142
- inv\_icm20690\_get\_st\_bias
  - lcm20690 control, 142
- inv\_icm20690\_get\_time\_us
  - lcm20690 driver, 129
- inv\_icm20690\_get\_whoami
  - lcm20690 driver setup, 151
- inv\_icm20690\_gyro\_fsr\_2\_reg
  - lcm20690 driver setup, 151
- inv\_icm20690\_has\_data\_ready
  - lcm20690 control, 143
- inv\_icm20690\_initialize
  - lcm20690 driver setup, 151
- inv\_icm20690\_is\_advanced\_features\_supported
  - lcm20690 driver setup, 152
- inv\_icm20690\_is\_sensor\_enabled
  - lcm20690 control, 143
- inv\_icm20690\_mems\_read\_reg
  - lcm20690 driver transport, 156
- inv\_icm20690\_mems\_write\_reg
  - lcm20690 driver transport, 156
- inv\_icm20690\_mems\_write\_reg\_one
  - lcm20690 driver transport, 157
- inv\_icm20690\_poll\_delay\_count
  - lcm20690 control, 143
- inv\_icm20690\_poll\_fifo\_data
  - lcm20690 control, 143
- inv\_icm20690\_poll\_fifo\_data\_setup
  - lcm20690 control, 144
- inv\_icm20690\_poll\_ois\_gyro\_data
  - lcm20690 control, 144
- inv\_icm20690\_poll\_sensor\_data\_reg
  - lcm20690 control, 144
- inv\_icm20690\_read\_akm\_scale
  - lcm20690 akm compass support, 133
- inv\_icm20690\_read\_secondary
  - lcm20690 secondary driver transport, 136
- inv\_icm20690\_reg\_2\_accel\_fsr
  - lcm20690 driver setup, 152
- inv\_icm20690\_reg\_2\_gyro\_fsr
  - lcm20690 driver setup, 152
- inv\_icm20690\_register\_aux\_compass
  - lcm20690 akm compass support, 133
- inv\_icm20690\_reset\_fifo
  - lcm20690 control, 145
- inv\_icm20690\_reset\_states
  - lcm20690 driver, 129
- inv\_icm20690\_reset\_states\_serif\_ois
  - lcm20690 driver, 130
- inv\_icm20690\_resume\_akm
  - lcm20690 akm compass support, 133
- inv\_icm20690\_run\_selftest
  - lcm20690 control, 145
- inv\_icm20690\_secondary\_disable\_i2c
  - lcm20690 secondary driver transport, 137
- inv\_icm20690\_secondary\_enable\_i2c

- lcm20690 secondary driver transport, [137](#)
- inv\_icm20690\_secondary\_stop\_channel
  - lcm20690 secondary driver transport, [137](#)
- inv\_icm20690\_serif, [180](#)
- inv\_icm20690\_set\_accel\_bandwidth
  - lcm20690 driver setup, [152](#)
- inv\_icm20690\_set\_accel\_fullscale
  - lcm20690 driver setup, [153](#)
- inv\_icm20690\_set\_accel\_ois\_fullscale
  - lcm20690 driver setup, [153](#)
- inv\_icm20690\_set\_chip\_power\_state
  - lcm20690 driver setup, [153](#)
- inv\_icm20690\_set\_divider
  - lcm20690 driver setup, [153](#)
- inv\_icm20690\_set\_fsyc\_bit\_location
  - lcm20690 driver setup, [154](#)
- inv\_icm20690\_set\_gyro\_bandwidth
  - lcm20690 driver setup, [154](#)
- inv\_icm20690\_set\_gyro\_fullscale
  - lcm20690 driver setup, [154](#)
- inv\_icm20690\_set\_gyro\_ois\_fullscale
  - lcm20690 driver setup, [154](#)
- inv\_icm20690\_set\_sensor\_period
  - lcm20690 control, [145](#)
- inv\_icm20690\_set\_slave\_compass\_id
  - lcm20690 control, [145](#)
- inv\_icm20690\_set\_st\_bias
  - lcm20690 control, [146](#)
- inv\_icm20690\_setup\_compass\_akm
  - lcm20690 akm compass support, [133](#)
- inv\_icm20690\_sleep
  - lcm20690 driver, [130](#)
- inv\_icm20690\_sleep\_us
  - lcm20690 driver, [130](#)
- inv\_icm20690\_soft\_reset
  - lcm20690 driver setup, [155](#)
- inv\_icm20690\_suspend\_akm
  - lcm20690 akm compass support, [134](#)
- inv\_icm20690\_write\_secondary
  - lcm20690 secondary driver transport, [138](#)
- inv\_msg
  - Message, [65](#)
- inv\_msg\_get\_level
  - Message, [65](#)
- inv\_msg\_printer\_default
  - Message, [66](#)
- inv\_msg\_setup
  - Message, [66](#)
- inv\_msg\_setup\_default
  - Message, [66](#)
- inv\_msg\_setup\_level
  - Message, [66](#)
- inv\_sensor\_config
  - Sensor Configuration, [26](#)
- inv\_sensor\_config\_BSCD\_t
  - Sensor Configuration, [24](#)
- inv\_sensor\_config\_BSCD, [181](#)
- inv\_sensor\_config\_bac, [180](#)
- inv\_sensor\_config\_bac\_t
  - Sensor Configuration, [23](#)
- inv\_sensor\_config\_context, [181](#)
- inv\_sensor\_config\_distance, [182](#)
- inv\_sensor\_config\_distance\_t
  - Sensor Configuration, [24](#)
- inv\_sensor\_config\_double\_tap, [182](#)
- inv\_sensor\_config\_double\_tap\_t
  - Sensor Configuration, [24](#)
- inv\_sensor\_config\_energy\_expenditure, [182](#)
- inv\_sensor\_config\_energy\_expenditure\_t
  - Sensor Configuration, [24](#)
- inv\_sensor\_config\_fsr, [183](#)
- inv\_sensor\_config\_gain, [183](#)
- inv\_sensor\_config\_mounting\_mtx, [184](#)
- inv\_sensor\_config\_mounting\_mtx\_t
  - Sensor Configuration, [25](#)
- inv\_sensor\_config\_offset, [184](#)
- inv\_sensor\_config\_offset\_t
  - Sensor Configuration, [25](#)
- inv\_sensor\_config\_powermode, [184](#)
- inv\_sensor\_config\_shake\_wrist, [185](#)
- inv\_sensor\_config\_shake\_wrist\_t
  - Sensor Configuration, [25](#)
- inv\_sensor\_config\_stepc, [185](#)
- inv\_sensor\_config\_stepc\_t
  - Sensor Configuration, [25](#)
- inv\_sensor\_event, [186](#)
  - accuracy\_flag, [191](#)
  - audio\_buffer, [191](#)
  - bias, [191](#)
  - cumulativeEEkcal, [191](#)
  - cumulativeEEemets, [191](#)
  - eis, [191](#)
  - event, [192](#)
  - floorsDown, [192](#)
  - floorsUp, [192](#)
  - hrm, [192](#)
  - instantEEkcal, [192](#)
  - instantEEemets, [192](#)
  - touch\_status, [192](#)
- inv\_sensor\_listener, [193](#)
- inv\_sensor\_listener\_event\_cb\_t
  - Sensor types, [19](#)
- inv\_sensor\_status
  - Sensor types, [20](#)
- inv\_sensor\_type
  - Sensor types, [20](#)
- inv\_smartmotion\_config\_setting
  - DeviceSmartMotion, [47](#)
- I
  - DeviceSmartMotion, [48](#)
- Message, [64](#)
  - INV\_MSG\_LEVEL, [65](#)
  - INV\_MSG, [65](#)
  - inv\_msg, [65](#)
  - inv\_msg\_get\_level, [65](#)

- inv\_msg\_printer\_default, 66
- inv\_msg\_setup, 66
- inv\_msg\_setup\_default, 66
- inv\_msg\_setup\_level, 66
- open
  - inv\_host\_serif, 170
- read\_reg
  - inv\_host\_serif, 170
- register\_interrupt\_callback
  - inv\_host\_serif, 171
- Sensor Configuration, 22
  - INV\_SENSOR\_CONFIG\_CONTEXT, 26
  - INV\_SENSOR\_CONFIG\_CUSTOM, 26
  - INV\_SENSOR\_CONFIG\_FSR, 26
  - INV\_SENSOR\_CONFIG\_GAIN, 26
  - INV\_SENSOR\_CONFIG\_MAX, 26
  - INV\_SENSOR\_CONFIG\_MOUNTING\_MATRIX, 26
  - INV\_SENSOR\_CONFIG\_OFFSET, 26
  - INV\_SENSOR\_CONFIG\_POWER\_MODE, 26
  - INV\_SENSOR\_CONFIG\_RESERVED, 26
  - INV\_SENSOR\_CONFIG\_RESET, 26
  - inv\_sensor\_config, 26
  - inv\_sensor\_config\_BSCD\_t, 24
  - inv\_sensor\_config\_bac\_t, 23
  - inv\_sensor\_config\_distance\_t, 24
  - inv\_sensor\_config\_double\_tap\_t, 24
  - inv\_sensor\_config\_energy\_expenditure\_t, 24
  - inv\_sensor\_config\_mounting\_mtx\_t, 25
  - inv\_sensor\_config\_offset\_t, 25
  - inv\_sensor\_config\_shake\_wrist\_t, 25
  - inv\_sensor\_config\_stepc\_t, 25
- Sensor types, 18
  - INV\_SENSOR\_STATUS\_DATA\_UPDATED, 20
  - INV\_SENSOR\_STATUS\_FLUSH\_COMPLETE, 20
  - INV\_SENSOR\_STATUS\_POLLED\_DATA, 20
  - INV\_SENSOR\_STATUS\_STATE\_CHANGED, 20
  - INV\_SENSOR\_TYPE\_3AXIS, 21
  - INV\_SENSOR\_TYPE\_ACCELEROMETER, 20
  - INV\_SENSOR\_TYPE\_AMBIENT\_TEMPERATURE, 20
  - INV\_SENSOR\_TYPE\_B2S, 21
  - INV\_SENSOR\_TYPE\_BAC\_EXTENDED, 21
  - INV\_SENSOR\_TYPE\_BAC\_STATISTICS, 21
  - INV\_SENSOR\_TYPE\_BAC, 21
  - INV\_SENSOR\_TYPE\_CUSTOM0, 21
  - INV\_SENSOR\_TYPE\_CUSTOM1, 21
  - INV\_SENSOR\_TYPE\_CUSTOM2, 21
  - INV\_SENSOR\_TYPE\_CUSTOM3, 21
  - INV\_SENSOR\_TYPE\_CUSTOM4, 21
  - INV\_SENSOR\_TYPE\_CUSTOM5, 21
  - INV\_SENSOR\_TYPE\_CUSTOM6, 21
  - INV\_SENSOR\_TYPE\_CUSTOM7, 21
  - INV\_SENSOR\_TYPE\_CUSTOM\_BSCD, 21
  - INV\_SENSOR\_TYPE\_CUSTOM\_PRESSURE, 21
  - INV\_SENSOR\_TYPE\_DATA\_ENCRYPTION, 21
  - INV\_SENSOR\_TYPE\_DISTANCE, 21
  - INV\_SENSOR\_TYPE\_DOUBLE\_TAP, 21
  - INV\_SENSOR\_TYPE\_EIS, 21
  - INV\_SENSOR\_TYPE\_ENERGY\_EXPENDITURE, 19
  - INV\_SENSOR\_TYPE\_ENERGY\_EXPENDITURE, 21
  - INV\_SENSOR\_TYPE\_FLOOR\_CLIMB\_COUNT↔ER, 21
  - INV\_SENSOR\_TYPE\_FSYNC\_EVENT, 21
  - INV\_SENSOR\_TYPE\_GAME\_ROTATION\_VECTOR↔TOR, 20
  - INV\_SENSOR\_TYPE\_GEOMAG\_ROTATION↔VECTOR, 21
  - INV\_SENSOR\_TYPE\_GLANCE\_GESTURE, 21
  - INV\_SENSOR\_TYPE\_GRAVITY, 20
  - INV\_SENSOR\_TYPE\_GYROMETER, 19
  - INV\_SENSOR\_TYPE\_GYROSCOPE, 20
  - INV\_SENSOR\_TYPE\_HEART\_RATE, 21
  - INV\_SENSOR\_TYPE\_HIGH\_RATE\_GYRO, 21
  - INV\_SENSOR\_TYPE\_HRM\_LOGGER, 21
  - INV\_SENSOR\_TYPE\_HRV, 21
  - INV\_SENSOR\_TYPE\_HUMIDITY, 20
  - INV\_SENSOR\_TYPE\_LIGHT, 20
  - INV\_SENSOR\_TYPE\_LINEAR\_ACCELERATION, 20
  - INV\_SENSOR\_TYPE\_MAGNETOMETER, 20
  - INV\_SENSOR\_TYPE\_MAX, 21
  - INV\_SENSOR\_TYPE\_META\_DATA, 19
  - INV\_SENSOR\_TYPE\_MIC, 21
  - INV\_SENSOR\_TYPE\_OIS, 21
  - INV\_SENSOR\_TYPE\_ORIENTATION, 20
  - INV\_SENSOR\_TYPE\_PDR, 21
  - INV\_SENSOR\_TYPE\_PICK\_UP\_GESTURE, 21
  - INV\_SENSOR\_TYPE\_PREDICTIVE\_QUATER↔NION, 21
  - INV\_SENSOR\_TYPE\_PRESSURE, 20
  - INV\_SENSOR\_TYPE\_PROXIMITY, 20
  - INV\_SENSOR\_TYPE\_RAW\_ACCELEROMETER, 21
  - INV\_SENSOR\_TYPE\_RAW\_GYROSCOPE, 21
  - INV\_SENSOR\_TYPE\_RAW\_MAGNETOMETER, 21
  - INV\_SENSOR\_TYPE\_RAW\_PPG, 21
  - INV\_SENSOR\_TYPE\_RAW\_TEMPERATURE, 21
  - INV\_SENSOR\_TYPE\_RESERVED, 20
  - INV\_SENSOR\_TYPE\_ROTATION\_VECTOR, 20
  - INV\_SENSOR\_TYPE\_SEDENTARY\_REMIND, 21
  - INV\_SENSOR\_TYPE\_SHAKE, 21
  - INV\_SENSOR\_TYPE\_SLEEP\_ANALYSIS, 21
  - INV\_SENSOR\_TYPE\_SMD, 20
  - INV\_SENSOR\_TYPE\_STEP\_COUNTER, 21
  - INV\_SENSOR\_TYPE\_STEP\_DETECTOR, 21
  - INV\_SENSOR\_TYPE\_TEMPERATURE, 20
  - INV\_SENSOR\_TYPE\_TILT\_DETECTOR, 21
  - INV\_SENSOR\_TYPE\_TSIMU, 21
  - INV\_SENSOR\_TYPE\_UNCAL\_GYROMETER, 19
  - INV\_SENSOR\_TYPE\_UNCAL\_GYROSCOPE, 20



INV\_SENSOR\_TYPE\_UNCAL\_MAGNETOMETER, [20](#)  
INV\_SENSOR\_TYPE\_WAKE\_GESTURE, [21](#)  
INV\_SENSOR\_TYPE\_WOM, [21](#)  
inv\_sensor\_listener\_event\_cb\_t, [19](#)  
inv\_sensor\_status, [20](#)  
inv\_sensor\_type, [20](#)

touch\_status  
    inv\_sensor\_event, [192](#)

Utils, [159](#)

write\_reg  
    inv\_host\_serif, [171](#)