Mercury System

# Framework User Manual

IoT and Connectivity Made Simple

Francesco Ficili
31/12/2018

| Revision Log | | | | |
|---|---|---|---|---|
| **Author** | **Date** | **Major** | **Minor** | **Description** |
| Francesco Ficili | 31/12/2018 | 1 | 0 | Initial Release for Framework version 1.0.0. |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# SUMMARY

1. INTRODUCTION ..... 4
2. WHAT THE MERCURY SYSTEM FRAMEWORK IS ..... 5
Framework Description ..... 5
The Framework Functionalities ..... 6
3. THE SYSTEM CONFIGURATION FILE (SYS_CFG.H) ..... 8
4. PML (PERIPHERAL MANAGEMENT LAYER) ..... 9
Modem Stacks ..... 9
WiFi Modem Stack ..... 9
BT Modem Stack ..... 16
GSM/GPRS Modem Stack ..... 17
I2C Stack ..... 20
UART Stack ..... 22
USB ..... 23
5. SSL (SYSTEM SERVICE LAYER) ..... 24
LED ..... 24
RTCM ..... 26
SYSM ..... 29
TERM ..... 31
6. OSL (OPERATIVE SYSTEM LAYER) ..... 32
OS Services ..... 32
OS Timers ..... 34
OS Alarms ..... 37

# 1. Introduction

This manual provides a complete reference guide to the Mercury System Framework. For a complete description of what the Mercury System (MS in short), what you can do with the system and other getting started information please refer to the document MS_GettingStartedGuide. This manual will go deeper in details into the Mercury System Framework (MSF in short) which is the SW framework for the development of applications using Mercury System.

## 2. What the Mercury System Framework is

The Mercury System (MS in short) is a modular system for the development of connectivity and IoT applications. The system uses various type of electronic boards (logic unit, modems, slave boards equipped with sensors and actuators, power boards...) and a complete SW framework to allow the realization of complex applications. Scalability, ease of use and modularity are key factors and are allowed by the use of a heterogeneous set of components that allow to assemble the system like a construction made with LEGO© bricks.

### Framework Description

The Mercury System Framework (MSF) is a layered SW framework specifically designed to support application development with Mercury System. It provides to the user a complete set of base functionalities to easily interface MS Slave Boards (SB) and Modem Boards (MB) as well as some infrastructural and system services. Figure 1 shows the layered Architecture of the MSF.

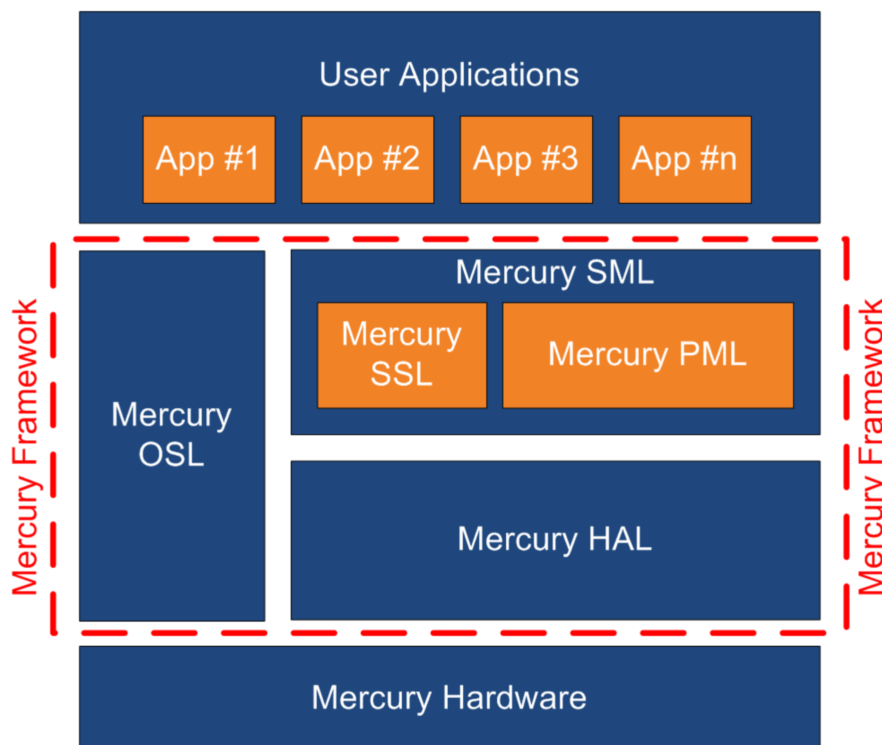


*Figure 1 - Mercury System Framework Architecture*

The framework is made up by the following components:

**HAL (Hardware Abstraction Layer):** the purpose of this layer is to abstract the HW dependencies to the upper layers.

**SML (System Management Layer):** the purpose of this layer is to provide services for the management of communication buses (I2C, UART) and for the management of Mercury System's Modem Board (WiFi, BT, GSM/GPRS). It also provides a set of System Services, like System Power Management, RTCC, USB terminal, etc. It's divided in two main components:

- PML: Peripheral Management Layer,
- SSL: System Services Layer.

**OSL (Operative System Layer):** this layer is made up by a lightweight RTOS that provides basic services to the system, like scheduling tables for the various tasks, Events, SW Timers, Alarms, etc…

## The Framework Functionalities

The Mercury System Framework provide a broad set of functionalities that helps the user in the developing of applications. The management of all buses and Modem communication stacks is provided along with services for the handling of the most useful microcontroller internal peripheral (RTCC, ADC, USB, Power Management, etc.). Moreover, a simple real time OS implementation with services likes schedule tables, SW timers, alarms, etc. is provided.

As shown in Figure 2, the user has to implement only the high-level application logic and schedule a period function to implement his own application:

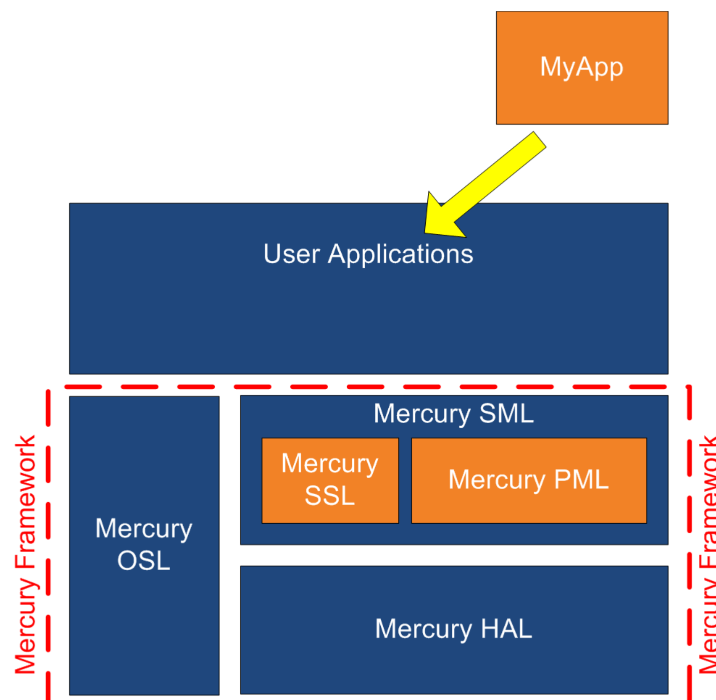


*Figure 2 - Example of user application positioning inside the MSF*

To get more information about features and compatibility of you MSF release, please check the MSF release notes (MS_FrameworkReleaseNotes_vx.x.x).

The Framework Version referenced by this manual is:

| Item | Major | Minor | Fix |
|---|---|---|---|
| MSF (Mercury System Framework) | 1 | 0 | 0 |

## 3. The System Configuration File (sys_cfg.h)

The MSF needs some basic static configuration to be set by the user, like the type of the modem used, the periodicity of the application, enable/disable status of certain modules, etc. These configurations are all stored in the file sys_cfg.h and this file must exist for each user application implemented using the MSF.

The list of relevant configurations is depicted in Table 1:

| Cfg Parameter Name | Possible Values | Description |
|---|---|---|
| USB_STS | STD_OFF<br>STD_ON | Enabling/disabling of USB device stack. |
| USB_CLASS_USED | USB_CLASS_CDC<br>USB_CLASS_HID | USB class to use. |
| TERM_TASK_STS | STD_OFF<br>STD_ON | Enabling/disabling of system terminal. |
| WIFI_MODULE | STD_ESP8266_MODULE<br>UPANEL_MODULE | WiFi module type (standard ESP module or uPanel option). |
| APP_TASK_STS | STD_OFF<br>STD_ON | Enabling/disabling of user application task (must be enabled to run the user app). |
| APP_TASK_SYMB | Valid function pointer | Name of the user function. |
| APP_TASK_PERIOD_MS | 1 to 65535 | Period of the user task in ms. |
| MODEM_USED | NO_MDM<br>GSM_GPRS_MDM<br>BT_MDM<br>WIFI_MDM | Type of modem to use. |

*Table 1 - Sys_Cfg config parameter list*

# 4. PML (Peripheral Management Layer)

The Peripheral Management Layer (or PML in short) is the MSF layer used to manage external peripheral through the Base Board communication channels. These peripherals are:

- Various types of Modem Board through the serial line on Mercury Modem Connector,
- Various types of Slave Boards (both on SBs or EBs) through I2C or serial lines o Mercury Slave Connector.

For the management of the Modem Boards, various complete stacks have been developed (one for each existing MB) and other will be added in the future. For the management of Slave devices on the Mercury Bus a complete I2C and UART stacks have been developed.

In addition to this the layer provides also the management of USB device stack.

## Modem Stacks

Currently the following Modem Stacks are available on MSF:

- Wifi Modem Stack (to be used with MB210),
- BT Modem Stack (to be used with MB310),
- GSM/GPRS Modem Stack (to be used with Futura GSM/GPRS modems, like FT1308M).

## Wifi Modem Stack

The Wifi Modem Stack is used to interface the Mercury WiFi modems (as, for instance, the MB210). The module provides API for the handling, association and creation of WiFi networks and for the transmission and reception of TCP and UDP packages.

### *API list:*

### *MdmWifi_SendWifiMsg*

| Service Name | MdmWifi_SendWifiMsg |
|---|---|
| Inputs | UINT8* TxBuffer – Pointer to TX buffer |
| Outputs | None |
| Description | API used to send an UART message to Wifi Modem, without providing message lenght. |
| Usage Examples | Send MyBuffer to Wifi Modem:<br><br>MdmWifi_SendWifiMsg(MyBuffer); |
| Notes | None |

## *MdmWifi_SendWifiMsgLen*

| Service Name | MdmWifi_SendWifiMsgLen |
| --- | --- |
| Inputs | UINT8* TxBuffer – Pointer to TX buffer<br>UINT8 DataLenght – Length of data to transmit |
| Outputs | None |
| Description | API used to send an UART message to Wifi Modem, providing message length. |
| Usage Examples | Send 10 bytes of MyBuffer to Wifi Modem:<br><br>MdmWifi_SendWifiMsg(MyBuffer,10); |
| Notes | None |

## *MdmWifi_ReceiveWifiMsg*

| Service Name | MdmWifi_ReceiveWifiMsg |
| --- | --- |
| Inputs | UINT8* RxBuffer – Pointer to RX buffer<br>UINT8 DataLenght – Number of received data bytes |
| Outputs | WifiMsg_NotReceived → No data received from modem<br>WifiMsg_Received → Some data received from modem |
| Description | API used to receive an UART message from WiFi Modem. If there are data received from the modem, the API will copy the received data to the user RX buffer (RxBuffer) provided and put also the number of bytes received on the user provided storage variable (DataLenght). |
| Usage Examples | Receive and copy data on WifiRxBuffer:<br><br>If ((MdmWifi_ReceiveWifiMsg(WifiRxBuffer,&RxDataLen)) ==<br>WifiMsg_Received)<br>{<br>/* Do something */<br>} |
| Notes | This API is used internally by the MdmWifi module, so the user is discouraged from using this API in the user app implementation, unless not strictly necessary. |

## *MdmWifiCmd_RestartModem*

| Service Name | MdmWifiCmd_RestartModem |
| --- | --- |
| Inputs | None |
| Outputs | None |
| Description | API used to restart Wifi Modem. |
| Usage Examples | Restart modem:<br><br>MdmWifiCmd_RestartModem(); |

| | |
|---|---|
| Notes | None |

### MdmWifiCmd_SetWifiMode

| | |
|---|---|
| Service Name | MdmWifiCmd_SetWifiMode |
| Inputs | UINT8 WifiMode – Possible values:<br>STATION_MODE,<br>SOFT_AP_MODE,<br>SOFT_AP_AND_STATION_MODE |
| Outputs | None |
| Description | API used to set the modem wifi mode (1: Station, 2: SoftAP, 3: SoftAP + Station). |
| Usage Examples | Set the modem to SoftAp Mode:<br><br>MdmWifiCmd_SetWifiMode(SOFT_AP_MODE); |
| Notes | None |

### MdmWifiCmd_JoinAccessPoint

| | |
|---|---|
| Service Name | MdmWifiCmd_JoinAccessPoint |
| Inputs | const UINT8 *AccessPoint<br>const UINT8 *Password |
| Outputs | None |
| Description | API used to join an existing access point with provided password. |
| Usage Examples | Join the access point “MyWifiAP” with Pwd “0123456789”:<br><br>MdmWifiCmd_JoinAccessPoint(“MyWifiAP”, “0123456789”); |
| Notes | None |

### MdmWifiCmd_QuitAccessPoint

| | |
|---|---|
| Service Name | MdmWifiCmd_QuitAccessPoint |
| Inputs | None |
| Outputs | None |
| Description | API used to quit a previously joined access point. |
| Usage Examples | Quit any previously joined AP:<br><br>MdmWifiCmd_QuitAccessPoint (); |
| Notes | None |

### MdmWifiCmd_SetTransferMode

| Service Name | MdmWifiCmd_SetTransferMode |
|---|---|
| Inputs | UINT8 Mode – Possible values:<br>TX_MODE_NORMAL<br>TX_MODE_UNVARNISHED |
| Outputs | STD_OK → Transfer mode correctly set<br>STD_NOT_OK → Invalid mode requested |
| Description | API used to set the transmission mode (normal or unvarnished). |
| Usage Examples | Set tx mode to normal:<br><br>MdmWifiCmd_SetTransferMode(TX_MODE_NORMAL); |
| Notes | None |

### MdmWifiCmd_SetConnectionMode

| Service Name | MdmWifiCmd_SetConnectionMode |
|---|---|
| Inputs | UINT8 Mode – Possible values:<br>CONN_MODE_SINGLE<br>CONN_MODE_MULTIPLE |
| Outputs | STD_OK → Conn mode correctly set<br>STD_NOT_OK → Invalid mode requested |
| Description | API used to set the connection mode (single or multiple). |
| Usage Examples | Set Conn mode to Multiple:<br><br>MdmWifiCmd_SetConnectionMode(CONN_MODE_MULTIPLE); |
| Notes | None |

### MdmWifiCmd_StartConnection

| Service Name | MdmWifiCmd_StartConnection |
|---|---|
| Inputs | UINT8 Mode – Possible values:<br>CONN_MODE_SINGLE<br>CONN_MODE_MULTIPLE<br>UINT8 Id – Connection ID: 0-4<br>UINT8 Type – Possible values:<br>PROTOCOL_UDP<br>PROTOCOL_TCP<br>UINT8 *Address – String with address of the Host to connect<br>UINT8 Port – Connection port |
| Outputs | None |
| Description | API used to start a TCP or UDP connection. |
| Usage Examples | Start a TCP connection to the host “dweet.io” on port 80: |

| | |
|---|---|
| | MdmWifiCmd_StartConnection(CONN_MODE_MULTIPLE,0, PROTOCOL_TCP, "dweet.io", 80); |
| Notes | None |

### MdmWifiCmd_SendData

| | |
|---|---|
| Service Name | MdmWifiCmd_SendData |
| Inputs | UINT8 Mode – Possible values:<br>CONN_MODE_SINGLE<br>CONN_MODE_MULTIPLE<br>UINT8 Id – Connection ID: 0-4<br>UINT8 Len – Length of data to send<br>UINT8 *Buffer – Pointer of the buffer to send |
| Outputs | None |
| Description | API used to send a TCP or UDP packet. |
| Usage Examples | Send the buffer Data of dimension Size on TCP or UDP channel:<br><br>MdmWifiCmd_SendData(CONN_MODE_MULTIPLE,0,Size,Data); |
| Notes | None |

### MdmWifiCmd_ReceiveWifiMsg

| | |
|---|---|
| Service Name | MdmWifiCmd_ReceiveWifiMsg |
| Inputs | UINT8* RxBuffer – Pointer to RX buffer<br>UINT8 DataLenght – Number of received data bytes |
| Outputs | WiFiRcv_DataNotReceived → No networkdata received<br>WiFiRcv_DataReceived → Some network data received |
| Description | API used to send a TCP or UDP packet. |
| Usage Examples | Receive network data from TCP or UDP channel:<br><br>If ((MdmWifiCmd_ReceiveWifiMsg(WifiRxBuffer,&RxDataLen)) == WiFiRcv_DataReceived)<br>{<br>/* Do something */<br>} |
| Notes | None |

### MdmWifiCmd_CloseConnection

| | |
|---|---|
| Service Name | MdmWifiCmd_CloseConnection |

| | |
|---|---|
| Inputs | UINT8 Mode – Possible values:<br>CONN_MODE_SINGLE<br>CONN_MODE_MULTIPLE<br>UINT8 Id – Connection ID: 0-4 |
| Outputs | None |
| Description | API used to close a connection |
| Usage Examples | Close connection of ID 0:<br><br>MdmWifiCmd_CloseConnection(CONN_MODE_MULTIPLE,0); |
| Notes | None |

### *MdmWifiCmd_ConfigureSoftAPMode*

| | |
|---|---|
| Service Name | MdmWifiCmd_ConfigureSoftAPMode |
| Inputs | void* ssid – Service Set Identifier (Wifi Network Name)<br>void* pwd – Wifi Network access password<br>UINT8 chid – Channel ID<br>UINT8 enc – Encoding type, possible values:<br>ENC_OPEN<br>WPA_PSK<br>WPA2_PSK<br>WPA_WPA2_PSK |
| Outputs | None |
| Description | API used to configure the softAP. |
| Usage Examples | Create an AP named “Mercury”, on ch 5 with password WPA2 “1234567890”.<br><br>MdmWifiCmd_ConfigureSoftAPMode("Mercury", "1234567890", 5,WPA2_PSK); |
| Notes | None |

### *MdmWifiCmd_ConfigureSoftAPIpAddress*

| | |
|---|---|
| Service Name | MdmWifiCmd_ConfigureSoftAPIpAddress |
| Inputs | void* ip – IP Address |
| Outputs | None |
| Description | API used to configure the sofAP IP Address. |
| Usage Examples | Set IP Address of SoftAP to 192.168.1.1:<br><br>MdmWifiCmd_ConfigureSoftAPIpAddress("192.168.1.1"); |
| Notes | None |

### MdmWifiCmd_ConfigureServer

| Service Name | MdmWifiCmd_ConfigureServer |
|---|---|
| Inputs | UINT8 Mode – Possible values:<br>DELETE_SERVER<br>CREATE_SERVER<br>UINT16 Port – Server port |
| Outputs | STD_OK → Server correctly created/deleted<br>STD_NOT_OK → Server not created/deleted |
| Description | API used to configure or delete a server. |
| Usage Examples | Create server on port 80:<br><br>MdmWifiCmd_ConfigureServer(CREATE_SERVER, 80); |
| Notes | None |

## BT Modem Stack

The BT Modem Stack is used to interface the Mercury BT modems (as, for instance, the MB310). The module provides API for the transmission and reception of BT packages and for the handling of BT module.

### *API list:*

### *MdmBt_SendBtMsg*

| Service Name | MdmBt_SendBtMsg |
|---|---|
| Inputs | UINT8* TxBuffer – Pointer to TX buffer<br>UINT8 DataLenght – Length of data to transmit |
| Outputs | None |
| Description | API used to send an message with BT Modem. |
| Usage Examples | Send the string "Hello" over BT:<br><br>MdmBt_SendBtMsg("Hello",5); |
| Notes | None |

### *MdmBt_ReceiveBtMsg*

| Service Name | MdmBt_ReceiveBtMsg |
|---|---|
| Inputs | UINT8* RxBuffer – Pointer to RX buffer<br>UINT8 DataLenght – Number of received data bytes |
| Outputs | BtMsg_NotReceived → No data received on BT<br>BtMsg_Received → Data received on BT |
| Description | API used to receive a message from BT Modem. |
| Usage Examples | Receive and copy data on BtRxBuffer:<br><br>if ((MdmBt_ReceiveBtMsg(BtRxBuffer,&RxDataLen)) == BtMsg_Received)<br>{<br>/* Do something */<br>} |
| Notes | None |

## GSM/GPRS Modem Stack

The GSM/GPRS Modem Stack is used to interface Futura GSM/GPRS modules like the FT1308M (based on SIM800 module). This module provides API for handling telephone calls, send and receive SMS and manage the GPRS network.

### *API list:*

### *Mdm_PinUnlock*

| Service Name | Mdm_PinUnlock |
|---|---|
| Inputs | const UINT8 *PIN – Pin to unlock the SIM |
| Outputs | None |
| Description | API to to unlock the SIM using PIN. |
| Usage Examples | Unlock with PIN “1234”:<br><br>Mdm_PinUnlock(“1234”); |
| Notes | None |

### *Mdm_MakePhoneCall*

| Service Name | Mdm_MakePhoneCall |
|---|---|
| Inputs | UINT8 *PhoneNumb<br>UINT8 PhoneNumbLen |
| Outputs | None |
| Description | API to make a phone call to specific number. |
| Usage Examples | Make a phone call to the number “1234567890”:<br><br>MakePhoneCall(“1234567890”, 10); |
| Notes | None |

### *Mdm_HangPhoneCall*

| Service Name | Mdm_HangPhoneCall |
|---|---|
| Inputs | None |
| Outputs | None |
| Description | API to close a phone call. |
| Usage Examples | Hang a phone call:<br><br>Mdm_HangPhoneCall(); |
| Notes | None |

### Mdm_GetPhoneCall

| Service Name | Mdm_GetPhoneCall |
|---|---|
| Inputs | None |
| Outputs | None |
| Description | API to get a phone call |
| Usage Examples | Get a phone call:<br><br>Mdm_GetPhoneCall(); |
| Notes | None |

### Mdm_IsRinging

| Service Name | Mdm_IsRinging |
|---|---|
| Inputs | None |
| Outputs | PhoneNotRinging → Not ringing<br>PhoneRinging → Ringing |
| Description | API to check if the phone is ringing. |
| Usage Examples | Check if the phone is ringing and get the call:<br><br>If (Mdm_IsRinging() == PhoneRinging)<br>{<br>  Mdm_GetPhoneCall();<br>} |
| Notes | None |

### Mdm_SetSmsFormat

| Service Name | Mdm_SetSmsFormat |
|---|---|
| Inputs | UINT8 TextFormat – Possible values:<br>SMS_MODE_TEXT_OFF<br>SMS_MODE_TEXT_ON |
| Outputs | None |
| Description | API to set the SMS format type (text ON/OFF). |
| Usage Examples | Set text mode:<br><br>Mdm_SetSmsFormat(SMS_MODE_TEXT_ON); |
| Notes | None |

### Mdm_RequestSmsData

| Service Name | Mdm_RequestSmsData |
|---|---|
| Inputs | None |
| Outputs | None |
| Description | API to request the SMS data to the modem |
| Usage Examples | Request SMS data: |

| | |
|---|---|
| | Mdm_RequestSmsData(); |
| Notes | None |

### *Mdm_GetSmsData*

| | |
|---|---|
| Service Name | Mdm_GetSmsData |
| Inputs | UINT8 *MessageText – Buffer where to store the SMS text |
| Outputs | SmsDataNotReady → SMS Data not yet ready<br>SmsDataReady → SMS data ready |
| Description | API to get the SMS data from the modem. |
| Usage Examples | - |
| Notes | None |

### *Mdm_SendSmsData*

| | |
|---|---|
| Service Name | Mdm_SendSmsData |
| Inputs | UINT8 *PhoneNmb<br>UINT8 PhoneNmbLen<br>UINT8 *MsgTxt<br>UINT8 MsgTxtLen |
| Outputs | None |
| Description | API to send an SMS. |
| Usage Examples | Send an SMS to the number "1234567890" with text "Ciao":<br><br>Mdm_SendSmsData("1234567890", 10, "Ciao", 4); |
| Notes | None |

### *Mdm_IsSmsReceived*

| | |
|---|---|
| Service Name | Mdm_IsSmsReceived |
| Inputs | None |
| Outputs | SmsNotReceived → No SMS received<br>SmsReceived → An SMS has been received |
| Description | API to check if an SMS has been received. |
| Usage Examples | Check if an SMS has been received:<br><br>If (Mdm_IsSmsReceived () == SmsReceived)<br>{<br>  Mdm_RequestSmsData();<br>} |
| Notes | None |

## I2C Stack

The I2C Stack is used to interface the I2C bus on Mercury system, in order to allow the communication with Mercury slaves (SBs and EBs with on board controller). The module provides API for transmission and reception of I2C packages.

### *API list:*

### *I2cSlv_SendI2cMsg*

| Service Name | I2cSlv_SendI2cMsg |
|---|---|
| Inputs | UINT8* TxBuffer – Pointer to TX buffer<br>UINT8 SlaveAddr – Address of the slave to transmit data to<br>UINT8 DataLenght – Length of data to transmit |
| Outputs | STD_OK → Tx OK<br>STD_NOT_OK → x Failed |
| Description | API used to send and I2c message to a specific slave device. The API returns the if the requested Tx operation was ok or failed. |
| Usage Examples | Send the command 0x50 0x01 to the slave address 0x01:<br><br>I2cTxBuffer[0] = 0x50;<br>I2cTxBuffer[1] = 0x01;<br>I2cSlv_SendI2cMsg(I2cTxBuffer,0x01,2); |
| Notes | None |

### *I2cSlv_ReceiveI2cMsg*

| Service Name | I2cSlv_ReceiveI2cMsg |
|---|---|
| Inputs | UINT8* RxBuffer– Pointer to RX buffer<br>UINT8 SlaveAddr – Address of the slave to transmit data to<br>UINT8 DataLenght – Length of data to transmit |
| Outputs | STD_OK → Rx OK<br>STD_NOT_OK → Rx Failed |
| Description | API used to receive and I2c message from a specific slave device. The API returns the if the requested Rx operation was ok or failed. |
| Usage Examples | Make a read request of 5 bytes to the slave 0x01:<br><br>I2cSlv_ReceiveI2cMsg(I2cRxBuffer, 0x01, 5); |
| Notes | The service is completely asynchronous, the read buffer will be filled with the read data once the I2C transaction will be completed. To check if the read operation is complete the API I2cSlv_I2cReadMsgSts must be used. |

## *I2cSlv_I2cReadMsgSts*

| Service Name | I2cSlv_I2cReadMsgSts |
|---|---|
| Inputs | None |
| Outputs | MessageNotReceived → The read operation is not completed<br>MessageReceived → The read operation is completed |
| Description | API used to check if a message has been received from the slave device. |
| Usage Examples | Check if the read operation is completed:<br><br>if (I2cSlv_I2cReadMsgSts() == MessageReceived)<br>{<br>/* Do something - I2cRxBuffer contains the received data */<br>} |
| Notes | None |

## *I2cSlv_GetI2cSts*

| Service Name | I2cSlv_GetI2cSts |
|---|---|
| Inputs | None |
| Outputs | I2cTxRxInProgress → Communication in progress<br>I2cTxRxComplete → Communication completed |
| Description | API used to get the global I2C status (TxRxbusy or Read/Write complete). |
| Usage Examples | Check the global I2C Communication status:<br><br>if (I2cSlv_GetI2cSts () == I2cTxRxComplete)<br>{<br>/* Do something */<br>} |
| Notes | None |

## UART Stack

The UART stack is not still implemented in the current release of the MSF.

## USB

The USB module provides some basic USB communication functionalities to the Base Board. It doesn't have user API in the current MSF release.

# 5. SSL (System Service Layer)

The System Services Layer (or SSL in short) is the MSF layer used to manage some basic system services. These services are:

- The on-board user LEDs,
- The internal RTCC,
- The system power management,
- The system terminal.

The MSF has some APIs for the basic management of each one of these services/modules.

## LED

The LED module is intended to provide to the user an high level management layer for the BBs on-board LEDs. It provides API for the setting of LEDs status and handling of LEDs blink and pulse behaviors.

### *API list:*

### *Led_SetLedBlinkTime*

| Service Name | Led_SetLedBlinkTime |
|---|---|
| Inputs | UINT8 Led – The LED to be controlled. Possible values:<br>LED_1<br>LED_2<br>LED_3<br>UINT16 OnTimeMs – Blink on time in ms<br>UINT16 OffTimeMs – Blink off time in ms |
| Outputs | None |
| Description | API to set the LED blink timing. This blink timing will be applied if the LED status is set to LED_STS_BLINK using the API Led_SetLedStatus. |
| Usage Examples | Set LED_1 blink timing to 50ms ON and 950ms OFF:<br><br>Led_SetLedBlinkTime(LED_1, 50, 950); |
| Notes | None |

### *Led_SetLedPulseTime*

| Service Name | Led_SetLedPulseTime |
|---|---|
| Inputs | UINT8 Led – The LED to be controlled. Possible values: |

| | |
|---|---|
| | LED_1<br>LED_2<br>LED_3<br>UINT16 PulseTimeMs – Pulse time in ms. |
| Outputs | None |
| Description | API to set the LED pulse timing. This pulse timing will be applied if the LED status is set to LED_STS_PULSEusing the API Led_SetLedStatus. |
| Usage Examples | Set LED_1 pulse timing to 100ms:<br><br>Led_SetLedPulseTime (LED_1, 100); |
| Notes | None |

### *Led_SetLedStatus*

| | |
|---|---|
| Service Name | Led_SetLedStatus |
| Inputs | UINT8 Led – The LED to be controlled. Possible values:<br>LED_1<br>LED_2<br>LED_3<br>LedStsType LedSts – Possible values:<br>LED_STS_OFF<br>LED_STS_ON<br>LED_STS_BLINK<br>LED_STS_PULSE |
| Outputs | None |
| Description | API to set the LED behavior. |
| Usage Examples | 1. Make the LED_1 blink 50ms on and 950 ms off:<br><br>Led_SetLedBlinkTime(LED_1, 50, 950);<br>Led_SetLedStatus(LED_1, LED_STS_BLINK);<br><br>2. Make the LED_1 pulse for 100ms:<br><br>Led_SetLedPulseTime (LED_1, 100);<br>Led_SetLedStatus(LED_1, LED_STS_PULSE);<br><br>3. Set LED_1 status ON:<br><br>Led_SetLedStatus(LED_1, LED_STS_ON); |
| Notes | None |

## RTCM

The RTCM module is intended to provide to the user an high level layer for the management of the internal RTCC. It provides API to set and get RTCC date/time and to set and get RTCC alarm date/time as well as an API to set an user action to be triggered once the RTCC alarm fires.

### *API list:*

### *Rtcm_SetRtccDate*

| Service Name | Rtcm_SetRtccDate |
|---|---|
| Inputs | RtccDateType Date – System Date/Time |
| Outputs | None |
| Description | API to set the RTCC date. |
| Usage Examples | Set RTCC date and time:<br><br>/* Set date and time */<br>Rtcm_SystemDate.Year = 2017;<br>Rtcm_SystemDate.Month = 4;<br>Rtcm_SystemDate.Day = 23;<br>Rtcm_SystemDate.Weekday = WEEKDAY_SUNDAY;<br>Rtcm_SystemDate.Hour = 0;<br>Rtcm_SystemDate.Minute = 0;<br>Rtcm_SystemDate.Second = 0;<br>Rtcm_SetRtccDate(Rtcm_SystemDate); |
| Notes | None |

### *Rtcm_GetRtccDate*

| Service Name | Rtcm_GetRtccDate |
|---|---|
| Inputs | None |
| Outputs | RtccDateType → System Date/Time |
| Description | API to get the current RTCC date. |
| Usage Examples | Get RTCC date and time:<br><br>/* Get RTCC date and time */<br>Rtcm_SystemDate = Rtcm_GetRtccDate(); |
| Notes | None |

### *Rtcm_SetRtccAlarm*

| Service Name | Rtcm_SetRtccAlarm |
|---|---|
| Inputs | RtccAlarmType Alarm – RTCC alarm |

| | |
|---|---|
| Outputs | None |
| Description | API to set the RTCC Alarm. |
| Usage Examples | Set RTCC alarm:<br><br>/* Set alarm */<br>Rtcm_RtccAlarm.AlrmMonth = 4;<br>Rtcm_RtccAlarm.AlrmDay = 23;<br>Rtcm_RtccAlarm.AlrmWeekday = WEEKDAY_SUNDAY;<br>Rtcm_RtccAlarm.AlrmHour = 0;<br>Rtcm_RtccAlarm.AlrmMinute = 1;<br>Rtcm_RtccAlarm.AlrmSecond = 0;<br>Rtcm_SetRtccAlarm(Rtcm_RtccAlarm); |
| Notes | None |

### *Rtcm_GetRtccAlarm*

| | |
|---|---|
| Service Name | Rtcm_GetRtccAlarm |
| Inputs | None |
| Outputs | RtccAlarmType → RTCC Alarm |
| Description | API to get the RTCC Alarm. |
| Usage Examples | Get RTCC alarm:<br><br>/* Get RTCC alarm */<br>Rtcm_Alarm = Rtcm_GetRtccAlarm(); |
| Notes | None |

### *Rtcm_SetAlarmAction*

| | |
|---|---|
| Service Name | Rtcm_SetAlarmAction |
| Inputs | Rtcc_CallbackType Action – Callback to be triggered when the alarm fires. Must be defined by the user. |
| Outputs | None |
| Description | API to set the action to be performed when the alarm fires. |
| Usage Examples | Registert an alarm action:<br><br>/* User alarm callback */<br>void Alarm (void)<br>{<br>  /* Set LED on */<br>  Led_SetLedStatus(LED_1, 1);<br>  /* Send alarm event */<br>  GenerateEvt(&AlarmEvent);<br>} |

| | /* Register alarm action */<br>Rtcm_SetAlarmAction(&Alarm); |
|---|---|
| Notes | None |

## SYSM

The SYSM module is intended to provide to the user an high level interface to handle the Base Board power settings (mainly low power modes entry).

### *API list:*

### *Sysm_IdleMode*

| Service Name | Sysm_IdleMode |
|---|---|
| Inputs | None |
| Outputs | None |
| Description | API to trigger the system IDLE mode (CPU off, peripherals on). This is the less power saving sleep mode. It can be waken-up by:<br>• Any enabled interrupt<br>• Wdg<br>• Reset (HW or SW) |
| Usage Examples | Trigger the IDLE mode:<br><br>Sysm_IdleMode(); |
| Notes | None |

### *Sysm_SleepMode*

| Service Name | Sysm_SleepMode |
|---|---|
| Inputs | None |
| Outputs | None |
| Description | API to trigger SLEEP mode (CPU and peripherals off). This is one of the two sleep mode. It can be waken-up some HW source only, in particular:<br>• Rtcc alarm<br>• Timer 1 interrupt<br>• INTx interrupt<br>• Wdg<br>• Reset (HW or SW) |
| Usage Examples | Trigger the SLEEP mode:<br><br>Sysm_SleepMode (); |
| Notes | None |

### *Sysm_DeepSleepMode*

| Service Name | Sysm_DeepSleepMode |
|---|---|
| Inputs | None |

| Outputs | None |
| --- | --- |
| Description | API to trigger the DEEP SLEEP mode (CPU and peripherals off). This is the highes power saving sleep mode. It can be waken-up some HW source only, in particular:<br>• Rtcc alarm<br>• INT0 interrupt<br>• DsWdg<br>• Reset (HW only) |
| Usage Examples | Trigger the DEEP SLEEP mode:<br><br>Sysm_DeepSleepMode(); |
| Notes | None |

## TERM

The TERM module provides some basic terminal functionalities to the Base Board. It doesn't have user API in the current MSF release.

# 6. OSL (Operative System Layer)

The MSF is based on a simple, non-preemptive real-time Operative System (also called Mercury OS), which provides some basic services like scheduling of the various framework and application main tasks, events, SW timers, alarms, etc. These basic functionalities are available for the user too,

## OS Services

The OS Services Module (os_ser) provides some basic APIs for events generation and reception. These two APIs relies on user-defined global variables of type EventStructureType* and provides an output of EventStatusType. For any event that the user wants to use a global variable of type EventStructureType must be declared. Then a corresponding event can be generated and received using the proper GenerateEvt or ReceiveEvt API.

### *API list:*

### *GenerateEvt*

| Service Name | GenerateEvt |
|---|---|
| Inputs | EventStructureType *Event – Pointer to the Event global variable |
| Outputs | EventStatusType – Possible values:<br>EventIdle<br>EventReceived<br>EventSent |
| Description | API to generate an event. The API takes an event variable passed by reference as an input. The same event could be received using the ReceiveEvt API. |
| Usage Examples | Generation of an user event:<br><br>EventStructureType MyEvent;<br><br>GenerateEvt(&MyEvent); |
| Notes | None |

### *ReceiveEvt*

| Service Name | ReceiveEvt |
|---|---|
| Inputs | EventStructureType *Event – Pointer to the Event global variable |
| Outputs | EventStatusType – Possible values:<br>EventIdle<br>EventReceived<br>EventSent |
| Description | API to receive an event. The API takes an event variable passed by reference as an input. The event had to be previously generated by a GenerateEvt API. |

| Usage Examples | Reception of an user event:<br><br>EventStructureType MyEvent;<br><br>If (ReceiveEvt(&MyEvent))<br>{<br>  /* Do something */<br>} |
| --- | --- |
| Notes | None |

## OS Timers

The Mercury OS provides some basic virtual timing services with a maximum resolution of 1ms, to be used for simple timing measurement, non-blocking SW delays, etc.

### *API list:*

### *OsTmr_StartTimer*

| Service Name | OsTmr_StartTimer |
|---|---|
| Inputs | SwTimerType *Timer<br>UINT32 Timeout |
| Outputs | None |
| Description | API to start a software timer. |
| Usage Examples | Start a SW timer with timeout of 10s:<br><br>SwTimerType MyTimer;<br><br>OsTmr_StartTimer(&MyTimer, 10000); |
| Notes | None |

### *OsTmr_StopTimer*

| Service Name | OsTmr_StopTimer |
|---|---|
| Inputs | SwTimerType *Timer |
| Outputs | None |
| Description | API to stop a software timer. |
| Usage Examples | Stop a previously started SW Timer:<br><br>OsTmr_StopTimer(&MyTimer); |
| Notes | None |

### *OsTmr_Wait*

| Service Name | OsTmr_Wait |
|---|---|
| Inputs | SwTimerType *WaitTimer<br>UINT32 DelayMs |
| Outputs | DelayNotExpired → The set delay is still not expired<br>DelayExpired → The set delay is expired |
| Description | API that implement a non-blocking delay function. It waits for the defined amount of time (in ms) passed as parameter. |
| Usage Examples | Set a LED on for 2s after an initial delay of 1s (in state machine fashion) |

| | |
|---|---|
| | /* Inside a periodically called task, with State static initialized to 0 */<br><br>switch(State)<br>{<br>  case 0:<br>    if (OsTmr_Wait(&WaitTimer, 1000))<br>    {<br>      State = 1;<br>      Led_SetLedStatus(LED_1, 1);<br>    }<br>    break;<br><br>  case 1:<br>    if (OsTmr_Wait(&WaitTimer, 2000))<br>    {<br>      State = 2;<br>      Led_SetLedStatus(LED_1, 0);<br>    }<br>    break;<br><br>  case 2:<br>    break;<br>} |
| Notes | None |

### *OsTmr_GetTimerStatus*

| | |
|---|---|
| Service Name | OsTmr_GetTimerStatus |
| Inputs | SwTimerType *Timer |
| Outputs | SwTmrNotExpired→ The sw timer is still not expired<br>SwTimerExpired→ The sw timer is expired<br>SwTimerDisabled → The sw timer is disabled (stopped) |
| Description | API that checks the software timer status. |
| Usage Examples | Check if a SW timer is expired:<br><br>/* Check if expired */<br>If (OsTmr_GetTimerStatus(&MyTimer) == SwTimerExpired)<br>{<br>  /* Do something */<br>} |
| Notes | None |

### *OsTmr_GetElapsedTime*

| | |
|---|---|
| Service Name | OsTmr_GetElapsedTime |

| Inputs | SwTimerType *Timer |
|---|---|
| Outputs | UINT32 → Elapsed time in ms |
| Description | API that gets the elapsed time since the sw timer started. |
| Usage Examples | Get elapsed time:<br><br>UINT32 ElapsedTimeMs;<br><br>/* Get elapsed time */<br>ElapsedTimeMs = OsTmr_GetElapsedTime(&MyTimer); |
| Notes | None |

### *OsTmr_GetRemainingTime*

| Service Name | OsTmr_GetRemainingTime |
|---|---|
| Inputs | SwTimerType *Timer |
| Outputs | UINT32 → Remaining time in ms |
| Description | API that gets the remaining time before a sw timer expires. |
| Usage Examples | Get remaining time:<br><br>UINT32 RemainingTimeMs;<br><br>/* Get remaining time */<br>RemainingTimeMs = OsTmr_GetRemainingTime (&MyTimer); |
| Notes | None |

## OS Alarms

Besides SW timers, the Mercury OS provides also an alarm module, that can set alarms which, once fired, could trigger the execution of a user callback. The callback must be defined by the user and it must be a void-void function.

The function which process the alarm will check if the function pointer passed is actually pointing to something, in order to avoid unexpected crashes of the system.

The maximum allowed number of alarms is a configuration parameter of the alarm module (OS_ALARM_NUMBER), and is statically defined at compile time. Then the desired alarm to address is identified by an ID (basically the position of the alarm structure inside the alarm list).

### *OsAlrm_SetOsAlarm*

| Service Name | OsAlrm_SetOsAlarm |
|---|---|
| Inputs | UINT16 OsAlarmId – ID of the alarm (from 0 to OS_ALARM_NUMBER)<br>UINT32 OsAlarmTimeout – Timeout in ms before the alarm fires<br>OsAlarmCallbackType AlarmCallback – User callback executed once the alarm fires |
| Outputs | None |
| Description | API to set an OS alarm. Once the alarm timeout expires the user callback will be automatically executed. |
| Usage Examples | Set the alarm of ID 1 with timeout of 5s and execution of the callback MyAlrmCbk once the alarm fires:<br><br>/* User callback implementation */<br>void MyAlrmCbk (void)<br>{<br>  /* My callback implementation */<br>}<br><br>/* Alarm set */<br>OsAlrm_SetOsAlarm(1, 5000, MyAlrmCbk); |
| Notes | None |

### *OsAlrm_ClearOsAlarm*

| Service Name | OsAlrm_ClearOsAlarm |
|---|---|
| Inputs | UINT16 OsAlarmId – ID of the alarm (from 0 to OS_ALARM_NUMBER) |
| Outputs | None |
| Description | API to clear an OS alarm. |
| Usage Examples | OS Alarm 1 cancellation: |

| | OsAlrm_ClearOsAlarm(1); |
|---|---|
| Notes | None |