

Mercury System

Framework User Manual

IoT and Connectivity Made Simple

Francesco Ficili
31/12/2018

Chapter: Introduction

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	18
GSM/GPRS MODEM STACK	21
I2C STACK	24
UART STACK	27
USB	28
5. SSL (SYSTEM SERVICE LAYER)	29
LED.....	29
EXT_INT	31
RTCM	33
SYSM.....	36
TERM	38
6. OSL (OPERATIVE SYSTEM LAYER).....	39
OS SERVICES	39
OS TIMERS	41
OS ALARMS	44

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.

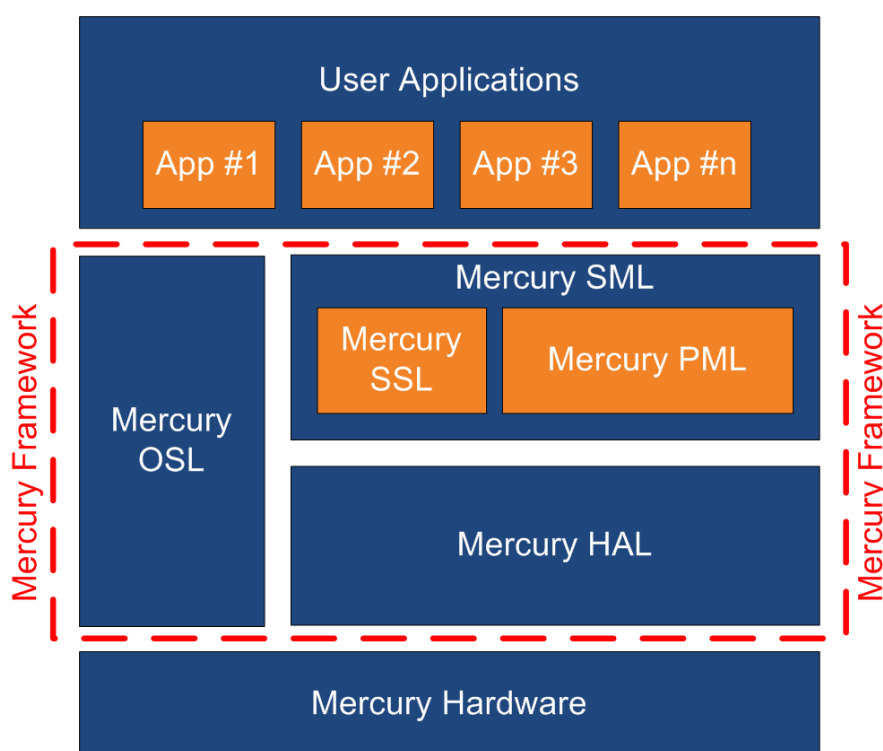


Figure 1 - Mercury System Framework Architecture

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:

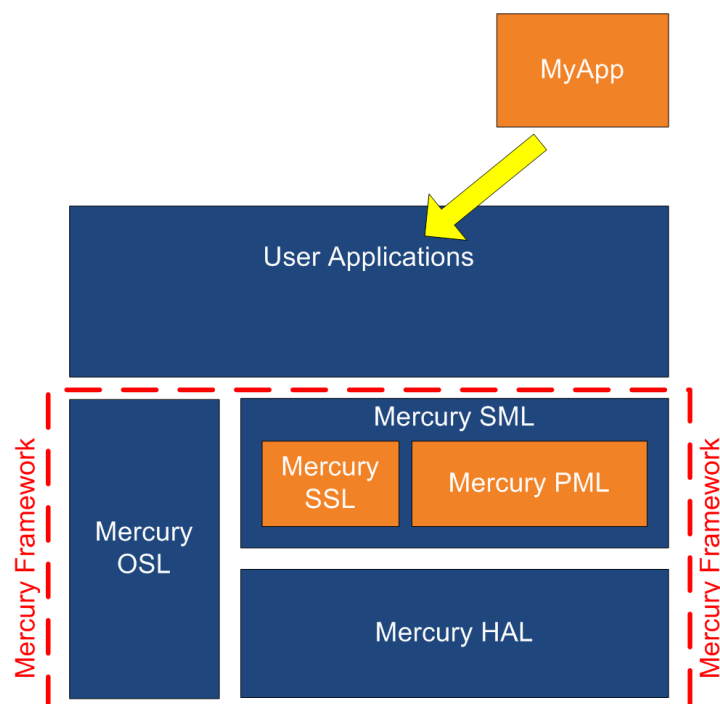


Figure 2 - Example of user application positioning inside the MSF

To get more information about features and compatibility of your 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	1	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 STD_ON	Enabling/disabling of USB device stack.
USB_CLASS_USED	USB_CLASS_CDC USB_CLASS_HID	USB class to use.
TERM_TASK_STS	STD_OFF STD_ON	Enabling/disabling of system terminal.
TERMINAL_MODE	COMMAND_MODE TRANSPARENT_MODE	Use the terminal Slave command mode ore modem transparent mode (this will be changed in the future).
WDG_STS	STD_OFF STD_ON	Enabling/disabling of watchdog.
WIFI_MODULE	STD_ESP8266_MODULE UPANEL_MODULE	WiFi module type (standard ESP module or uPanel option).
APP_TASK_STS	STD_OFF 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 GSM_GPRS_MDM BT_MDM 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: MdmWifi_SendWifiMsg(MyBuffer);
Notes	None

MdmWifi_SendWifiMsgLen

Service Name	MdmWifi_SendWifiMsgLen
Inputs	UINT8* TxBuffer – Pointer to TX buffer UINT8 DataLength – 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: MdmWifi_SendWifiMsg(MyBuffer,10);
Notes	None

MdmWifi_ReceiveWifiMsg

Service Name	MdmWifi_ReceiveWifiMsg
Inputs	UINT8* RxBuffer – Pointer to RX buffer UINT8 DataLength – Number of received data bytes
Outputs	WifiMsg_NotReceived → No data received from modem 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 (DataLength).
Usage Examples	Receive and copy data on WifiRxBuffer: If ((MdmWifi_ReceiveWifiMsg(WifiRxBuffer,&RxDataLen)) == WifiMsg_Received) { /* Do something */ }
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: MdmWifiCmd_RestartModem();

Notes	None

MdmWifiCmd_SetWifiMode

Service Name	MdmWifiCmd_SetWifiMode
Inputs	UINT8 WifiMode – Possible values: STATION_MODE, SOFT_AP_MODE, 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: MdmWifiCmd_SetWifiMode(SOFT_AP_MODE);
Notes	None

MdmWifiCmd_JoinAccessPoint

Service Name	MdmWifiCmd_JoinAccessPoint
Inputs	const UINT8 *AccessPoint 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”: 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: MdmWifiCmd_QuitAccessPoint ();
Notes	None

MdmWifiCmd_SetTransferMode

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

MdmWifiCmd_SetConnectionMode

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

MdmWifiCmd_StartConnection

Service Name	MdmWifiCmd_StartConnection
Inputs	UINT8 Mode – Possible values: CONN_MODE_SINGLE CONN_MODE_MULTIPLE UINT8 Id – Connection ID: 0-4 UINT8 Type – Possible values: PROTOCOL_UDP PROTOCOL_TCP UINT8 *Address – String with address of the Host to connect 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: CONN_MODE_SINGLE CONN_MODE_MULTIPLE UINT8 Id – Connection ID: 0-4 UINT8 Len – Length of data to send 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: MdmWifiCmd_SendData(CONN_MODE_MULTIPLE,0,Size,Data);
Notes	None

MdmWifiCmd_ReceiveWifiMsg

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

MdmWifiCmd_CloseConnection

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

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

MdmWifiCmd_ConfigureSoftAPMode

Service Name	MdmWifiCmd_ConfigureSoftAPMode
Inputs	void* ssid – Service Set Identifier (Wifi Network Name) void* pwd – Wifi Network access password UINT8 chid – Channel ID UINT8 enc – Encoding type, possible values: ENC_OPEN WPA_PSK WPA2_PSK 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”. <pre>MdmWifiCmd_ConfigureSoftAPMode("Mercury", "1234567890", 5,WPA2_PSK);</pre>
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: <pre>MdmWifiCmd_ConfigureSoftAPIpAddress("192.168.1.1");</pre>
Notes	None

MdmWifiCmd_ConfigureServer

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

MdmWifiHttp_MdmJoinNetwork

Service Name	MdmWifiHttp_MdmJoinNetwork
Inputs	const UINT8 *AccessPoint – Name of the AP to connect to const UINT8 *Password – Password for AP connection
Outputs	None
Description	API to initialize the wifi modem in Join Network mode.
Usage Examples	Join an existing network: MdmWifiHttp_MdmJoinNetwork("MyAccessPoint","password");
Notes	None

MdmWifiHttp_IsMdmJoinNetworkCompleted

Service Name	MdmWifiHttp_IsMdmJoinNetworkCompleted
Inputs	None
Outputs	STD_TRUE → Join Network completed STD_FALSE → Join Network not completed
Description	API to get if MdmWifiHttp_MdmJoinNetwork is completed.
Usage Examples	Check if the network join is completed: if (MdmWifiHttp_IsMdmJoinNetworkCompleted()) { /* Join Network completed */ }
Notes	None

MdmWifiHttp_GetMethod

Service Name	MdmWifiHttp_GetMethod
Inputs	const UINT8 *GetString – Data string of GET method const UINT8 *Host – Host of GET method
Outputs	None
Description	API to request a GET method service.
Usage Examples	GET Method: MdmWifiHttp_GetMethod(GetData,HOST);
Notes	None

MdmWifiHttp_IsGetMethodCompleted

Service Name	MdmWifiHttp_IsGetMethodCompleted
Inputs	None
Outputs	STD_TRUE → GET Method completed STD_FALSE → GET Method not completed
Description	API to get if MdmWifiHttp_GetMethod is completed.
Usage Examples	Check if the GET method has completed: if (MdmWifiHttp_IsGetMethodCompleted ()) { /* GET completed */ }
Notes	None

MdmWifiHttp_PostMethod

Service Name	MdmWifiHttp_PostMethod
Inputs	const UINT8 *PostString– Data string of POST method const UINT8 *Host – Host of POST method
Outputs	None
Description	API to request a POST method service.
Usage Examples	POST Method: MdmWifiHttp_PostMethod (PostData,HOST);
Notes	None

MdmWifiHttp_IsPostMethodCompleted

Service Name	MdmWifiHttp_IsPostMethodCompleted
Inputs	None

Outputs	STD_TRUE → GET Method completed STD_FALSE → GET Method not completed
Description	API to get if MdmWifiHttp_PostMethod is completed.
Usage Examples	Check if the POST method has completed: <pre>if (MdmWifiHttp_IsPostMethodCompleted ()) { /* GET completed */ }</pre>
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_SendBtMsgLen

Service Name	MdmBt_SendBtMsgLen
Inputs	UINT8* TxBuffer – Pointer to TX buffer UINT8 DataLength – Length of data to transmit
Outputs	None
Description	API used to send a message to BT Modem with Length parameter.
Usage Examples	Send the string “Hello” over BT: <pre> uint8 *Data[] = {'H','e','l','l','o'}; MdmBt_SendBtMsgLen(Data,5); </pre>
Notes	None

MdmBt_SendBtMsg

Service Name	MdmBt_SendBtMsg
Inputs	UINT8* TxBuffer – Pointer to TX buffer
Outputs	None
Description	API used to send a message to BT Modem without Length parameter.
Usage Examples	Send the string “Hello” over BT: <pre> MdmBt_SendBtMsg(“Hello”); </pre>
Notes	None

MdmBt_ReceiveBtMsg

Service Name	MdmBt_ReceiveBtMsg
Inputs	UINT8* RxBuffer – Pointer to RX buffer UINT8 DataLength – Number of received data bytes
Outputs	BtMsg_NotReceived → No data received on BT BtMsg_Received → Data received on BT
Description	API used to receive a message from BT Modem.
Usage Examples	Receive and copy data on BtRxBuffer:

	<pre> if ((MdmBt_ReceiveBtMsg(BtRxBuffer,&RxDataLen)) == BtMsg_Received) { /* Do something */ } </pre>
Notes	None

MdmBt_SetAtMode

Service Name	MdmBt_SetAtMode
Inputs	None
Outputs	None
Description	API used to set BT Modem AT mode.
Usage Examples	Set AT mode: MdmBt_SetAtMode();
Notes	None

MdmBt_SetComMode

Service Name	MdmBt_SetComMode
Inputs	None
Outputs	None
Description	API used to set BT Modem COM mode.
Usage Examples	Set COM mode: MdmBt_SetComMode ();
Notes	None

MdmBt_ModemReset

Service Name	MdmBt_ModemReset
Inputs	None
Outputs	None
Description	API used to reset the BT Modem. This command only works on AT mode.
Usage Examples	Reset BT modem: MdmBt_ModemReset ();
Notes	None

MdmBt_SetModuleName

Service Name	MdmBt_SetModuleName
Inputs	UINT8* Name – Pointer to module name string
Outputs	None
Description	API used to set the BT module name. This command only works on AT mode.
Usage Examples	Name the BT modem as “Mercury”: MdmBt_SetModuleName (“Mercury”);
Notes	None

MdmBt_SetModuleMode

Service Name	MdmBt_SetModuleName
Inputs	BtModuleModeType Mode – Available modes: <ul style="list-style-type: none">- MODE_SLAVE- MODE_MASTER- MODE_SLAVE_LOOP
Outputs	None
Description	API used to set the BT module Mode. This command only works on AT mode.
Usage Examples	Set slave mode: MdmBt_SetModuleMode(MODE_SLAVE);
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”: Mdm_PinUnlock(“1234”);
Notes	None

Mdm_MakePhoneCall

Service Name	Mdm_MakePhoneCall
Inputs	UINT8 *PhoneNumb UINT8 PhoneNumbLen
Outputs	None
Description	API to make a phone call to specific number.
Usage Examples	Make a phone call to the number “1234567890”: 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: 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: Mdm_GetPhoneCall();
Notes	None

Mdm_IsRinging

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

Mdm_SetSmsFormat

Service Name	Mdm_SetSmsFormat
Inputs	UINT8 TextFormat – Possible values: SMS_MODE_TEXT_OFF SMS_MODE_TEXT_ON
Outputs	None
Description	API to set the SMS format type (text ON/OFF).
Usage Examples	Set text mode: 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 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 UINT8 PhoneNmbLen UINT8 *MsgTxt UINT8 MsgTxtLen
Outputs	None
Description	API to send an SMS.
Usage Examples	Send an SMS to the number “1234567890” with text “Ciao”: Mdm_SendSmsData(“1234567890”, 10, “Ciao”, 4);
Notes	None

Mdm_IsSmsReceived

Service Name	Mdm_IsSmsReceived
Inputs	None
Outputs	SmsNotReceived → No SMS received 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: If (Mdm_IsSmsReceived () == SmsReceived) { Mdm_RequestSmsData(); }
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 UINT8 SlaveAddr – Address of the slave to transmit data to UINT8 DataLenght – Length of data to transmit
Outputs	STD_OK → Tx OK 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: I2cTxBuffer[0] = 0x50; I2cTxBuffer[1] = 0x01; I2cSlv_SendI2cMsg(I2cTxBuffer,0x01,2);
Notes	None

I2cSlv_ReceiveI2cMsg

Service Name	I2cSlv_ReceiveI2cMsg
Inputs	UINT8* RxBuffer– Pointer to RX buffer UINT8 SlaveAddr – Address of the slave to transmit data to UINT8 DataLenght – Length of data to transmit
Outputs	STD_OK → Rx OK 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: 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 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: <pre>if (I2cSlv_I2cReadMsgSts() == MessageReceived) { /* Do something - I2cRxBuffer contains the received data */ }</pre>
Notes	None

I2cSlv_GetI2cSts

Service Name	I2cSlv_GetI2cSts
Inputs	None
Outputs	I2cTxRxInProgress → Communication in progress 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: <pre>if (I2cSlv_GetI2cSts () == I2cTxRxComplete) { /* Do something */ }</pre>
Notes	None

I2cSlv_QueryI2cData

Service Name	I2cSlv_QueryI2cData
Inputs	UINT8* TxBuffer – Pointer to TX buffer UINT8 TxLen – TX buffer len UINT8* RxBuffer – Pointer to RX buffer UINT8 RxLen – RX buffer len UINT8 SlaveAddr – Slave address
Outputs	OP_IN_PROGRESS → Communication in progress OP_COMPLETE → Communication completed OP_TIMEOUT → Timeout
Description	API used to query an I2c Data to a slave. This API works in state-machine fashion, the user has to keep calling the API until either OP_COMPLETE or OP_TIMEOUT is returned.
Usage Examples	

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: LED_1 LED_2 LED_3 UINT16 OnTimeMs – Blink on time in ms 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: 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 LED_2 LED_3 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_PULSE using the API Led_SetLedStatus.
Usage Examples	Set LED_1 pulse timing to 100ms: Led_SetLedPulseTime (LED_1, 100);
Notes	None

Led_SetLedStatus

Service Name	Led_SetLedStatus
Inputs	UINT8 Led – The LED to be controlled. Possible values: LED_1 LED_2 LED_3 LedStsType LedSts – Possible values: LED_STS_OFF LED_STS_ON LED_STS_BLINK 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: Led_SetLedBlinkTime(LED_1, 50, 950); Led_SetLedStatus(LED_1, LED_STS_BLINK); 2. Make the LED_1 pulse for 100ms: Led_SetLedPulseTime (LED_1, 100); Led_SetLedStatus(LED_1, LED_STS_PULSE); 3. Set LED_1 status ON: Led_SetLedStatus(LED_1, LED_STS_ON);
Notes	None

EXT_INT

The EXT_INT module is intended to provide to the user an high level management layer for the handling of BBs external pin interrupt sources. These PINs are called Int0 and Int1 and are present in the Mercury Connector and some SBs uses them to signal to the BB alarms or attention requests. The module provides API for registering callbacks to be executed when an external interrupt is triggered.

API list:

ExtInt_SetInt0Action

Service Name	ExtInt_SetInt0Action
Inputs	ExtInt_CallbackType Callback – Function pointer to the external interrupt handling callback.
Outputs	None
Description	API used to register action to service the Int0 external interrupt.
Usage Examples	<p>Register an handling callback for Int0 external interrupt.</p> <pre> Void MyInt0Callback (void) { /* implement the desired cb code here */ } /* Register the callback */ ExtInt_SetInt0Action(MyInt0Callback); </pre>
Notes	None

ExtInt_SetInt1Action

Service Name	ExtInt_SetInt1Action
Inputs	ExtInt_CallbackType Callback – Function pointer to the external interrupt handling callback.
Outputs	None
Description	API used to register action to service the Int0 external interrupt.
Usage Examples	<p>Register an handling callback for Int1 external interrupt.</p> <pre> Void MyInt1Callback (void) { /* implement the desired cb code here */ } /* Register the callback */ ExtInt_SetInt0Action(MyInt1Callback); </pre>
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: <pre> /* Set date and time */ Rtcm_SystemDate.Year = 2017; Rtcm_SystemDate.Month = 4; Rtcm_SystemDate.Day = 23; Rtcm_SystemDate.Weekday = WEEKDAY_SUNDAY; Rtcm_SystemDate.Hour = 0; Rtcm_SystemDate.Minute = 0; Rtcm_SystemDate.Second = 0; Rtcm_SetRtccDate(Rtcm_SystemDate); </pre>
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: <pre> /* Get RTCC date and time */ Rtcm_SystemDate = Rtcm_GetRtccDate(); </pre>
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: <pre> /* Set alarm */ Rtcm_RtccAlarm.AlrmMonth = 4; Rtcm_RtccAlarm.AlrmDay = 23; Rtcm_RtccAlarm.AlrmWeekday = WEEKDAY_SUNDAY; Rtcm_RtccAlarm.AlrmHour = 0; Rtcm_RtccAlarm.AlrmMinute = 1; Rtcm_RtccAlarm.AlrmSecond = 0; Rtcm_SetRtccAlarm(Rtcm_RtccAlarm); </pre>
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: <pre> /* Get RTCC alarm */ Rtcm_Alarm = Rtcm_GetRtccAlarm(); </pre>
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: <pre> /* User alarm callback */ void Alarm (void) { /* Set LED on */ } </pre>

	<pre>Led_SetLedStatus(LED_1, 1); /* Send alarm event */ GenerateEvt(&AlarmEvent); } /* Register alarm action */ Rtcm_SetAlarmAction(&Alarm);</pre>
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: <ul style="list-style-type: none"> Any enabled interrupt Wdg Reset (HW or SW)
Usage Examples	Trigger the IDLE mode: 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: <ul style="list-style-type: none"> Rtcc alarm Timer 1 interrupt INTx interrupt Wdg Reset (HW or SW)
Usage Examples	Trigger the SLEEP mode: Sysm_SleepMode ();
Notes	None

Sysm_DeepSleepMode

Service Name	Sysm_DeepSleepMode
Inputs	None

Outputs	None
Description	<p>API to trigger the DEEP SLEEP mode (CPU and peripherals off). This is the highest power saving sleep mode. It can be waken-up some HW source only, in particular:</p> <ul style="list-style-type: none">• Rtcc alarm• INTO interrupt• DsWdg• Reset (HW only)
Usage Examples	<p>Trigger the DEEP SLEEP mode:</p> <pre>Sysm_DeepSleepMode();</pre>
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: EventIdle EventReceived 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: EventStructureType MyEvent; GenerateEvt(&MyEvent);
Notes	None

ReceiveEvt

Service Name	ReceiveEvt
Inputs	EventStructureType *Event – Pointer to the Event global variable
Outputs	EventStatusType – Possible values: EventIdle EventReceived 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: EventStructureType MyEvent; If (ReceiveEvt(&MyEvent)) { /* Do something */ }
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 UINT32 Timeout
Outputs	None
Description	API to start a software timer.
Usage Examples	Start a SW timer with timeout of 10s: SwTimerType MyTimer; 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: OsTmr_StopTimer(&MyTimer);
Notes	None

OsTmr_Wait

Service Name	OsTmr_Wait
Inputs	SwTimerType *WaitTimer UINT32 DelayMs
Outputs	DelayNotExpired → The set delay is still not expired 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)

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

OsTmr_GetTimerStatus

Service Name	OsTmr_GetTimerStatus
Inputs	SwTimerType *Timer
Outputs	SwTmrNotExpired → The sw timer is still not expired SwTimerExpired → The sw timer is expired SwTimerDisabled → The sw timer is disabled (stopped)
Description	API that checks the software timer status.
Usage Examples	Check if a SW timer is expired: <pre> /* Check if expired */ If (OsTmr_GetTimerStatus(&MyTimer) == SwTimerExpired) { /* Do something */ } </pre>
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	<p>Get elapsed time:</p> <pre> UINT32 ElapsedTimeMs; /* Get elapsed time */ ElapsedTimeMs = OsTmr_GetElapsedTime(&MyTimer); </pre>
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	<p>Get remaining time:</p> <pre> UINT32 RemainingTimeMs; /* Get remaining time */ RemainingTimeMs = OsTmr_GetRemainingTime (&MyTimer); </pre>
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) UINT32 OsAlarmTimeout – Timeout in ms before the alarm fires 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: <pre>/* User callback implementation */ void MyAlrmCbk (void) { /* My callback implementation */ } /* Alarm set */ OsAlrm_SetOsAlarm(1, 5000, MyAlrmCbk);</pre>
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