USI Host User's Guide

Introduction

The Microchip Universal Serial Interface (USI) is a wrapper that serializes components and services of the different interfaces described in the PRIME or G3 specification. Microchip PRIME or G3 firmware stacks connect easily with external devices using this wrapper. All Microchip PLC PC tools use this USI interface.

This User's Guide describes how to configure a Host device to use the source code of the USI for PRIME or G3, and how to customize the source code to fulfill the customer requirements.

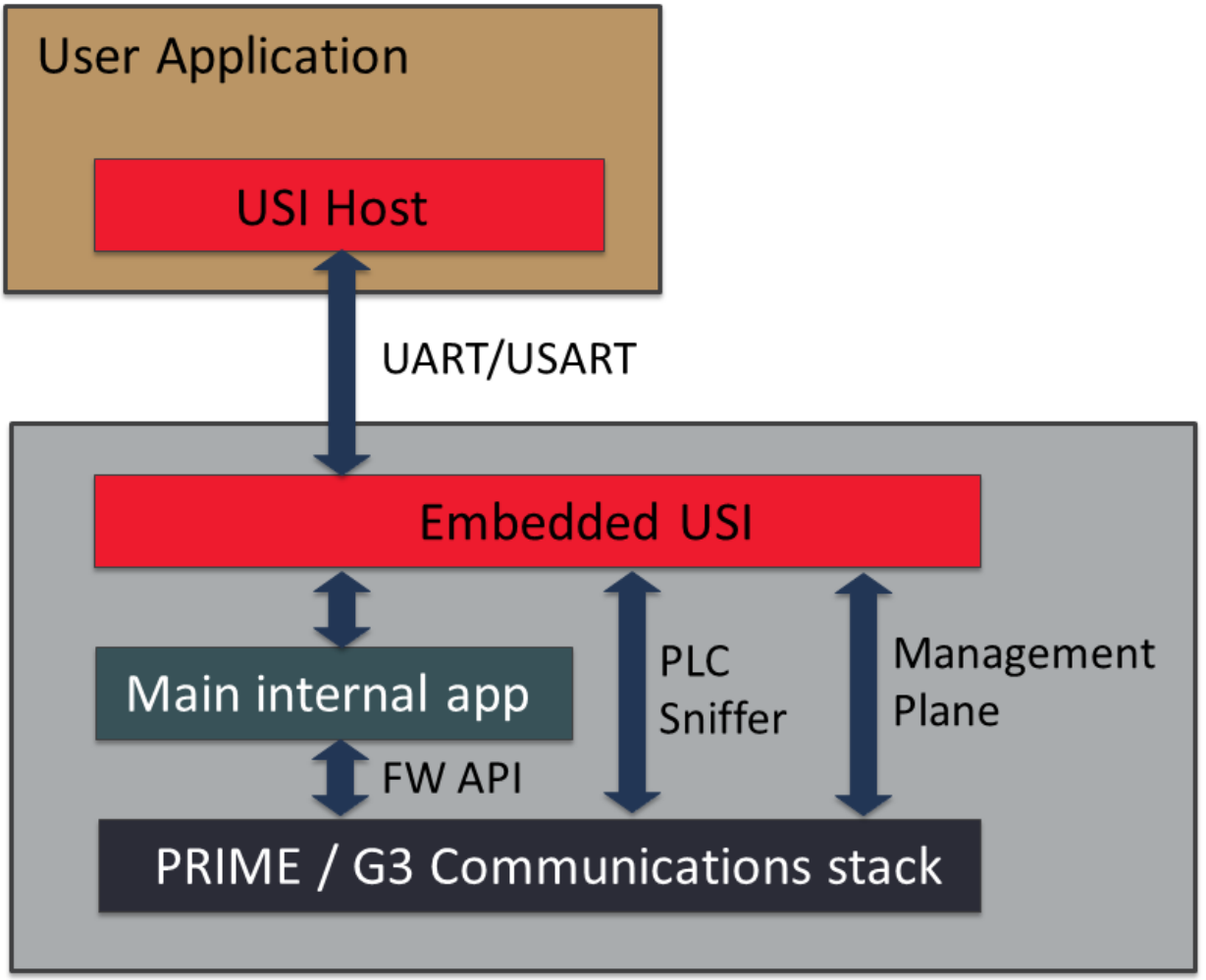
Table of Contents

1. PLC Universal Serial Interface

1.1 Overview

The USI enables communication between any external user application and the embedded firmware stacks through multiple serial ports. Therefore, when an external application communicates with the PRIME or G3 stack is implemented through a serial device, the typical resulting structure is shown in the figure below.

Figure 1-1. Structure of the USI Architecture



As indicated in the preceding image, the USI is divided into two parts:

- **Embedded USI**: The wrapper part of the Hardware Abstraction Layer (HAL) is responsible for the interface between the firmware stack and the serial communications channel. It can also be used by the main internal application to communicate through the serial port.
- **USI Host**: An example of wrapper to be integrated in the external application which allows to interface with the Microchip PLC firmware stacks using the serial communication channel. It is written in C Programming Language, and it can run in an embedded system (either bare-metal or Linux®) or in a general purpose computer. The user can also develop their own independent implementation of the USI Host following the indications of this user's guide and examples.

Important: This User's Guide describes the USI Host example implementation written in C Language. Another possibility to communicate with the USI serialized interface of the G3 or PRIME stacks is to use the *python libs*. These libraries provide a simple way to develop scripts in Python Language that make use of the G3 or PRIME primitives through a serial connection.

1.2 Functional Description

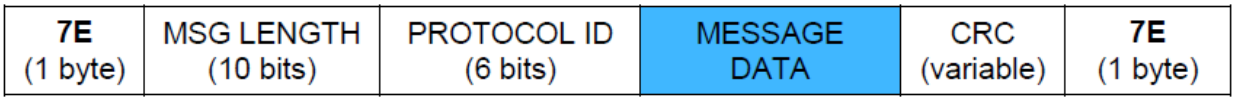
For serial transmissions from the firmware stack (PRIME or G3) *'uplink'*, the embedded USI provides a function that packs and sends each primitive through the serial link to the external application. Upon reception, the USI Host unpacks the message, extracts received parameters, and calls a user configured callback function.

For serial transmission to the firmware stack (PRIME or G3) *'downlink'*, the USI Host provides the external application with a set of functions identical to the ones described in the PRIME or G3 stacks. When any of these functions are invoked, the primitive is packed and sent through the serial link to the remote device. Upon reception, the Embedded USI unpacks it, extracts received parameters, and transfers the control to the corresponding stack module.

1.3 Frame Format

The USI frame format is based on the Serial Communications Profile of the Management Plane defined in the PRIME specification as shown in the figure below.

Figure 1-2. USI General Frame Format



The frame starts and ends with 0x7E (delimiter). The frame delimiters will be escaped if they appear in message data. The description of each field is as follows:

- MSG LENGTH: Length in bytes of the message data (message length, protocol ID and CRC bytes are excluded).
- PROTOCOL ID: Protocol in the frame.
- MESSAGE DATA: Variable length field that contains the data of the exchanged primitive.
- CRC: Error correction code for the message. The CRC field can have a different length depending on the protocol.

Table 1-1. USI Protocols and Associated CRC Size

Protocol	Description	ID	CRC Size (bits)
PROTOCOLS_MNGP_PRIME	Management Plane defined in the PRIME specification	0x00 - 0x07	32
PROTOCOL_SNIF_PRIME	PRIME Sniffer	0x13	16
PROTOCOL_PHY_SERIAL_PRIME	PRIME PHY serial layer	0x1F	16
PROTOCOL_PHY_TESTER	Used for PHY Tester Tool provided with the evaluation kit in order to serialize the API of the PHY layer	0x22	16
PROTOCOL_SNIF_G3	G3 Sniffer	0x23	16
PROTOCOL_MAC_G3	G3 MAC Layer serialization	0x24	16
PROTOCOL_ADP_G3	G3 ADP Layer serialization	0x25	16
PROTOCOL_COORD_G3	G3 Coordinator serialization	0x26	16

.....continued			
Protocol	**Description**	**ID**	**CRC Size (bits)**
PROTOCOL_PHY_RF215_TESTER	Used to interact like the PHY tester tool with the RF215 PHY layer	0x28	16
PROTOCOL_PRIME_API	PRIME API	0x30	8
PROTOCOL_INTERNAL	Reserved for internal use	0x3E	User defined
PROTOCOL_USER_DEFINED	Defined by the user if required	0xFE	User defined

1.4 USI PRIME Protocols

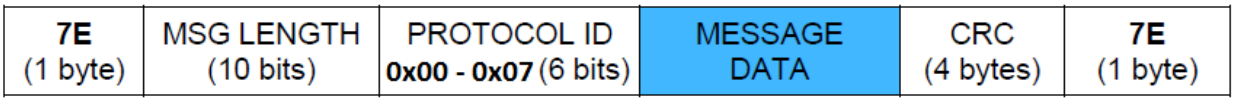
USI is able to serialize the following PRIME interfaces and services:

- PRIME Management Plane
- PRIME Sniffer
- PRIME API
- User Application

1.4.1 MNGP PRIME

This service refers to the different protocols defined in the Serial Communication Profile of the Management Plane described in the PRIME specification. The frame format is as follows:

Figure 1-3. MNGP PRIME USI Frame Format



Important: The PRIME certification requires the protocol accessible through a serial port.

The management functions described in the PRIME specification are as follows:

Table 1-2. USI MNGP PRIME Protocols

Protocols MNGP PRIME	ID	Description
PROTOCOL_MNGP_PRIME_GETQRY	0x00	This protocol is used to get a PIB with information from the node.
PROTOCOL_MNGP_PRIME_GETRSP	0x01	This protocol is the response to PROTOCOL_MNGP_PRIME_GETQRY.
PROTOCOL_MNGP_PRIME_SET	0x02	This protocol is used to set a PIB and thus modify the behavior of the node.
PROTOCOL_MNGP_PRIME_RESET	0x03	This protocol is used to reset statistics.
PROTOCOL_MNGP_PRIME_REBOOT	0x04	This protocol is used to reboot the node.
PROTOCOL_MNGP_PRIME_FU	0x05	This protocol is used to exchange FU protocol frames. In this way, it is possible to perform a FU process through the serial port.
PROTOCOL_MNGP_PRIME_GETQRY_EN	0x06	This protocol is used to get a PIB with information from the node in an enhanced way.
PROTOCOL_MNGP_PRIME_GETRSP_EN	0x07	This protocol is the response to PROTOCOL_MNGP_PRIME_GETQRY_EN.

1.4.2 PRIME Sniffer

The PRIME Sniffer is a service of Microchip PRIME FW Stack that uses the PHY layer to provide received PLC or RF traffic from the PRIME network. The USI is able to serialize and treat this service independently. This serialization can be directly provided to Microchip PLC PC Tools to be analyzed or saved for later use.

The following figures show the USI frame format of the sniffer frames. In this case, the field MESSAGE DATA that appears in the USI general frame format (See Figure 1-3) is divided into two fields: HEADER and PDU SNIFFER MESSAGE

Figure 1-4. PRIME Sniffer USI Frame Format

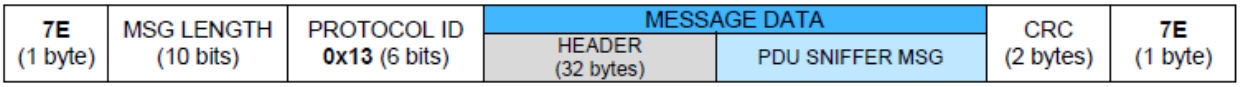
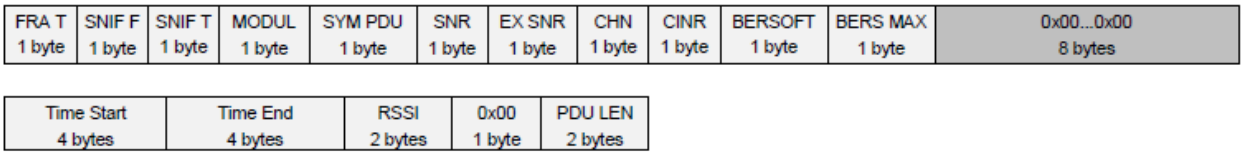


Figure 1-5. PRIME Sniffer USI Frame Header Field



As indicated in the prededing images, sniffer frames contain the received PDU (MAC encapsulation following the PRIME specification) and some additional information related to the PHY layer, which is included in the header part:

- FRA T:
  - SPEC1_3: PDU type of the received frame (A, B, BC), see values in the *sniffer_if.h.h* file
  - SPEC1_4: PDU type of the received frame (A, B, BC), see values in the *sniffer_if.h.h* and *rf215_sniffer_if_prime.c* file
- SNIF F: Sniffer frame version: 0x14 for current version
- SNIF T:
  - SPEC1_3: Sniffer type version: 0x11 for PL360
  - SPEC1_4: Sniffer type version: 0x11 for PL360, 0x13 for RF
- MODUL:
  - SPEC1_3: Modulation scheme of the received frame, see modulation values in the *atpl360_comm.h* file. The modulation scheme of frames received in the serial PHY layer is set to 0x0F.
  - SPEC1_4: Modulation scheme of the received frame, see modulation values in the *atpl360_comm.h* and *rf215_sniffer_if_prime.c* files.
- SYM PDU: Length of the PDU in PHY symbols
- SNR (Note 1): PRIME defined measurement of the SNR (from 0 to 7)
- EX SNR (Note 1): High precision SNR
- CHN (Note 2): Channel in which the frame has been received
- CINR (Note 1): Minimum Carrier to Interference Noise Ratio
- BERSOFT (Note 1): Viterbi soft bit error rate value
- BERS MAX (Note 1): Viterbi soft bit error rate maximum value
- Time Start or Time End: High precision internal counter to measure length (time) of the PDUs in microseconds in the PL360 platform
- RSSI: Average RSSI in dBuV
- PDU LEN: Length of the PDU in bytes

Notes:

1. Not available in RF. The padding field changes accordingly.
2. The CHN in RF has two bytes.

For additional informaiton about the PHY information, refer these documents:

- SPEC1_3: *PL360* and *PL460*
- SPEC1_4: *PL360*, *PL460* and *AT86RF215*

1.4.3 PRIME API

This protocol consists of the serialization of the PRIME API primitives. The Microchip PRIME Stack can provide the Application Programming Interface (PRIME API) through a serial interface as an independent protocol of USI. It is only available when the user application contains the modem example.

Figure 1-6. PRIME API USI Frame Format

7E	MSG LENGTH (10 bits)	PROTOCOL ID 0x30 (6 bits)	MESSAGE DATA			CRC (1 byte)	7E
			LENGTH Extended (1 bit)	PRIME API COMMAND (7 bits)	Primitive function parameters		

The following three fields are available within the general MESSAGE DATA field:

- LENGTH Extended: As the information contained in the message data can exceed the size reserved for MSG length (10 bits), a bit has been added to increase the message length size. In this field the most significant bit of the message length is codified.
- PRIME API COMMAND: This field refers to the primitive included in the message, using the same primitives described in the PRIME API interface description. The values for those primitives are defined in enumerator `prime_api_cmd_t` in the *modem.h* file.
- Primitive function parameters: The serialization of each primitive directly concatenates the different parameters included in the primitive function, with the most significant byte of a variable always on the left. The only exception is that the length of buffers is always placed before the buffer itself so that the data can be inserted and extracted easily. This is applicable to all primitives in the PRIME API.
  - Figure 1-7 shows how a serialized primitive looks. It is based on the MAC_ESTABLISH.request, which is mapped into the following type:

```
typedef void (*mac_establish_request_t)(uint8_t *puc_eui48, uint8_t uc_type, uint8_t
*puc_data, uint16_t us_data_len, uint8_t uc_arq, uint8_t uc_cfbytes);
```

Figure 1-7. Message Data for MAC_ESTABLISH.request Primitive

MAC_ADDR (6 bytes)	CON_TYPE (1 byte)	DATA_LEN (2 bytes)	DATA (variable)	ARQ (1 byte)	CFP_BYTES (1 byte)

Important: The Base Management primitives already include the PRIME API command as the first function parameter, so exceptionally sending it twice is not required.

1.5 USI G3 Protocols

The application example *adp_mac_serialized* included in the Microchip G3 stack contains a USI interface that is able to serialize the following G3 interfaces and services:

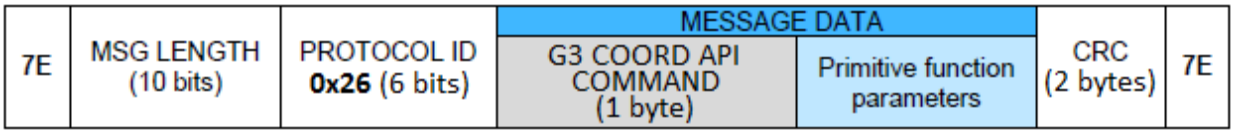
- G3 Coordinator

- G3 ADP
- G3 MAC
- G3 Sniffer

1.5.1 G3 Coordinator Access

This protocol consists of the serialization of the G3 coordinator API different primitives.

Figure 1-8. G3 COORD API USI Frame Format



The following two fields are available within the general MESSAGE DATA field:

- G3 COORD API COMMAND: Available commands are enumerated in Table 1-3 and in the `G3.h` file.
- Primitive function parameters: The serialization of each primitive concatenates the different parameters included in the function. This serialization order matches with the parameter order indicated in the G3 specification for each primitive. Its type and length can be seen in the examples provided.

Table 1-3. G3 Coordinator API Commands

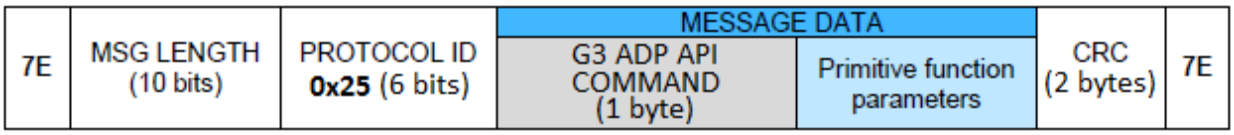
G3 Coordinator API Command	ID
G3_SERIAL_MSG_COORD_INITIALIZE	0x01
G3_SERIAL_MSG_COORD_SET_REQUEST	0x02
G3_SERIAL_MSG_COORD_GET_REQUEST	0x03
G3_SERIAL_MSG_COORD_KICK_REQUEST	0x04
G3_SERIAL_MSG_COORD_REKEYING_REQUEST	0x05
G3_SERIAL_MSG_COORD_SET_CONFIRM	0x06
G3_SERIAL_MSG_COORD_GET_CONFIRM	0x07
G3_SERIAL_MSG_COORD_JOIN_INDICATION	0x08
G3_SERIAL_MSG_COORD_LEAVE_INDICATION	0x09

Note: All the primitives described above have similar names to the functions described in the G3 specification.

1.5.2 G3 ADP Access

This protocol consists of the serialization of the G3 ADP API different primitives.

Figure 1-9. G3 ADP API USI Frame Format



The following two fields are available within the general MESSAGE DATA field:

- G3 ADP API COMMAND: Available commands are enumerated in Table 1-4, Table 1-5 and in the `G3.h` file.

- Primitive function parameters: The serialization of each primitive concatenates the different parameters included in the function. This serialization order matches with the parameter order indicated in the G3 specification for each primitive. Its type and length can be seen in the examples provided.

Table 1-4. G3 ADP API Commands

G3 ADP API Command	ID
G3_SERIAL_MSG_ADP_INITIALIZE	0x0A
G3_SERIAL_MSG_ADP_DATA_REQUEST	0x0B
G3_SERIAL_MSG_ADP_DISCOVERY_REQUEST	0x0C
G3_SERIAL_MSG_ADP_NETWORK_START_REQUEST	0x0D
G3_SERIAL_MSG_ADP_NETWORK_JOIN_REQUEST	0x0E
G3_SERIAL_MSG_ADP_NETWORK_LEAVE_REQUEST	0x0F
G3_SERIAL_MSG_ADP_RESET_REQUEST	0x10
G3_SERIAL_MSG_ADP_SET_REQUEST	0x11
G3_SERIAL_MSG_ADP_GET_REQUEST	0x12
G3_SERIAL_MSG_ADP_LBP_REQUEST	0x13
G3_SERIAL_MSG_ADP_ROUTE_DISCOVERY_REQUEST	0x14
G3_SERIAL_MSG_ADP_PATH_DISCOVERY_REQUEST	0x15
G3_SERIAL_MSG_ADP_MAC_SET_REQUEST	0x16
G3_SERIAL_MSG_ADP_MAC_GET_REQUEST	0x17
G3_SERIAL_MSG_ADP_NO_IP_DATA_REQUEST	0x18
G3_SERIAL_MSG_ADP_DATA_CONFIRM	0x1E
G3_SERIAL_MSG_ADP_DATA_INDICATION	0x1F
G3_SERIAL_MSG_ADP_NETWORK_STATUS_INDICATION	0x20
G3_SERIAL_MSG_ADP_DISCOVERY_CONFIRM	0x21
G3_SERIAL_MSG_ADP_NETWORK_START_CONFIRM	0x22
G3_SERIAL_MSG_ADP_NETWORK_JOIN_CONFIRM	0x23
G3_SERIAL_MSG_ADP_NETWORK_LEAVE_CONFIRM	0x24
G3_SERIAL_MSG_ADP_NETWORK_LEAVE_INDICATION	0x25
G3_SERIAL_MSG_ADP_RESET_CONFIRM	0x26
G3_SERIAL_MSG_ADP_SET_CONFIRM	0x27
G3_SERIAL_MSG_ADP_GET_CONFIRM	0x28
G3_SERIAL_MSG_ADP_LBP_CONFIRM	0x29
G3_SERIAL_MSG_ADP_LBP_INDICATION	0x2A
G3_SERIAL_MSG_ADP_ROUTE_DISCOVERY_CONFIRM	0x2B
G3_SERIAL_MSG_ADP_PATH_DISCOVERY_CONFIRM	0x2C
G3_SERIAL_MSG_ADP_MAC_SET_CONFIRM	0x2D
G3_SERIAL_MSG_ADP_MAC_GET_CONFIRM	0x2E
G3_SERIAL_MSG_ADP_BUFFER_INDICATION	0x2F
G3_SERIAL_MSG_ADP_DISCOVERY_INDICATION	0x30
G3_SERIAL_MSG_ADP_PREQ_INDICATION	0x31
G3_SERIAL_MSG_ADP_UPD_NON_VOLATILE_DATA_INDICATION	0x32
G3_SERIAL_MSG_ADP_ROUTE_NOT_FOUND_INDICATION	0x33

Note: All of the primitives described above have similar names to the functions described in the G3 specification.

Last G3 releases include additional primitives to handle LBP functionality on G3 ADP serialization.

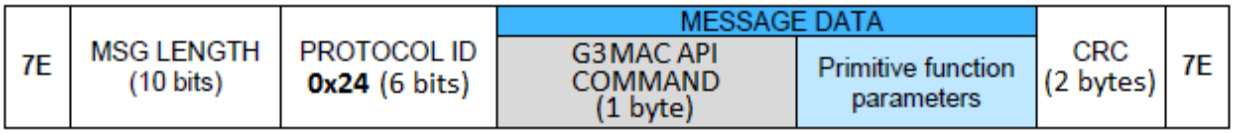
Table 1-5. G3 ADP API LBP Extension Commands

G3 ADP API LBP Extension Commands	ID
G3_SERIAL_MSG_LBP_SET_REQUEST	0x3C
G3_SERIAL_MSG_LBP_DEV_FORCE_REGISTER	0x3D
G3_SERIAL_MSG_LBP_COORD_KICK_DEVICE	0x3E
G3_SERIAL_MSG_LBP_COORD_REKEY	0x3F
G3_SERIAL_MSG_LBP_COORD_SET_REKEY_PHASE	0x40
G3_SERIAL_MSG_LBP_COORD_ACTIVATE_NEW_KEY	0x41
G3_SERIAL_MSG_LBP_COORD_SHORT_ADDRESS_ASSIGN	0x42
G3_SERIAL_MSG_LBP_SET_CONFIRM	0x46
G3_SERIAL_MSG_LBP_COORD_JOIN_REQUEST_INDICATION	0x47
G3_SERIAL_MSG_LBP_COORD_JOIN_COMPLETE_INDICATION	0x48
G3_SERIAL_MSG_LBP_COORD_LEAVE_INDICATION	0x49

1.5.3 G3 MAC

This protocol consists of the serialization of the G3 MAC API different primitives.

Figure 1-10. G3 MAC API USI Frame Format



The following two fields are availabe within the general MESSAGE DATA field:

- G3 MAC API COMMAND: Available commands are enumerated in Table 1-6 and Table 1-7 tables and in the `G3.h` file.
- Primitive function parameters: The serialization of each primitive concatenates the different parameters included in the function. This serialization order matches with the parameter order indicated in the G3 specification for each primitive. Its type and length can be seen in the examples provided.

Table 1-6. G3 MAC PLC API Commands

G3 MAC PLC API Command	ID
G3_SERIAL_MSG_MAC_INITIALIZE	0x32
G3_SERIAL_MSG_MAC_DATA_REQUEST	0x33
G3_SERIAL_MSG_MAC_GET_REQUEST	0x34
G3_SERIAL_MSG_MAC_SET_REQUEST	0x35
G3_SERIAL_MSG_MAC_RESET_REQUEST	0x36
G3_SERIAL_MSG_MAC_SCAN_REQUEST	0x37
G3_SERIAL_MSG_MAC_START_REQUEST	0x38
G3_SERIAL_MSG_MAC_DATA_CONFIRM	0x3C
G3_SERIAL_MSG_MAC_DATA_INDICATION	0x3D
G3_SERIAL_MSG_MAC_GET_CONFIRM	0x3E
G3_SERIAL_MSG_MAC_SET_CONFIRM	0x3F
G3_SERIAL_MSG_MAC_RESET_CONFIRM	0x40
G3_SERIAL_MSG_MAC_SCAN_CONFIRM	0x41
G3_SERIAL_MSG_MAC_BEACON_NOTIFY	0x42
G3_SERIAL_MSG_MAC_START_CONFIRM	0x43

.....continued	
G3 MAC PLC API Command	**ID**
G3_SERIAL_MSG_MAC_COMM_STATUS_INDICATION	0x44
G3_SERIAL_MSG_MAC_SNIFFER_INDICATION	0x45

Table 1-7. G3 MAC RF API Commands

G3 MAC RF API Command	ID
G3_SERIAL_MSG_MAC_INITIALIZE_RF	0x1E
G3_SERIAL_MSG_MAC_DATA_REQUEST_RF	0x1F
G3_SERIAL_MSG_MAC_GET_REQUEST_RF	0x20
G3_SERIAL_MSG_MAC_SET_REQUEST_RF	0x21
G3_SERIAL_MSG_MAC_RESET_REQUEST_RF	0x22
G3_SERIAL_MSG_MAC_SCAN_REQUEST_RF	0x23
G3_SERIAL_MSG_MAC_START_REQUEST_RF	0x24
G3_SERIAL_MSG_MAC_DATA_CONFIRM_RF	0x28
G3_SERIAL_MSG_MAC_DATA_INDICATION_RF	0x29
G3_SERIAL_MSG_MAC_GET_CONFIRM_RF	0x2A
G3_SERIAL_MSG_MAC_SET_CONFIRM_RF	0x2B
G3_SERIAL_MSG_MAC_RESET_CONFIRM_RF	0x2C
G3_SERIAL_MSG_MAC_SCAN_CONFIRM_RF	0x2D
G3_SERIAL_MSG_MAC_BEACON_NOTIFY_RF	0x2E
G3_SERIAL_MSG_MAC_START_CONFIRM_RF	0x2F
G3_SERIAL_MSG_MAC_COMM_STATUS_INDICATION_RF	0x30

Note: All the primitives described above have similar names to the functions described in the G3 specification.

1.5.4 G3 Sniffer

G3 Sniffer is a service of the Microchip G3 FW Stack that uses the PHY layer to provide received PLC or RF traffic from the G3 network. The USI is able to serialize and treat this service independently. This serialization can be directly provided to Microchip PLC PC Tools to be analyzed or saved for later use.

The following figures show the USI frame format of the sniffer frames. In this protocol case the field MESSAGE DATA that appears in the USI general frame format (See Figure 1-2) is divided into two fields: header and PDU SNIFFER MSG.

Figure 1-11. G3 Sniffer USI Frame Format

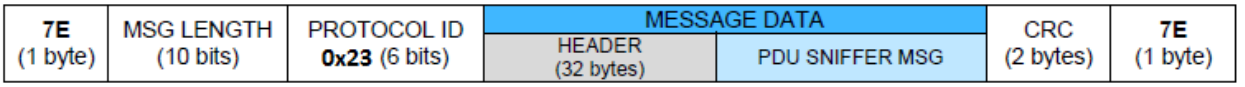
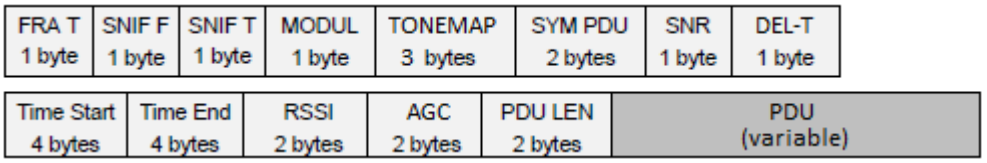


Figure 1-12. G3 Sniffer USI Frame Header Field



Sniffer frames contain the PDU received (MAC encapsulation following G3 specification) and some extra information related to the PHY layer, which is included in the header part as given below:

- FRA T: PDU type of the received frame for G3. Currently 0x00
- SNIF F: Sniffer frame version (0x2)
- SNIF T: Sniffer type version: 0x12 for ATPL360 and 0x14 for RF215
- MODUL:
  - PLC: Modulation type (high) and scheme (low) of the received frame, see modulation values in the `atpl360_comm.h` file
  - RF: Modulation Scheme of the received frame, see modulation values in the *rf215_sniffer_if_g3.c* file
- TONEMAP:
  - PLC: 3 bytes for the TONE-MAP
  - RF: First byte corresponds with the Modulation Type depending on the RF configuration and second byte corresponds with FCS Correction flag, see modulation values in the *rf215_sniffer_if_g3.c* file
- SYM PDU: Length of the PDU in PHY symbols
- SNR: G3 defined measurement of the SNR (LQI). No sense on RF
- DEL-T: Delimiter type. No sense on RF
- Time Start/Time End: High precision internal counter to measure length (time) of the PDUs in microseconds in the PL360 platform
- RSSI: Average RSSI in dBuV
- AGC: Automatic Gain Control (AGC) Factor, no sense on RF
- PDU LEN: Length of the PDU in bytes

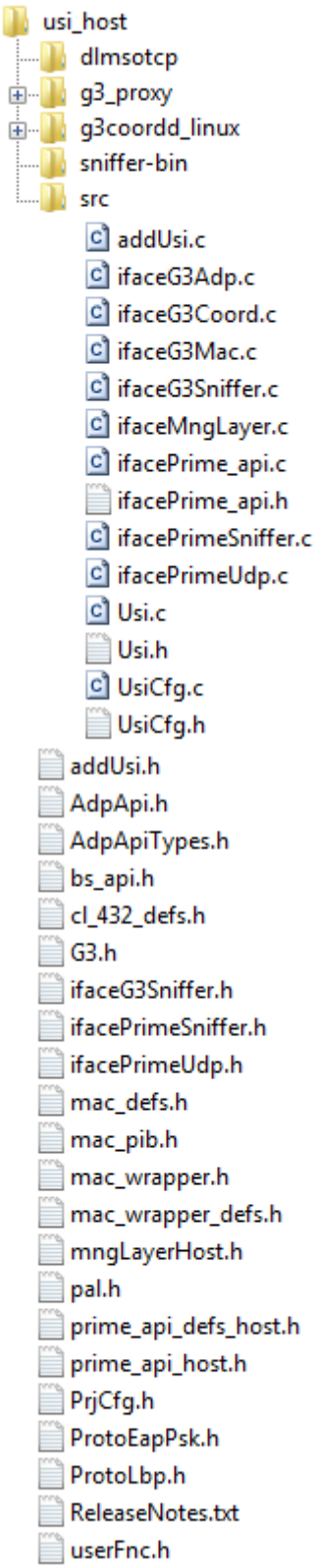
2. Configuration Steps

2.1 USI Host Example Package

The Microchip USI Host package contains the following files and folders (Figure 2-1):

- Root directory: Public interfaces that must be included in the user project, and must be modified or considered by the user. The following sections explain how to use these files.
- *src*: Source code of the USI Host itself. It must be included in the user project.
- *dlmsotcp*: Example described in "AN3285 Serialized DLMS over TCP PRIME Gateway Application Note".
- *primeBN_linux*: Example described in "AN4327 Serialized PRIME Network Manager on Linux®Application Note".
- *g3_proxy*: Example of application that acts as USI through a TCP port, decodes the G3 commands and sends them through the serial port using USI Host.
- *g3coordd_linux*: Example described in "AN3154 Serialized G3-PLC Coordinator on Linux® Application Note" for G3 releases previous to SEF 2.1.0.
- *g3coordd_harmony_linux*: The same than g3coord_linux example for G3 release SEF 2.1.0 and Harmony.
- *sniffer-bin*: Example of sniffer implementation.

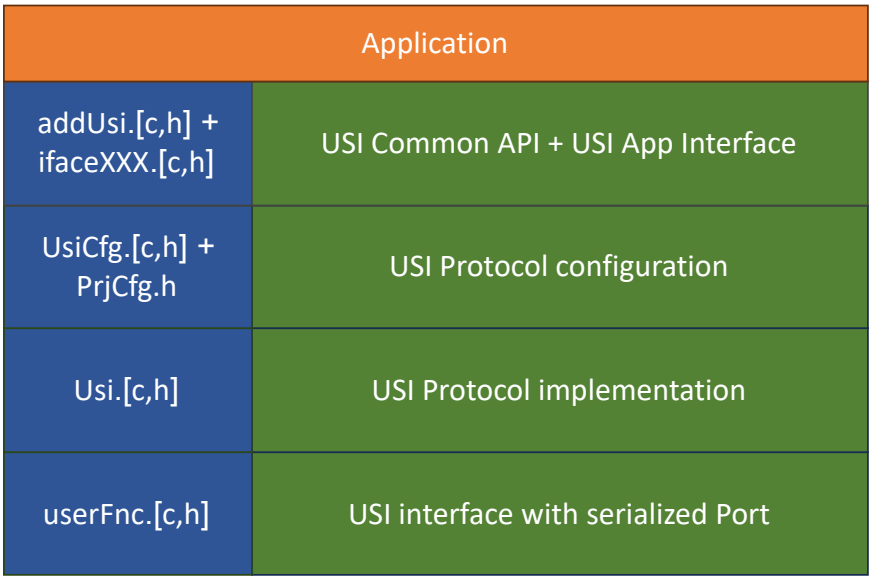
Figure 2-1. USI Host package folder structure



The complete directory must be included in the user project.

The different files required on any application based on USI Host (Figure 2-2) are explained in the following figure.

Figure 2-2. Architecture of an application based on USI Host



2.2 Configuration Header: `PrjCfg.h`

The `PrjCfg.h` header file is used for the USI Host configuration, and the user must include this file with the same name. In the USI Host repository there is a file named `PrjCfg.h` with the example below:

```
#define NUM_PORTS                    2
#define PORT_0                       CONF_PORT(UART_TYPE,3, 115200, 4096, 4096)
#define PORT_1                       CONF_PORT(UART_TYPE,4, 115200, 4096, 4096)
#define NUM_PROTOCOLS                3
#define USE_MNGP_PRIME_PORT          0
#define USE_PROTOCOL_PRIME_API       0
#define USE_PROTOCOL_SNIF_PRIME_PORT 0
/*#define USE_PROTOCOL_PHY_SERIAL_PRIME*/
/*#define USE_PROTOCOL_ADP_G3_PORT*/
/*#define USE_PROTOCOL_COORD_G3_PORT*/
/*#define USE_PROTOCOL_MAC_G3_PORT*/
/*#define USE_PROTOCOL_SNIF_G3_PORT*/
```

`NUM_PORTS` defines the number of ports to be used and users can use up to 4 ports. Ports can be configured in the header file with the macro CONF_PORT defined in the `PrjCfg.h` file, where:

- *Type:* Port type (*UART_TYPE, USART_TYPE or COM_TYPE*)
- *Channel:* COM port number
- *Baudrate:* Port speed
- *txSize:* Buffer size for transmission
- *rxSize:* Buffer size for reception

The available protocols implemented in USI Host are MNGP_PRIME, PRIME_API, PRIME_SNIFFER and PHI_SERIAL for PRIME; and ADP, Coordinator and Sniffer for G3. To use a protocol, the definition of USE_PROTOCOL_xxx must be set with the number of port used for that protocol. Each protocol can be enabled using the appropriate macro. The value of the definition sets the port used (0 to 3) for that protocol.

The port configuration can also be changed once the program is running, using the function `addUsi_ConfigurePort (uint8_t logPort, uint8_t commPort, uint32_t speed)`.

2.3 User Defined Functions: `userFnc.h`

The user must implement the following functions and these functions will typically interface to a serial port to read a character or transmit a message, and depend on the platform where the USI Host runs.

```
/**@brief Open communication port (parameters are defined in PrjCfg.h)
  @param port_type: Port Type (UART_TYPE, USART_TYPE, COM_TYPE)
  @param commPort: Physical Communication Port
  @param speed:Communication speed
  @return 0 on success.
*/
int8_t addUsi_Open(uint8_t port_type, uint8_t port, uint32_t bauds);

/**@brief Transmit a message through the interface
  @param port: port number configured in PrjCfg.h
  @param msg: buffer holding the message
  @param msglen:buffer length
  @return the number of bytes written.
*/
uint16_t addUsi_TxMsg(uint8_t port, uint8_t *msg, uint16_t msglen);

/**@brief Read char from port
  @param port: port number configured in PrjCfg.h
  @param c: pointer to a character
  @return 0 if a charater has been read, otherwise -1.
*/
int8_t addUsi_RxChar(uint8_t port, uint8_t *c);
```

2.4 Initialization

The user must call *addUsi_Init()* at the beginning. If the port to be used is different from the port defined in the `PrjCfg.h` file, it must be configured before initializing the USI Host with the function `addUsiConfigurePort()`.

2.5 Protocol Interfaces and Setting Callbacks

The user can set the callback functions which will be called when a confirm or indication is received from the device.

2.5.1 PRIME API

Important: PRIME API is defined in the "PRIME Specification" documentation. For implementation details, refer to Microchip "PRIME 1.x FW Stack for Service Nodes for Base Node" document.

PRIME API functions are defined in *prime_api_host.h*. This file contains the interface to PRIME API.

There are two type of functions:

- Request functions to send messages or data:

```
prime_cl_null_mlme_get_request(uint16_t us_pib_attrib)
```

- Callbacks to manage confirm and indication messages.

The callbacks are initialized with the following functions:

```
prime_cl_null_set_callbacks(prime_cl_null_callbacks_t *px_prime_cbs);
prime_cl_432_set_callbacks(prime_cl_432_callbacks_t *px_prime_cbs);
```

Where the argument is a pointer to a struct with the callback functions:

```
typedef struct {
    prime_cl_null_establish_ind_cb_t prime_cl_null_establish_ind_cb;
    prime_cl_null_establish_cfm_cb_t prime_cl_null_establish_cfm_cb;
```

```
        prime_cl_null_release_ind_cb_t prime_cl_null_release_ind_cb;
        prime_cl_null_release_cfm_cb_t prime_cl_null_release_cfm_cb;
        prime_cl_null_join_ind_cb_t prime_cl_null_join_ind_cb;
        prime_cl_null_join_cfm_cb_t prime_cl_null_join_cfm_cb;
        prime_cl_null_leave_ind_cb_t prime_cl_null_leave_ind_cb;
        prime_cl_null_leave_cfm_cb_t prime_cl_null_leave_cfm_cb;
        prime_cl_null_data_ind_cb_t prime_cl_null_data_ind_cb;
        prime_cl_null_data_cfm_cb_t prime_cl_null_data_cfm_cb;
        prime_cl_null_plme_reset_cfm_cb_t prime_cl_null_plme_reset_cfm_cb;
        prime_cl_null_plme_sleep_cfm_cb_t prime_cl_null_plme_sleep_cfm_cb;
        prime_cl_null_plme_resume_cfm_cb_t prime_cl_null_plme_resume_cfm_cb;
        prime_cl_null_plme_testmode_cfm_cb_t prime_cl_null_plme_testmode_cfm_cb;
        prime_cl_null_plme_get_cfm_cb_t prime_cl_null_plme_get_cfm_cb;
        prime_cl_null_plme_set_cfm_cb_t prime_cl_null_plme_set_cfm_cb;
        prime_cl_null_mlme_register_ind_cb_t prime_cl_null_mlme_register_ind_cb;
        prime_cl_null_mlme_register_cfm_cb_t prime_cl_null_mlme_register_cfm_cb;
        prime_cl_null_mlme_unregister_ind_cb_t prime_cl_null_mlme_unregister_ind_cb;
        prime_cl_null_mlme_unregister_cfm_cb_t prime_cl_null_mlme_unregister_cfm_cb;
        prime_cl_null_mlme_promote_ind_cb_t prime_cl_null_mlme_promote_ind_cb;
        prime_cl_null_mlme_mp_promote_ind_cb_t prime_cl_null_mlme_mp_promote_ind_cb;
        prime_cl_null_mlme_promote_cfm_cb_t prime_cl_null_mlme_promote_cfm_cb;
        prime_cl_null_mlme_mp_promote_cfm_cb_t prime_cl_null_mlme_mp_promote_cfm_cb;
        prime_cl_null_mlme_demote_ind_cb_t prime_cl_null_mlme_demote_ind_cb;
        prime_cl_null_mlme_mp_demote_ind_cb_t prime_cl_null_mlme_mp_demote_ind_cb;
        prime_cl_null_mlme_demote_cfm_cb_t prime_cl_null_mlme_demote_cfm_cb;
        prime_cl_null_mlme_mp_demote_cfm_cb_t prime_cl_null_mlme_mp_demote_cfm_cb;
        prime_cl_null_mlme_reset_cfm_cb_t prime_cl_null_mlme_reset_cfm_cb;
        prime_cl_null_mlme_get_cfm_cb_t prime_cl_null_mlme_get_cfm_cb;
        prime_cl_null_mlme_list_get_cfm_cb_t prime_cl_null_mlme_list_get_cfm_cb;
        prime_cl_null_mlme_set_cfm_cb_t prime_cl_null_mlme_set_cfm_cb;
    } prime_cl_null_callbacks_t;
```

Each callback must have the appropriate parameters. The definition of the callback format is in the *prime_api_defs_host.h* file. If a callback is set to NULL, the corresponding confirm or indication message will not be forwarded to any callback function.

```
    typedef struct {
     prime_cl_432_establish_cfm_cb_t prime_cl_432_establish_cfm_cb;
     prime_cl_432_release_cfm_cb_t prime_cl_432_release_cfm_cb;
     prime_cl_432_dl_data_ind_cb_t prime_cl_432_dl_data_ind_cb;
     prime_cl_432_dl_data_cfm_cb_t prime_cl_432_dl_data_cfm_cb;
     prime_cl_432_dl_leave_ind_cb_t prime_cl_432_dl_leave_ind_cb;
     prime_cl_432_dl_join_ind_cb_t prime_cl_432_dl_join_ind_cb;
    } prime_cl_432_callbacks_t;
```

2.5.2 Base Management

Important: This API is only defined for Base Nodes. Refer to Microchip "PRIME 1.x FW Stack for Base Node documentation" for additional information.

This API defines the interface functions for several Base Node features, such as the Firmware Update Protocol, Network Events, Prime Profile (for remote PIBs access), Zero Cross, and Whitelist Management. The protocol interface and callback interfaces of these features are defined in the *prime_api_host.h* file.

The base management callbacks are initialized with the following function:

```
void bmng_set_callbacks(prime_bmng_callbacks_t *px_fup_cbs);
```

Where the argument is a pointer to a struct with all callback functions:

```
    typedef struct {
        bmng_fup_ack_ind_cb_t                   fup_ack_ind_cb;
        bmng_fup_error_ind_cb_t                 fup_error_ind_cb;
        bmng_fup_version_ind_cb_t               fup_version_ind_cb;
        bmng_fup_status_ind_cb_t                fup_status_ind_cb;
        bmng_fup_kill_ind_cb_t                  fup_kill_ind_cb;
        bmng_network_event_ind_cb_t             network_event_ind_cb;
```

```
        bmng_pprof_ack_ind_cb_t                 pprof_ack_ind_cb;
        bmng_pprof_get_response_cb_t            pprof_get_response_cb;
        bmng_pprof_get_enhanced_response_cb_t   pprof_get_enhanced_response_cb;
        bmng_pprof_zerocross_response_cb_t      pprof_zerocross_response_cb;
        bmng_pprof_zc_diff_response_cb_t        pprof_zc_diff_response_cb;
        bmng_whitelist_ack_cb_t                 whitelist_ack_cb;
    } prime_bmng_callbacks_t;
```

Each callback must have the appropriate parameters. The definition of the callback format is in the *prime_api_defs_host.h* file. If a callback is set to NULL, the corresponding confirm or indication message will not be forwarded to any callback function.

2.5.3 PRIME Management Protocol

Management Protocol (MNGP) is the standard protocol defined by the PRIME specification. The interface is defined in the *mngLayerHost.h file*.

To send a message using Management Protocol, follow these steps:

1. Create a message*: mngLay_NewMsg(uint8_tcmd);*.
2. Add one or more of these queries (of the same kind):
   - Get PIB: *mngLay_AddGetPibQuery(uint16_t pib, uint8_t index);*
   - Set PIB: *mngLay_AddSetPib(uint16_t pib, uint16_t length, uint8_t* msg);*
   - Reset Statistics: *mngLay_AddResetStats(uint16_t pib, uint8_t index);*
   - FW Upgrade Message: *mngLay_AddFUMsg(uint16_t length, uint8_t* msg);*
   - Bridge Message: *mngLay_BridgeMsg(uint16_t length, uint8_t* msg);*
3. Send request message: *mngLay_SendMsg();*.

The command in *mngLay_NewMsg* must be one of the following:

```
/// Protocol ID to serialize (USI)
#define MNGP_PRIME                      0x00
#define MNGP_PRIME_GETQRY               0x00
#define MNGP_PRIME_GETRSP               0x01
#define MNGP_PRIME_SET                  0x02
#define MNGP_PRIME_RESET                0x03
#define MNGP_PRIME_REBOOT               0x04
#define MNGP_PRIME_FU                   0x05
#define MNGP_PRIME_EN_PIBQRY            0x06
#define MNGP_PRIME_EN_PIBRSP            0x07
```

The query must be consistent with the protocol ID selected in *mngLay_NewMsg.* An example for sending a *MNGP_PRIME_GETQRY* message is shown below:

```
uint16_t pib_attrib;
mngLay_NewMsg(MNGP_PRIME_GETQRY);
mngLay_AddGetPibQuery(pib_attrib, 0);
mngLay_SendMsg();
```

There is only one callback, which is set by the following function:

```
void mngp_set_rsp_cb(void (*sap_handler)(uint8_t* ptrMsg, uint16_t len));
```

The only MNGP command sent by the device is `MNGP_PRIME_GET_RESP`. This callback will retrieve a buffer with the whole MNGP message (USI header and CRC are removed). The user must know the protocol and process the message.

2.5.4 PRIME Sniffer

This interface is simple and is defined in the *ifacePrimeSniffer.h* file. Only two following functions are available:

- One function to set the callback where sniffer messages will be sent:

```
void prime_sniffer_set_cb(void (*sap_handler)(uint8_t* msg, uint16_t len));
```

- One function to set the channel:

```
void prime_sniffer_set_channel(uint8_t uc_channel);
```

2.5.5 G3 ADP API

Important: The G3 ADP API is defined in the G3 standard and in Microchip "G3 FW Stack User's Guide". Refer to those documents about API definitions.

The G3 ADP API functions are defined in the *AdpApi.h* file. This file contains the interface with G3 ADP API, which is the entry point to the G3 stack. This file is a copy of the embedded, the G3 USI Host interface is same as the one embedded stack provides.

There are two type of functions:

- Request functions to send messages or data:

```
void AdpGetRequest(uint32_t u32AttributeId, uint16_t u16AttributeIndex);
```

- Callbacks to manage confirm and indication messages received.

When a confirm or an indication message is received, the corresponding callback is called. The callbacks are initialized when the ADP layer is initialized:

```
    void AdpInitialize(struct TAdpNotifications *pNotifications, enum TAdpBand band, enum
EAdpDeviceType);
```

where,

- The first argument is a pointer to a struct with the callback functions:

```
struct TAdpNotifications {
    AdpDataConfirm fnctAdpDataConfirm;
    AdpDataIndication fnctAdpDataIndication;
    AdpDiscoveryConfirm fnctAdpDiscoveryConfirm;
    AdpDiscoveryIndication fnctAdpDiscoveryIndication;
    AdpNetworkStartConfirm fnctAdpNetworkStartConfirm;
    AdpNetworkJoinConfirm fnctAdpNetworkJoinConfirm;
    AdpNetworkLeaveIndication fnctAdpNetworkLeaveIndication;
    AdpNetworkLeaveConfirm fnctAdpNetworkLeaveConfirm;
    AdpResetConfirm fnctAdpResetConfirm;
    AdpSetConfirm fnctAdpSetConfirm;
    AdpMacSetConfirm fnctAdpMacSetConfirm;
    AdpGetConfirm fnctAdpGetConfirm;
    AdpMacGetConfirm fnctAdpMacGetConfirm;
    AdpLbpConfirm fnctAdpLbpConfirm;
    AdpLbpIndication fnctAdpLbpIndication;
    AdpRouteDiscoveryConfirm fnctAdpRouteDiscoveryConfirm;
    AdpPathDiscoveryConfirm fnctAdpPathDiscoveryConfirm;
    AdpNetworkStatusIndication fnctAdpNetworkStatusIndication;
    AdpBufferIndication fnctAdpBufferIndication;
    AdpPREQIndication fnctAdpPREQIndication;
    AdpUpdNonVolatileDataIndication fnctAdpUpdNonVolatileDataIndication;
    AdpRouteNotFoundIndication fnctAdpRouteNotFoundIndication;
};
```

  Each callback must have the appropriate parameters. The definition of the callback format is in the same file (*AdpApi.h)*.

  If a callback is set to NULL, the corresponding confirm or indication message will not be received by any callback function.

- The second argument is the frequency band target, whose type (`TAdpBand`) is defined in the *AdpApiTypes.h file.* Valid values are as follows:
  - *ADP_BAND_CENELEC_A*
  - *ADP_BAND_CENELEC_B*
  - *ADP_BAND_FCC*
  - *ADP_BAND_ARIB*
- The third argument is the device type allows to run a Device or Coordinator device through the ADP Serialization.

2.5.6 G3 MAC API

Important: The G3 MAC API is defined in the G3 standard and in Microchip "G3 FW Stack User's Guide". Refer to those documents about API definitions.

The G3 ADP API functions are defined in the *mac_wrapper.h* file. This file contains the interface with G3 MAC API. This file is a copy of the embedded, the G3 USI Host interface is the same as the one embedded stack provides.

There are two type of functions:

- Request functions to send messages or data:

```
void MacWrapperMlmeGetRequest(struct TMacWrpGetRequest *pParameters);
```

- Callbacks to manage confirm and indication messages received.

When a confirm or an indication message is received, the corresponding callback is called. The callbacks are initialized when the MAC layer is initialized:

```
void MacWrapperInitialize(struct TMacWrpNotifications *pNotifications, uint8_t u8Band);
```

where:

- The first argument is a pointer to a struct with the callback functions:

```
struct TMacWrpNotifications {
    MacWrpDataConfirm m_MacWrpDataConfirm;
    MacWrpDataIndication m_MacWrpDataIndication;
    MacWrpGetConfirm m_MacWrpGetConfirm;
    MacWrpSetConfirm m_MacWrpSetConfirm;
    MacWrpResetConfirm m_MacWrpResetConfirm;
    MacWrpBeaconNotify m_MacWrpBeaconNotifyIndication;
    MacWrpScanConfirm m_MacWrpScanConfirm;
    MacWrpStartConfirm m_MacWrpStartConfirm;
    MacWrpCommStatusIndication m_MacWrpCommStatusIndication;
    MacWrpSnifferIndication m_MacWrpSnifferIndication;
};
```

  Each callback must have the appropriate parameters. The definition of the callback format is in the same file (*mac_wrapper.h)*.

  If a callback is set to NULL, the corresponding confirm or indication message will not be received by any callback function.
- The second argument is the frequency band target (`uint8_t`). Valid values are as follows:
  - *0 = CENELEC_A*
  - *1 = CENELEC_B*
  - *2 = FCC*
  - *3 = ARIB*

2.5.7 G3 COORD API

The G3 COORD API functions are defined in the *bs_api.h* file. This file contains the interface with G3 bootstrap API, which is the entry point to the G3 bootstrap process used by the coordinator. This file is a copy of the embedded: the G3 USI Host interface is the same as the one the embedded stack provides.

There are request functions:

```
void bs_lbp_get_param(uint32_t ul_attribute_id, uint16_t us_attribute_idx,
    struct t_bs_lbp_get_param_confirm *p_get_confirm);
void bs_lbp_set_param(uint32_t ul_attribute_id, uint16_t us_attribute_idx, uint8_t
uc_attribute_len, const uint8_t *puc_attribute_value,
        struct t_bs_lbp_set_param_confirm *p_set_confirm);
void bs_lbp_launch_rekeying();
void bs_lbp_kick_device(uint16_t us_short_address);
```

When a confirm or an indication message is received, the corresponding callback is called.

Confirm callbacks are set in the corresponding request. For example, in the former *bs_lbp_get_param* request function, the last parameter is a pointer confirm callback function.

Indication callbacks are set using functions defined in the same file *(bs_api.h)*:

```
    voidbs_lbp_leave_ind_set_cb(pf_app_leave_ind_cb_t pf_handler);
    voidbs_lbp_join_ind_set_cb(pf_app_join_ind_cb_t pf_handler);
```

Each callback must have the appropriate parameters. The definition of the callback format is in the same file (*bs_api.h)*.

If a callback is set to NULL, the corresponding confirm or indication message will not be received by any callback function.

Important: The bootstrapping (BS) part of the G3 Coordinator application is not handled by USI protocol PROTOCOL_COORD_G3; this part has to be implemented locally where the user application runs. This approach allows more control of the status of G3 devices on the network, and reduces the overhead to request this information through USI. An example of bootstrapping implementation is included in the *g3coordd_linux* example application, in the folder “g3coordd_linux\g3\bootstrap”.

2.5.8 G3 Sniffer

This interface is simple and is defined in the *ifaceG3Sniffer.h* file. Only two functions are available:

- One function to set the callback where sniffer messages will be sent:

```
    void g3_sniffer_set_cb(void (*sap_handler)(uint8_t* msg, uint16_t len));
```

- One function to set the channel:

```
    void g3_sniffer_set_channel(uint8_t uc_channel);
```

2.6 USI Host Process

The *addUsi_Init()* function initializes the USI Host and starts it. This function must be launched before using the USI Host, typically on the system or application initialization.

The *addUsi_Process()* function processes USI messages in Tx and Rx. It transmits pending messages and retrieves data from the serial port. Once a full message is received, it will process it, calling the corresponding callbacks if needed. This function must be called periodically, for example every millisecond.

The *addUsi_ConfigurePort()* function allows to change the default port configuration (defined in the `PrjCfg.h` file) once the program is running.

3. Revision History

3.1 Revision A - March 2024

Document	Initial release PRIME and G3. Included references to hybrid protocols and new applications.

Microchip Information

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable". Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure

that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, TimeCesium, TimeHub, TimePictra, TimeProvider, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, Clockstudio, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, EyeOpen, GridTime, IdealBridge, IGaT, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, IntelliMOS, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, MarginLink, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mSiC, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, Power MOS IV, Power MOS 7, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, Trusted Time, TSHARC, Turing, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

ISBN: 978-1-6683-4164-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office	**Australia - Sydney**	**India - Bangalore**	**Austria - Wels**
2355 West Chandler Blvd.	Tel: 61-2-9868-6733	Tel: 91-80-3090-4444	Tel: 43-7242-2244-39
Chandler, AZ 85224-6199	**China - Beijing**	**India - New Delhi**	Fax: 43-7242-2244-393
Tel: 480-792-7200	Tel: 86-10-8569-7000	Tel: 91-11-4160-8631	**Denmark - Copenhagen**
Fax: 480-792-7277	**China - Chengdu**	**India - Pune**	Tel: 45-4485-5910
Technical Support:	Tel: 86-28-8665-5511	Tel: 91-20-4121-0141	Fax: 45-4485-2829
www.microchip.com/support	**China - Chongqing**	**Japan - Osaka**	**Finland - Espoo**
Web Address:	Tel: 86-23-8980-9588	Tel: 81-6-6152-7160	Tel: 358-9-4520-820
www.microchip.com	**China - Dongguan**	**Japan - Tokyo**	**France - Paris**
Atlanta	Tel: 86-769-8702-9880	Tel: 81-3-6880- 3770	Tel: 33-1-69-53-63-20
Duluth, GA	**China - Guangzhou**	**Korea - Daegu**	Fax: 33-1-69-30-90-79
Tel: 678-957-9614	Tel: 86-20-8755-8029	Tel: 82-53-744-4301	**Germany - Garching**
Fax: 678-957-1455	**China - Hangzhou**	**Korea - Seoul**	Tel: 49-8931-9700
Austin, TX	Tel: 86-571-8792-8115	Tel: 82-2-554-7200	**Germany - Haan**
Tel: 512-257-3370	**China - Hong Kong SAR**	**Malaysia - Kuala Lumpur**	Tel: 49-2129-3766400
Boston	Tel: 852-2943-5100	Tel: 60-3-7651-7906	**Germany - Heilbronn**
Westborough, MA	**China - Nanjing**	**Malaysia - Penang**	Tel: 49-7131-72400
Tel: 774-760-0087	Tel: 86-25-8473-2460	Tel: 60-4-227-8870	**Germany - Karlsruhe**
Fax: 774-760-0088	**China - Qingdao**	**Philippines - Manila**	Tel: 49-721-625370
Chicago	Tel: 86-532-8502-7355	Tel: 63-2-634-9065	**Germany - Munich**
Itasca, IL	**China - Shanghai**	**Singapore**	Tel: 49-89-627-144-0
Tel: 630-285-0071	Tel: 86-21-3326-8000	Tel: 65-6334-8870	Fax: 49-89-627-144-44
Fax: 630-285-0075	**China - Shenyang**	**Taiwan - Hsin Chu**	**Germany - Rosenheim**
Dallas	Tel: 86-24-2334-2829	Tel: 886-3-577-8366	Tel: 49-8031-354-560
Addison, TX	**China - Shenzhen**	**Taiwan - Kaohsiung**	**Israel - Ra'anana**
Tel: 972-818-7423	Tel: 86-755-8864-2200	Tel: 886-7-213-7830	Tel: 972-9-744-7705
Fax: 972-818-2924	**China - Suzhou**	**Taiwan - Taipei**	**Italy - Milan**
Detroit	Tel: 86-186-6233-1526	Tel: 886-2-2508-8600	Tel: 39-0331-742611
Novi, MI	**China - Wuhan**	**Thailand - Bangkok**	Fax: 39-0331-466781
Tel: 248-848-4000	Tel: 86-27-5980-5300	Tel: 66-2-694-1351	**Italy - Padova**
Houston, TX	**China - Xian**	**Vietnam - Ho Chi Minh**	Tel: 39-049-7625286
Tel: 281-894-5983	Tel: 86-29-8833-7252	Tel: 84-28-5448-2100	**Netherlands - Drunen**
Indianapolis	**China - Xiamen**		Tel: 31-416-690399
Noblesville, IN	Tel: 86-592-2388138		Fax: 31-416-690340
Tel: 317-773-8323	**China - Zhuhai**		**Norway - Trondheim**
Fax: 317-773-5453	Tel: 86-756-3210040		Tel: 47-72884388
Tel: 317-536-2380			**Poland - Warsaw**
Los Angeles			Tel: 48-22-3325737
Mission Viejo, CA			**Romania - Bucharest**
Tel: 949-462-9523			Tel: 40-21-407-87-50
Fax: 949-462-9608			**Spain - Madrid**
Tel: 951-273-7800			Tel: 34-91-708-08-90
Raleigh, NC			Fax: 34-91-708-08-91
Tel: 919-844-7510			**Sweden - Gothenberg**
New York, NY			Tel: 46-31-704-60-40
Tel: 631-435-6000			**Sweden - Stockholm**
San Jose, CA			Tel: 46-8-5090-4654
Tel: 408-735-9110			**UK - Wokingham**
Tel: 408-436-4270			Tel: 44-118-921-5800
Canada - Toronto			Fax: 44-118-921-5820
Tel: 905-695-1980			
Fax: 905-695-2078			