Telematics Gateway
API Reference Manual

API Reference Manual

iG-PRGOT-UM-R3.0-REL2.8_Telematics_Gateway_Library_API_Reference_Manual

22nd July 2025

AUTHOR(s)

Name	Function	Organisation	Date	Signature
Tanushree	Junior Engineer	iWave Systems Technologies Pvt. Ltd.	22-07-2025	

APPROVAL

Name	Function	Organisation	Date	Signature
Mohamed Yasar Arafath	Project Manager	iWave Global	22-07-2025	

Distribution	iWave Global

Contact info

Name	Telephone	E-mail
iWave Global Pvt. Ltd. 7/B, 29th Main, BTM Layout, 2nd stage, Bangalore-560076, India	+91-80-2678-4775 +91-80-2668-3700	

Document Identification

Project Name	Telematics Gateway
Document Name	iG-PRGOT-UM-R3.0-REL2.8_Telematics_Gateway_Library_API_Reference_Manual
Revision No	REV3.0/REL2.8
Status	Release 2.8
Audience	iWave Global

Document Revision History

Document Number		iG-PRGOT-UM-R3.0-REL2.8_Telematics_Gateway_Library_API_Reference_Manual
Release	**Date**	**Description**
Rev1.0/Rel1.0	28th Apr 2023	Initial Version
Rev2.0/Rel2.0	09th June 2023	Draft API Reference User Manual
Rev2.0/Rel2.1	13th Sept 2023	Added notes for get_gps_data API
Rev3.0/Rel2.2	31st Jan 2024	Added CAN written error
Rev3.0/Rel2.3	16th Jan 2024	Added RTC commands for date and time
Rev3.0/Rel2.4	3rd April 2024	Updated RTC commands
Rev3.0/Rel2.5	12th Nov 2024	Updated MCU related API Added ADC1 and ADC2 related API
Rev3.0/Rel2.6	11th Dec 2024	Added automotive ethernet and RS485 API
Rev3.0/Rel2.7	24th March 2025	Updated format
Rev3.0/Rel2.8	22nd July 2025	Added config_rtc_wakeup API and sample application of MCU

TABLE OF CONTENTS

1. INTRODUCTION **9**

1.1 PURPOSE AND SCOPE **9**

1.2 GLOSSARY **9**

2 FUNCTIONAL DESCRIPTION **10**

2.1 GENERAL APIS 10

2.1.1 init 10

2.1.2 get_time 11

2.1.3 check_adc1_voltage 11

2.1.4 check_adc2_voltage 11

2.1.5 led_enable 12

2.1.6 led_disable 12

2.1.7 restart_device 12

2.1.8 i2c_write 13

2.1.9 i2c_read 13

2.1.10 get_mac_address 14

2.1.11 get_cpu_id 14

2.1.12 read_digital_in 14

2.1.13 write_digital_out 15

2.1.14 config_timer_wakeup 15

2.1.15 config_rtc_wakeup 16

2.1.16 push_device_to_sleep 16

2.1.17 ign_pin_status_check_enable 16

2.1.18 ignition_pin_status 17

2.1.19 ign_pin_status_check_disable 17

2.1.20 config_ignition_wakeup 17

2.1.21 config_sms_wakeup 18

2.1.22 version_read 18

2.1.23 deinit 18

2.2 CELLULAR MODULE APIS 19

2.2.1 gsm_modem_on 19

2.2.2 gsm_modem_off 19

2.2.3 get_gsm_imei 19
2.2.4 check_gsm_nw_connection 19
2.2.5 check_gsm_modem_status 20
2.2.6 set_gsm_flight_mode_on 20
2.2.7 set_gsm_flight_mode_off 20
2.2.8 establish_connection 20
2.2.9 get_gsm_sim_status 21
2.2.10 get_gsm_sim_iccid 21
2.2.11 set_gsm_network_mode 21
2.2.12 get_gsm_signal_strength 22
2.2.13 get_gsm_nw_reg 22
2.2.14 gsm_at_cmd 22
2.2.15 gsm_apn_configuration 23
2.2.16 check_network_connection 23
2.2.17 GSM_set_to_message_init 24
2.2.18 network_monitor_disable 24
2.2.19 unread_message 24
2.2.20 read_message 24
2.2.21 send_sms 25
2.2.22 delete_message 25
2.2.23 delete_all_messages 25
2.3 CELLULAR MODULE SPECIFIC RETURN CODES 26
2.4 CAN APIS 27
2.4.1 can_init 27
2.4.2 can_fd_init 27
2.4.3 set_can_mask_and_filter 28
2.4.4 can_write 28
2.4.5 can_read 29
2.4.6 config_can_wakeup 29
2.4.7 can_deinit 29
2.5 CAN SPECIFIC RETURN CODES 30
2.6 ETHERNET APIS 31
2.6.1 eth_init 31
2.6.2 eth_deinit 31
2.7 ETHERNET SPECIFIC RETURN CODES 31

2.8 ACCELEROMETER APIS 32
2.8.1 acc_init 32
2.8.2 acc_deinit 32
2.8.3 accelerometer_read 32
2.8.4 set_acc_low_pass_filter 32
2.8.5 set_acc_sampling_frequency 33
2.8.6 set_acc_wakeup_threshold 35
2.8.7 config_acc_wakeup 36
2.8.8 acc_temp_read 36
2.9 ACCELEROMETER SPECIFIC RETURN CODES 36
2.10 GYROSCOPE APIS 37
2.10.1 gyro_init 37
2.10.2 gyro_deinit 37
2.10.3 gyroscope_read 37
2.10.4 set_gyro_sampling_frequency 38
2.10.5 set_gyro_low_pass_filter 39
2.11 GYROSCOPE SPECIFIC RETURN CODES 39
2.12 GPS APIS 40
2.12.1 gps_init 40
2.12.2 gps_deinit 40
2.12.3 agps_init 41
2.12.4 get_gps_data 42
2.13 GPS SPECIFIC RETURN CODES 43
2.14 BATTERY APIS 44
2.14.1 i_battery_init 44
2.14.2 i_get_battery_status 44
2.14.3 i_battery_get_voltage 45
2.14.4 i_battery_get_health 45
2.14.5 battery_connect_config 45
2.14.6 battery_charge_state_config 46
2.14.7 get_power_source 46
2.14.8 i_get_battery_temp 47
2.15 BATTERY SPECIFIC RETURN CODES 47
2.16 WIFI APIS 48
2.16.1 wifi_init 48
2.16.2 wifi_deinit 48

2.17 WiFi specific return codes 49
2.18 Bluetooth APIs 50
2.18.1 ble_init 50
2.18.2 ble_deinit 50
2.19 Bluetooth specific return codes 50
2.20 RS232 APIs 51
2.20.1 rs232_init 51
2.20.2 rs232_deinit 51
2.20.3 rs232_write 51
2.20.4 rs232_read 52
2.21 RS485 related APIs 52
2.21.1 Rs485_init 52
2.21.2 rs485_deinit 52
2.21.3 rs485_write 53
2.21.4 Rs485_read 53
2.22 MCU related APIs 54
2.22.1 MCU_sleep_mode 54
2.22.2 MCU_Wakeup_Source_Request 55
2.22.3 MCU_FW_Version_Read_Request 55
2.23 MCU Specific Return Codes 56
3 GENERAL RETURN CODES 57
4 STRUCTURES 58
4.1 _accelerometer_api_priv 58
4.2 _gyroscope_api_priv 58
4.3 _magnetometer_api_priv 58
4.4 i2c_cmd_frame_struct 59
4.5 i2c_cmd_decode_struct 59
5 API SEQUENCE 61
6 APPLICATION DEVELOPMENT 83
Headers 83
Library 83
Cross_Compiler 83
8. TECHNICAL SUPPORT 84

List of Figures

FIGURE 1: ACCELEROMETER ODR SETTING 34
FIGURE 2: ACCELEROMETER THRESHOLD SETTINGS 35
FIGURE 3: GYROSCOPE ODR SETTINGS 38

List of Tables

TABLE 1: ACRONYMS & ABBREVIATIONS 9
TABLE 2: HEX VALUES OF SET_ACC_SAMPLING_FREQUENCY 34

1. INTRODUCTION

1.1 Purpose and Scope

The purpose of this document is to describe the API's supported by the Telematics Gateway library for developing an application. Developers can use this document while developing application using Telematics Gateway library.

1.2 Glossary

Acronyms	Abbreviations
API	Application Programming Interface
APN	Access Point Name
CAN	Controller Area Network
GSM	Global System for Mobile Communication
I2C	Inter-Integrated Circuit
IPC	Inter Process Communication
SIM	Subscriber Identity Module

Table 1: Acronyms & Abbreviations

2 Functional Description

This chapter describes APIs supported by the Telematics Gateway library.

NOTE:

- Use single application for all the library calls (libTelematics_GW.so) especially related to Cellular and GNSS Modem.
- In-case of any of the cellular modem USBs are being used by any of the process, kill the process before the modem de-initialization.
- All the below API's return code are with respect to C language. Print the return values in hexadecimal.
- If python is used, then 2's compliment of hexadecimal value must be done to get the proper return value.

Example:

```
ret= init ()
print (hex((ret + (1 << 32)) % (1 << 32))
```

2.1 General APIs

2.1.1 init

init	
Description	This function initializes the IPC mechanism required by Telematics Gateway library.
Syntax	**int init (int network_enable)**
Arguments	network_enable **-** Automatic network monitor feature will be enabled/disabled based on the value passed on to this variable. • If network_enable = 0: Automatic network monitor feature would be disabled. • If network_enable = 1: Automatic network monitor feature will be enabled.
Return	Refer *General Return Codes* section.
Example	ret=init (0);
NOTE	• **If "0" is passed as argument:** o 4G module initialization and connection establishment would have to handled manually using the provided APIs. (Refer *Sequence to be followed if 0 is passed to init ()* for calling sequence). • **If "1" is passed as argument:** o 4G module initialization and connection establishment will be internally without any external intervention. o Please note that when this feature is enabled,

	"network_monitor_disable ()" would have to be called before calling the main deinitialization function (deinit ()). ○ This can be used only if it is needed since there are few constraints like some API cannot be used since it will be used internally by network manager thread. ○ Below are the APIs which has limitations with init (1): ▪ establish_connection ▪ set_gsm_flight_mode_on ▪ set_gsm_flight_mode_off ▪ gsm_modem_on ▪ gsm_modem_off ▪ set_gsm_network_mode

2.1.2 get_time

get_time	
Description	This API reads the date & time from the Telematics Gateway.
Syntax	void get_time (char *buf)
Arguments	char * buf - pointer to character array to store date & time of the Telematics Gateway.
Return	None
Example	char buf [50] = {0}; ret = get_time(&buf);
NOTE	-

2.1.3 check_adc1_voltage

check_adc_voltage	
Description	This API reads the ADC external supply voltage.
Syntax	int check_adc1_voltage (double *volt)
Arguments	double *volt - Voltage value.
Return	Refer *General Return Codes* section.
Example	double volt; ret = check_adc1_voltage(&volt);

2.1.4 check_adc2_voltage

check_adc_voltage	
Description	This API reads the ADC external supply voltage.
Syntax	int check_adc2_voltage (double *volt)
Arguments	double *volt - Voltage value.

Return	Refer *General Return Codes* section.
Example	double volt; ret = check_adc2_voltage(&volt);

2.1.5 led_enable

led_enable	
Description	This API is used to turn on the green LED.
Syntax	int led_enable ();
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = led_enable ();
NOTE	-

2.1.6 led_disable

led_disable	
Description	This API is used to turn off the green LED.
Syntax	int led_disable ();
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = led_disable ();
NOTE	-

2.1.7 restart_device

restart_device	
Description	This API is used to restart the Telematics Gateway.
Syntax	int restart_device ();
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = restart_device ();
NOTE	-

2.1.8 i2c_write

i2c_write	
Description	This API is used to write values to i2c registers.
Syntax	int i2c_write (int bus_num, uint8_t slave_addr, uint8_t data_addr, uint8_t value)
Arguments	• int bus_num - i2c bus number • uint8_t slave_addr - slave address • uint8_t data_addr - data address of register • uint8_t value - value to be written to the data address
Return	Refer *General Return Codes* section.

Example	#define SLAVE_ADDR 0x6a
	#define DATA_ADDR 0x20
	#define VALUE 0x86
	int main () {
	int bus_num =1;
	ret = i2c_write (bus_num, SLAVE_ADDR, DATA_ADDR, VALUE);
	}

NOTE	• The bus number should be valid. • The addresses passed to the API should be in hex format and should be valid.

2.1.9 i2c_read

i2c_read	
Description	This API is used to read data from i2c registers.
Syntax	int i2c_read (int bus_num, uint8_t slave_addr, uint8_t data_addr, uint8_t *value)
Arguments	• int bus_num - i2c bus number • uint8_t slave_addr - slave address • uint8_t data_addr - data address of register • uint8_t *value - this variable gives the value in the data address register
Return	Refer *General Return Codes* section.

Example	#define SLAVE_ADDR 0x6a
	#define DATA_ADDR 0x20
	int main () {
	int bus_num = 1;
	uint8_t resp;
	ret = i2c_read (bus_num, SLAVE_ADDR, DATA_ADDR, &resp);
	}

NOTE	• The bus number should be valid.

	• The addresses passed to the API should be in hex format and should be valid.

2.1.10 get_mac_address

get_mac_address	
Description	This API used to get the MAC address of interfaces.
Syntax	int get_mac_address (char *interface, char *mac_address)
Arguments	• char *interface - interface name. Ex: “wlan0”, “eth0”, “hci0”, etc.
Return	Refer *General Return Codes* section.
Example	char wifi_addr [30]; ret = get_mac_adddress (“wlan0”, wifi_addr);
NOTE	-

2.1.11 get_cpu_id

get_cpu_id	
Description	This API used to get the CPU ID.
Syntax	int get_cpu_id (char *cpuid, int cpuid_len)
Arguments	• char *cpuid - array to store the CPU id. • int cpuid_len - Size/Length of the array to store the CPU ID.
Return	Refer *General Return Codes* section.

Example	char cpu_id [50] = {0};
	int cpuid_length = 40;
	ret = get_cpu_id (cpu_id, cpuid_length);

NOTE	Array size cannot be less than 23.

2.1.12 read_digital_in

read_digital_in	
Description	This API is used to read the Digital Input in Telematics Device.
Syntax	int read_digital_in (int din, int *state)
Arguments	• int din – Name of DIN. 1 – DIN1

	2 – DIN2 • Int *state – To store the status of the Digital Input.
Return	Refer *General Return Codes* section.

Example	int din1 = 1;
	int state.
	ret = read_digital_in (din1, state);

2.1.13 write_digital_out

write_digital_out	
Description	This API is used to write the Digital Output from Telematics Device.
Syntax	int write_digital_out (int dout, int state)
Arguments	• int din – Name of DIN. 1 – DOUT1 2 – DOUT2 • Int state – The status to write to the Digital Output. 0 – OFF 1 – ON
Return	Refer *General Return Codes* section.

Example	int dout1 = 1;
	int state = 1;
	ret = write_digital_out(dout1, state);

2.1.14 config_timer_wakeup

config_timer_wakeup	
Description	This API is used to enable or disable timer wakeup.
Syntax	int config_timer_wakeup (int option, int timer)
Arguments	• int option - option to enable or disable timer wakeup 1 - enable 0 - disable • int timer - timer value in seconds Ex: 10
Return	Refer *General Return Codes* section.
Example	int option = 1; ret = config_timer_wakeup (option, 60);

2.1.15 config_rtc_wakeup

config_timer_wakeup	
Description	This API is used to enable or disable rtc wakeup.
Syntax	int config_rtc_wakeup (int option, int timer)
Arguments	• int option - option to enable or disable timer wakeup 1 - enable 0 - disable • int timer - timer value in seconds Ex: 10
Return	Refer *General Return Codes* section.
Example	int option = 1; ret = config_rtc_wakeup (option, 60);

2.1.16 push_device_to_sleep

push_device_to_sleep	
Description	This API is used to put device to sleep mode.
Syntax	int push_device_to_sleep ()
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = push_device_to_sleep ();
NOTE	• Any wakeup API must be called before calling this API. • If none of the wakeup APIs are called the device need to be power cycled manually.

2.1.17 ign_pin_status_check_enable

ign_pin_status_check_enable	
Description	This API is used to enable the ignition event status thread.
Syntax	**int ign_pin_status_check_enable ()**
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = ign_pin_status_check_enable ();
NOTE	-

2.1.18 ignition_pin_status

ignition_pin_status	
Description	This API is used to check the ignition pin status.
Syntax	int ignition_pin_status ()
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = ignition_pin_status ();
NOTE	-

2.1.19 ign_pin_status_check_disable

ign_pin_status_check_disable	
Description	This API is used to disable the ignition event status thread.
Syntax	**int ign_pin_status_check_disable ()**
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = ign_pin_status_check_disable ();
NOTE	• ign_pin_status_check_enable () should be called before ignition_pin_status (). • **Following are the return code expected:** ○ ign_pin_status returns 1 in the presence of ignition. ○ ign_pin_status returns -1 in the absence of ignition. ○ If ign_pin_status_check_enable () is not called before ignition_pin_status (), status value is always 0. (Even when ignition status is toggled). ○ If ign_pin_status_check_disable () is called before ign_pin_status (), status value is always 0.

2.1.20 config_ignition_wakeup

config_ignition_wakeup	
Description	This API is used to enable or disable ignition wakeup.
Syntax	int config_ignition_wakeup (int option)
Arguments	• int option - option to enable or disable ignition wakeup 1 - enable 0 - disable
Return	Refer *General Return Codes* section.
Example	int option = 1; ret = config_ignition_wakeup (option);

2.1.21 config_sms_wakeup

config_sms_wakeup	
Description	This API is used to enable or disable the SMS wakeup.
Syntax	int config_sms_wakeup (int option)
Arguments	• int option - option to enable or disable SMS wakeup. 1 - enable 0 – disable
Return	Refer *General Return Codes* section.
Example	int option = 1; ret = config_sms_wakeup (option);
Note	▪ Currently supported in EG95-EX Modules only.

2.1.22 version_read

version_read	
Description	This API is used to read the version of library, Software DF and Kernel version.
Syntax	char *version_read ()
Arguments	None
Return	Refer *General Return Codes* section.
Example	char *version = version_read ();

2.1.23 deinit

deinit	
Description	Please note that when this feature is enabled, “network_monitor_disable ()” API should be called before calling the main deinitialization function (deinit ()). This function de-initializes the IPC mechanism required by Telematics library.
Syntax	int deinit ()
Arguments	None
Return	Refer *General Return Codes* section.
Example	ret = deinit ();
NOTE	This API will deinitialize Ethernet and turn off LED.

2.2 Cellular module APIs

2.2.1 gsm_modem_on

gsm_modem_on	
Description	This API is used to turn on the GSM modem.
Syntax	int gsm_modem_on (char *cpin, int length)
Arguments	• char *cpin - CPIN value of the SIM that is being used. If the SIM does not have CPIN value, “0000” can be passed to the same argument. • int length - Size/Length of the CPIN value that is being passed.
Return	Refer *Cellular module specific return codes* section.
Example	ret = gsm_modem_on (“0000”, 4);

2.2.2 gsm_modem_off

gsm_modem_off	
Description	This API is used to turn off the GSM modem.
Syntax	int gsm_modem_off ()
Arguments	None
Return	Refer *Cellular module specific return codes* section.
Example	ret = gsm_modem_off ();

2.2.3 get_gsm_imei

get_gsm_imei	
Description	This API is used to get the IMEI number of the GSM module.
Syntax	int get_gsm_imei (char *imei_no, int length)
Arguments	• char *imei_no - array to store the IMEI number. • int length - Size/Length of the array to store the IMEI number.
Return	Refer *Cellular module specific return codes* section.
Example	char imei [20] = {0}; ret = get_gsm_imei (imei, sizeof (imei));
NOTE	• Array size cannot be less than 16. • gsm_modem_on () should be called before calling this API.

2.2.4 check_gsm_nw_connection

check_gsm_nw_connection	
Description	This function is used to check whether is internet connectivity is active or not.
Syntax	int check_gsm_nw_connection ()
Arguments	None
Return	Refer *Cellular module specific return codes* section.

Example	ret = check_gsm_nw_connection ();
NOTE	gsm_modem_on () should be called before calling this API.

2.2.5 check_gsm_modem_status

check_gsm_modem_status	
Description	This API is used to check whether modem is on or off.
Syntax	int check_gsm_modem_status ()
Arguments	None
Return	0 if modem is ON. Refer *Cellular module specific return codes* section.
Example	ret = check_gsm_modem_status ();
NOTE	gsm_modem_on () should be called before calling this API.

2.2.6 set_gsm_flight_mode_on

set_gsm_flight_mode_on	
Description	This API is used to set the flight mode ON.
Syntax	int set_gsm_flight_mode_on ()
Arguments	None
Return	Refer *Cellular module specific return codes* section.
Example	ret = set_gsm_flight_mode_on ();
NOTE	gsm_modem_on () should be called before calling this API.

2.2.7 set_gsm_flight_mode_off

set_gsm_flight_mode_off	
Description	This API is used to set the flight mode OFF.
Syntax	int set_gsm_flight_mode_off ()
Arguments	None
Return	Refer *Cellular module specific return codes* section.
Example	ret = set_gsm_flight_mode_off ();
NOTE	gsm_modem_on () should be called before calling this API.

2.2.8 establish_connection

establish_connection	
Description	This API establishes ppp network connection.
Syntax	int establish_connection ()

Arguments	None
Return	Refer *Cellular module specific return codes* section.
Example	ret = establish_connection ();
NOTE	gsm_modem_on () should be called before calling this API.

2.2.9 get_gsm_sim_status

get_gsm_sim_status	
Description	This API will check whether the sim is inserted in device or not.
Syntax	int get_gsm_sim_status (int *sim_status_val)
Arguments	sim_status_val- integer pointer which stores the current sim status.
Return	Refer *Cellular module specific return codes* section.
Example	int sim_status_val; ret = get_gsm_sim_status (&sim_status_val);
NOTE	gsm_modem_on () should be called before calling this API.

2.2.10 get_gsm_sim_iccid

get_gsm_sim_iccid	
Description	This API to get the ICCID of the GSM module.
Syntax	int get_gsm_sim_iccid (char *iccid, int length)
Arguments	• char *iccid - array to store the ICCID of the GSM module. • int length - Size/Length of the array to store the ICCID of the GSM module.
Return	Refer *Cellular module specific return codes* section.
Example	char iccid [30] = “”; ret = get_gsm_sim_iccid (iccid, sizeof(iccid);
NOTE	• Array size cannot be less than 25. • gsm_modem_on () should be called before calling this API.

2.2.11 set_gsm_network_mode

set_gsm_network_mode	
Description	This API is used to change the network mode.
Syntax	int set_gsm_network_mode (int type)
Arguments	int type - contains the network mode to be set.
Return	Refer *Cellular module specific return codes* section.
Example	int option = 1; ret = set_gsm_network_mode (int option)

NOTE	• gsm_modem_on () and establish_connection () should be called before calling this API.
	  ○ Auto = 1
	  ○ 2G = 2
	  ○ 3G = 3
	  ○ 4G = 4

2.2.12 get_gsm_signal_strength

get_gsm_signal_strength	
Description	This API to get the signal range details of the network.
Syntax	int get_gsm_signal_strength (char *signal_lvl, int length)
Arguments	• char *signal_lvl - array to store the signal range. • int length - Size/Length of the array to store the signal range.
Return	Refer *Cellular module specific return codes* section.
Example	char arr_sig_strength [20] = {0}; ret = get_gsm_signal_strength (arr_sig_strength, sizeof(arr_sig_strength));
NOTE	• gsm_modem_on () should be called before calling this API. • Make sure that sim is inserted before calling this API.

2.2.13 get_gsm_nw_reg

get_gsm_nw_reg	
Description	This API to get the SIM registration details.
Syntax	int get_gsm_nw_reg (char * cell_id, int cell_id_len, char *lac, int lac_len)
Arguments	• char *cell_id - array to store the cell identification. • int cell_id_len - Size/Length of the array to store the cell identification • char *lac - array to store location area code. • int lac_len - Size/Length of the array to store location area code
Return	Refer *Cellular module specific return codes* section.
Example	char cell_id [20] = {0}; char lac [20] = {0}; ret = get_gsm_nw_reg (cell_id, sizeof(cell_id), lac, sizeof(lac));
NOTE	• gsm_modem_on () should be called before calling this API. • Make sure that sim is inserted before calling this API. • Array size cannot be less than 20.

2.2.14 gsm_at_cmd

gsm_at_cmd	
Description	This API is used to send AT commands and receive the response.
Syntax	int gsm_at_cmd (char *at_cmd, char *resp, int length, int max_resp_time)

Arguments	• char *at_cmd - AT command to be sent. • char *resp - Array to store the AT command response. • int length - Size/Length of array to store AT command response. • int max_resp_time - Response wait time in milliseconds (default max_resp_time can be considered as 300ms (300000), if it is not mentioned in AT command reference manual).
Return	Refer *Cellular module specific return codes* section.
Example	char at_cmd [] = “at+pin?”; char resp [200]; int length; ret = gsm_at_cmd (at_cmd, &resp, length, 500000); //500000 is 500ms
NOTE	• gsm_modem_on () should be called before calling this API. • Example for AT commands: AT+CPIN? , AT+CREG?

2.2.15 gsm_apn_configuration

gsm_apn_configuration	
Description	This API is used to change the APN configurations of the SIM.
Syntax	void gsm_apn_configuration (char *apn_name, char *atd_num, char *username, char *password)
Arguments	• char *apn_name - APN name to be set. Ex: “airtelgprs.com” • char *atd_num - ATD number to be set. Ex: “ATDT*99***1#” • char *username - Username of the SIM. • char *password - Password of SIM.
Return	None
Example	char apn [] = “airtelgprs.com” char atd_num [] = “ATDT*99***1#” char username [] = “xxxxxx” char password [] = “yyyyyy” ret = gsm_apn_configuration (&apn, &atd_num, &username, &password);
NOTE	• gsm_modem_on () should be called before calling this API. • Pass NULL if there is no Username and Password for sim.

2.2.16 check_network_connection

check_network_connection	
Description	This API is used to check the network availability.
Syntax	int check_network_connection ()
Arguments	None
Return	Refer *Cellular module specific return codes* section.
Example	ret = check_network_connection ();
NOTE	-

2.2.17 GSM_set_to_message_init

GSM_set_to_message_init	
Description	This API is used to set the GSM to text message mode.
Syntax	int GSM_set_to_message_init ()
Arguments	None
Return	Refer *Cellular module specific return codes* section.
Example	ret = GSM_set_to_message_init ();
NOTE	-

2.2.18 network_monitor_disable

network_monitor_disable	
Description	This API is used to disable the Automatic 4G monitor feature.
Syntax	int network_monitor_disable ()
Arguments	None
Return	Refer *Cellular module specific return codes* section.
Example	ret = network_monitor_disable ();
NOTE	This API would have to be called before calling the deinit ().

2.2.19 unread_message

unread_message	
Description	This API is used to read all unread SMS.
Syntax	int unread_message (char *msg_buf, int length, int max_resp_time)
Arguments	• char *msg_buf - array to store all unread messages. • Length - Size/Length of the array to store all unread messages • max_resp_time - response time to read response from GSM module. (In millisec)
Return	Refer *Cellular module specific return codes* section.
Example	char resp_buffer [1024]; ret = unread_message (resp_buffer, sizeof(resp_buffer), 800000);

2.2.20 read_message

read_message	
Description	This API is used to read all read SMS.
Syntax	int read_message (char *msg_buf, int length, int max_resp_time)

Arguments	• char *msg_buf - array to store all read messages. • Length - Size/Length of the array to store all read messages • max_resp_time - response time to read response from GSM module. (In microseconds)
Return	Refer *Cellular module specific return codes* section.
Example	char resp_buffer [1024]; ret = read_message (resp_buffer, sizeof(resp_buffer), 800000);

2.2.21 send_sms

send_sms	
Description	This API is used to send SMS to specified mobile number.
Syntax	int send_sms (char *msg_response, char *sender_number, int max_resp_time)
Arguments	• char *msg_response - array stored with message to send. • char *sender_number - array stored with mobile number in which message must be to send. • max_resp_time - response time to read response from GSM module. (In microseconds)
Return	Refer *Cellular module specific return codes* section.
Example	ret = send_sms (“Hi, I have a user application”, “7089234567”, “800000);
NOTE	gsm_modem_on () and establish_connection () should be called before calling send_sms ().

2.2.22 delete_message

delete_message	
Description	This API is used to delete SMS using index number.
Syntax	int delete_message (int index, int max_resp_time)
Arguments	• int index - index number of the message to be deleted. • max_resp_time - response time to read response from GSM module. (In microseconds)
Return	Refer *Cellular module specific return codes* section.
Example	ret = delete_message (1, 800000);

2.2.23 delete_all_messages

delete_all_messages	
Description	This API is used to delete all read SMS.
Syntax	int delete_all_messages (int max_resp_time)
Arguments	max_resp_time - response time to read response from GSM module. (In microseconds)
Return	Refer *Cellular module specific return codes* section.

Example	ret = delete_all_messages (1, 800000);

2.3 Cellular module specific return codes

Return Codes	Description
E0010001	GSM port initialization error
E0010002	GSM port write error
E0010003	GSM port read error
E0010004	GSM port disconnection error
E0010005	GSM AT port initialization error
E0010006	GSM AT port write error
E0010007	GSM AT port read error
E0010008	IMEI read timeout error
E0010009	Network connection down
E001000A	SIM is not detected
E001000B	SIM detected
E001000C	SIM status unknown
E1010001	GSM SIM registration error
E1010002	Failed to obtain signal strength
E1010003	Failed to obtain SIM ICCID
E1010004	GSM SMS send error
E1010005	GSM SMS send buffer invalid
E1010006	GSM SMS delete using index error
E1010007	GSM SMS delete all read messages error
E1010008	Data Overflow occurred - Length of buffer passed insufficient
E1010009	GSM Network mode enabled (Network related handling done internally)
E2010001	GSM semaphore initialization error
E2010002	SIM is not valid
E2010003	GSM NTP Fail
E2010004	GSM SMS initialization error
E2010005	GSM SMS read all unread SMS error
E2010006	GSM SMS read all read SMS error

For other possible general return codes refer *General Return Codes* section.

2.4 CAN APIs

2.4.1 can_init

can_init	
Description	This API initializes CAN interface.
Syntax	int can_init (const char *name, int bitrate)
Arguments	• const char *name - string contains CAN interface name (Ex: can0, can1 or can2) for initializing CAN. • int bitrate - variable which holds the baudrate to be set to the CAN interface.
Return	Refer *CAN specific return codes* section

Example	char name = “can0”;
	int bitrate = 500000;
	ret = can_init (name, bitrate);

NOTE	

2.4.2 can_fd_init

can_fd_init	
Description	This API initializes CAN_FD interface.
Syntax	int can_fd_init (const char *name, int bitrate, int dbitrate, int txqueuelen)
Arguments	• const char *name – Variable which holds the CAN name • int bitrate - Variable which holds the baudrate to be set to the CAN FD interface. • int dbitrate - Variable which holds the data bitrate to be set to the CAN FD interface. • int txqueuelen - Number of packets in the transmission queue.
Return	Refer *CAN specific return codes* section

Example	const char *name = CAN3
	
	int bitrate = 500000;
	int dbitrate = 1000000;
	int txqueuelen = 25000;
	ret = can_fd_init (name,bitrate, dbitrate, txqueuelen);

NOTE	• To use CANFD(CAN3 and CAN4) follow the below steps • Insert the CANFD module by using below command: # *insmod tcan4x5x.ko*

2.4.3 set_can_mask_and_filter

set_can_mask_and_filter	
Description	This API is used for setting filters for the CAN bus using CAN ids. For each filter mask must be defined. This API would be called before calling can_read API. All Filters can set single time.
Syntax	int set_can_mask_and_filter (uint32_t *mask, uint32_t *filter, int no_of_filter);
Arguments	• uint32_t *mask: pointer array which holds the CAN masks. • uint32_t *filter: pointer array which holds the CAN filter ID's. • int no_of_filter: variable which holds the number of filters to set.
Return	Refer *CAN specific return codes* section

Example	uint32_t mask[7] = {113, 502, 202, 534, 615, 7E8, 18DAF110};
	uint32_t filter[7] = {113, 502, 202, 534, 615, 7E8, 18DAF110};
	int no_of_filter = 7;
	rc = can_init("can0", 250000);
	rc = set_can_mask_and_filter(mask, filter, no_of_filter);

NOTE	

2.4.4 can_write

can_write	
Description	This API is used to send CAN request with CAN ID with required mode and PID to the can bus.
Syntax	int can_write (char *name, char *data)
Arguments	• char *data - data to be written to the can bus. • char *name - can interface name. Ex: "can0"
Return	Refer *CAN specific return codes* section
Example	char name [] = "can0"; char data [] = "123#AABBCCDD"; ret = can_write (name, data);
NOTE	can_init () should be called before calling can_write () API.

2.4.5 can_read

can_read	
Description	This API is used to read can response messages for the requested CAN PIDs if any request is sent. If no request sent then broadcast messages will be receiving.
Syntax	int can_read (char *name, struct canfd_frame *frame)
Arguments	• char *name - can interface name. Ex: “can0” • struct canfd_frame *frame - to store received CAN response data.
Return	Length of CAN data
Example	char name [] = “can0”; struct canfd_frame frame; ret = can_read (name, &frame);
NOTE	can_init () should be called before calling can_read () API.

2.4.6 config_can_wakeup

config_can_wakeup	
Description	This API is used to enable or disable CAN wakeup.
Syntax	int config_can_wakeup (char *can_name, int option)
Arguments	• CAN interface name. Ex: “can0” • int option - option to enable or disable CAN wakeup 1 - enable 0 – disable
Return	Refer *CAN specific return codes* section

Example	char can_name [] = “can0”;
	int option = 1;
	ret = config_can_wakeup(option);

NOTE	can_init () should be called before calling this API.

2.4.7 can_deinit

can_deinit	
Description	This API de-initializes the CAN interface.
Syntax	int can_deinit (const char *name)
Arguments	const char *name - string contains CAN interface name (Ex: can0 or can1) for deinitializing CAN.
Return	Refer *CAN specific return codes* section
Example	char name [] = “can0”; ret = can_deinit(name);
NOTE	can_init () should be called before calling this API.

2.5CAN specific return codes

Return Codes	Description
90010001	CAN baud rate set error
90010002	CAN initialization error
90010003	CAN interface not found
90010004	CAN Semaphore initialization error
90010005	CAN de-initialization error
90010006	CAN Wakeup enabled error
90010007	CAN0 Wakeup file close error
90010008	CAN1 Wakeup file close error
90010009	CAN2 Wakeup file close error
9001000A	CAN read timeout
9001000B	CAN invalid bitrate
9001000C	CAN Module load error
9001000D	CAN3 Wakeup file close error
9001000E	CAN4 Wakeup file close error

For other possible general return codes refer *General Return Codes* section.

2.6 Ethernet APIs

2.6.1 eth_init

eth_init	
Description	This API initializes Ethernet interface.
Syntax	int eth_init (char *iface)
Arguments	char *iface – string contains ethernet interface (eth0 or eth1)
Return	Refer *Ethernet specific return codes* section.
Example	ret = eth_init (“eth0”)
NOTE	-

2.6.2 eth_deinit

eth_deinit	
Description	This API de-initializes Ethernet interface.
Syntax	int eth_deinit (char *iface)
Arguments	char *iface – string contains ethernet interface (eth0 or eth1)
Return	Refer *Ethernet specific return codes* section.
Example	ret = eth_deinit (“eth0”)
NOTE	-

2.7 Ethernet specific return codes

Return Codes	Description
60010001	ETH0 MAC Address initialization error
60010002	ETH1 MAC Address initialization error
60010003	Ethernet interface initialization error
60010004	Ethernet interface de-initialization error

For other possible general return codes refer *General Return Codes* section.

2.8 Accelerometer APIs

2.8.1 acc_init

acc_init	
Description	This API initializes accelerometer.
Syntax	int acc_init ()
Arguments	None
Return	Refer **Error! Reference source not found.** section.
Example	ret = acc_init ()

2.8.2 acc_deinit

acc_deinit	
Description	This API de-initializes accelerometer.
Syntax	int acc_deinit ()
Arguments	None
Return	Refer **Error! Reference source not found.** section.
Example	ret = acc_deinit ()

2.8.3 accelerometer_read

accelerometer_read	
Description	This API reads the accelerometer values from the Telematics, and returns x-axis, y-axis and z-axis data in the provided structure. The output data will be in unit m/s² (meter per second square). **acc->x** holds x-axis value, **acc->y** holds y-axis value and **acc->z** holds z-axis values.
Syntax	int accelerometer_read (accelerometer_api_priv *acc)
Arguments	▪ accelerometer_api_priv *acc - pointer to structure accelerometer_api_priv
Return	Refer **Error! Reference source not found.** section.
Example	accelerometer_api_priv g_adata; ret = accelerometer_read(&g_adata);
NOTE	acc_init () must be called before calling this API.

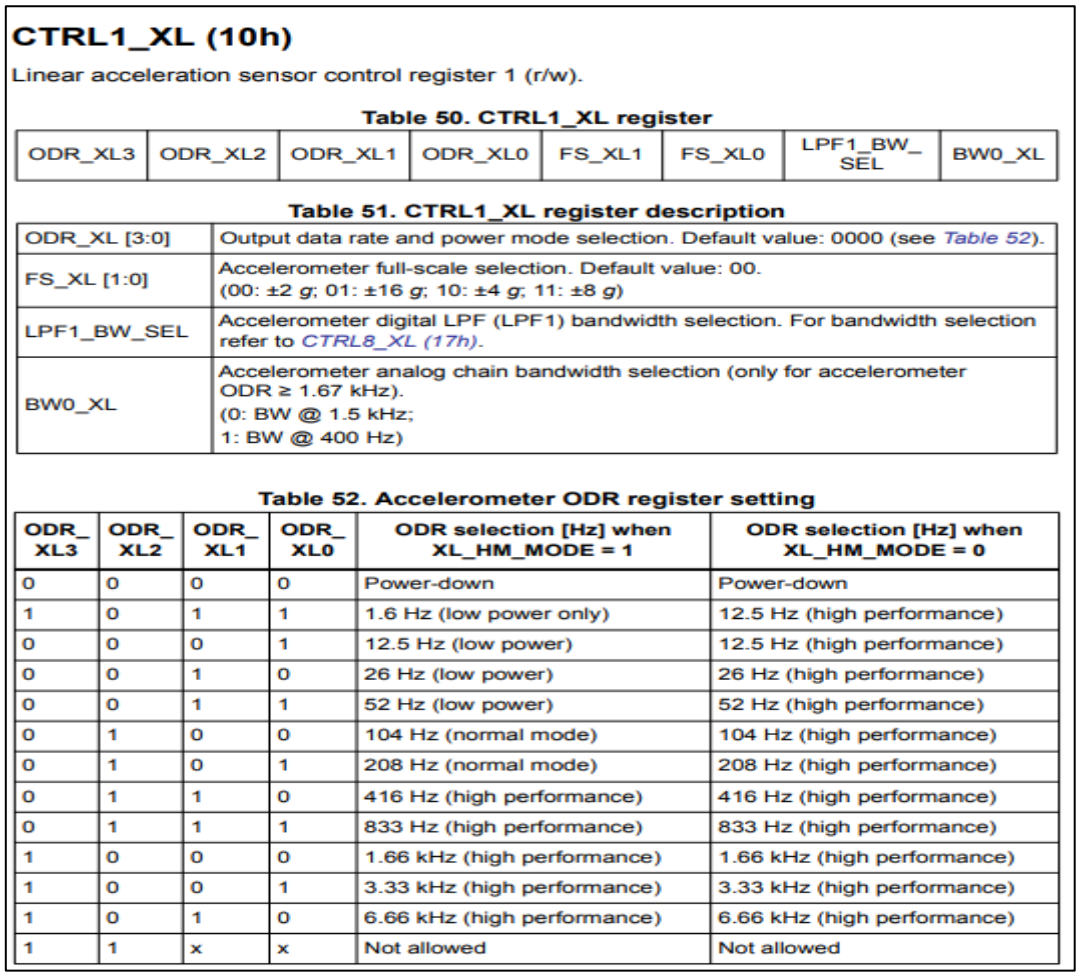
2.8.4 set_acc_low_pass_filter

set_acc_low_pass_filter	
Description	This API is used to change the low pass filter of accelerometer sensor.
Syntax	int set_acc_low_pass_filter (uint8_t value)
Arguments	uint8_t value - low pass filter value to be written.

Return	Refer**Error! Reference source not found.** section.
Example	ret = set_acc_low_pass_filter(0x80);
NOTE	• acc_init () must be called before calling this API. • The value passed to the API should in hex format. • For set_acc_low_pass_filter pass 0x80 as a default value.

2.8.5 set_acc_sampling_frequency

set_acc_sampling_frequency	
Description	This API is used to change the sampling frequency of accelerometer sensor.
Syntax	int set_acc_sampling_frequency (uint8_t value)
Arguments	uint8_t value - sampling frequency value to be written.
Return	Refer **Error! Reference source not found.** section.
Example	ret = set_acc_sampling_frequency(0x70);
NOTE	• acc_init () must be called before calling this API. • The value passed to the API should in hex format. Refer **Error! Reference source not found.** for supported frequencies and corresponding values.



CTRL1_XL (10h)

Linear acceleration sensor control register 1 (r/w).

Table 50. CTRL1_XL register

ODR_XL3	ODR_XL2	ODR_XL1	ODR_XL0	FS_XL1	FS_XL0	LPF1_BW_SEL	BW0_XL

Table 51. CTRL1_XL register description

ODR_XL [3:0]	Output data rate and power mode selection. Default value: 0000 (see *Table 52*).
FS_XL [1:0]	Accelerometer full-scale selection. Default value: 00. (00: ±2 *g*; 01: ±16 *g*; 10: ±4 *g*; 11: ±8 *g*)
LPF1_BW_SEL	Accelerometer digital LPF (LPF1) bandwidth selection. For bandwidth selection refer to *CTRL8_XL (17h)*.
BW0_XL	Accelerometer analog chain bandwidth selection (only for accelerometer ODR ≥ 1.67 kHz). (0: BW @ 1.5 kHz; 1: BW @ 400 Hz)

Table 52. Accelerometer ODR register setting

ODR_XL3	ODR_XL2	ODR_XL1	ODR_XL0	ODR selection [Hz] when XL_HM_MODE = 1	ODR selection [Hz] when XL_HM_MODE = 0
0	0	0	0	Power-down	Power-down
1	0	1	1	1.6 Hz (low power only)	12.5 Hz (high performance)
0	0	0	1	12.5 Hz (low power)	12.5 Hz (high performance)
0	0	1	0	26 Hz (low power)	26 Hz (high performance)
0	0	1	1	52 Hz (low power)	52 Hz (high performance)
0	1	0	0	104 Hz (normal mode)	104 Hz (high performance)
0	1	0	1	208 Hz (normal mode)	208 Hz (high performance)
0	1	1	0	416 Hz (high performance)	416 Hz (high performance)
0	1	1	1	833 Hz (high performance)	833 Hz (high performance)
1	0	0	0	1.66 kHz (high performance)	1.66 kHz (high performance)
1	0	0	1	3.33 kHz (high performance)	3.33 kHz (high performance)
1	0	1	0	6.66 kHz (high performance)	6.66 kHz (high performance)
1	1	x	x	Not allowed	Not allowed

Figure 1: Accelerometer ODR setting

- XL_HM_MODE on default is set to 0 that is high performance mode. As per the image we need to change only the CTRL1_XL register based on our requirement.
- LPF1_SW_SEL and BW0_XL is 0 by default.
- Below is the table with the example hex value argument that can be set to the set_acc_sampling_frequency:

Table 2: Hex values of set_acc_sampling_frequency

Hexa Value	Binary Values							
	ODRXL_3	ODRXL_2	ODRXL_1	ODRXL_0	FSXL_1	FSXL_0	LPF1_BW_SEL	BW0_XL
B0	1	0	1	1	0	0	0	0
70	0	1	1	1	0	0	0	0
78	0	1	1	1	1	0	0	0
7C	0	1	1	1	1	1	0	0
74	0	1	1	1	0	1	0	0

2.8.6 set_acc_wakeup_threshold

set_acc_wakeup_threshold	
Description	This API is used to change the wakeup threshold value of accelerometer sensor.
Syntax	int set_acc_wakeup_threshold (uint8_t value)
Arguments	uint8_t value - wake up threshold value to be written.
Return	Refer **Error! Reference source not found.** section.
Example	ret = set_acc_wakeup_threshold(0x81);
NOTE	• acc_init () must be called before calling this API. • The value passed to the API should in hex format and should be valid.

TAP_THS_6D (59h)

Portrait/landscape position and tap function threshold register (r/w).

Table 185. TAP_THS_6D register

D4D_EN	SIXD_THS 1	SIXD_THS 0	TAP_THS 4	TAP_THS 3	TAP_THS 2	TAP_THS 1	TAP_THS 0

Table 186. TAP_THS_6D register description

D4D_EN	4D orientation detection enable. Z-axis position detection is disabled. Default value: 0 (0: enabled; 1: disabled)
SIXD_THS[1:0]	Threshold for 4D/6D function. Default value: 00 For details, refer to *Table 187*.
TAP_THS[4:0]	Threshold for tap recognition. Default value: 00000 1 LSb corresponds to $FS\_XL/2^5$

Table 187. Threshold for D4D/D6D function

SIXD_THS[1:0]	Threshold value
00	80 degrees
01	70 degrees
10	60 degrees
11	50 degrees

Figure 2: Accelerometer Threshold settings

2.8.7 config_acc_wakeup

config_acc_wakeup	
Description	This API is used to enable or disable accelerometer wakeup.
Syntax	int config_acc_wakeup (int option)
Arguments	• int option - option to enable or disable accelerometer wakeup 1 - enable 0 - disable
Return	Refer **Error! Reference source not found.** section.
Example	int option = 1; ret = config_acc_wakeup(option);
NOTE	acc_init () must be called before calling this API.

2.8.8 acc_temp_read

acc_temp_read	
Description	This API is used for reading the ambient board temperature using accelerometer-temperature sensor.
Syntax	int acc_temp_read (float_t *temperature)
Arguments	• float_t *temperature-An empty float_t pointer to store the temperature.
Return	Refer **Error! Reference source not found.** section.
Example	float temperature; ret = acc_temp_read(&temperature);

2.9 Accelerometer specific return codes

Return Codes	Description
C0010001	**Accelerometer X axis initialization error**
C0010002	**Accelerometer Y axis initialization error**
C0010003	**Accelerometer Z axis initialization error**
C0010004	**Accelerometer buffer initialization error**
C0010005	**Accelerometer build channel array error**
C0010006	**Accelerometer scan size error**
C0010007	**Accelerometer interrupt disable error**
C0010008	**Accelerometer i2c register configuration error**
C0010009	**Accelerometer semaphore initialization error**

For other possible general return codes refer *General Return Codes* section.

2.10 Gyroscope APIs

2.10.1 gyro_init

gyro_init	
Description	This function initializes gyroscope.
Syntax	int gyro_init ()
Arguments	None
Return	Refer *Gyroscope specific return codes* section.
Example	ret = gyro_init ();

2.10.2 gyro_deinit

gyro_deinit	
Description	This function de-initializes gyroscope.
Syntax	int gyro_deinit ()
Arguments	None
Return	Refer *Gyroscope specific return codes* section.
Example	ret = gyro_deinit ();

2.10.3 gyroscope_read

gyroscope_read	
Description	This API reads the gyroscope values from the Telematics, and returns x-axis, y-axis and z-axis data in the provided structure. The output data will be in unit RPS (radian per second). **g_data->x** holds x-axis value, **g_data->y** holds y-axis value and **g_data->z** holds z-axis values.
Syntax	int gyroscope_read (gyroscope_api_priv *g_data)
Arguments	gyroscope_api_priv *g_data - pointer to the structure gyroscope_api_priv
Return	Refer *Gyroscope specific return codes* section.
Example	gyroscope_api_priv g_data; ret = gyroscope_read(&g_data);
NOTE	gyro_init () must be called before calling this API.

2.10.4 set_gyro_sampling_frequency

set_gyro_sampling_frequency	
Description	This API is used to change the sampling frequency of gyroscope sensor.
Syntax	int set_gyro_sampling_frequency (uint8_t value)
Arguments	uint8_t value - sampling frequency value to be written.
Return	Refer *Gyroscope specific return codes* section.
Example	ret = set_gyro_sampling_frequency (0x70);
NOTE	• gyro_init () must be called before calling this API. • The value passed to the API should in hex format. Refer **Error! Reference source not found.** for supported frequencies and corresponding values.

9.14 CTRL2_G (11h)

Angular rate sensor control register 2 (r/w).

Table 53. CTRL2_G register

ODR_G3	ODR_G2	ODR_G1	ODR_G0	FS_G1	FS_G0	FS_125	0(1)

1. This bit must be set to '0' for the correct operation of the device.

Table 54. CTRL2_G register description

ODR_G [3:0]	Gyroscope output data rate selection. Default value: 0000 (Refer to *Table 55*)
FS_G [1:0]	Gyroscope full-scale selection. Default value: 00 (00: 250 dps; 01: 500 dps; 10: 1000 dps; 11: 2000 dps)
FS_125	Gyroscope full-scale at 125 dps. Default value: 0 (0: disabled; 1: enabled)

Table 55. Gyroscope ODR configuration setting

ODR_G3	ODR_G2	ODR_G1	ODR_G0	ODR [Hz] when G_HM_MODE = 1	ODR [Hz] when G_HM_MODE = 0
0	0	0	0	Power down	Power down
0	0	0	1	12.5 Hz (low power)	12.5 Hz (high performance)
0	0	1	0	26 Hz (low power)	26 Hz (high performance)
0	0	1	1	52 Hz (low power)	52 Hz (high performance)
0	1	0	0	104 Hz (normal mode)	104 Hz (high performance)
0	1	0	1	208 Hz (normal mode)	208 Hz (high performance)
0	1	1	0	416 Hz (high performance)	416 Hz (high performance)
0	1	1	1	833 Hz (high performance)	833 Hz (high performance)
1	0	0	0	1.66 kHz (high performance)	1.66 kHz (high performance)
1	0	0	1	3.33 kHz (high performance	3.33 kHz (high performance)
1	0	1	0	6.66 kHz (high performance	6.66 kHz (high performance)
1	0	1	1	Not available	Not available

Figure 3: Gyroscope ODR settings

2.10.5 set_gyro_low_pass_filter

set_gyro_low_pass_filter	
Description	This API is used to change the low pass filter of gyroscope sensor.
Syntax	int set_gyro_low_pass_filter (uint8_t value)
Arguments	uint8_t value - low pass filter value to be written.
Return	Refer *Gyroscope specific return codes* section.
Example	ret = set_gyro_low_pass_filter(0x80);
NOTE	• gyro_init () must be called before calling this API. • The value passed to the API should in hex format.

2.11 Gyroscope specific return codes

Return Codes	Description
B0010001	**Gyroscope X axis initialization error**
B0010002	**Gyroscope Y axis initialization error**
B0010003	**Gyroscope Z axis initialization error**
B0010004	**Gyroscope buffer initialization error**
B0010005	**Gyroscope build channel array error**
B0010006	**Gyroscope scan size error**
B0010007	**Gyroscope i2c register configuration error**
B0010008	**Gyroscope semaphore initialization error**

For other possible general return codes, refer*General Return Codes* section.

2.12 GPS APIs

2.12.1 gps_init

gps_init	
Description	This function initializes GPS.
Syntax	int gps_init ()
Arguments	None
Return	Refer GPS specific return codes section.
Example	ret = gps_init ();
NOTE	▪ gsm_modem_on () must be called before gps_init () and agps_init () can be called before calling gps_init () for the faster fix.

2.12.2 gps_deinit

gps_deinit	
Description	This function de-initializes GPS.
Syntax	int gps_deinit ()
Arguments	None
Return	Refer

	GPS specific return codes section.
Example	ret = gps_deinit ();
NOTE	-

2.12.3 agps_init

agps_init	
Description	This function initializes AGPS.
Syntax	int agps_init ()
Arguments	None
Return	Refer GPS specific return codes section.
Example	ret = agps_init ();

NOTE	• This API fetches latest extra data from the server. So, this function may take some time.
	• Before AGPS init: (Both BG96 and EG95)
	  ○ In case init (0) is called for device initialization, then establish_connection () API to be called.
	  ○ In case of init (1), use check_network_connection () API to check PPP link is Up.
	• After AGPS init:
	  ○ In case of BG96 Module if init (0) is called establish_connection () API to be called.
	  ○ In case of init (1) Network manager thread will take care of PPP link.
	• In case of EG95 series module no need to call establish_connection () API after agps_init ().

2.12.4 get_gps_data

get_gps_data	
Description	This API reads the gps data from the Telematics and returns the data in the character array recv_data.
Syntax	int get_gps_data (char * nmea, size_t * g_nbytes, char *recv_data, int length)
Arguments	• char * nmea - type of nmea message to be read. Ex: “GPRMC”, “GPGSA” etc. • size_t * g_nbytes - pointer to variable which contains number of bytes read • char *recv_data - pointer to character array (128 bytes) to store the received data • int length - Size/Length of the character array to store the received data
Return	Refer GPS specific return codes section.
Example	char recv_data [200] = {0}; Size_t len = 0; int ret = 0; sleep (1); ret = get_gps_data (“GPRMC”, &len, recv_data, sizeof(recv_data));
NOTE	▪ NMEA message will be stored to the recv_data buffer. gps_init () must be called before calling this API. ▪ Once the GPS initialization is done, add a sleep of minimum 1 second before calling this API. ▪ Add a sleep of minimum 1 second between each get_gps_data() API calls. Otherwise, the API will return "0xA001000B" indicating the timeout error. ▪ Ignore the first data read after the GPS initialization as it can be an older data.

2.13 GPS specific return codes

Return Codes	Description
A0010001	GPS data port initialization error
A0010002	GPS data port de-initialization error
A0010003	GPS data port disconnection error
A0010004	AGPS enable error
A0010005	AGPS set time error
A0010006	AGPS delete existing data fail
A0010007	AGPS HTTP configuration enable fail
A0010008	AGPS QIACT enable error
A0010009	AGPS URL size error
A001000B	Timeout Error.
A1010001	AGPS downloaded data invalid
A1010002	AGPS read data invalid
A1010003	AGPS load data invalid
A1010004	AGPS Semaphore initialization error
A1010005	AGPS QICSGP error
A1010006	AGPS listing existing data error
A1010007	AGPS QNTP error
A1010008	GPS port already exists

For other possible general return codes, refer *General Return Codes*section.

2.14 Battery APIs

2.14.1 i_battery_init

i_battery_init	
Description	This function initializes Battery.
Syntax	int i_battery_init ()
Arguments	None
Return	Refer *Battery specific return codes* section.
Example	ret = i_battery_init ();
NOTE	This API fetches latest extra data from the server. So, this may take some time.

2.14.2 i_get_battery_status

i_get_battery_status	
Description	This API is used to check the battery charging status.
Syntax	int i_get_battery_status (int *b_chrg_status)
Arguments	int *b_chrg_status - Variable to store the battery status
Return	• b_chrg _status = 0x1003, if battery not connected • b_chrg _status = 0x1004, if battery fully charged • b_chrg _status = 0x1005, if battery is charging • b_chrg _status = 0x1006, if battery is not charging • b_chrg _status = 0x1007, if battery is unstable • Refer *Battery specific return codes* section.
Example	int batter_chrg_status; ret = i_get_battery_status (& batter_chrg_status);
NOTE	• If the battery charge status is being shown as UNSTABLE_STATE(0x1007) but the proper voltage is displayed and battery heath is good, it implies the battery is in Shipment mode. • Battery charge status will be in UNSTABLE STATE (0x1007) if battery charging is disabled.

2.14.3 i_battery_get_voltage

i_battery_get_voltage	
Description	This API is used to check the battery voltage.
Syntax	int i_battery_get_voltage (double *i_bat_volt)
Arguments	double *i_bat_volt - address of variable to store voltage value.
Return	Refer *Battery specific return codes* section.
Example	double i_bat_volt; ret = i_battery_get_voltage (&i_bat_volt);
NOTE	i_bat_volt contains the battery voltage value.

2.14.4 i_battery_get_health

i_battery_get_health	
Description	This API is used to check the battery health status.
Syntax	int i_battery_get_health ()
Arguments	None
Return	• 0x1001, if battery is in good health. • 0x1002, if battery is not in good health • Refer *Battery specific return codes* section.
Example	ret = i_battery_get_health ();
NOTE	i_bat_volt contains the battery voltage value.

2.14.5 battery_connect_config

battery_connect_config	
Description	This API is used to either disable/enable shipment mode.
Syntax	int battery_connect_config (int con_status)
Arguments	• int con_status: Pass this value to enable or disable the Shipment Mode. 0 - Enable the shipment mode after 10 seconds. 1 - Disable the shipment mode. 2 - Enable the shipment mode immediately.
Return	Refer *Battery specific return codes* section.
Example	int con_status = 2; ret = battery_connect_config(con_status);
NOTE	• If *0* is passed, shipment mode will be enabled after 10 seconds. (i.e., disabling the power flow from the battery). • If *1* is passed, shipment mode will be disabled (i.e., enabling all the functionalities related to battery (charging, discharging and other functionalities)). • If *2* is passed, shipment mode will be enabled immediately (i.e.,

	Disables the power flow from the battery).

2.14.6 battery_charge_state_config

battery_charge_state_config	
Description	This API is used to either disable/enable battery charging.
Syntax	int battery_charge_state_config (int state)
Arguments	int state: state must be either low/high
Return	Refer *Battery specific return codes* section.
Example	int state = 1; ret = battery_charge_state_config (state);
NOTE	• If low (0) is passed, battery charging will be enabled. • If high (1) is passed, battery charging will be disabled.

2.14.7 get_power_source

get_power_source	
Description	This API is used to check the power source status.
Syntax	int get_power_source ()
Arguments	None
Return	• 0x1008, if connected to external power • 0x1009, if connected to internal power • Refer *Battery specific return codes* section.
Example	ret = get_power_source ();

NOTE	• Sequence to be followed for battery
	  ○ init ().
	  ○ battery_init ().
	  ○ Any API related to battery.

2.14.8 i_get_battery_temp

i_get_battery_temp	
Description	This API is used to check the battery temperature.
Syntax	int i_get_battery_temp (int *b_temp_status)
Arguments	int *b_temp_status - Variable to store the battery temperature value.
Return	• 0x100A, if battery is in Normal Temperature • 0x100B, if battery is in Warm Temperature • 0x100C, if battery is in Cool Temperature • 0x100D, if battery is in Cold Temperature • 0x100E, if battery is in Hot Temperature • Refer *Battery specific return codes* section.
Example	int b_temp_status; ret = i_get_battery_temp (&b_temp_status);
NOTE	• b_temp_status contains the battery temperature value.

2.15 Battery specific return codes

Return Codes	Description
D0010001	**Battery voltage node open error**
D0010002	**Battery voltage node read error**
D0010003	**Battery health node open error**
D0010004	**Battery health node read error**
D0010005	**Battery not connected**
D0010006	**Battery is fully connected**
D0010007	**Battery is charging**
D0010008	**Battery is not charging**
D0010009	**Battery unstable state**

For other possible general return codes, refer *General Return Codes* section.

2.16 WiFi APIs

2.16.1 wifi_init

wifi_init	
Description	This API is used to initialize WiFi interface with hostapd or station mode.
Syntax	int wifi_init (int mode)
Arguments	• int mode - enable hostapd or station mode. 1 - hostapd mode 0 - station mode
Return	Refer *WiFi specific return codes* section.
Example	int mode = 0; ret = wifi_init(mode);
NOTE	• In-order to avoid getting the above error code and to avoid enabling Wi-Fi on boot up, service needs to be disabled or modified. • In-order to enable routing of the 4G internet connection through the Wi-Fi interface, please ensure that 4G is initialized with an internet connection before initializing the Wi-Fi interface. • Routing 4G internet connection through the Wi-Fi interface can be availed only when the Wi-Fi is initialized in Access Point Mode.

2.16.2 wifi_deinit

wifi_deinit	
Description	This API is used to deinitialize WiFi interface with hostapd or station mode.
Syntax	int wifi_deinit ()
Arguments	None
Return	Refer *WiFi specific return codes* section.
Example	ret = wifi_deinit ();
NOTE	-

2.17 WiFi specific return codes

Return Codes	Description
80010001	WiFi interface initialization error
80010002	WiFi interface deinitialization error
80010003	WiFi interface Up error
80010004	WiFi hostapd mode initialization error
80010005	WiFi station mode initialization error
80010006	WiFi hostapd mode deinitialization error
80010007	WiFi station mode deinitialization error
80010008	WiFi IP set error
80010009	WiFi hostapd mode udhcpd configuration error

For other possible general return codes, refer *General Return Codes* section.

2.18 Bluetooth APIs

2.18.1 ble_init

ble_init	
Description	This API is used to initialize Bluetooth Interface.
Syntax	int ble_init ()
Arguments	None
Return	Refer *Bluetooth specific return codes* section.
Example	ret = ble_init ();
NOTE	-

2.18.2 ble_deinit

ble_deinit	
Description	This API is used to de-initialize Bluetooth Interface.
Syntax	int ble_deinit ()
Arguments	None
Return	Refer *Bluetooth specific return codes* section.
Example	ret = ble_deinit ();
NOTE	-

2.19 Bluetooth specific return codes

Return Codes	Description
70010001	**Bluetooth interface initialization error**
70010002	**Bluetooth interface deinitialization error**

For other possible general return codes, refer *General Return Codes*section.

2.20 RS232 APIs

2.20.1 rs232_init

rs232_init	
Description	This API is used to initialize RS232 UART.
Syntax	int rs232_init (int baudrate)
Arguments	• int Baudrate – Baudrate for the RS232 UART.
Return	Refer *General Return Codes* section.
Example	ret = rs232_init (9600);

2.20.2 rs232_deinit

rs232_deinit	
Description	This API is used to de-initialize RS232 UART.
Syntax	int rs232_deinit ()
Arguments	None
Return	Refer *General Return Codes* section.
Example	int ret = 0; ret = rs232_deinit ();

2.20.3 rs232_write

rs232_write	
Description	This API is used to write the data to RS232 UART.
Syntax	int rs232_write (char *buf, size_t sz)
Arguments	• char *buf – To store the RS232 data. • Size_t sz – Size of the input buf in bytes.
Return	Refer *General Return Codes* section.

Example	char buf2[1048] = “Telematics”;
	int ret = 0;
	ret = rs232_write (buf2, 17);

NOTE	▪ rs232_init must be called before calling this API.

2.20.4 rs232_read

rs232_read	
Description	This API is used to read the data from RS232 UART.
Syntax	int rs232_read (char *buf, long int sz)
Arguments	• char *buf – To store the RS232 data. • Size_t sz – Maximum number of bytes to be read from RS232 UART.
Return	Refer *General Return Codes* section.

Example	char buf1[1048];
	int ret = 0;
	ret = rs232_read (buf1, 17);

NOTE	▪ rs232_init must be called before calling this API.

2.21 RS485 related APIs

2.21.1 Rs485_init

Rs485_init	
Description	This API is used to initialize RS485 UART.
Syntax	int rs485_init (int baudrate)
Arguments	• int Baudrate – Baudrate for the RS485 UART.
Return	Refer *General Return Codes* section.
Example	ret = rs485_init (9600);

2.21.2 rs485_deinit

Rs485_deinit	
Description	This API is used to de-initialize RS485 UART.
Syntax	int rs485_deinit ()
Arguments	None
Return	Refer *General Return Codes* section.
Example	int ret = 0; ret = rs485_deinit ();

2.21.3 rs485_write

Rs485_write	
Description	This API is used to write the data to RS485 UART.
Syntax	int rs485_write (char *buf, size_t sz)
Arguments	• char *buf – To store the RS485 data. • Size_t sz – Size of the input buf in bytes.
Return	Refer *General Return Codes* section.

Example	char buf2[1048] = “Telematics”;
	int ret = 0;
	ret = rs485_write (buf2, 17);

NOTE	▪ Rs485_init must be called before calling this API.

2.21.4 Rs485_read

Rs485_read	
Description	This API is used to read the data from RS485 UART.
Syntax	int rs485_read (char *buf, long int sz, int mode)
Arguments	• char *buf – To store the RS485 data. • Size_t sz – Maximum number of bytes to be read from RS485 UART. • Int mode – used to select either Half or Full duplex operation
Return	Refer *General Return Codes* section.

Example	char buf1[1048];
	int ret = 0;
	ret = rs485_read (buf1, 17,1);

NOTE	▪ Rs485_init must be called before calling this API. ▪ Mode = 0: Half duplex ▪ Mode = 1: Full duplex

2.22 MCU related APIs

2.22.1 MCU_sleep_mode

Note: If using *MCU_sleep_mode ()* API, please ensure that the below APIs are not used:

- *push_device_to_sleep ()*
- *config_timer_wakeup ()*
- *config_ignition_wakeup ()*
- *config_can_wakeup ()*
- *config_acc_wakeup ()*

<table>
<tr><th colspan="2">MCU_sleep_mode</th></tr>
<tr><td>Description</td><td>This API is used to setup the MCU sleep mode.</td></tr>
<tr><td>Syntax</td><td>int MCU_sleep_mode (uint8_t sleep_mode, uint8_t wakeup_source, uint32_t *timer)</td></tr>
<tr><td>Arguments</td><td>
• uint8_t sleep_mode - The field indicates the sleep wakeup source to be enabled. Below are the 2 sleep modes available.
<table>
<tr><th>Hex Value</th><th>MCU Sleep Mode</th></tr>
<tr><td>0x00</td><td>Sleep Mode</td></tr>
<tr><td>0x04</td><td>Deep Power Down Mode</td></tr>
</table>
▪ uint8_t wakeup_source - This field indicates the various wakeup sources to be enabled. Below are the various sleep wakeup sources available.
<table>
<tr><th>Bit Information</th><th>MCU Wakeup Sources</th></tr>
<tr><td>Bit7(MSB)</td><td>Timer Wakeup enable bit</td></tr>
<tr><td>Bit6</td><td>Ignition Wakeup enable bit</td></tr>
<tr><td>Bit5</td><td>Accelerometer Wakeup enable bit</td></tr>
</table>
Example: For Ignition and accelerometer wakeup, the below value is to be passed to this API:

0x60 (0b01100000)

• uint32_t timer - This field contains the timer (in seconds) for sleep. This is applicable only if the Timer wakeup is enabled.
</td></tr>
<tr><td>Return</td><td>Refer</td></tr>
</table>

	MCU Specific Return Codes section.

Example	int ret = 0;
	uint8_t sleep_mode = 1
	uint8_t wakeup_source = 0x40;
	uint32_t *timer = 0;
	ret = MCU_sleep_mode (sleep_mode, wakeup_source, timer);

NOTE	The timer wakeup is not supported in Deep Power Down Mode. If only Timer wakeup is requested in Deep Power Down Mode, the API will return error.

2.22.2 MCU_Wakeup_Source_Request

MCU_Wakeup_Source_Request	
Description	This API is used to request the source of sleep wakeup from MCU.
Syntax	int MCU_Wakeup_Source_Request (uint8_t *wake_source)
Arguments	uint8_t *wake_source - Variable to store the Source of wakeup. Below are the available wakeup source and their corresponding hex values. **Hex Value** / **MCU Wakeup Source** 0x80 – Wakeup source was Timer 0x40 – Wakeup source was Ignition 0x20 – Wakeup source was Accelerometer Ex: If the wakeup source is Ignition, then the value will be ***0x40***.
Return	Refer MCU Specific Return Codes section.

Example	int ret =0;
	uint8_t wake_source = 0;
	ret = MCU_Wakeup_Source_Request(&wake_source);

2.22.3 MCU_FW_Version_Read_Request

MCU_FW_Version_Read_Request	
Description	This API is used for reading the MCU Firmware Version.
Syntax	int MCU_FW_Version_Read_Request (char *mcu_fw_version)
Arguments	char * mcu_fw_version – Character pointer to hold the MCU Firmware version.
Return	Refer

	MCU Specific Return Codes section.

Example	int ret = 0;
	char mcu_fw_version [50] = {0};
	ret = MCU_FW_Version_Read_Request(mcu_fw_version);

2.23 MCU Specific Return Codes

Return Codes	Description
0	Success
-1	Failure
F3110001	CPU to MCU i2c write error.
F3110002	CPU to MCU i2c read error.
F3210001	Buffer Overflow error.
F3210002	Null data error.
F3210003	Memory Allocation error.
F3210004	Memory Write error.
F3310001	Timer Sleep Wakeup Error.
F3310002	Ignition Sleep Wakeup Error.
F3310003	Accelerometer Sleep Wakeup Error.
F3310004	CAN Sleep Wakeup Error.
F3410001	MCU Invalid Response ID Error.
F3410002	MCU Invalid Sleep Mode Error.
F3410003	MCU Invalid Wakeup Source Error.
F3410004	MCU Invalid arguments Error.
F3410005	MCU Invalid FW Version Error.

Note: MCU related APIs may return some Standard errors.

3 General Return Codes

Error Number	Description
0	Success
-1	Failure
-2	Invalid argument passed
100A	Invalid baudrate passed
F0010001	Invalid GPIO number
F0010002	GPIO export file open error
F0010003	GPIO export file write error
F0010004	GPIO direction file open error
F0010005	GPIO direction file write error
F0010006	No access to GPIO direction file
F0010007	GPIO value file open error - to read value
F0010008	GPIO value file read error
F0010009	No access to GPIO value file - to read value
F0010010	GPIO value file open error - to write value
F1010001	GPIO value file write error
F1010002	No access to GPIO value file - to write value
F1010003	GPIO read file open error
F1010004	Invalid GPIO read
F1010005	No access to GPIO file
F1010006	GPIO event number read error
F2010001	Invalid serial port
F2010002	Serial port initialization error
F2010003	Serial port write error
F2010004	Serial port read error
F2010005	Serial port baud rate set error
F2010006	Serial port deinitialization error
F3010001	Invalid i2c bus
F3010002	I2c file open error
F4010001	Error in reading MAC address
F4010002	Error in accessing interface
F4010003	Error reading the file

4 Structures

This chapter describes structures in the Telematics Gateway library.

4.1_accelerometer_api_priv

```
typedef struct _accelerometer_api_priv {
  int fd;
  double x, y, z, acc;
} accelerometer_api_priv;
```

This structure holds the accelerometer data,

fd = accelerometer device node

x = x-axis value

y = y-axis value

z = z - axis value

4.2_gyroscope_api_priv

```
typedef struct _gyroscope_api_priv {
double x, y, z;
} gyroscope_api_priv;
```

This structure holds the gyroscope data,

x = x axis value

y = y axis value

z = z axis value

4.3_magnetometer_api_priv

```
typedef struct _magnetometer_api_priv {
double x, y, z;
} magnetometer_api_priv;
```

This structure holds the magnetometer data,

x = x axis value

y = y axis value

z = z axis value

4.4i2c_cmd_frame_struct

```
typedef struct i2c_cmd_frame_struct
{
uint8_t i2c_addr;
uint8_t reg;
uint8_t command_id;
uint8_t length;
uint8_t *data;
} i2c_cmd_frame;
```

This structure holds the i2c command information to be framed,

i2c_addr - MCU slave address

reg - Register to write the i2c data

command_id - Command ID for the i2c command.

length - Length of the data

*data - Data with “length” number of bytes

4.5i2c_cmd_decode_struct

```
typedef struct i2c_cmd_decode_struct
{
uint8_t response_id;
uint8_t err_code[4];
uint8_t length;
uint8_t *data;
```

} i2c_cmd_decode;

This structure holds the decoded i2c frame data,

response_id - The response ID from the MCU

err_code[4] - 4bytes of return code from the MCU

length - Length of the data

*data - Data with “length” number of bytes.

5 API Sequence

Please follow the below sequence in application, while calling the Telematics Gateway library APIs.

- init()
- gsm_modem_on
- agps_init
- gps_init
- can_init
-

5.1 Sequence to be followed if 0 is passed to init ()

5.1.1 gsm_modem_on ()
5.1.2 check_gsm_modem_status ()
5.1.3 establish_connection ()
5.1.4 check_network_connection ()
5.1.5 check_gsm_nw_connection ()
5.1.6 get_gsm_signal_strength ()
5.1.7 get_gsm_nw_reg ()

5.2 Sample Application Source:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "common.h"
#include "gps.h"
#include <fcntl.h>
#include <termios.h>
#include <signal.h>
#include <stdint.h>
#include <linux/netlink.h>
#include <linux/if_link.h>
#include <linux/input.h>
#include <linux/rtnetlink.h>
#include <linux/can.h>
#include <netdb.h>
#include <math.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <unistd.h>
#include <errno.h>
#include "gsm.h"
#include "error_nos.h"
#include "can.h"
#include "gyroscope.h"
#include "accelerometer.h"
                #include "rs232.h"
```

```
#include "rs485.h"
#include <time.h>
#include "battery.h"

#define ENABLE 1
#define DISABLE 0
#define WIFI_HOSTAPD_MODE 1
#define WIFI_STATION_MODE 2
#define CAN_250 250000
#define CAN_500 500000

#define TIME_STAMP      1
#define CUR_LATITUDE    2
#define HEMISPHERE      3
#define CUR_LONGITUDE   4
#define GREENWICH       5
#define FIX_STATUS      7
#define ALTITUDE        9
#define GEOID          11

#define UART_PORT "/dev/ttyLP2"
#define UART_BAUDRATE 9600

int ParseRMC(struct gps_rmc_t *gps_rmc, size_t nbytes)
{
int j,k,l;
char field[20][100] = {{0}};
static double prev_lat = 0.0, prev_lon = 0.0;

if (gps_rmc == NULL || nbytes <= 0)
        return -1;

printf ("ParseRMC rmc_nmea %s \n",gps_rmc->nmea);
for(j=0,k=0,l=0;j<nbytes;j++,l++)
{
        for(;(gps_rmc->nmea[j] != ',') && (gps_rmc->nmea[j] != '\0');k++,j++)
        {
                field[l][k] = gps_rmc->nmea[j];
                if(l > 16)
                        break;
        }
        field[l][k] = '\0';
        k=0;
}

if(*field[RMA_FIX_STATUS] != 'A'){
        printf("\nGPS Not Valid\n");
        gps_rmc->latitude = prev_lat;
        gps_rmc->longitude = prev_lon;
        gps_rmc->gps_valid = 0;
}else{
        printf("GPS is Valid!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! \n");
```

```
	gps_rmc->hour = atoi(field[TIME_STAMP]) / 10000;
	gps_rmc->minutes = (atoi(field[TIME_STAMP])/100) % 100;
	gps_rmc->seconds = atoi(field[TIME_STAMP]) % 100;

	gps_rmc->latitude = (fmod(atof(field[RMA_CUR_LATITUDE]),100.00)/60) +
(atoi(field[RMA_CUR_LATITUDE])/100);
	prev_lat = gps_rmc->latitude;
	if (*field[RMA_HEMISPHERE] != 'N')
		gps_rmc->latitude = -gps_rmc->latitude;

	gps_rmc->longitude = (fmod(atof(field[RMA_CUR_LONGITUDE]),100.00)/60 ) +
(atoi(field[RMA_CUR_LONGITUDE])/100);
	prev_lon = gps_rmc->longitude;
	if (*field[RMA_GREENWICH] != 'E')
		gps_rmc->longitude = -gps_rmc->longitude;

	gps_rmc->speed = atof(field[RMA_KNOT_SPEED]);

	gps_rmc->direction = atof(field[RMA_DIRECTION]);

	gps_rmc->gps_valid = 1;
	printf ("speed is %f knots\n",gps_rmc->speed);
	gps_rmc->speed = gps_rmc->speed * 1.852;
	printf ("gps_rmc->latitude : %f gps_rmc->longitude: %f  gps_rmc->speed : %fkm/hr gps_rmc-
>direction:%f deg\n",gps_rmc->latitude,gps_rmc->longitude, gps_rmc->speed, gps_rmc->direction);
}
return gps_rmc->gps_valid;
}


int ParseGGA(struct gps_gga_t *gps_gga, size_t nbytes)
{
int j = 0;
int k = 0;
int l = 0;
int ret = 0;
char field[15][100] = {{0}};
if (gps_gga == NULL || nbytes <= 0)
{
	ret = -1;
}
else
{
	for(j=0,k=0,l=0;j<nbytes;j++,l++)
	{
		for(;(gps_gga->nmea[j] != ',') && (gps_gga->nmea[j] != '\0');k++,j++)
		{
			field[l][k] = gps_gga->nmea[j];
		}
		field[l][k] = '\0';
		k=0;
```

```
        }
        gps_gga->altitude = atof(field[ALTITUDE]);
        gps_gga->geoid = atof(field[GEOID]);
}
printf ("Altitude %f Geoid %f \n",gps_gga->altitude, gps_gga->geoid);
return ret;
}

/ Printing GPS read data /
int gps_data_ap(char buf[200], size_t nbytes)
{
int j,k,l;
char field[15][100] = {0};

printf("\nGPS-DATA[");
for(j=0;j<nbytes;j++)
{
        printf("%c", buf[j]);
}
printf("]\n");

return 0;
}

int main()
{
int rc = 0;
int sim_reg_count = 0;
char arr_sig_strngth[100] = {0};
char cell_id[50] = {0},lac[30] = {0};
int sig_strength = 0;
int arr_sig_strength = 0;
int network_link_count = 0;
int network_connection_count = 0;
int ntp_server_update_count = 0;
int link_status = 0;
char iccid[ 64 ] = " ";
char buf[ 100 ] = " ";
char can_name[10] = {};

char resp_buffer[ 1024 ];
char imei[100] = {0};
char recv_data[200] = {0};
int count = 0;
size_t len = 0;
int n = 0;
int i = 0;
int battery_chrg_status = 0;
double battery_voltage = 0;
 double i_bat_volt;
accelerometer_api_priv g_adata;
gyroscope_api_priv g_data;
```

```
int cpuid_length = 40;
char cpu_id[16] = {0};

char can_id[10] = {0};
char can_data[10] = {0};
char can_inf[5] = {0};
int ret = 0;
struct canfd_frame frame;
struct gps_rmc_t gps_rmc;
struct gps_gga_t gps_gga;
int Serial_Port_Number=0;
int rs232_fd;
int rs485_fd;
long int rs232_write_data;
long int rs232_read_data;
long int rs485_write_data;
long int rs485_read_data;
char buf_rs232[1048576] = {0};
char buf_rs485[1048576] = {0};
char buf1[1048576];
char buf2[1048576]="ABCDEFGHIJ";
int rs232_count;
int rs485_count;
int state = 1;
int con_status=1;


printf("\n#############[INIT]###############\n");
rc = init(0);
if(rc == -1)
        printf("init() failed\n");
else
        printf("**********************init() done\n");

rc = i_battery_init( );
     printf("i_battery_init rc value %x\n", rc );

     rc = i_battery_get_health();
     printf("i_battery_get_helath rc value %x\n",rc);


     rc= i_battery_get_voltage(&i_bat_volt);
      printf("i_battery_get_voltage rc value %x\n",rc);
printf("the battery voltage is i_battery_get_voltage = %f\n",i_bat_volt);

      rc = battery_charge_state_config(state);
      printf("battery_charge_state_config rc value %x\n",rc);


rc= get_power_source();
printf("get_power_source rc value %x\n", rc);
```

```
	rc=i_get_battery_status(&battery_chrg_status);
	printf("i_get_battery_status rc value %x\n", rc);
	printf("the battery status is batter_chrg_status=%x\n",battery_chrg_status);

rc = battery_connect_config(con_status);
 printf("battery_connect_config rc value %x\n", rc);

rc= eth_init(“eth0”);
printf("eth_init rc value %x\n",rc);

rc = eth_deinit(“eth0”);
printf("eth_deinit rc value %x\n",rc);

memset( cpu_id, 0, sizeof( cpu_id ) );
rc = get_cpu_id(cpu_id);
printf("get_cpu_id rc value %x\n", rc );
printf("get_cpu_id buffer %s\n", cpu_id );
printf("\n#############[GSM Modem]###############\n");

/ checking whether modem is on or off /
rc = check_gsm_modem_status();
if(rc != 0)
{
	/ Turning on the GSM modem /
	rc = gsm_modem_on( "0000", 4 );
	printf("gsm_modem_on() Return value = %x\n",rc);
}
printf("\nLINE %d\n", __LINE__);

/ Getting the IMEI number of the GSM module. /
memset( imei, 0, sizeof( imei ) );
rc = get_gsm_imei(imei, sizeof( imei ) );
printf("imei[%s] rc = %x\n", imei, rc);

/ Getting the ICCID of the GSM module /
memset( iccid, 0, sizeof( iccid ) );
rc = get_gsm_sim_iccid(iccid, sizeof(iccid));
printf("iccid[%s] = %x\n", iccid, rc);

/ Getting the signal range details of the network /
memset( arr_sig_strngth, 0, sizeof( arr_sig_strngth ) );
rc = get_gsm_signal_strength(arr_sig_strngth, sizeof( arr_sig_strngth ));
printf("gsm signal[%s] = %x\n",arr_sig_strngth, rc);

/ Getting the SIM registration details /
memset( cell_id, 0, sizeof( cell_id ) );
memset( lac, 0, sizeof( lac ) );
rc = get_gsm_nw_reg(cell_id, sizeof( cell_id ), lac, sizeof( lac ) );
printf("CELL_ID %s LAC %s\n",cell_id,lac);
#if 1
while (1){
	rc = establish_connection();
```

```
        if(rc == 0) {
                rc = check_network_connection();
                if (rc == 0 )
                {
                        rc = check_gsm_nw_connection();
                        break;
                }
                else{
                        printf("Trying to Connect!!!!\n");
                        sleep(1);
                        if (count > 1){
                                printf("Network Connection Not Established\n");
                                break;
                        }
                }
        }
        else
        {
                sleep(1);
                if (count > 10){
                        printf("Network Connection Not Established\n");
                        break;
                }
        }
        count++;
}

/ Fetching latest xtradata from the server /

//      rc = agps_init();
//      printf("quectel_agps_init() Return value = %x\n",rc);
#endif

/ Initializing gps /
rc = gps_init( );
printf("gps_init() Return value = %x\n", rc );

/ Reading the gps data from the OBDII device /
memset( &gps_rmc, 0, sizeof( gps_rmc ) );
len = 0;

rc = get_gps_data("GPRMC", &len, gps_rmc.nmea, sizeof( gps_rmc.nmea ));
printf("get_gps_data rc value %x\n", rc);
printf("get_gps_data len value %d\n", len);
printf("get_gps_data gps_rmc.nmea value %s\n", gps_rmc.nmea);

ParseRMC(&gps_rmc, len);
memset( &gps_gga, 0, sizeof( gps_gga ) );
len = 0;

rc = get_gps_data("GPGGA", &len, gps_gga.nmea, sizeof( gps_gga.nmea ));
printf("get_gps_data GPGGA rc value %x\n", rc);
```

```
printf("get_gps_data len GPGGA value %d\n", len);
printf("get_gps_data gps_gga.nmea value %s\n", gps_gga.nmea);

rc = ParseGGA(&gps_gga, len);
sleep( 1 );

rc = can_init("can0", 250000);
printf("can_init return value = %x\n",rc);
rc= can_write("can0", "7e0#02010C0000000000");
printf("can_write() Return value = %x\n",rc);


rc = can_init("can1", 250000);
      printf("can_init return value = %x\n",rc);
 rc= can_write("can1", "7e0#02010C0000000000");
      printf("can_write() Return value = %x\n",rc);

rc = can_init("can2", 250000);
      printf("can_init return value = %x\n",rc);
 rc= can_write("can2", "7e0#02010C0000000000");
      printf("can_write() Return value = %x\n",rc);


rc = can_init("can3", 250000);
      printf("can_init return value = %x\n",rc);
 rc= can_write("can3", "7e0#02010C0000000000");
      printf("can_write() Return value = %x\n",rc);


sleep(2);
rc = can_deinit( "can0" );
printf("can_deinit() Return value = %x\n",rc);

sleep(2);
      rc = can_deinit( "can1" );
      printf("can_deinit() Return value = %x\n",rc);


sleep(2);
      rc = can_deinit( "can2" );
      printf("can_deinit() Return value = %x\n",rc);


sleep(2);
      rc = can_deinit( "can3" );
      printf("can_deinit() Return value = %x\n",rc);


rc = wifi_init( 1 );
printf("wifi_init() Return value = %x\n",rc);

rc = wifi_deinit();
```

```
printf("wifi_deinit() Return value = %x\n",rc);

rc = ble_init();
printf("ble_init() Return value = %x\n",rc);

rc = ble_deinit();
printf("ble_deinit() Return value = %x\n",rc);

rc = set_gsm_flight_mode_on();
printf("set_gsm_flight_mode_on() return value %x\n",rc);
sleep(2);

rc = set_gsm_flight_mode_off();
printf("set_gsm_flight_mode_off() return value %x\n",rc);

/ De-initializing gps /
rc = gps_deinit( );
printf("gps_deinit rc value 0x%x\n", rc );
sleep(1);
rc = gsm_modem_off();
printf("gsm_modem_off rc value 0x%x\n", rc );
#if 0
printf("\n#############[Accelerometer]###############\n");
/          Initializing accelerometer /
rc = acc_init();
printf("\nAccelerometer Initialisation return value = 0x%x\n",rc);
n = 0;
while(1)
{
        n = n+1;
        / Reading the accelerometer values from the OBDII device /
        accelerometer_read(&g_adata);
        printf ("Acc x-axis: %f\ty-axis: %f\tz-axis: %f\n",g_adata.x,g_adata.y,g_adata.z);
        if(n >= 10)
                break;
}
#endif


//printf("\n#############[Gyroscope]###############\n");
/ Initializing gyroscope /

#if 0
rc = gyro_init();
printf("\n Gyroscope Initialisation return value = 0x%x\n",rc);
n =0;
while(1)
{
        n = n+1;
        / Reading the gyroscope values from the OBDII device /
        gyroscope_read(&g_data);
        printf ("Gyro x-axis: %f\ty-axis: %f\tz-axis: %f\n",g_data.x,g_data.y,g_data.z);
```

```
        if(n >= 10)
                break;
}

/ De-initializing gyroscope /
rc = gyro_deinit();
printf("gyro_deinit rc value %x\n", rc );
#endif

/       De-initializing accelerometer /
//rc = acc_deinit();
/       printf("acc_deinit rc value %x\n", rc ); /

/ RS232 Initialization /
memset(buf1,0,sizeof(buf1));
rs232_fd = rs232_init( UART_BAUDRATE );
if(rs232_fd < 0)
{
        printf("Error rs232_init\n");
}
else
{
        printf("RS232_init Success\n");
}
sleep(1);

/ RS232 WRITE /
rs232_write_data = rs232_write(buf2, 11);
if(rs232_write_data < 0)
{
        printf("Error in writing RS232 Data=%d\n", rs232_write_data);
}
else
{
        printf("RS232 Write Success \n");
        printf("RS232 Write Return value=%d\n",rs232_write_data);
}
sleep(1);

/ RS232 READ /
rs232_read_data = rs232_read(buf1, 11);
sleep(1);
printf("RS232 Read Data: %s\r\n",buf1);
if(rs232_read_data < 0)
{
        printf("Error in reading RS232 Data = %x\n", rs232_read_data);
}
else
{
        printf("RS232 READ Success\n");
}
```

```
/* RS232 De-initialization*/
rc = rs232_deinit();
printf("RS232 Deinit Return value %d\n", rc );

/         RS485 Initialization /
rs485_fd = rs485_init( UART_BAUDRATE );
if(rs485_fd < 0)
{
        printf("Error rs485_init\n");
}
else
{
        printf("RS485_init Success\n");
}
sleep(1);

/         RS485 WRITE /
rs485_write_data = rs485_write(buf2, 11);
if(rs485_write_data< 0)
{
        printf("Error in writing RS485 Data = %d\n", rs485_write_data);
}
else
{
        printf("RS485 Write Success\n");
        printf("RS485 Write Return value = %d\n", rs485_write_data);
}
sleep(1);

/ RS485 READ /
rs485_read_data = rs485_read(buf1, 11);
printf("RS485 Read Data: %s\r\n",buf1);
if(rs485_read_data < 0)
{
        printf("Error in reading RS485 Data=%x\n", rs485_read_data);
}
else
{
        printf("RS485 READ Success\n");
}
sleep(1);

/         RS485 De-initialization /
rc = rs485_deinit();
printf("RS485 Deinit Return value %d\n", rc );


printf("\n#############[DEINIT]###############\n");
rc = deinit();
if(rc == -1)
        printf("deinit() failed\n");
else
```

```
        printf("**********de_init() done\n");

    }
```

5.3 MCU Sleep Source:

```
#include "mcu_i2c_common.h"
#include "accelerometer.h"
#include "battery.h"
#include "common.h"
#include "error_nos.h"
#include "can.h"
#include "gps.h"
#include "gsm.h"

#define I2C_TEST 0
#define RTC_NODE_PATH "/sys/class/rtc/rtc1/wakealarm"

/* Debug Prints Enable */
#define COMMON_DEBUG_PRINTS 1
#define DEBUG_EN 0
#if DEBUG_EN
#define DEBUG_PRINTS 1
#define DEC_TO_HEX_DEBUG_EN 1
#define MCU_I2C_BUF_FRAME_DEBUG_EN 1
#define MCU_I2C_READ_DEBUG_EN 1
#define MCU_I2C_WRITE_DEBUG_EN 1
#endif

char CAN[6];
int external_rtc_flag = 1;
/*
 * API              : prepare_sleep_mode()
 * Description      : API to disable all the interfaces before entering the Sleep Mode to \
 *                  : reduce the current consumption.
 * Arguments        : None
 * Return Value     : 0 for success and error code for failure.
 * */
int prepare_sleep_mode()
{
        int ret = OBD2_LIB_FAILURE;
        ret = check_gsm_modem_status();
        if (ret == OBD2_LIB_SUCCESS)
        {
                /* Turning off teh GPS */
                ret = gps_deinit();
                /* Turning Off the GSM modem */
                ret = gsm_modem_off();
        }

        /* Deinitialize the WiFi */
        ret = wifi_deinit();
```

```
	/* Deinitialize the Bluetooth */
	ret = ble_deinit();

	/* Deinitialize the interfaces */
	// ret = deinit();

	return ret;
}

/*
 * API		: enable_requested_wakeup_sources( uint8_t wakeup_source, uint32_t *timer )
 * Description	: API to enable the requested wakeup sources for MCU sleep wakeup. Other \
 *		: wakeup sources will be disabled.
 * Arguments	: uint8_t *wake_source - Variable to store the Wakeup source
 * Return Value	: 0 for success and error code for failure.
 * */
int enable_requested_wakeup_sources(uint8_t wakeup_source, uint32_t *timer)
{
	int ret = OBD2_LIB_SUCCESS;
#if 1
	if (access(RTC_NODE_PATH, F_OK) == 0)
	{
		external_rtc_flag = 1;
	}
	else
	{
		external_rtc_flag = 0;
	}
#endif
	/* For safer side, we are disabling the CAN0 and CAN1 wakeup */
	ret = config_can_wakeup(CAN0, DISABLE);
	if (ret == OBD2_LIB_SUCCESS)
	{
		ret = config_can_wakeup(CAN1, DISABLE);
		if (ret == OBD2_LIB_SUCCESS)
		{
			// Do Nothing
			ret = config_can_wakeup(CAN2, DISABLE);
			if (ret == OBD2_LIB_SUCCESS)
			{
				// Do Nothing
			}
			else
			{
				// Do Nothing
			}
		}
		else
		{
			// Do Nothing
		}
```

```
	}
	else
	{
		// Do Nothing
	}

	if (ret == OBD2_LIB_SUCCESS)
	{
		/* Check for the Ignition Interrupt */
		if (wakeup_source & CPU_MCU_IGN_WAKEUP_REQUEST_BIT_MASK)
		{
			ret = config_ignition_wakeup(ENABLE);
#if DEBUG_PRINTS
			printf("%s : config_ignition_wakeup() enable returned |%d|\n", __func__, ret);
#endif
		}
		else
		{
			ret = config_ignition_wakeup(DISABLE);
#if DEBUG_PRINTS
			printf("%s : config_ignition_wakeup() disable returned |%d|\n", __func__, ret);
#endif
		}

		if (ret == OBD2_LIB_SUCCESS)
		{
			/* Check for the Accelerometer Interrupt */
			if (wakeup_source & CPU_MCU_ACC_WAKEUP_REQUEST_BIT_MASK)
			{
				ret = config_acc_wakeup(ENABLE);
#if DEBUG_PRINTS
				printf("%s : config_acc_wakeup() enable returned |%d|\n", __func__, ret);
#endif
			}
			else
			{
				ret = config_acc_wakeup(DISABLE);
				if (ret == OBD2_LIB_SUCCESS)
				{
					//			ret = acc_deinit();
				}
#if DEBUG_PRINTS
				printf("%s : config_acc_wakeup() disable returned |%d|\n", __func__, ret);
#endif
			}

			if (ret == OBD2_LIB_SUCCESS)
			{
				/* Check for the CAN FD Interrupt */
				if (wakeup_source & CPU_MCU_CAN_WAKEUP_REQUEST_BIT_MASK)
				{
					ret = config_can_wakeup(CAN, ENABLE);
```

```
#if DEBUG_PRINTS
                                        printf("%s : config_can_wakeup() enable returned |%d|\n",
__func__, ret);
#endif
                                }
                                else
                                {
                                        ret = config_can_wakeup(CAN3, DISABLE);
                                        ret = config_can_wakeup(CAN4, DISABLE);
#if DEBUG_PRINTS
                                        printf("%s : config_can_wakeup() disable returned |%d|\n",
__func__, ret);
#endif
                                        //                                              ret =
!access(CAN3_WAKEUP_FILE_NAME, F_OK) ? config_can_wakeup(CAN3, DISABLE) : ret;
                                        //                                              ret =
!access(CAN4_WAKEUP_FILE_NAME, F_OK) ? config_can_wakeup(CAN4, DISABLE) : ret;
                                        // #if DEBUG_PRINTS
                                        //                                              printf("%s :
config_can_wakeup() disable returned |%d|\n", __func__, ret);
                                        // #endif
                                }

                                if (ret == OBD2_LIB_SUCCESS)
                                {
                                        /* Check for the Timer Interrupt */
                                        if (wakeup_source &
CPU_MCU_TIMER_WAKEUP_REQUEST_BIT_MASK)
                                        {
                                                ret = config_timer_wakeup(ENABLE, *timer);
#if DEBUG_PRINTS
                                                printf("%s : config_timer_wakeup() disable returned
|%d|\n", __func__, ret);
#endif
                                        }

                                        /* Support for the RTC Interrupt */
                                        else if (wakeup_source &
CPU_MCU_RTC_WAKEUP_REQUEST_BIT_MASK)
                                        {

                                                if (external_rtc_flag)
                                                {
#if DEBUG_PRINTS
                                                        printf("External RTC detected. Enabling the RTC
wakeup\n");
#endif

                                                        ret = config_rtc_wakeup(ENABLE, *timer);

#if DEBUG_PRINTS
                                                        printf("%s : config_rtc_wakeup() enable returned
|%d|\n", __func__, ret);
#endif
```

```
                                        }
                                        else
                                        {
                                                ret = E_IF_INVALID;
                                                printf("%s : RTC Not found. Error Code is |%x|\n",
__func__, ret);
                                        }
                                }
                                else
                                {
                                        if (external_rtc_flag)
                                        {
#if DEBUG_PRINTS
                                                printf("External RTC detected. Disabling the RTC
wakeup\n");
#endif
                                                //                                              ret
= config_rtc_wakeup( DISABLE, *timer );
                                                printf("the rtc ret value is ret=%d\n", ret);
#if DEBUG_PRINTS
                                                printf("%s : config_rtc_wakeup() disable returned
|%d|\n", __func__, ret);
#endif
                                        }
                                        else
                                        {
                                                // Do Nothing
                                        }

//                                      ret = config_timer_wakeup( DISABLE, DISABLE );
#if DEBUG_PRINTS
                                        printf("%s : config_timer_wakeup() disable returned
|%d|\n", __func__, ret);
#endif
                                }

                                /*              if( ret == OBD2_LIB_SUCCESS )
                                                {
                                                 Enable the MCU GPIO interrupt for wakeup
                                                ret = config_mcu_wakeup( ENABLE );
                        #if DEBUG_PRINTS
                                                printf("%s : config_mcu_wakeup() enable
returned |%d|\n", __func__, ret);
                        #endif
                                                }
                                                else
                                                {
                                                // Do Nothing
                                                }*/
                                }
                                else
                                {
```

```
                                        // Do Nothing
                                }
                        }
                        else
                        {
                                // Do Nothing
                        }
                }
                else
                {
                        // Do Nothing
                }
        }
        else
        {
                // Do Nothing
        }

        return ret;
}

int main()
{
        int ret = 0, i = 0;
        int choice = 0;
        int adc = 0;
        float voltage = 0;
        uint8_t sleep_mode = 0;
        uint8_t wakeup_source = 0;
        uint8_t wakeup_choice = 0;
        char mcu_fw_version[35] = {0};
        int timer = 0;
        int bitrate = 0;
        int dbitrate = 0;
        int txqueuelen = 250000;
        uint32_t bat_monitor_interval = 0;
        float_t temperature;

        i2c_cmd_frame i2c_frame_buf;
        i2c_cmd_decode i2c_decode_buf;

        uint32_t bus_num = 0;
        uint8_t slave_addr = 0x76;
        uint8_t data_addr = 0x02;
        uint8_t i2c_frame[256] = {0};

        FILE *fp = NULL;
        char buffer[120];
        char can_buf[120] = {0};
        int can_is_up = 0;

        i = 1;
```

```
		printf("\n\t%d. MCU Sleep Request\n\t%d. MCU Sleep Source Request\n\t%d. MCU Firmware
Version\n\t%d. MCU ADC Read\n\n",
				i, (i + 1), (i + 2), (i + 3));
		printf("Enter the choice : ");
		ret = scanf("%d", &choice);
		switch (choice)
		{
		case 1:
			/* Sleep Request */
			printf("\t\t1. Normal Sleep\n\t\t2. Deep Power Down\n\n");
			fflush(stdin);
			printf("Enter the sleep_mode: ");
			ret = scanf(" %hhx", &sleep_mode);

			switch (sleep_mode)
			{
			case 1:
				sleep_mode = 1;
				break;
			case 2:
				sleep_mode = 4;
				break;
			default:
				sleep_mode = 0;
				printf("Enter valid Sleep mode\n\n");
				break;
			}

			if (sleep_mode > OBD2_LIB_SUCCESS)
			{
				i = 1;
				fflush(stdin);

				if (sleep_mode == 4)
				{
					printf("\t\t%d. RTC Wakeup\n\t\t%d. Ignition\n\t\t%d.
Accelerometer\n\t\t%d. Custom Wakeup sources\n",
						i, (i + 1), (i + 2), (i + 3));

					printf("Enter the wakeup_source: ");
					ret = scanf(" %hhx", &wakeup_choice);

					if (wakeup_choice == 1)
					{
						wakeup_choice = 2;
					}
					else if (wakeup_choice == 2)
					{
						wakeup_choice = 3;
					}
					else if (wakeup_choice == 3)
					{
```

```
                        wakeup_choice = 4;
                }
                else if (wakeup_choice == 4)
                {
                        wakeup_choice = 6;
                }
        }
        else
        {
                printf("\t\t%d. Timer\n\t\t%d. RTC Wakeup\n\t\t%d. Ignition\n\t\t%d.
Accelerometer\n\t\t%d. CAN FD\n\t\t%d. Custom Wakeup sources\n",
                                i, (i + 1), (i + 2), (i + 3), (i + 4), (i + 5));

                printf("Enter the wakeup_source: ");
                ret = scanf(" %hhx", &wakeup_choice);
        }

        switch (wakeup_choice)
        {
        case 1:
                wakeup_source = CPU_MCU_TIMER_WAKEUP_REQUEST_BIT_MASK;
                break;
        case 2:
                wakeup_source = CPU_MCU_RTC_WAKEUP_REQUEST_BIT_MASK;
                break;
        case 3:
                wakeup_source = CPU_MCU_IGN_WAKEUP_REQUEST_BIT_MASK;
                break;
        case 4:
                wakeup_source = CPU_MCU_ACC_WAKEUP_REQUEST_BIT_MASK;
                break;
        case 5:
                wakeup_source = CPU_MCU_CAN_WAKEUP_REQUEST_BIT_MASK;
                break;
        case 6:
                fflush(stdin);
                printf("Enter the combination of wakeup sources: ");
                ret = scanf(" %hhx", &wakeup_source);
                break;
        default:
                wakeup_source = 0x0;
                printf("Enter valid Wakeup Source\n\n");
                break;
        }

        if ((wakeup_source & CPU_MCU_TIMER_WAKEUP_REQUEST_BIT_MASK) ||
(wakeup_source & CPU_MCU_RTC_WAKEUP_REQUEST_BIT_MASK))
        {
                fflush(stdin);
                printf("Enter the timer for wakeup: ");
                ret = scanf("%d", &timer);
        }
```

```
                else
                {
                        // Do Nothing
                }

                if (wakeup_source & CPU_MCU_CAN_WAKEUP_REQUEST_BIT_MASK)
                {
                        fflush(stdin);
                        printf("Enter the Speed for CAN FD interface: ");
                        ret = scanf(" %d", &bitrate);
                        printf("Enter the DBitrate for CAN FD interface: ");
                        ret = scanf(" %d", &dbitrate);

                        printf("Enter the can for wakeup source(can3/can4): ");
                        ret = scanf(" %s", CAN);

                        if (system("ifconfig | grep can3") || system("ifconfig | grep can4"))
                        {
                                ret = can_fd_init(CAN, bitrate, dbitrate, txqueuelen);
                        }
                }
                else
                {
                        // Do Nothing
                }
                if ((wakeup_source > 0x0) && !((wakeup_source &
CPU_MCU_TIMER_WAKEUP_REQUEST_BIT_MASK) && (wakeup_source &
CPU_MCU_RTC_WAKEUP_REQUEST_BIT_MASK)))
                {
                        /*
                         * API to disable all the interfaces before entering the \
                         * Sleep Mode to reduce the current consumption.
                         * */
                        ret = prepare_sleep_mode();
                        /*
                         * Enable the requested Wakeup Sources only. The wakeup \
                         * sources that are not requested will be disabled.
                         * */
                        ret = enable_requested_wakeup_sources(wakeup_source, &timer);

                        if (ret == OBD2_LIB_SUCCESS)
                        {
                                ret = MCU_sleep_mode(sleep_mode, wakeup_source);
                                if ((ret == OBD2_LIB_SUCCESS) && ((wakeup_source &
CPU_MCU_CAN_WAKEUP_REQUEST_BIT_MASK)))
                                {
                                        ret = can_deinit(CAN3);
                                        ret = can_deinit(CAN4);
                                }
                                else
                                {
                                        // Do Nothing
```

```
                                        }
                                        ret = eth_init("eth0");
                                }
                                else
                                {
                                        // Do Nothing
                                }
                        }
                        else
                        {
                                ret = INVALID_ARGS;
                        }
                }
                else
                {
                        // Do Nothing
                }
                break;
        case 2:
                /* Wakeup Source Request */
                wakeup_source = 0;
                ret = MCU_Wakeup_Source_Request(&wakeup_source);
                printf("The MCU wakeup source is |%x|\n\n", wakeup_source);
                break;
        case 3:
                /* MCU Firmware Version Read */
                ret = MCU_FW_Version_Read_Request(mcu_fw_version);
                if (ret == OBD2_LIB_SUCCESS)
                        printf("The MCU Firmware Version is |%s|\n\n", mcu_fw_version);
                break;
        case 4:
                printf("\n\t1. 12V ADC\n\t2. Battery ADC\n\t3. ADC1\n\t4. ADC2\nEnter the ADC to be read:
");
                ret = scanf(" %d", &adc);
                ret = MCU_ADC_Read_Request(adc, &voltage);
                if (ret == OBD2_LIB_SUCCESS)
                        printf("The MCU ADC Voltage is |%f|\n\n", voltage);
                break;
        default:
                ret = OBD2_LIB_FAILURE;
                printf("Not implemented\n\n");
                break;
        }

        if (ret == OBD2_LIB_SUCCESS)
        {
                printf("mcu_test_app : SUCCESS with return |%x|\n", ret);
        }
        else
        {
                printf("mcu_test_app : FAILURE with return |%x|\n", ret);
        }
```

```
        return ret;
}
```

6 Application Development

To develop an application for Telematics Gateway, please follow the below steps,

Headers

- Include the *common.h, gps.h, can.h, accelerometer.h, gyroscope.h, error_nos.h*, *gsm.h*, *rs232.h*, *rs485.h* and *debug.h* header files to the application.

Library

- Link the libTelematics_GW.so library to the application using "*-l libTelematics_GW.so*" option during compilation.

Cross_Compiler

Follow the below steps to compile the Application for Telematics Gateway.

- Cross Compiler can be found in below path.
  /<path to iW-PRGOT-DF-01-RX.Y-RELX.Y-6.1.22>/iW-PRGOT-DF-01-RX.Y-RELX.Y-6.1.22/Cross-Compiler/fsl-imx-xwayland-glibc-x86_64-core-image-minimal-armv8a-imx8dxla1-iwg46m-2gb-toolchain-6.1-mickledore.sh
- Run the cross-compiler script and enter the target directory to install SDK
  - Note: Default SDK directory is */opt/fsl-imx-xwayland/6.1-mickledore*
- Include Cross-Compiler headers to your application by adding below options in your *Makefile*

  -I /opt/fsl-imx-xwayland/6.1-mickledore/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/
- Copy the released libTelematics_GW.so library file to Application directory

  $ cp libTelematics_GW.so <path to Application folder>
- Link libTelematics_GW.so library to your application by adding below option in your *Makefile*

  -L<Path To libTelematics_GW.so > -l libTelematics_GW.so
- Copy the *lib_telematics_gw.so* library to the below path in the root filesystem in the Telematics Gateway.

  /usr/lib/
- Setup the environment in terminal/shell window before compiling the application using the below command:

  $ source /opt/fsl-imx-xwayland/6.1-mickledore/environment-setup-armv8a-poky-linux

8. TECHNICAL SUPPORT

iWave Systems technical support team is committed to provide the best possible support for our customers so that our Hardware and Software can be easily migrated and used.

For assistance, contact our Technical Support team at,

Email : support.telematics@iwave-global.com

Website : *www.iwavesystems.com*

Address : iWave Systems Technologies Pvt. Ltd.

7/B, 29th Main, BTM Layout 2nd Stage,

Bangalore, Karnataka,

India - 560076

A Global Leader in Embedded Systems Engineering and Solutions

Since 1999, we have pioneered leadership in embedded systems technology, establishing ourselves as a strategic embedded technology partner for advanced solutions. Our comprehensive portfolio encompasses ARM and FPGA System on Modules, COTS FPGA solutions, and ODM solutions which include Telematics, Gateways & HMI Solutions.

Beyond our robust product ecosystem, we provide comprehensive ODM support with specialized custom design and manufacturing capabilities, enabling customers to accelerate and optimize their product development roadmaps. With a strategic focus on industrial, automotive, medical, and avionics markets, we deliver innovative technology solutions to global clients.

mktg@iwave-global.com

iWave
Bangalore, India

iWave USA
Campbell, California

iWave Global
Ras Al Khaimah, UAE

iWave Global GmbH
Ratingen, Germany

iWave Europe
Rotterdam, Netherlands

iWave Japan
Yokohama, Kanagawa

iWave Korea
Gyeonggi,-do, Korea

iWave APAC
Taipei City, Taiwan

iWave Global