

Introduction

This document is the user guide for the implementation of the PRIME 1.3 Stack for Base Node developed by Microchip.

The mechanisms and functionalities of the PRIME specification are the basis for the entire PRIME FW stack implementation. Therefore, it is highly recommended to use it as a reference. Basic concepts that are introduced by the PRIME specification are assumed to be known within this document.

Features

- Implementation of PRIME 1.3 stack for Base Node
- Support of several microcontroller families including SAME70 and PIC32CXMTx
- Support of PLC modems including PL360 and PL460
- This document applies to PRIME FW stack version 1.3.10

Table of Contents

Introduction.....	1
Features.....	1
1. Overview.....	4
1.1. General Architecture.....	4
1.2. Event-Driven Operation.....	5
2. Understanding the Firmware Package.....	6
2.1. PRIME Firmware Package Contents.....	6
3. PRIME FW Stack.....	10
3.1. Introduction.....	10
4. Managing the PRIME FW Stack.....	13
4.1. Hardware Resources Usage.....	13
4.2. PRIME Interfaces.....	14
4.3. PRIME Integration Requirements.....	18
4.4. Main Function Structure.....	19
4.5. Configuration Parameters.....	22
5. Data Exchange.....	29
5.1. Null SSCS.....	29
5.2. IEC 61334-4-32.....	29
5.3. Automatic Repeat Request (ARQ).....	29
6. PRIME Management Plane.....	33
7. Base Management.....	35
7.1. Overview.....	35
7.2. Firmware Upgrade Protocol.....	35
7.3. Network Events.....	41
7.4. API for PRIME Profile in Management Plane.....	41
7.5. Whitelist Management.....	41
8. Toolchain.....	43
8.1. General Prerequisites.....	43
8.2. Building the Applications.....	43
9. Supported Platforms.....	44
9.1. Supported MCU Families.....	44
9.2. Supported Devices.....	44
9.3. Supported Boards.....	44
10. PICS.....	45
10.1. Major Roles for Devices Compliant with PRIME.....	45
11. API of PHY and PAL Layers.....	47
11.1. PLC PHY SAP.....	47

11.2. Serial PHY SAP.....	47
11.3. PAL SAP.....	47
11.4. PAL Primitives.....	48
12. API of PRIME FW Stack.....	50
12.1. MAC Primitives.....	50
12.2. IEC 61334-4-32 Primitives.....	53
12.3. Base Management Primitives.....	54
12.4. PIB Objects Specification and Access.....	56
13. Example Applications.....	62
13.1. PRIME Base Modem.....	62
13.2. Base Slave Modem.....	62
13.3. PHY Applications.....	63
14. Serialization with Embedded USI.....	65
14.1. USI Frame Format.....	65
14.2. USI PRIME Protocols.....	66
14.3. Embedded USI Configuration.....	68
15. Auxiliary Nodes Connection in Multi-transformer Substations.....	71
16. Abbreviations.....	73
17. References.....	75
18. Microchip Revision History.....	76
18.1. Revision C - 03/2024.....	76
18.2. Revision B - 02/2024.....	76
18.3. Rev A - 07/2020.....	77
18.4. Old Revision History.....	78
Microchip Information.....	80
The Microchip Website.....	80
Product Change Notification Service.....	80
Customer Support.....	80
Microchip Devices Code Protection Feature.....	80
Legal Notice.....	80
Trademarks.....	81
Quality Management System.....	82
Worldwide Sales and Service.....	83

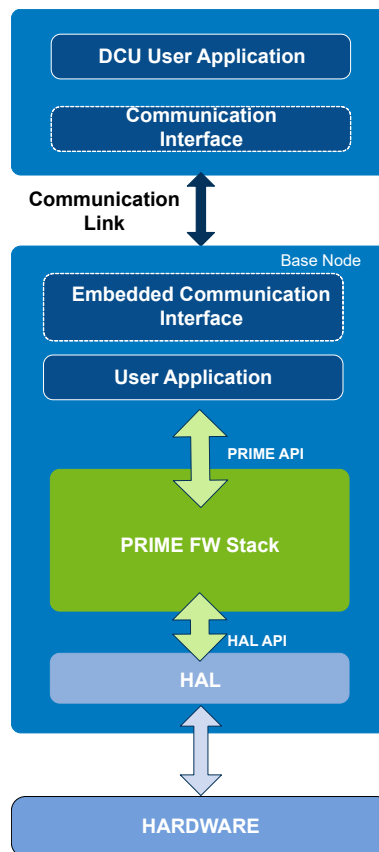
1. Overview

1.1 General Architecture

The PRIME FW stack general architecture follows the suggested separation of the network stack into logical layers, as described in the PRIME specification. Besides the core stack containing the protocol implementation, the PRIME general architecture contains additional layers implementing shared services (for example, serial interfaces, vendor specific configurations), a Hardware Abstraction Layer (HAL) and user application examples.

The general architecture of a user application including the PRIME FW stack is shown in the next figure. As it can be seen, the PRIME FW stack is part of the Base Node architecture.

Figure 1-1. Block Diagram of a DCU General Architecture with the PRIME FW Stack



Microchip provides an example project of the PRIME FW stack with a user application in the Base Node. Structure, requirements and features of the user application are described in chapter [Managing the PRIME FW Stack](#). The API of the PRIME FW stack is described in chapter [API of PRIME FW Stack](#). In addition, Microchip provides different user application examples to ease user development (see chapter [Example Applications](#)).

1.1.1 User Application

The user application is the main application in the Base Node system and therefore it must not be confused with the DCU user application. It is responsible for managing the hardware and the external resources and for setting up and running the whole system. The PRIME FW stack provides connectivity to the user application through the PRIME API (see chapter [API of PRIME FW Stack](#)),

whereas the user application communicates with the main DCU user application through a serial link, Ethernet, etc.

The user application can configure the PRIME FW stack by means of configuration files and PIB attributes.



Important: The HAL is also part of the user application in the Base Node. The pointer to the HAL functions must be passed to the PRIME FW stack at initialization. Users are also responsible for initializing, starting and running the HAL.

1.1.2 PRIME FW Stack

The PRIME firmware stack described in this document constitutes Microchip's implementation of a PRIME 1.3 Base Node.

This stack interacts with the user application through the PRIME API (see chapter [API of PRIME FW Stack](#)), with peripherals and specific software services through the Hardware Abstraction Layer (see chapter [HAL API](#)) and with the PL360 Host Controller component (used to communicate with the PLC PHY layer implemented in the PL360 device of the PL360/PL460 platform).

1.2 Event-Driven Operation

Microchip implementation of the PRIME FW stack is event-driven. Every time an API function is called, an asynchronous notification message is received after the function task is completed. This notification is delivered by means of the callback associated to the called function. Programmatically, the user application provides the underlying layers with a function pointer, which the layers below call after the request is serviced.

In such an event-driven system, user code related to PRIME functions executes in a callback that must be registered with the PRIME FW stack by the user application. Thus, the user application functions related to PRIME run entirely in stack-invoked callbacks. More information about callbacks in the PRIME general architecture can be found in chapter [PRIME API](#).

2. Understanding the Firmware Package

The following chapter describes the content of the PRIME firmware package, required common modules and drivers, and explains some general guidelines about how the various software layers are structured.

2.1 PRIME Firmware Package Contents

Microchip provides PRIME packages for every platform that supports PRIME. These packages include PRIME applications that communicate using the PRIME network and applications that only make use of the PRIME PHY layer, so that users can easily access all available user applications and their related files. Provided example applications are described in chapter [Example Applications](#).

Users are responsible for configuring and compiling the applications (see chapter [Toolchain](#)) and assigning memory resources.

2.1.1 PRIME FW Stack Library

Microchip provides several PRIME FW stack libraries for the Base Node. All of them are fully functional and the only difference resides in the number of supported nodes and the included PHY layers.

- Extra small library, that supports 25 nodes with a PLC PHY layer, is a limited library, provided for evaluation purposes only, and shall not be implemented in a final commercial product.
- Small library, that supports 150 nodes with PLC PHY layer, is intended for a hardware design with internal memory.
- Medium library, that supports 300 nodes with PLC PHY layer.
- Large library, that supports 1500 nodes with PLC PHY layer, is intended for a hardware design with external memory in the Xplained platform.
- Extra large library, that supports 2,000 nodes, is intended for a hardware design with external memory. This library includes the PLC PHY layer and the Serial PHY layer to support the connection of auxiliary nodes. For additional information on functionality, refer to the Chapter [Auxiliary Nodes Connection in Multi-transformer Substations](#).

Additionally, the PRIME FW stack library for a Service Node can be used to support the connection of auxiliary nodes by simply enabling the Serial PHY layer so that the node becomes a Base Slave Node.

Table 2-1. PRIME FW Stack Libraries and Platforms

Board	Device	Node Number	Library
PL360BN (external memory)	SAME70Q21	2000	libsamM7-prime13-bn_xl
SAME70 Xplained	SAME70Q21	1500	libsamM7-prime14-bn_l
PIC32CXMTSH-DB	PIC32CX2051MTSH128	300	libsamM4-prime14-bn_m
PIC32CXMTG-EK	PIC32CX2051MTG128	300	libsamM4-prime14-bn_m
PL360BN (internal memory)	SAME70Q21	150	libsamM7-prime13-bn_s
SAMG55 Xplained (Note 1)	SAMG55J19	25	libsamM4-prime13-bn_xs
PL360BN (Note 2)	SAME70Q21	-	libsamM4-prime13-sn-iar

Notes:

1. For evaluation purposes only.
2. For the Base Slave Node only.

2.1.2 Project Examples

Every project example contains the corresponding PRIME FW stack library, the PL360 Host Controller, the user application example, and the configuration files described in chapter [Configuration Parameters](#).



Important: The Serial PHY layer is only available in examples that use the full library and in the Base Node Slave.

This project can be reused whenever a new PRIME library is available, however, it is necessary to exchange the library file in the project and update the PRIME firmware version in the corresponding configuration file. For additional information, refer to the Chapter [Firmware Version Information](#).

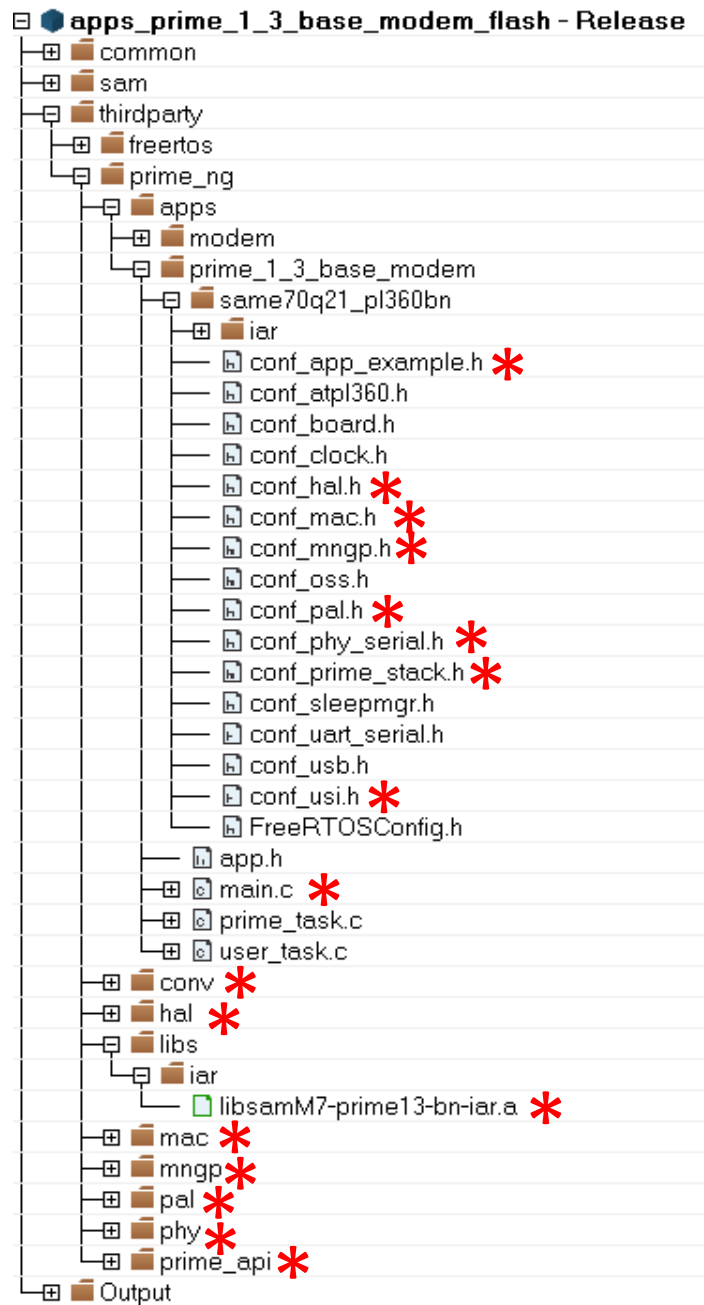
Microchip provides a project example, called Modem, which shows how to manage the PRIME FW stack from an external application by serializing the PRIME API. See chapter [Example Applications](#) for detailed information about the operation of this example.

The Modem project can be found in the package following this path:

```
..\thirdparty\prime_ng\apps\prime_1_3_base_modem\same70q21_pl360bn\iar\apps_prime_1_3_base_modem.eww
```

The structure of this project in IAR Embedded Workbench[®] is shown in the following figure. Files and folders required to develop a user application over the PRIME FW stack are marked with red asterisks.

Figure 2-1. Folder Structure of the Modem Example Project



Important: Users must not modify any of the provided files except for the configuration files, the HAL and the board definition (in folder `Isam\boards\`) in order to adapt them to their hardware design. Users are free to use the example applications as templates to create new user applications. Users can also develop their own applications in the *apps* directory.

Microchip also provides a Modem project for the Base Slave Node to connect auxiliary nodes (see chapter [Auxiliary Nodes Connection in Multi-transformer Substations](#)). The Base Slave Node project can be found in the package following this path:

```

..\thirdparty\prime_ng\apps\prime_1_3_base_slave_modem\same70q21_pl360bn\iar\
apps_prime_1_3_base_slave_modem.eww

```


Additionally, Microchip also provides several PHY applications that show to manage the PRIME PHY layer exclusively. See chapter [Example Applications](#) for more information.

2.1.3 PL360 Firmware

For the PL360/PL460 platform, Microchip provides a binary file to be loaded into the PL360 device, where the PLC PHY layer of the PRIME FW stack runs. Details about loading the PL360 firmware can be found in the "PL360 Host Controller User Guide".

2.1.4 USI Host Example

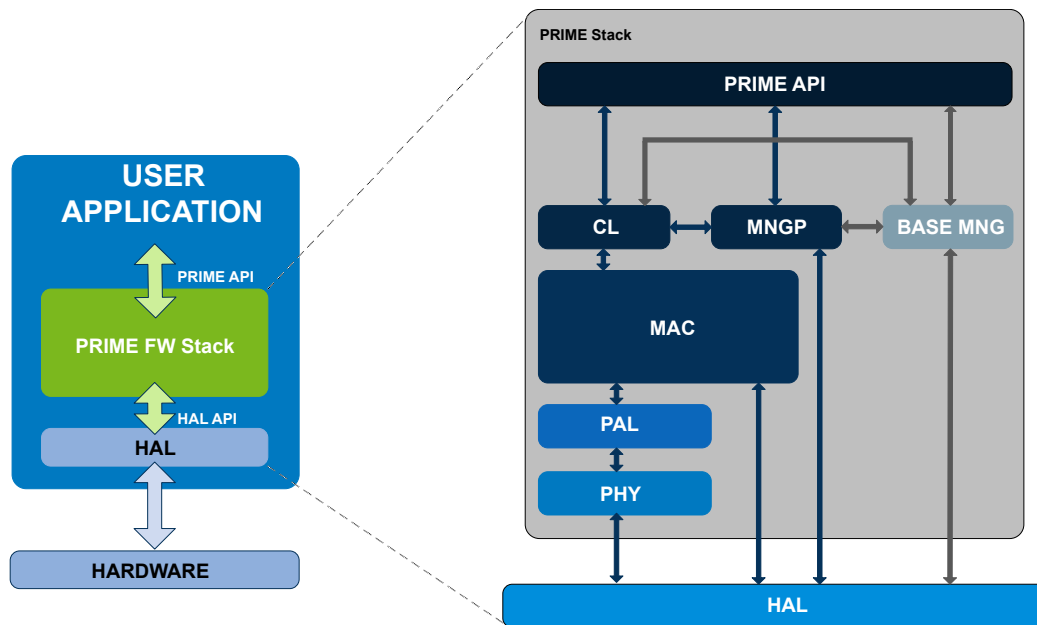
Microchip provides the required files to develop the host controller side of the serialization of the PRIME FW stack. These files are included in the USI (Universal Serial Interface) Host. Details about usage of serialization can be found in chapter [Example Applications](#). For the USI Host, see the *USI Host User Guide*.

3. PRIME FW Stack

3.1 Introduction

The PRIME Firmware Stack follows a layered approach based on the PRIME specification. The following figure shows the stack architecture.

Figure 3-1. Block Diagram of the PRIME FW Stack Architecture



Microchip provides a library which contains the MAC layer, the Convergence Layer and the Management Plane as described in the PRIME specification. It also includes a proprietary Base Management module to handle the interface between the Base Node and the main DCU user application and enhance it with additional functionalities. The PHY is not the PHY layer as such but the host controller component.

Note that the PHY and PAL layer boxes in the above diagram can be split into more than one PHY and PAL layers depending on the application. See chapter [3.1.2. Physical Abstraction Layer \(PAL\)](#).

It is important to note that the only entry point to the PRIME FW stack from the user application is through the PRIME API, which contains the interfaces defined in the PRIME specification as well as stack control functions. The API of the PRIME FW stack is described in chapter [PRIME API](#) and in chapter [API of PRIME FW Stack](#). The PRIME FW stack accesses the hardware through the user application with the HAL API described in chapter [HAL API](#).

Additionally, Microchip provides the source code of the PL360 Host Controller that manages the interface of the PLC PHY layer in the PL360 device, the Serial PHY layer and the PAL layer.

The PHY layer, the MAC layer and the Convergence Layer have all their own version numbers, which can be accessed through the corresponding vendor specific PIB attributes `PIB_PHY_SW_VERSION` (0x8080), `PIB_MAC_INTERNAL_SW_VERSION` (0x8126) and `PIB_CL_INTERNAL_SW_VERSION` (0x8201). These PIB attributes are described in chapter [PIB Objects Specification and Access](#).

The PRIME FW stack modules from the bottom up are briefly described in the following sections. Basic concepts that are discussed in the PRIME specification are assumed to be known. The

conformance statement about the current implementation of the PRIME protocol is found in chapter [Protocol Implementation Conformance Statement](#).

3.1.1 Physical Layer (PHY)

The Physical Layer (PHY) handles the transmission and reception of MPDUs at the physical level between neighbor nodes.

On the transmission side, the PHY layer receives its inputs from the MAC layer through the PAL. At the end of transmission, the data is passed to the physical medium via the HAL.

On the reception side, the PHY layer receives its inputs from the physical medium via the HAL. At the end of reception path, the data flows to the MAC layer through the PAL.

By providing an interface through the PAL, the MAC layer is then independent from the underlying modem. Besides that, the PHY interface can be used by basic applications without requiring the MAC layer (see a PHY application example in chapter [PHY Applications](#)). The API of the PHY layer is described in chapter [API of PHY and PAL Layers](#).

The PHY layer can be used for PLC communications as well as for serial communications. An example of the Serial PHY layer can be found in chapter [Auxiliary Nodes Connection in Multi-transformer Substations](#)). The selection of the right PHY layer is performed by the PAL according to the indications given by the MAC layer.

The PHY layer maintains a set of attributes that provide detailed information about its operation. The PIB attributes related to the PHY layer are stored in the PHY PIB storage and the PHY layer provides an interface to access and update their values through the PHY Layer Management Entity (PLME). The PRIME FW stack supports all mandatory PHY PIB attributes as described in the PRIME specification as well as some vendor specific ones (see [Table 12-10](#)).

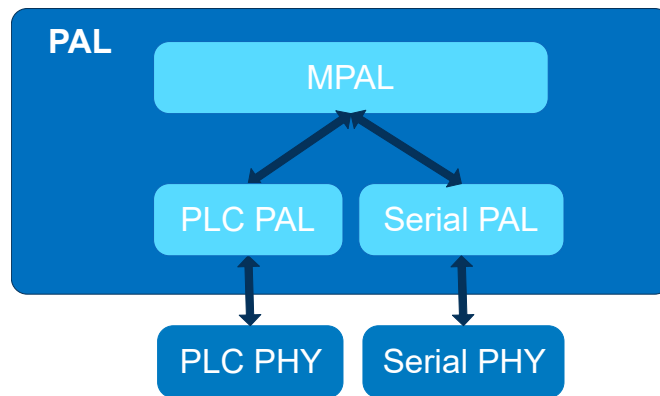
3.1.2 Physical Abstraction Layer (PAL)

The Physical Abstraction Layer (PAL) provides an interface between the PHY layer and the MAC layer, so that the MAC layer is independent from the PHY layer and the underlying modem.

The PAL is based on the PHY primitives for data and control planes described in the PRIME specification, although the current implementation offers enhanced versions of some of them to take a full advantage of the features available in the PHY layer. These features improve the operation of the MAC layer.

A description of the PHY data and management primitives available in the PAL as well as the extended features can be found in chapter [API of PHY and PAL Layers](#).

Taking into account that there could be several PHY layers, such as the PLC PHY and the Serial PHY, the MPAL (MultiPhy Abstraction Layer) provides a link between the PAL interface and the selected PHY layer. The MPAL selects the correct PHY layer by means of the channel indicated by the MAC layer.

Figure 3-2. Block Diagram of the PAL

3.1.3 Medium Access Control Layer (MAC)

The Medium Access Control (MAC) layer provides core MAC functionalities of system access, bandwidth allocation, connection establishment/maintenance and topology resolution according to the PRIME specification.

The MAC layer maintains a set of attributes that provide detailed information about its operation. The PIB attributes related to the MAC layer are stored in the MAC PIB storage and the MAC layer provides an interface to access and update their values through the MAC Layer Management Entity (MLME). The PRIME FW stack supports all mandatory MAC PIB attributes as described in the PRIME specification as well as some vendor specific ones (see [Table 12-11](#)).

3.1.4 Convergence Layer (CL)

The Convergence Layer (CL) associates the data received from a user application to its proper MAC connection. It is composed by a common part to segment and reassemble packets and one or more service specific sublayers (e.g. IEC 61334-4-32). Information about data exchange using the CL can be found in chapter [Data Exchange](#).

3.1.5 Management Plane (MNGP)

The Management Plane enables a local or remote control entity to perform actions on a node. These actions include providing access to internal parameters defined by PIB attributes as well as managing the firmware upgrade inside the stack. More information about this layer can be found in chapter [PRIME Node Management Plane](#).

The PRIME certification conformance tests require accessing the PIB attributes and this is achieved via the Management Plane.

3.1.6 Base Management (Base MNG)

The Base Management module is a Microchip proprietary extension over the PRIME specification to increase the functionalities of a Base Node. The PRIME specification does not always define the interfaces between the Base Node and the DCU user application, so new interfaces must be implemented as required.

The Base Management handles the Firmware Upgrade Protocol interface and access to the PRIME Profile. It also notifies about network events, such as node registrations and unregistrations, and manages the whitelist. More information about this module can be found in chapter [Base Management](#).

4. Managing the PRIME FW Stack

The purpose of this section is to indicate requirements and provide recommendations for integrating the PRIME FW stack into the user application.

The following issues must be taken into account:

- Hardware requirements
- Interfaces with the PRIME FW stack (PRIME API and HAL API)
- Integration requirements
- Main function structure

4.1 Hardware Resources Usage

MCU hardware resources include microcontroller peripherals, buses, timers, IRQ lines, I/O registers, and so on. Since many of these interfaces have corresponding APIs in the HAL, users are encouraged to use the high-level APIs instead of the low-level register interfaces to ensure that the resource usage does not overlap with that of the PRIME FW stack. The hardware resources reserved for internal use by the PRIME FW stack are the following:

- Mandatory timers: The 1 μ s service requires one timer TCx to have a common timer reference between the PHY and the MAC layer.
- Exceptional timers: During certification, when a UART or USART is required (see Chapter [Serial Communication Profile of PRIME Management Plane](#)), a timer TCx is used.
- SPI: The PLC PHY layer requires one SPI port.
- DMA: DMA0 is used.
- Interrupts: The PLC PHY layer uses a GPIO as PLC External Interrupt pin (EINT).
- External memory in SAME70 (for full library): PMC and SDRAM.



Important: Hardware resources reserved for use by the PRIME FW stack must not be accessed by the user application.

4.1.1 Data Storage

The PRIME FW stack requires storage of data in a non-volatile storage area that must be provided by users. Non-volatile data are given below:

- The MAC address (see chapter [MAC Address](#))
- The BN parameters (see chapter [Base Node Parameters](#))
- The PRIME mode (see chapter [PRIME Mode](#))
- The PHY parameters (see chapter [Communication Channel and other PHY Parameters](#))

In addition, the PRIME FW stack imposes memory requirements to the user application in the Base Node regarding the memory. In order to upgrade a Service Node, the corresponding firmware image must be stored in RAM or Flash memory.

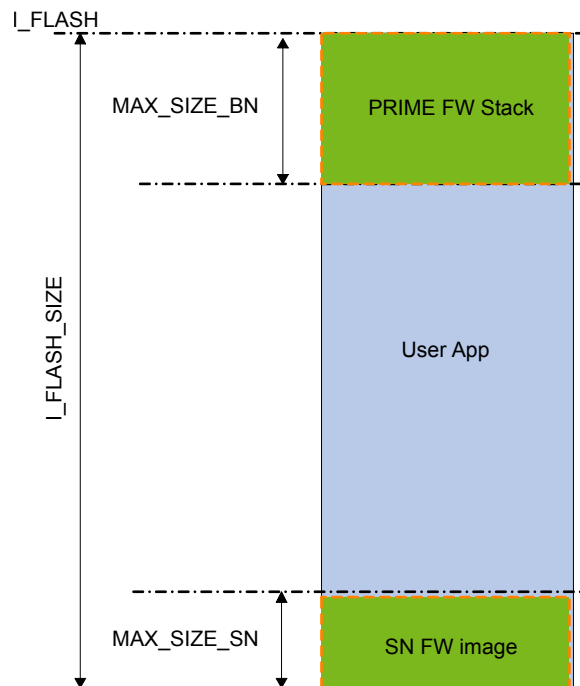


Important: The location and the size of the reserved memory area for the SN firmware image to upgrade must be set up by the users according to their system and requirements.

The figure below illustrates the memory allocation configured in the provided application project examples. The memory addresses can be configured in the *conf_app_example.h* file.

```
/* Define PRIME size in flash. Could be movable */
#define PRIME_MAX_SIZE_SN (0x00060000u)
#define PRIME_MAX_SIZE_BN (0x00040000u)
#define PRIME_FLASH_LOCATION_SN (I_FLASH + I_FLASH_SIZE - PRIME_MAX_SIZE_SN)
```

Figure 4-1. Flash Example for PRIME



The usage of external memory implies a correct initialization of variables. This means that variables used before the external memory is initialized must be initialized before. Since the projects provided by Microchip initialize the external memory at program initialization and not at start-up, variables like the heap are initialized in the linker script of each project.

4.2 PRIME Interfaces

As shown in [Figure 1-1](#), the PRIME FW stack has two interfaces: one towards the user application and another one towards the HAL.

4.2.1 PRIME API

The PRIME API is the only interface that the user application must use to interact with the PRIME FW stack.

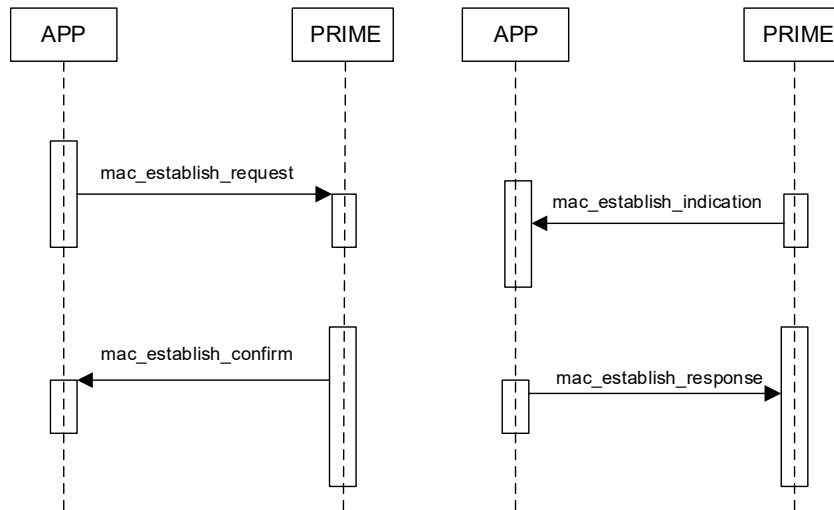
The PRIME API is based on a request/confirm mechanism, which is a particular instance of an event-driven programming model. A *request* is an asynchronous call to the underlying stack to perform some action on behalf of the user application; a *confirm* is the callback that executes when that action has been completed and the result of that action is available.

Apart from *request/confirm* pairs, there are cases when the application needs to be notified of an external event that is not a reply to any specific *request*. For this, there are a number of user-defined callbacks named *indications* that are invoked by the stack asynchronously. Note that during the execution of an *indication* callback, it is not permitted to invoke any *request* as this could lead to the generation of a callback chain.



Important: Callback functions must be registered in the PRIME FW stack by the user application upon initialization (see chapter [Main Function Structure](#)).

Figure 4-2. API Calls in Event-Driven Programming



The *confirm* callback can be divided in two types:

- **Asynchronous:** For most operations, *request* execution takes a considerable amount of time and a *confirm* callback function can be called seconds after the request was issued. During this time, the application must postpone other *requests* of the same type until the first *request* is completed. (i.e.: previous *confirm* has been received). Furthermore, it is not recommended to execute any other action after invoking the *request*. Considering that the PRIME FW stack always replies with a *confirm*, the application must wait for the corresponding *confirm* before performing a new *request* of the same type
- **Immediate:** For PLME and MLME operations, the *confirm* callback function is called right away from the *request* during the same execution cycle, i.e. it acts as a normal function call and it is thus possible to perform other actions after the *request*. Note that for any operation (not only PLME or MLME), when the result is not successful, the confirm callback function is called immediately instead of asynchronously

The complete PRIME API structure is located relatively to the address indicated by the `prime_api` pointer. It is defined as a set of macros in the header file *prime_api.h*. The function parameters are defined in several *defs* header files depending on the layer the function is related to. The user application can simply call these functions without modifying them. For more information, see chapter [API of PRIME FW Stack](#).

4.2.1.1 Request/Confirm Example

Consider the establishment of a MAC connection from a node. The user application needs to call the following function:

```
prime_cl_null_establish_request(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);
```

Previously, it has registered the callback to the corresponding confirm (see chapter [PRIME FW Stack Initialization](#)), which must be a function of the following type:

```
typedef void (*mac_establish_cfm_cb_t)(uint16_t us_con_handle, mac_establish_confirm_result_t
uc_result, uint8_t *puc_eui48, uint8_t uc_type, uint8_t *puc_data, uint16_t us_data_len);
```

The example illustrates a particular instance of using a request/confirm mechanism but all other uses follow the same approach.

4.2.1.2 Indication/Response Example

If the user application wants to be informed about the establishment of a MAC connection in the node, it must register a callback to the corresponding indication (see chapter [PRIME Callback Functions](#)), which must be a function of the following type:

```
typedef void (*mac_establish_ind_cb_t)(uint16_t us_con_handle, uint8_t *puc_eui48, uint8_t uc_type, uint8_t *puc_data, uint16_t us_data_len, uint8_t uc_cfbytes);
```

Indications might or might not need a response. In this example, the callback code for the indication must include a call to the following function:

```
prime_cl_null_establish_response(uint16_t us_con_handle, mac_establish_response_answer_t uc_answer, uint8_t *puc_data, uint16_t us_data_len);
```

The example illustrates a particular instance of using an indication mechanism but all other uses follow the same approach.

4.2.2 HAL API

Apart from the PRIME FW stack, Microchip provides an open code of the Hardware Abstraction Layer (HAL). The HAL functionalities are used both by the user application as well as by the PRIME FW stack.

The HAL can be allocated at any address within the region of the user application, which must keep a pointer to the HAL API functions. This pointer must be passed to the PRIME FW stack at initialization.



The API of the HAL, defined in header file *hal.h*, must not be changed under any circumstances.



Important: Note that the provided HAL source code is only an implementation example. Users should modify the function code according to their hardware and specifications.

The HAL contains all platform (i.e., MCU and board) specific functionality (required by the PRIME FW stack) and provides interfaces to the upper modules. Therefore, all upper modules are independent from the underlying platform.

The HAL provides interfaces to several components, such as the PLC access functionality through SPI or access to persistent storage (for example, serial Flash) among others. These components are implemented as software blocks which interact with the hardware.

The HAL also implements software components that may or may not interact with the hardware. For example, the CRC component or the serial interface to handle the different serial interfaces described in the PRIME specification through one or more UART ports. The Universal Serial Interface (USI) is an example of implementation of this serial interface. For additional information about services in the USI, refer to the Application Note "PLC Universal Serial Interface".

The following table shows all HAL functions used by the PRIME FW stack, the file where they are implemented and a brief description of their usage. Note that there might be other functions not described here but available in *hal.h*. Those functions are irrelevant in this context.

Table 4-1. HAL API

Function	File	Description
hal_restart_system	hal.c	Restart the complete system.
hal_pcrc_calc	hal_pcrc.c	Calculate the CRC of the input buffer according to the PRIME specification.
hal_pcrc_config_sna	hal_pcrc.c	Configure the SNA for the CRC.
hal_fu_data_read	hal_fu.c	Read data from memory during FU.
hal_fu_data_write	hal_fu.c	Write data in memory during FU.
hal_fu_start	hal_fu.c	Initialize and unlock memory for a FU process.
hal_fu_crc_calculate	hal_fu.c	Calculate the CRC of the received file.
hal_fu_crc_set_callback	hal_fu.c	Specify the callback function to provide the calculated CRC.
hal_plc_init ¹	hal_plc.c	Initialize the PLC interface.
hal_plc_reset ¹	hal_plc.c	Reset the internal PLC modem.
hal_plc_set_handler ¹	hal_plc.c	Specify the callback function for the PLC interrupt.
hal_plc_tx_signal	hal_plc.c	Indicate a transmission event.
hal_plc_rx_signal	hal_plc.c	Indicate a reception event.
hal_get_config_info ²	hal_cfg.c	Read configuration parameters. This function disables all interrupts when accessing the User Signature.
hal_set_config_info ²	hal_cfg.c	Write configuration parameters. This function disables all interrupts when accessing the User Signature.
hal_usi_set_callback	hal_usi.c	Specify the callback function for a given protocol.
hal_usi_send_cmd	hal_usi.c	Transmit data through the serial interface.
hal_trng_init	hal_trng.c	Initialize the true random number generator.
hal_trng_read	hal_trng.c	Read information from the true random number generator.
hal_debug_report	hal.c	Report a debug error.
hal_plc_send_boot_cmd ³	hal_plc.c	Send a boot command. This function disables all interrupts.
hal_plc_send_wrrd_cmd ³	hal_plc.c	Send a write/read command. This function disables all interrupts.
hal_plc_enable_interrupt ³	hal_plc.c	Enable or disable the PLC interrupt.
hal_plc_delay ³	hal_plc.c	Delay execution.
hal_pib_get_request	hal_pib.c	Get user specific PIB value.
hal_pib_get_request_set_callback	hal_pib.c	Specify the callback function to get a user specific PIB value.
hal_pib_set_request	hal_pib.c	Set user specific PIB value.
hal_pib_set_request_set_callback	hal_pib.c	Specify the callback function to return the result of setting a user specific PIB value.
hal_plc_set_stby_mode ³	hal_plc.c	Set the stand-by mode.
hal_plc_get_thermal_warning ³	hal_plc.c	Check if there is a thermal warning active.
timer_1us_get	timer_1us.c	Get the current time in microseconds.
timer_1us_set_int	timer_1us.c	Set an interrupt for the specified time in microseconds.
timer_1us_cancel_int	timer_1us.c	Cancel a programmed time interrupt.
timer_1us_enable_interrupt	timer_1us.c	Enable/disable the TC interrupt.

Notes:

1.



Microchip does not recommend changing these functions as they are related to the PLC interrupt.

2.



Important: The Microchip implementation of these functions uses the User Signature in the MCU to store some configuration parameters. This is just an example. Users can store such parameters in any non-volatile storage area of their choice and then they must update these functions accordingly.

3. These functions are only available in the PL360/PL460 platform.

4.3 PRIME Integration Requirements

The goal of this section is to provide recommendations, requirements and limitations to be taken into account in the application development with the PRIME FW stack. Users are strongly advised to follow these guidelines.



Failure to comply with the requirements may result in an anomalous and/or unexpected behavior of the PRIME node.

4.3.1 Task Manager, Priorities and Preemption

A major aspect of application development is managing the control flow and ensuring that different parts of the application do not interfere with each other's execution. The PRIME FW stack can be integrated in the user application like any other task.



Important: The PRIME task must have at least the same or a higher priority than any other user task.

If several user tasks are defined, then it is highly recommended to configure the user tasks with a lower priority than the PRIME task. The reason to do this is to ensure that the PRIME task is executed as soon as the system tick arrives, in the defined timer rate. This synchronization is critical for the PRIME FW stack.



Important: Call the PRIME FW stack process every 10ms as a maximum period, ideally every 5ms.

4.3.2 Other Coding Requirements

The following requirements must be taken into account for a proper system performance:

- Avoid invoking a request function inside a callback function. This increases the call stack size and may provoke endless loops. For example, do not request to send data in the same callback function that handles the confirm of a data request.
- Avoid invoking a request to a node if the previous confirm has not been received yet, if the connection is being closed or if there has been a reset.

- Set to NULL all unused callback function pointers:

```
memset(&mac_cbs, NULL, sizeof(mac_cbs));
memset(&cl432_cbs, NULL, sizeof(cl432_cbs));
memset(&bmng_cbs, NULL, sizeof(bmng_cbs));
```

- Provide the callback function pointers again after an MLME_RESET.confirm primitive.
- Do not modify critical regions in the HAL. The PRIME process already disables any interrupts that could affect its operation (all interrupts with priority 2 and higher). When the PRIME process has finished, disabled interrupts are enabled again.
- Do not modify the functions and parameters in the PRIME API and the HAL API.
- Modify the available user configuration files according to your needs.
- Give a MAC address to the board. If the PRIME FW stack cannot find a MAC address, it enters into MTP mode.
- Initialize and refresh the watchdog to avoid hangings of any application during execution.
- Use the supply monitor controller, if available in the board, to avoid malfunctions.
- Do not use size optimization to avoid malfunctions.

4.4 Main Function Structure

Every user application contains a main function, which is, as usual, the starting point of the application. A basic main function is presented below.

Example of main function

```
int main(void)
{
    /* Function to setup clocking. */
    sysclk_init();

    /* Ensure all priority bits are assigned as preemption priority bits.*/
    NVIC_SetPriorityGrouping(__NVIC_PRIO_BITS);

    /* Library function to setup for the evaluation kit being used.*/
    board_init();

    /* Configure external 32khz clock if needed (depends on board) */
    pmc_switch_sclk_to_32kxtal(0);

    /* Enable Sleep manager */
    sleepmgr_init();

    /* Initialize SDRAM controller */
    _prv_setup_sdram();

    /* Configure supply monitor */
    hal_setup_supply_monitor(CONTINUOUS_MONITORING, THRESHOLD_3V04);

    /* Initialize flash: 6 wait states for flash writing. */
    flash_init(FLASH_ACCESS_MODE_128, CHIP_FLASH_WRITE_WAIT_STATE);

    /* Set up watchdog */
    hal_watchdog_setup(WATCHDOG_TIME);

    /* Configure console */
    _configure_dbg_console();
    puts(STRING_HEADER);

    /* Initialize PRIME stack */
    prime_init((hal_api_t *) &hal_api);

    /* Init HAL PLC signalling */
    hal_plc_set_tx_signalling_handler(_blink_plc_tx_activity_led);
    hal_plc_set_rx_signalling_handler(_blink_plc_rx_activity_led);

    /* Init user application (callbacks to PRIME) */
    app_init();

    while (1) {
```

```

/* Restart watchdog */
wdt_restart(WDT);

/* Process HAL layer */
hal_process();

/* Process PRIME stack */
prime_process();

/* Process user application */
app_process();
}
}

```

A developer can add additional code into the body of the function, but the main function should always follow the structure provided:

1. Invoke the `prime_init()` function to initialize the PRIME FW stack
2. Invoke the `prime_process()` function in the infinite loop to pass control to the task manager

The task manager begins invoking the task handlers of each layer in order of priority (from highest to lowest), eventually invoking the application task handler. Following the initial call to the application task handler, the control flow passes between the PRIME FW stack and the callbacks, as shown in [Figure 4-2](#).



Important: The pointer to the HAL functions must be passed to the PRIME FW stack at initialization. Users are also responsible for initializing, starting and running the HAL.



Important: In order to avoid hangings of any application during execution, it is recommended to initialize and refresh the watchdog. This is a driver available in any Microchip board.

In order to start using the PRIME FW stack, it is necessary to initialize different parameters and to call the corresponding initialization functions. These actions must follow the order indicated in the next sections.

4.4.1 PLC Signaling (optional)

If users want to be notified about PLC transmission and reception events, the corresponding callback functions must be provided to the HAL.

4.4.1.1 Providing the Pointer to the Transmission Event Callback Function

To set up the callback for transmission events, the following function must be used:

```
void hal_plc_set_tx_signalling_handler(void (*p_handler)(void));
```

Parameters:

- *p_handler*: Pointer address to the callback function

4.4.1.2 Providing the Pointer to the Reception Event Callback Function

To set up the callback for reception events, the following function must be used:

```
void hal_plc_set_rx_signalling_handler(void (*p_handler)(void));
```

Parameters:

- *p_handler*: Pointer address to the callback function

4.4.2 PRIME FW Stack Initialization

After setting up pointers, the user application can call the initialization function of the PRIME FW stack. Additionally, since the HAL is part of the application, the application must provide the HAL pointer to the PRIME FW stack at this point. The function used for this purpose is the following:

```
prime_init(void *px_hal_api);
```

Parameters:

- *px_hal_api*: Pointer to the HAL API



Important: This function initializes the complete PRIME FW stack, from the PHY layer to the CL shown in [Block Diagram of the PRIME FW Stack Architecture](#).

4.4.3 PRIME Callback Functions

When the user application is initialized, it must set up required callback function pointers in order to be informed about confirm and indication primitives from the MAC layer and the CL. This must be performed after the PRIME FW stack has been initialized. The following functions are used:

```
void prime_cl_null_set_callbacks(mac_set_callbacks_t *mac_cbs);
```

Parameters:

- *mac_cbs*: Pointer to the callback structure of the MAC layer (see chapter [Callback Functions in the MAC Layer](#) for details).

```
void prime_cl_432_set_callbacks(cl_432_set_callbacks_t *cl432_cbs);
```

Parameters:

- *cl432_cbs*: Pointer to the callback structure of the 4.32 SSCS (see chapter [IEC 61334-4-32 Primitives](#) for details).

```
void bmng_set_callbacks(bmng_callbacks_t *px_base_mng_cbs);
```

Parameters:

- *px_base_mng_cbs*: Pointer to the callback structure of the Base Management (see chapter [Callback Functions in the Base Management](#) for details).

This is an example of callback function pointer set-up:

```
mac_callbacks_t mac_cbs;
cl_432_callbacks_t cl432_cbs;
bmng_callbacks_t bmng_callbacks;

/* Initialize all callback pointers to NULL */
memset(&mac_cbs, NULL, sizeof(mac_cbs));
memset(&cl432_cbs, NULL, sizeof(cl432_cbs));
memset(&bmng_callbacks, NULL, sizeof(bmng_callbacks));

/* Initialize callback pointers for MAC */
mac_callbacks.mac_data_cfm_cb = _user_app_data_cfm_cb;
mac_callbacks.mac_data_ind_cb = _user_app_data_ind_cb;
mac_callbacks.mac_establish_cfm_cb = _user_app_establish_cfm_cb;
mac_callbacks.mac_establish_ind_cb = _user_app_establish_ind_cb;
mac_callbacks.mac_join_cfm_cb = _user_app_join_cfm_cb;
```

```

mac_callbacks.mac_join_ind_cb = _user_app_join_ind_cb;
mac_callbacks.mac_leave_cfm_cb = _user_app_leave_cfm_cb;
mac_callbacks.mac_leave_ind_cb = _user_app_leave_ind_cb;
mac_callbacks.mac_release_cfm_cb = _user_app_release_cfm_cb;
mac_callbacks.mac_release_ind_cb = _user_app_release_ind_cb;
mac_callbacks.mlme_get_cfm_cb = _user_app_mlme_get_cfm_cb;
mac_callbacks.mlme_list_get_cfm_cb = _user_app_mlme_list_get_cfm_cb;
mac_callbacks.mlme_promote_cfm_cb = _user_app_mlme_promote_cfm_cb;
mac_callbacks.mlme_reset_cfm_cb = _user_app_mlme_reset_cfm_cb;
mac_callbacks.mlme_set_cfm_cb = _user_app_mlme_set_cfm_cb;

prime_cl_null_set_callbacks(&mac_cbs);

/* Initialize callback pointers for 4.32 */
cl432_callbacks.cl_432_dl_data_cfm_cb = _user_app_cl_432_dl_data_cfm_cb;
cl432_callbacks.cl_432_dl_data_ind_cb = _user_app_cl_432_dl_data_ind_cb;
cl432_callbacks.cl_432_join_ind_cb = _user_app_cl_432_join_ind_cb;
cl432_callbacks.cl_432_leave_ind_cb = _user_app_cl_432_leave_ind_cb;

prime_cl_432_set_callbacks(&cl432_cbs);

/* Initialize callback pointers for Base Management */
bmng_callbacks.fup_ack_cb = _user_app_bmng_fup_ack_cb;
bmng_callbacks.fup_error_ind_cb = _user_app_bmng_fup_error_ind_cb;
bmng_callbacks.fup_kill_ind_cb = _user_app_bmng_fup_kill_ind_cb;
bmng_callbacks.fup_status_ind_cb = _user_app_bmng_fup_status_ind_cb;
bmng_callbacks.fup_version_ind_cb = _user_app_bmng_fup_version_ind_cb;
bmng_callbacks.network_event_ind_cb = _user_app_bmng_net_event_ind_cb;
bmng_callbacks.pprof_ack_cb = _user_app_bmng_pprof_ack_cb;
bmng_callbacks.pprof_get_response_cb = _user_app_bmng_pprof_get_response_cb;
bmng_callbacks.pprof_get_zc_response_cb = _user_app_bmng_pprof_get_zc_response_cb;
bmng_callbacks.pprof_zc_diff_response_cb = _user_app_bmng_pprof_zc_diff_response_cb;
bmng_callbacks.whitelist_ack_cb = _user_app_bmng_whitelist_ack_cb;

prime_bmng_set_callbacks(&bmng_callbacks);

```

4.4.4 PRIME FW Stack Process

Every program cycle, the PRIME FW stack must perform several actions for its normal operation. This is achieved by calling the following function:

```
prime_process(void);
```

This function processes events and invokes the corresponding callback functions.

4.5 Configuration Parameters

The PRIME FW stack can be configured to suit user requirements. This is achieved with several configuration files, which allow:

- The stack to configure the required stack resources according to the application needs based on the required functionality, and
- The application to configure its own resources

Note that some parameters must be configured during the Manufacturing Test Procedure (MTP) using PIB attributes.

Some parameters are subject to change during execution and it could be needed to store them in non-volatile memory so that they can be recovered after a system restart. See chapter [4.1.1. Data Storage](#) for more information.

4.5.1 PRIME Stack User Configuration Parameters

4.5.1.1 Enabling Base Node API

The file *conf_prime_stack.h* includes a definition to enhance the PRIME API with functions related to the Base Node. In the provided examples, this line is not commented, so that the API of the Base Node is included.

```
/* Enable API for base node */
#define PRIME_API_BN
```

4.5.1.2 Firmware Version Information

In the provided PRIME FW stack library, the first 20 bytes contain the Vendor, Model and Version information corresponding to the PRIME FW stack library. These metadata are useful to identify and control which library is being used. Users can edit this information in file *conf_prime_stack.h*.

```
/* Firmware Information */
#define PRIME_FW_VENDOR    "MCHP"
#define PRIME_FW_MODEL     "PL360BN"
#define PRIME_FW_VERSION   "B13.10.04\0\0\0\0\0\0\0"

/* Prime PIB firmware information. FW Version is used as PIB version */
#define PRIME_PIB_VENDOR   0x0000
#define PRIME_PIB_MODEL    0x3D3F
```

The modifiable attributes are the following:

Table 4-2. PRIME FW Information Attributes

Attribute	Definition	Format
PRIME_FW_VENDOR	Vendor name, e.g. "MCHP".	ASCII. Maximum 16 bytes.
PRIME_FW_MODEL	Product name, e.g. "PL360BN".	ASCII. Maximum 16 bytes.
PRIME_FW_VERSION	Version identifier as defined in PIB 0x0075 (PIB_MAC_APP_FW_VERSION), e.g. "B13.10.04\0\0\0\0\0\0\0".	Numerical, ASCII recommended. It must be 16 bytes long.
PRIME_PIB_VENDOR	Vendor name as defined in PIB 0x0076 (PIB_MAC_APP_VENDOR_ID), e.g. 0x0000.	Numerical, two bytes size. This value is assigned by the PRIME Alliance.
PRIME_PIB_MODEL	Product name as defined in PIB 0x0077 (PIB_MAC_APP_PRODUCT_ID), e.g. "70".	Numerical or ASCII, two bytes size.

4.5.1.3 Communication Channel and other PHY Parameters

Microchip provides solutions that cover frequencies from 41 kHz up to 89 kHz and are compliant with different applicable regulations in the PLC domain.

The table below shows the available frequency bands with their associated PLC coupling boards. To configure the PLC coupling in PL360 and PL460, see the corresponding chapters.

Table 4-3. Frequency Bands

Channel	Frequency band (kHz)	PLC coupling board (PL360)	PLC coupling (PL460)
1	41 - 89	ATPLCOUP000 ATPLCOUP001 ATPLCOUP003 ATPLCOUP004 ATPLCOUP007 ATPLCOUP011	Main branch or auxiliary branch

The PRIME FW stack needs PHY parameters to be kept safe in a non-volatile storage area in case the system restarts for any reason.

These PHY parameters are the PLC communication channel.

The PRIME FW stack uses a configuration key to determine whether these parameters are available in the non-volatile storage or not. The structure with the configuration key and the PHY parameters that must be kept in the non-volatile storage area is defined in file *hal.h* as follows:

```
/** Configuration key to manage PHY params */
#define HAL_PRIME_PHY_CONFIG_KEY    0xAA99

/** Type to manage PHY params */
typedef struct {
    uint16_t ul_cfg_key;
    uint8_t txrxChannel;
} x_phy_cfg_t;
```

Whenever the system restarts, the PRIME FW stack checks the configuration key stored in the non-volatile storage area. If the stored key matches the value defined by `HAL_PRIME_PHY_CONFIG_KEY`, then the PRIME FW stack considers that the stored values are valid and reads them from the non-volatile storage area. However, if the stored key does not match `HAL_PRIME_PHY_CONFIG_KEY`, then the stored values are discarded and the PRIME FW stack reads and stores the default values from the PHY layer.

The PLC communication channel can be changed by users using vendor specific PIB attribute `PIB_PHY_TX_CHANNEL` (0x8090) described in [Table 12-10](#). This update is only permitted under MTP mode and requires a reset to take effect. The changed value is always stored in the non-volatile storage area. After a reset, since the configuration key matches the defined value `HAL_PRIME_PHY_CONFIG_KEY`, the last value stored in the non-volatile storage area will be the one used by the stack.



Important: Do not change the value assigned to the configuration key `HAL_PRIME_PHY_CONFIG_KEY` because this is a compilation constant in the PRIME FW stack library.

The PRIME FW stack invokes the functions included in file *hal_cfg.c* of the HAL with the configuration type set to `HAL_PHYCFG_OFFSET_USER_SIGN` in order to access the PHY parameters in the non-volatile storage area.

4.5.1.4 PLC Coupling (PL360 Platform)

When using the PL360 platform, the hardware coupling implemented in the user board must only be taken into account when using `ATPLCOUP011`. In that case, the parameter `PAL_ENABLE_C11_CFG` must be defined in file *conf_pal.h*.

```
/* If coupling 11 is chosen */
#define PAL_ENABLE_C11_CFG
```

4.5.1.5 PLC Coupling (PL460 Platform)

When using the PL460 platform, the default configuration uses the auxiliary branch to transmit in channel 1. To use the hardware variant in order to transmit in channel 1 using the main branch, the parameter `PAL_ENABLE_PL460_CEN_A_CFG` must be defined in file *conf_pal.h*.

```
/* PL460 + PLCOUP007 (Single Branch) / PL460 CEN-A (Single Branch) configuration */
#define PAL_ENABLE_PL460_CEN_A_CFG
```

4.5.1.6 Band Plan in PLC

Although the PLC PHY layer can handle several communication channels, for this version of the PRIME specification only channel 1 is supported, as shown in [Table 4-3](#). This is set in `USER_BAND_PLAN` in file *conf_pal.h*.

```
/* Define the band plan */
#define USER_BAND_PLAN    (CHANNEL1)
```


4.5.1.7 MAC Address

A MAC address is mandatory for a normal operation of the PRIME FW stack. The MAC address identifies the node in the network and therefore it must be kept safe in a non-volatile storage area in case the system restarts for any reason.



Important: Note that whenever the system restarts and the PRIME FW stack cannot find a valid MAC address, it enters into MTP mode.

The PRIME FW stack uses a configuration key to determine whether the MAC address is available in the non-volatile storage or not. The structure with the configuration key and the MAC address that must be kept in the non-volatile storage area is defined in file *hal.h* as follows:

```
/** Configuration key to manage MAC address */
#define HAL_MAC_CONFIG_KEY    0xAA55

/** Type to manage MAC address */
typedef struct {
    uint16_t us_cfg_key;
    uint8_t uc_mac[6];
} x_mac_cfg_t;
```

Whenever the system restarts, the PRIME FW stack checks the configuration key stored in the non-volatile storage area. If the stored key does not match the value defined by `HAL_MAC_CONFIG_KEY`, the PRIME FW stack enters into MTP mode. Users can then set the MAC address using the vendor specific PIB 0x8100 (PIB_MTP_MAC_EUI_48 in [Table 12-11](#)). The MAC address is then stored in the non-volatile storage area. If the stored key matches `HAL_MAC_CONFIG_KEY`, the MAC address is read and the PRIME FW stack starts in normal operating mode.



Important: Do not change the value assigned to the configuration key `HAL_MAC_CONFIG_KEY` because this is a compilation constant in the PRIME FW stack library.

The PRIME FW stack invokes the functions included in file *hal_cfg.c* of the HAL with the configuration type set to `HAL_MACCFG_OFFSET_USER_SIGN` in order to access the MAC address in the non-volatile storage area.

4.5.1.8 Enabling PLC PHY Layer

The PLC PHY layer is enabled in file *conf_pal.h* by defining `PAL_PLC`.

```
/* Definition of available PHY layers */
#define PAL_PLC
```

4.5.1.9 Enabling Serial PHY Layer

In a multi-transformer substation, when the Base Node must communicate with an auxiliary node without using the PLC PHY layer, the Serial PHY layer comes into effect (see chapter [Auxiliary Nodes Connection in Multi-transformer Substations](#)). The Serial PHY layer is enabled in file *conf_pal.h* by defining `PAL_SERIAL`.

```
/* Definition of available PHY layers */
#define PAL_SERIAL
```

When using the serial PHY layer, users need to define the port number in file *conf_phy_serial.h*. See chapter [Linking of PRIME PHY Serial](#) for details.

The Serial PHY frame format can be found in chapter [PRIME PHY Serial Frame Format](#).

4.5.1.10 Sniffer Serialization

The different PHY layers include an embedded sniffer, which provides the traffic via serial interface so that it can be analyzed if necessary.

To set the port number for the serialized sniffer in *conf_pal.h*, see chapter [Linking of PRIME Sniffer](#).

To configure the sniffer for the PLC PHY layer, users must enable the code related to the embedded sniffer in file *conf_atpl360.h*:

```
/* Enable ADDONS module */
#define ATPL360_ADDONS_ENABLE
```

To configure the sniffer for the Serial PHY layer, users must enable the code related to the embedded sniffer in file *conf_phy_serial.h*:

```
/* Enable addons for sniffer */
#define PHY_SERIAL_ADDONS_ENABLE
```



Important: When the embedded sniffer code is included, the sniffer serialization can be enabled and disabled through the vendor specific PIB attribute 0x8106 (PIB_PHY_SNIFFER_ENABLED in [Table 12-10](#)).

The sniffer frame format can be found in chapter [PRIME Sniffer Frame Format](#).

4.5.1.11 Serial Communication Profile of PRIME Management Plane

The Management Plane can be accessed through the Serial Communication Profile, as described in the PRIME specification. If users want to use the Serial Communication Profile, they need to define the port number for the serialization in file *conf_mngp.h*. See chapter [Linking of Serial Communication Profile of PRIME Management Plane](#) for details.



Important: Note that the Serial Communication Profile of the Management Plane is required for PRIME certification. The baud rate must be set to 57600 bauds. See the PRIME Certification test book for more information.

4.5.1.12 Base Node Parameters

The PRIME FW stack is implemented in such a way that it does not start assigning LNIIDs from the beginning whenever it is restarted. What it does is setting an offset depending on the maximum number of supported nodes and starting from that number at restart. Therefore it must be kept safe in a non-volatile storage area.

The PRIME FW stack uses a configuration key to determine whether or not the BN parameters are available in the non-volatile storage. The structure with the configuration key and the LNIID offset that must be kept in the non-volatile storage is defined in file *hal.h* as follows:

```
/** Configuration key to manage BN parameters */
#define HAL_PRIME_BN_INFO_CONFIG_KEY_13 0xAA55

/** Type to manage BN parameters */
typedef struct {
    uint16_t key;
    uint16_t mac_lnid_offset;
} x_bn_info_cfg_t;
```

Whenever the system restarts, the PRIME FW stack checks the configuration key stored in the non-volatile storage area. If the stored key does not match the value defined by `HAL_PRIME_BN_INFO_CONFIG_KEY_13`, the LNIID offset is set to 0 and then stored in the non-volatile storage area. If the stored key matches `HAL_PRIME_BN_INFO_CONFIG_KEY_13`, the LNIID offset is read, the next offset is calculated and then stored in the non-volatile storage area.



Important: Do not change the value assigned to the configuration key `HAL_PRIME_BN_INFO_CONFIG_KEY_13` because this is a compilation constant in the PRIME FW stack library.

The PRIME FW stack invokes the functions included in file *hal_cfg.c* of the HAL with the configuration type set to `HAL_BNINFO_OFFSET_USER_SIGN` in order to access the MAC address in the non-volatile storage area.

4.5.1.13 PRIME Mode

The PRIME FW stack library must be configured for the right type of node (board mode) and for the correct version of the PRIME specification. This information must be kept safe in a non-volatile storage area so that it can be recovered in case of restart.

The PRIME FW stack uses a configuration key to determine whether the PRIME mode information is available in the non-volatile storage or not. The structure with the configuration key and the PRIME mode information that must be kept in the non-volatile storage area is defined in file *hal.h* as follows:

```
/** Configuration key to manage PRIME mode */
#define HAL_PRIME_MODE_CONFIG_KEY    0xA55A

/** Type to manage PRIME mode configuration.
 * board_mode indicates board function (PRIME_BN or PRIME_SN)
 * prime_version indicates protocol version (PRIME_1_3, PRIME_1_4 or PRIME_BC)
 */
typedef struct {
    uint16_t key;
    uint8_t prime_version;
    uint8_t board_mode;
} x_prime_mode_info_cfg_t;
```

Whenever the system restarts, the PRIME FW stack checks the configuration key stored in the non-volatile storage area. If the stored key does not match the value defined by `HAL_PRIME_MODE_CONFIG_KEY`, the PRIME FW stack reads and stores the default values defined in file *conf_mac.h*. If the stored key matches `HAL_PRIME_MODE_CONFIG_KEY` but any of the stored values does not match the value in file *conf_mac.h*, the values defined in *conf_mac.h* are read and stored.



Important: Do not change the value assigned to the configuration key `HAL_PRIME_MODE_CONFIG_KEY` because this is a compilation constant in the PRIME FW stack library.

The PRIME FW stack invokes the functions included in file *hal_cfg.c* of the HAL with the configuration type set to `HAL_PRIME_MODE_OFFSET_USER_SIGN` in order to access the MAC address in the non-volatile storage area.

4.5.1.14 Zero Cross Detection in PLC

When using the PL360/PL460 platform, if there is a zero cross detection circuit in the user board, it must be configured appropriately with the parameters available in the PL360 device. For additional information, refer to the "PL360 Host Controller User Guide".

4.5.1.15 Network Behavior

The behavior of the PRIME FW stack can be altered by means of the PIB attributes described in the PRIME specification.

The PRIME specification defines default values for some of these PIB attributes but the current implementation is not always using the default ones. These modified values have been set to achieve maximum performance on real networks.

The following table lists all read-write PIB attributes together with their default values according to the PRIME specification and the values configured in the PRIME FW stack. Users are free to modify any of these values through MLME or Management Plane primitives in order to customize the PRIME FW stack according to their needs. These PIB attributes can be found in file *mac_pib.h*.

Table 4-4. Configured PIB Values in the PRIME FW Stack

Attribute	ID	PRIME default value	Implementation value
PIB_MAC_BEACONS_PER_FRAME	0x0013	5	1
PIB_MAC_SCP_MAX_TX_ATTEMPTS	0x0014	5	5
PIB_MAC_CTL_RE_TX_TIMER	0x0015	15	2
PIB_MAC_MAX_CTL_RE_TX	0x0018	3	5
PIB_MAC_EMA_SMOOTHING	0x0019	3	0

4.5.2 Application Configuration Parameters

4.5.2.1 Example Configuration – *conf_app_example.h*

In this file, users need to define anything related to their application. For example, the reserved memory addresses, communication ports, etc.

4.5.2.2 HAL Configuration – *conf_hal.h*

Users need to define in this file the configurable hardware parameters according to their HAL code. These can be timers, buffer sizes, interruptions, etc. (see chapter [Hardware Resources Usage](#)).

4.5.2.3 PRIME Stack Configuration – *conf_prime_stack.h*

This file enables parameter `PRIME_API_SEPARATED_APPS` to indicate that the PRIME FW stack is separated from the user application. Currently, Microchip only provides support for non-separated applications; that is, one project contains both the PRIME FW stack library as well as the user application.

4.5.2.4 USI Configuration – *conf_usi.h*

This file configures the characteristics of the ports used for serializations. More information about this file can be found in chapter [Definition and Configuration of Serial Ports](#).

5. Data Exchange

Users are free to use the provided specific convergence sublayers or access directly the MAC layer to establish a MAC connection to exchange data between application peers.

Please refer to PRIME specification for detailed information about the semantics of the primitives available in the MAC service access point.

Please note that an Automatic Repeat Request (ARQ) mechanism is available in order to provide guaranteed communications between peers.



Important: Remember that all requests to exchange data are followed by a confirm and that the application must always wait for it before performing a new request. The confirm can be immediate or be delayed for a considerable amount of time. For communications with ARQ, the invocation of the confirm callback could take up to several minutes. More information about the request/confirm mechanism can be found in chapter [PRIME API](#).

5.1 Null SSCS

Users can directly access the MAC layer through the Null SSCS.

Available signalling primitives can be found in [Table 12-1](#).

Available data primitives can be found in [Table 12-2](#).

5.2 IEC 61334-4-32

Users can make use of the provided IEC 61334-4-32 compliant SSCS.

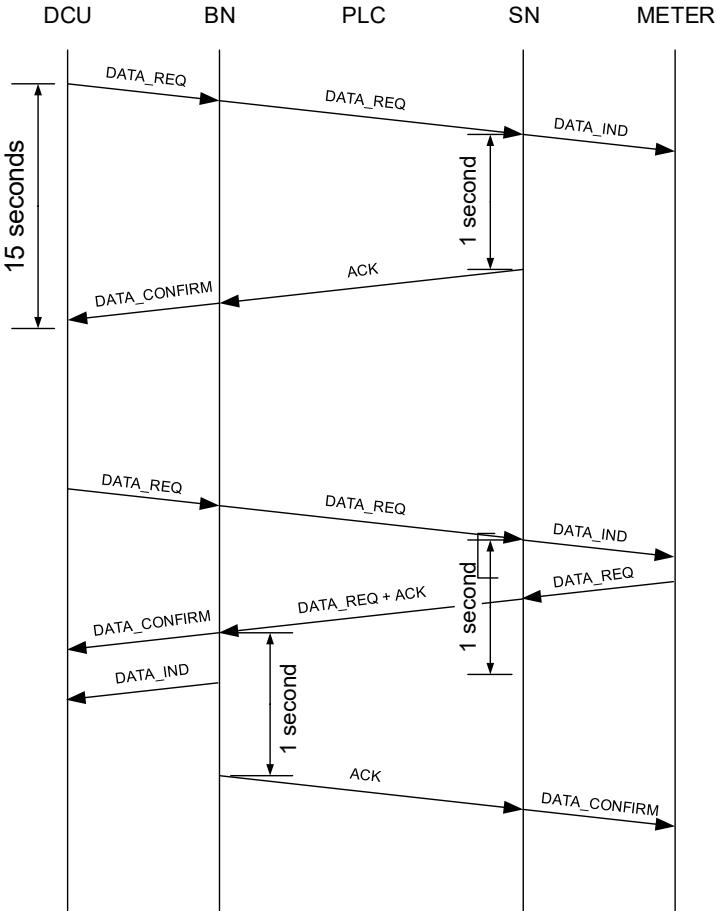
Available IEC 61334-4-34 primitives can be found in [Table 12-5](#).

5.3 Automatic Repeat Request (ARQ)

The PRIME specification defines two kinds of ARQ: windowed and “*Stop and wait*”. The current implementation of the PRIME FW stack only supports windowed ARQ with a window size fixed to 4.

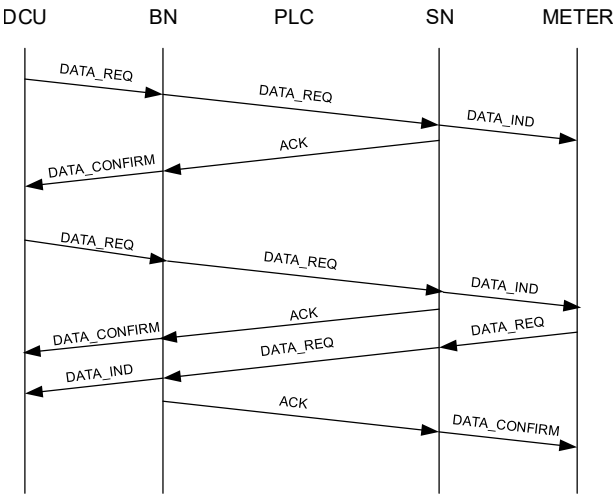
The following figure shows how ARQ works. If the application needs more than one second to send the reply, the Service Node sends the ACK after that time. If the response is received within one second, the PRIME FW stack piggybacks the ACK in the response.

Figure 5-1. ARQ with One Fragment and Piggyback



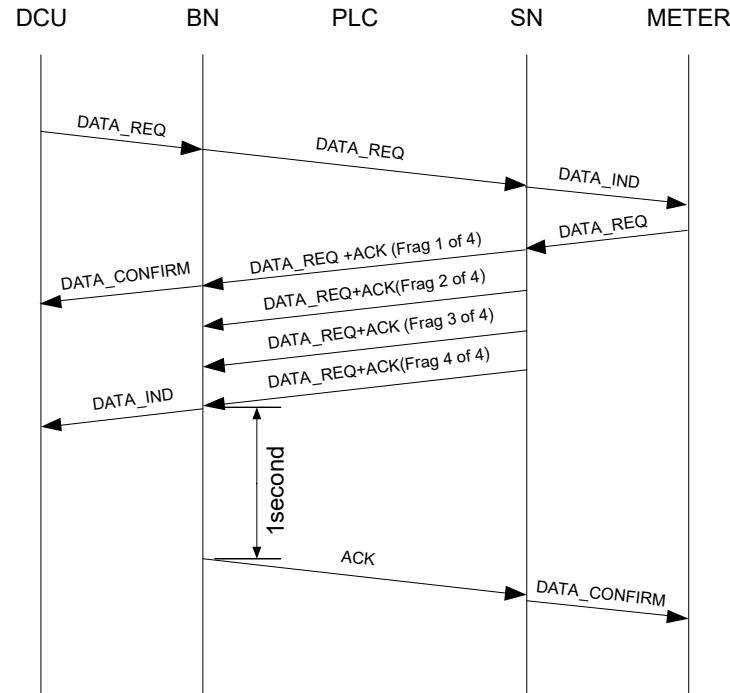
In certain situations, it might be possible that the ACK is sent as soon as the request is received, i.e. it is never piggybacked. That is shown in the next figure.

Figure 5-2. ARQ with One Fragment



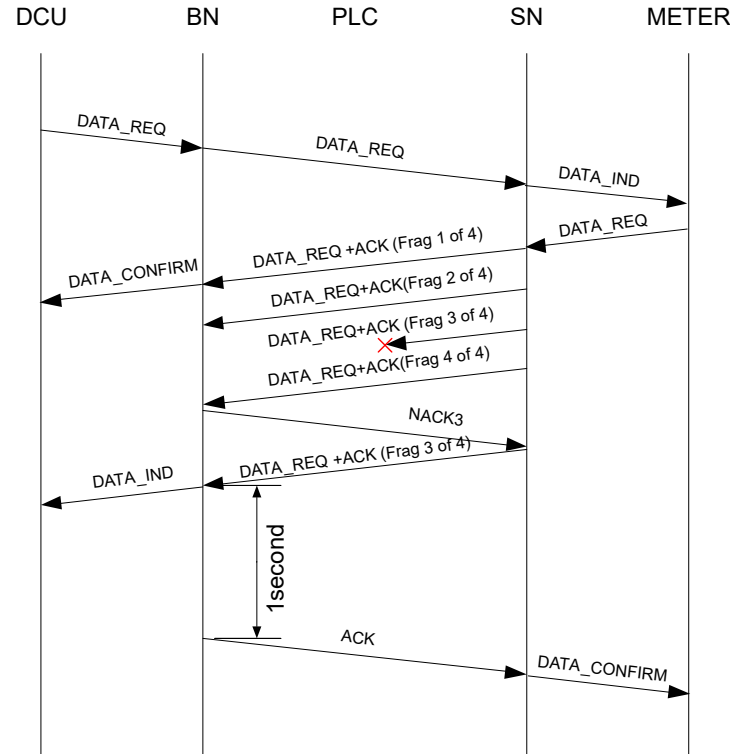
For long fragmented responses, there is an ACK piggybacked to each fragment and the BN sends the ACK after receiving the last fragment, as shown in the following figure.

Figure 5-3. ARQ with Four Fragments



It is possible that a fragment with its corresponding ACK is lost. In that case, a NACK with the missing fragment is sent, so that the Service Node can resend it. This is shown in the following figure.

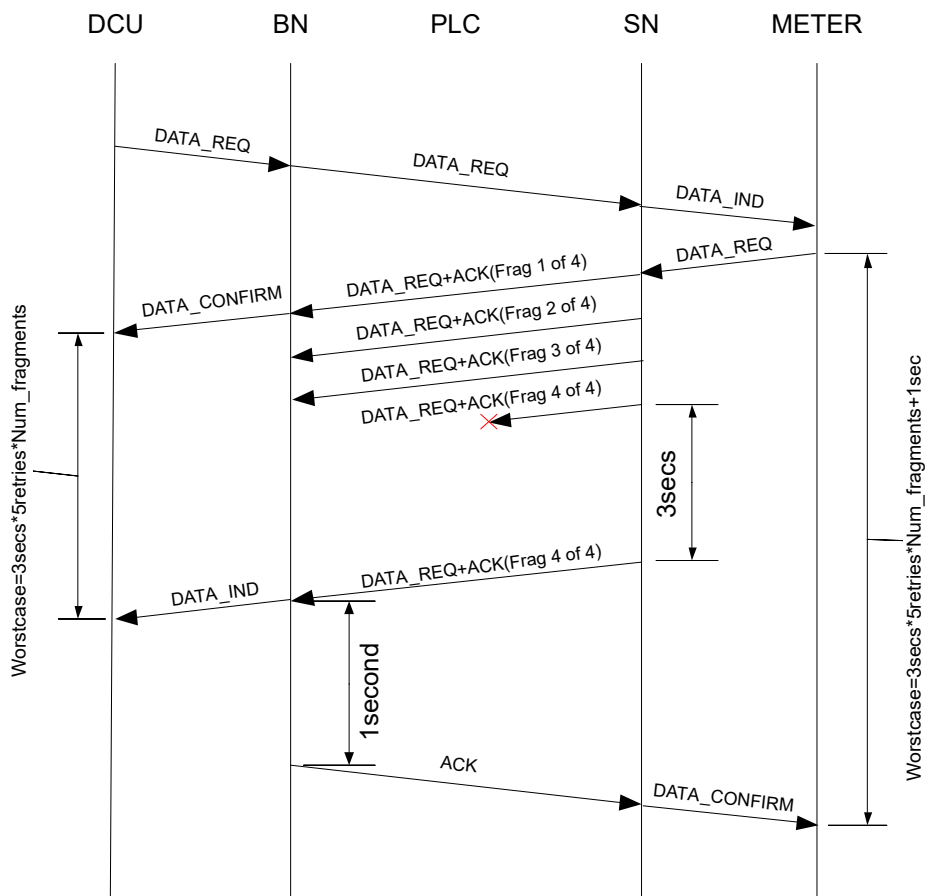
Figure 5-4. ARQ with Four Fragments and Retransmission



The worst case scenario occurs when the last fragment is lost because then the Service Node does not receive any ACK or NACK from the Base Node. In this case, the Service Node waits for 3 seconds

before retransmitting the last fragment. If the ACK or NACK is still not received, the Service Node makes up to 5 retransmissions. After all the retransmissions, the disconnection process starts.

Figure 5-5. ARQ with Four Fragments and Maximum Number of Retransmissions



6. PRIME Management Plane

The Management Plane enables a local or remote control entity to perform actions on a node. These actions include providing access to internal parameters defined by PIB attributes as well as managing the firmware upgrade.

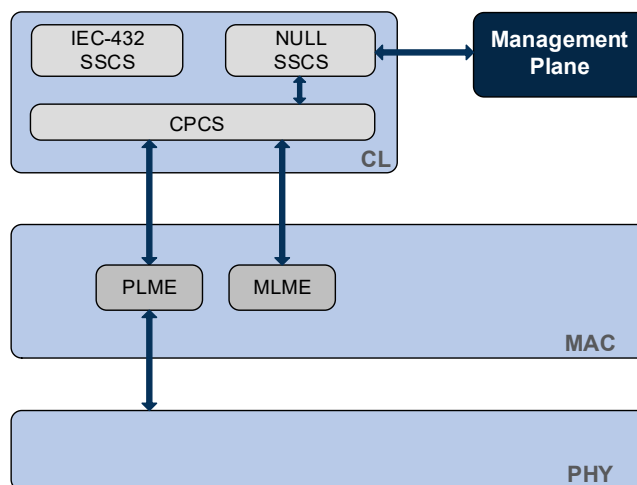
Node management in the PRIME FW stack is accomplished through a set of attributes. Attributes are defined for both PHY and MAC layers. The set of these management attributes is called PLC Information Base (PIB). Some attributes are read-only, while others are read-write. For details about implementation of PIB attributes described in the PRIME specification, see chapter [Major Capabilities of the MAC Layer](#) and chapter [Major Capabilities of the Management Plane](#). The PRIME FW stack also implements vendor specific PIB attributes, which are described in chapter [PIB Objects Specification and Access](#). Additionally, Microchip offers an interface through the HAL that enables the implementation of user specific PIBs in the application. Those user specific PIBs can only be defined with a size of 4 bytes in the ID range 0xF000 - 0xFCFF.

The Management Plane in the Base Node cannot only be used to control the Base Node, but also to access the Management Planes in the Service Nodes and perform actions on them. In order to fulfill both actions, the control entity can access the Management Plane by using the two communication profiles defined in the PRIME specification:

- **PRIME Profile.** It is used by the local control entity to perform remote actions on a Service Node. The Base Node receives the requests in the local Management Plane from the Base Management (see chapter [API for PRIME Profile in Management Plane](#)) and passes them to the Null SSCS in the CL (see chapter [Null SSCS](#)) so that they are sent by PLC to the specific Service Node
- **Serial Communication Profile.** It is implemented as a service in the USI of the HAL and is used by the local control entity. For more information about the different services available in the USI, see chapter [Serialization with Embedded USI](#)

Whenever the Management Plane receives a request from any of the specified profiles, it passes it to the MAC layer using the functions of the Null SSCS in the CL (see chapter [Null SSCS](#)). This access is depicted in the next figure.

Figure 6-1. Management Plane Access to the MAC Layer





Important: Please note that the PRIME Certification Tool needs to access the Management Plane through the Serial Communication Profile and therefore it must be configured appropriately (see [Serial Communication Profile of PRIME Management Plane](#)).

7. Base Management

7.1 Overview

The Base Management Protocol is a Microchip proprietary extension over the PRIME specification to increase the functionalities of a Base Node. The PRIME specification does not always define the interfaces between the Base Node and the DCU user application, so new interfaces must be implemented as required.

The Base Management API is defined in chapter [Base Management Primitives](#).

Available functionalities are:

- Firmware Upgrade Protocol interface. This interface lets the DCU user application configure the firmware upgrade of the registered devices in the network
- Network events. The Base Node notifies about events such as new registrations, unregistrations, etc
- Access to the PRIME Profile in the Management Plane. This interface allows the DCU user application to manage PIBs in a remote node
- Whitelist management. If the whitelist mechanism is enabled, this interface allows the DCU user application to add and remove nodes from the whitelist



Important: Note that the DCU user application is not in the Base Node with the PRIME FW stack. For simplicity reasons, whenever DCU user application is mentioned in the next sections, it means that the DCU user application is commanding the actions to the user application in the Base Node. The user application in the Base Node must use the Base Management API defined in chapter [Base Management Primitives](#). It is up to the users to define the interface between the DCU user application and the user application in the Base Node in order to handle the Base Management API.

7.2 Firmware Upgrade Protocol

The FUP defines the steps required to set up and initiate a firmware upgrade in a PRIME PLC network and also monitor the upgrade process from the Base Node. The PRIME FW stack implements the PRIME FU process in the Management Plane as described in the PRIME specification. Please refer to the PRIME specification for details.

The FUP uses the following configuration parameters:

- List of Service Nodes to upgrade
- Binary file configuration (Vendor, Model and Version)
- PLC protocol options:
 - ARQ: enables ARQ
 - Multicast: enables multicasting to transfer the firmware image through the PLC
 - Page size: size of the data chunks used during the PLC phase of the FU
 - Restart delay: time that a Service Node waits before restarting with the new image
 - Safety timer: time a Service Node must wait before reverting to a former firmware image when the new image is not confirmed by the Base Node

The FUP has a list of message commands related to primitives and callbacks that are defined in [FUP Primitives in the Base Node](#).

7.2.1 FUP Stages

The FUP is divided into the following stages:

- Setup: The DCU user application indicates to the PRIME FW Stack which Service Nodes must be upgraded and configures the upgrade options
- File transfer: The DCU user application sends the binary file to the PRIME FW Stack
- Firmware upgrade: The PRIME FW Stack proceeds with the firmware upgrade as defined in the PRIME specification

7.2.1.1 Setup Stage

In the setup stage, the Service Nodes that will be upgraded and the options for the protocol are defined. The DCU user application must execute the next sequence of actions:

1. Configure the PLC FUP options: ARQ, Multicast, Page Size, Delay Restart and Safety Timer
2. Clear the BN's target list
3. Add device targets from the list of registered devices in the BN. Devices are added one-by-one
4. Configure the upgrade rule: select which nodes will be upgraded based on their Vendor/Model
5. Set the new firmware version information: Vendor, Model and Version

Unregistered Service Nodes that have been registered previously can also be added to the FU list. When they register again, they will be automatically added to the FU process by the Base Node.

[Setup Sequence Diagram](#) shows the sequence of primitives exchanged between the DCU user application and the Base Node in this stage. [FUP Primitives in the Setup Stage](#) describes those FUP primitives, whose parameters are defined in [FUP Primitives in the Base Node](#).

Table 7-1. FUP Primitives in the Setup Stage

FUP primitive	Description
FUP_CLEAR_TARGET_LIST.request	This primitive clears the target list.
FUP_ADD_TARGET.request	This primitive adds an SN to the target list.
FUP_SET_FW_DATA.request	This primitive sets the new version information: Vendor, Model and Version.
FUP_SET_UPG_OPTIONS.request	This primitive configures the protocol options: <ul style="list-style-type: none"> • ARQ: to enable (1) or disable (0) the usage of ARQ in PLC communications • Multicast: to enable (1) or disable (0) the usage of a multicast group for the firmware upgrade • Page size: automatic (0), 32 bytes (1), 64 bytes (2), 128 bytes (3), 192 bytes (4) • Delay restart: time before restarting with the new firmware (0 - 65536 seconds) • Safety timer: time to test the new firmware (0 - 65536 seconds)
FUP_SET_MATCH_RULE.request	This primitive sets the Vendor and Model applicable for the firmware upgrade. If a rule is given for Vendor and/or Model (the corresponding bits of 0000 0MV0 are set), only the SNs in the target list matching the configured Vendor and/or Model will be upgraded.
FUP_ACK.indication	This primitive acknowledges the last request made and informs of any errors.

The possible return values in the FUP_ACK.indication are the following:

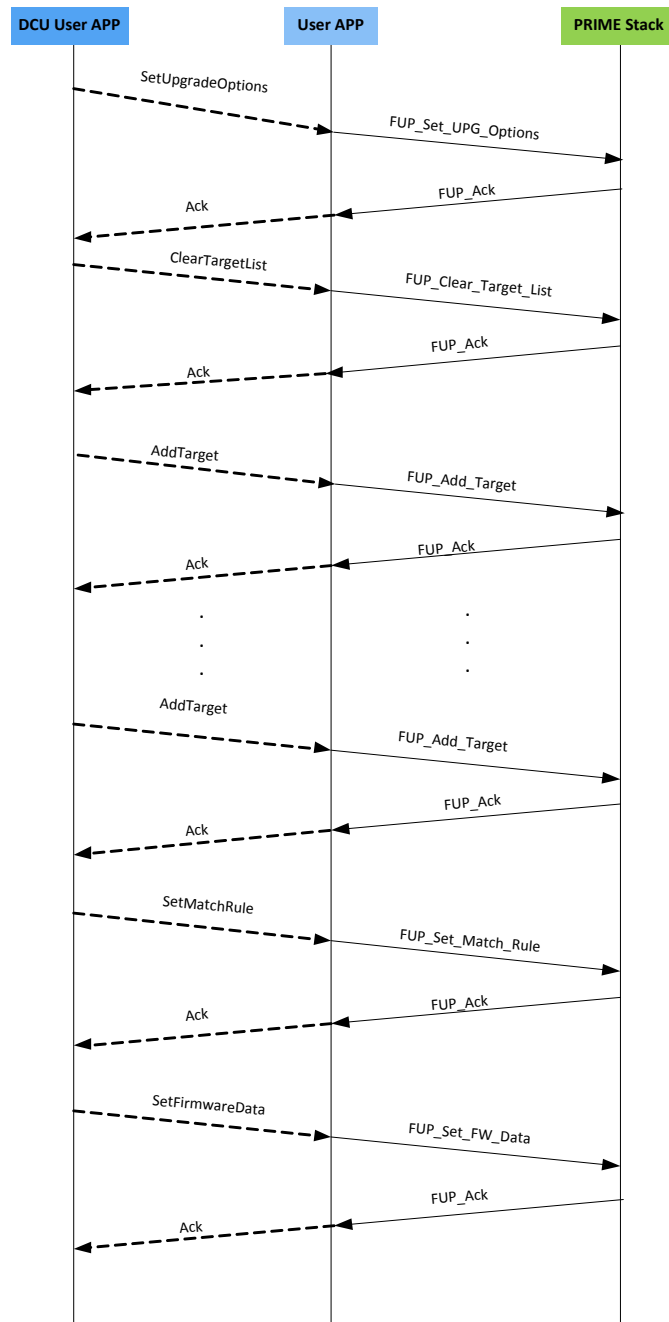
```
/** FUP ACK codes */
typedef enum {
    FUP_ACK_OK = 0,
```

```

FUP_ACK_ERROR,
FUP_ACK_ERROR_MAC,
FUP_ACK_ERROR_MODEL,
FUP_ACK_ERROR_CRC,
FUP_ACK_ERROR_DATA,
FUP_ACK_ERROR_CRC_FILE,
FUP_ACK_CRC_ONGOING,
FUP_ACK_FU_ONGOING
} fup_ack_code_t;

```

Figure 7-1. Setup Sequence Diagram



7.2.1.2 File Transfer Stage

The file transfer algorithm is based on the TFTP protocol (Trivial File Transfer Protocol). First, the DCU user application sends an initialization message to the Base Node with the following information:

- Firmware size: Size of the firmware file.
- Frame size: Size of the data fragments to be sent.
- Firmware CRC: PRIME CRC-32 of the firmware file.

After the initialization request, the file is sent to the Base Node in fragments of fixed length, the indicated frame size. Each fragment contains part of the file and must be acknowledged by the Base Node before sending the next fragment. A fragment of smaller size than the frame size signals the termination of the file transfer.



Important: If the last fragment has the same length as the fragment size, the DCU user application must send a fragment of size zero, as indicated in the TFTP protocol.

After the file transfer is completed, the Base Node computes the PRIME CRC-32 of the received file. This CRC-32 is compared to the value received in the initialization message. The DCU user application must ask the Base Node for the result. If it is not correct, the upgrade is cancelled by the Base Node.

[File Transfer Sequence Diagram](#) shows the sequence of primitives exchanged between the DCU user application and the Base Node in this stage. Note that the PRIME FW stack invokes several functions of the HAL in order to store the file and calculate its CRC. [FUP Primitives in the File Transfer Stage](#) describes those FUP primitives, whose parameters are defined in [FUP Primitives in the Base Node](#).



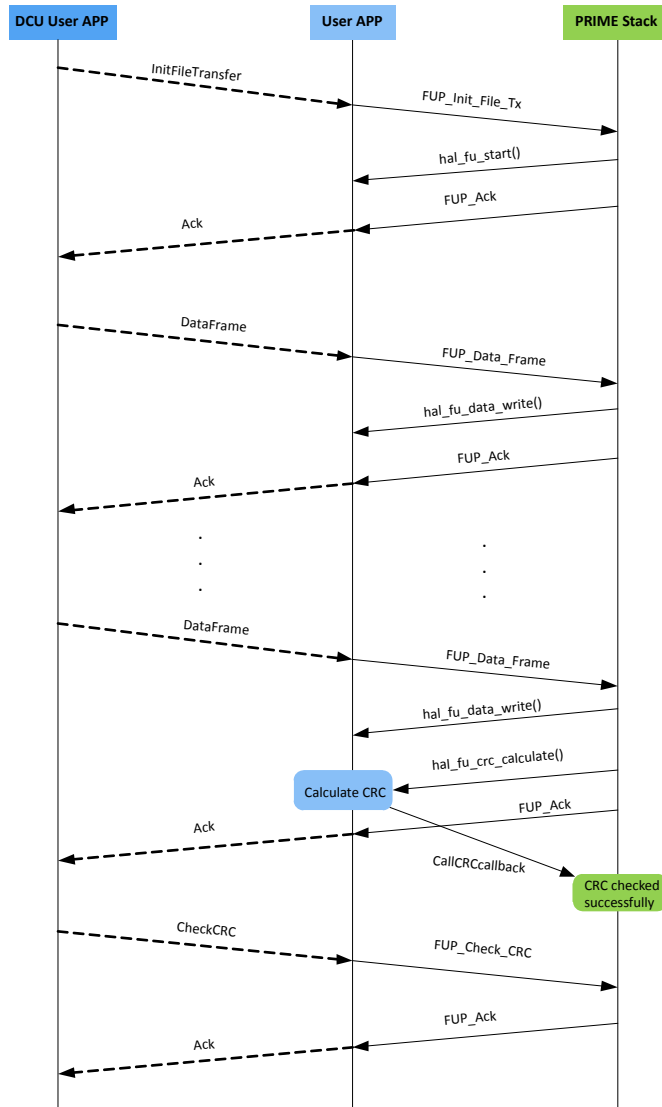
Important: Users are responsible for reserving a memory area with enough space to store the firmware file (see chapter [Hardware Resources Usage](#)).

Table 7-2. FUP Primitives in the File Transfer Stage

FUP primitive	Description
FUP_INIT_FILE_TX.request	This primitive indicates the beginning of a file transfer and configures the following parameters: <ul style="list-style-type: none"> • The size of the firmware file • The size of the data fragments • The CRC-32 of the firmware file
FUP_DATA_FRAME.request	This primitive forwards a fragment of the firmware file.
FUP_CHECK_CRC.request	This primitive requests the result of the CRC calculation.
FUP_ACK.indication	This primitive acknowledges the last request made and informs of any errors. When acknowledging a data frame, it also provides the number of data fragment.

The possible return values in the FUP_ACK.indication are described in the previous chapter.

Figure 7-2. File Transfer Sequence Diagram



As mentioned before, the PRIME FW stack invokes several functions of the HAL in order to store the file and calculate its CRC. Users are responsible for implementing the following functions according to their design.

CRC Result Callback

The HAL is responsible for calculating the CRC of the received file when requested by the PRIME FW stack. The calculated result is provided invoking a callback function whose pointer is set at initialization by the PRIME FW stack with the following function:

```
void hal_fu_crc_set_callback(void (*p_handler)(uint32_t ul_crc));
```

Parameters:

- *p_handler*: Pointer address to the callback function

Starting the File Transfer

The PRIME FW stack indicates it to the user application with the following function:

```
void hal_fu_start(hal_fu_info_t *x_fu_info);
```

Parameters:

- *x_fu_info*: Pointer to FU information:

```
typedef struct {
    uint32_t image_size;
    uint8_t page_size;
    hal_fu_signature_algorithm_t sign_algorithm;
    uint16_t sign_length;
} hal_fu_info_t;
```

The HAL stores the received information and erases the region where the file will be safely stored.

Note: In this version of the specification, the signature of the firmware file is not supported.

Writing Firmware Data

When the PRIME FW stack has received a frame of the file, it requests the HAL to write it with the following function:

```
uint8_t hal_fu_data_write(uint32_t addr, uint8_t *puc_buf, uint16_t us_size);
```

Parameters:

- *addr*: Address of page to write
- *puc_buf*: Pointer to page to write
- *us_size*: Page size

Result: 1 if there is no error. Otherwise, 0.

CRC Calculation

When the file has been received, the PRIME FW stack needs to verify that the received file is correct and thus it requests the HAL to calculate its CRC with the following function:

```
void hal_fu_crc_calculate(void);
```

The HAL can calculate the CRC over the complete file, but, as this can take some time for large files, it is recommended to calculate it page-by-page in a process function. The result is returned through the CRC callback function as defined previously.

7.2.1.3 Firmware Upgrade Stage

If the previous stages have been completed correctly, it is safe to trigger the PLC FU process with primitive FUP_START_FU.request. During this stage, the Base Node follows the process defined by the PRIME specification to upgrade the firmware in the Service Nodes.

Once the PLC FU stage has started, the DCU user application can cancel the FU process any time with primitive FUP_ABORT_FU.request. The cancellation can apply to all Service Nodes or just to a certain Service Node. New Service Nodes can also be added to the FU process any time with the primitive FUP_ADD_TARGET.request described in the setup stage. All these requests are acknowledged by the primitive FUP_ACK.indication.

Furthermore, the DCU user application can also request the version (FUP_GET_VERSION.request) and FU state (FUP_GET_STATE.request) of any SN. The response to these primitives is received in the corresponding indication primitive.

The indication primitives, which provide information about the status of the upgrade process asynchronously, are issued with callback functions. If interested in receiving these notifications, the user application must set up the corresponding callback function pointers at initialization. [Table 7-3](#) describes the FUP indication primitives, whose parameters are defined in defined in [FUP Primitives in the Base Node](#). To set up the callback function pointers, see chapter [Callback Functions in the Base Management](#).

Table 7-3. FUP Indication Primitives in the Firmware Upgrade State

FUP primitive	Description
FUP_STATUS.indication	This primitive gives information about the progress of the FU in an SN by providing the SN FU state as defined in the PRIME specification. Additionally, the FUP has added the status <code>FUP_STATE_ENDED_NOTIFICATION</code> that corresponds to the Idle state once the new firmware has been confirmed. When the node is in <code>FUP_NODE_STATE_RECEIVING</code> , this primitive also returns the number of pages received by the SN and confirmed by the BN (the real number of received pages could be bigger). To mark the end of the FU process, this primitive is issued with state <code>FUP_STATE_ENDED_NOTIFICATION</code> for the broadcast MAC address (FF:FF:FF:FF:FF:FF).
FUP_VERSION.indication	This primitive provides information about Vendor, Model and Version from an SN.
FUP_KILL.indication	This primitive indicates that an SN has cancelled its firmware upgrade.
FUP_ERROR.indication	[DEPRECATED]

7.3 Network Events

The Base Node notifies about asynchronous events related to the Service Nodes whenever the corresponding callback function (`BMNG_NETWORK_EVENT.indication`, defined in [Table 12-7](#)) has been set up by the user application. The events indicated by the Base Node are the following:

- State changes in the Service Node:
 - registration (`BMNG_NET_EVENT_REGISTER`)
 - unregistration (`BMNG_NET_EVENT_UNREGISTER`)
 - promotion (`BMNG_NET_EVENT_PROMOTE`)
 - demotion (`BMNG_NET_EVENT_DEMOTE`)
- Alive events (`BMNG_NET_EVENT_ALIVE`): whenever the Base Node receives an alive message from a Service Node
- Reboot events (`BMNG_NET_EVENT_REBOOT`): whenever the PRIME FW stack is started in the Base Node
- Unknown Service Node (`BMNG_NET_EVENT_UNKNOWN_NODE`): whenever the Base Node receives a registration message from a Service Node which is not included in the whitelist. This only applies when the whitelist mechanism is enabled

7.4 API for PRIME Profile in Management Plane

The PRIME Profile is a communication protocol in the Management Plane that permits the DCU user application to manage a remote Service Node through PRIME standard and Microchip-defined PIBs, refer to the Chapter [PRIME Management Plane](#). This communication profile is defined in the PRIME specification, but the interface to the DCU user application is not. Therefore, Microchip has defined a proprietary API to allow access to the PRIME Profile in the Management Plane. This API with its corresponding primitives and callback functions is defined in [Table 12-8](#).

This API has been enhanced with a request to calculate the difference between the zero cross times in the Base Node and a Service Node, which is useful in a phase detection algorithm.

7.5 Whitelist Management

When the user application has enabled the use of the whitelist mechanism via the Microchip defined PIB `PIB_MAC_WHITELIST_ENABLED` (0x8151), the whitelist management interface allows addition and removal of Service Nodes to the whitelist. This API with its corresponding primitives and callback functions is defined in [Table 12-9](#).

By default, the use of the whitelist is disabled. When it is enabled by the user application, all Service Nodes that were already registered are automatically added to the whitelist.

When a Service Node is removed from the whitelist, it is also unregistered from the network.

8. Toolchain

The following sections describe the required tools and toolchain for the development and build process and how the provided example applications can be built.

8.1 General Prerequisites

The following tools and toolchains are used for building the applications from this firmware package: IAR Embedded Workbench for Arm® V9.20 (see <http://www.iar.com/>).

8.2 Building the Applications

8.2.1 Using IAR Embedded Workbench

All provided projects can be built and loaded into the board using the IAR Embedded Workbench directly.

For each example application described in Chapter [Example Applications](#), follow these steps:

1. Open the corresponding project.
2. Configure the project as required (see Chapter [Configuration Parameters](#)).
3. Build the project.
4. Program the board.

In the PL360/PL460 platform, the PL360 firmware is also required for a system to be operative in PLC but it is directly provided as a binary file.



Important: Make sure that the Flash memory is erased before programming it.

9. Supported Platforms

This chapter describes which hardware platforms are currently supported with the PRIME firmware package. A platform usually comprises of three major components:

- An MCU
- A modem chip (this may be integrated into the MCU for single chips)
- A specific hardware that contains the MCU and/or the modem chip

9.1 Supported MCU Families

The only supported generic MCU family are the SAME70 and PIC32CXMTx platforms. The SAMG55 platforms are intended for evaluation purposes only.

The dedicated code for each device of the family can be found in the corresponding subdirectories.

9.2 Supported Devices

Currently the only supported devices are SAME70Q21B with PL360, SAME70 Xplained with PL460, PIC32CX2051MTSH128 with PL460, and PIC32CX2051MTG128 with PL460. The following are intended for evaluation purposes only: SAMG55J19 with PL460.

9.3 Supported Boards

The currently supported boards and combinations are given below:

- Base Node reference design PL360BN, implementing SAME70Q21B host plus PL360 PLC device
- SAME70 Xplained with PL460-EK on Xplained port
- PIC32CXMTSH-DB with PL460-EK on Xplained port
- PIC32CXMTG-EK with PL460-EK on Xplained port
- SAMG55 Xplained with PL460-EK on Xplained port (BN for evaluation purposes only)

10. PICS

This chapter lists the conformance of the Microchip PRIME implementation with the requirements and optional features as defined by the PRIME specification document.

A Protocol Implementation Conformance Statement (PICS) is a declaration listing the capabilities and options supported by an implementation. The PICS is based on a list of options and values, defined in the PRIME specification and in the test suites used by the certification process.

10.1 Major Roles for Devices Compliant with PRIME

10.1.1 Major Capabilities of the PHY Layer

The current PHY layer is the implementation of the PRIME PHY layer as specified in PRIME specification versions 1.3.6 and 1.4.

10.1.1.1 PHY Frames

The current PHY layer supports frame types A, B and BC specified in PRIME specification version 1.4.

10.1.1.2 PLME Primitives

The current implementation of the PHY layer does not support the following optional functionalities of the PRIME specification: suspend and resume (primitives PLME_SLEEP.request and PLME_RESUME.request) and test mode (primitive PLME_TESTMODE.request).

10.1.2 Major Capabilities of the MAC Layer

The current MAC layer is compliant with PRIME specification version 1.3.6 and is valid for a Base Node.

The following table shows which optional features of the PRIME specification are currently implemented in the MAC layer provided by Microchip.

Table 10-1. Implementation of Optional Features in the MAC Layer

Feature	Implementation
Direct Connection	Not available
ARQ	Available
Packet Aggregation	Not available
Multicasting	Available ¹
Contention-Free Service	Not available
Security Profile 1	Not available
Roaming	Not available

Note:

1. Switch Nodes handle multicast packets as broadcast packets since the tracking of nodes in a Switch Node is not performed.

10.1.2.1 PLC Information Base

All mandatory PHY and MAC PIB attributes defined in PRIME specification version 1.3.6 for a Base Node are available in the current MAC layer through PLME and MLME.

Optional PIB attributes from PRIME specification version 1.3.6 currently implemented in PLME and MLME can be found in file *mac_pib.h*.

For details about vendor specific PIB attributes, see chapter [PIB Objects Specification and Access](#).

10.1.3 Major Capabilities of the Convergence Layer

The current Convergence layer is compliant with PRIME specification version 1.3.6 and is valid for a Base Node.

The following table shows which Service-Specific Convergence Sublayers are currently implemented in the CL provided by Microchip.

Table 10-2. Implementation of SSCS

SSCS	Implementation
Null	Available
IPv4	Not available
IEC 61334-4-32	Available
IPv6	Not available

10.1.4 Major Capabilities of the Management Plane

10.1.4.1 PLC Information Base

All mandatory PIB attributes defined in PRIME specification version 1.3.6 for a Base Node are available in the current Management Plane of the PRIME FW stack.

Optional PIB attributes from PRIME specification version 1.3.6 currently implemented in the Management Plane can be found in file *mac_pib.h*.

For details about vendor specific PIB attributes, see chapter [PIB Objects Specification and Access](#).

10.1.4.2 Communications Profiles

The Management Plane implements both the interface over the Null SSCS (PRIME Profile) and the interface over the local serial link (Serial Communication Profile) according to PRIME specification version 1.3.6.

11. API of PHY and PAL Layers

The following API is to be used by applications which do not integrate the MAC layer, i.e. that only require transmission and reception of messages through the PHY layer.

11.1 PLC PHY SAP

The PLC PHY SAP corresponds to the PHY API described in header files *atpl360.h* and *atpl360_comm.h*. More information about management, communication and configuration functions of the PHY API can be found in User Guide *PL360 Host Controller*.

11.2 Serial PHY SAP

The Serial PHY SAP corresponds to the PHY API described in the header file *phy_serial.h*. It is the same as the PLC PHY SAP, although some parameters are not used in the Serial PHY layer.

11.3 PAL SAP

The PAL SAP corresponds to the PAL API described in the header file *pal.h*.

11.3.1 Initialization Function

The PAL layer must always be initialized when the system starts the execution. The following function is used for that purpose:

```
void pal_init(void);
```

Depending on the PAL configuration, this function initializes the transmission handlers according to the available PHY layers.

11.3.2 Process Function

Every program cycle the PAL layer must check transmission and reception events. This is achieved by calling the following function:

```
void pal_process(void);
```

This function is also responsible for invoking the corresponding callback functions.

11.3.3 Callback Functions

When a previously requested transmission finishes (data confirm) and when a frame is received (data indication), the PAL layer informs about it by calling the corresponding callback function. To set up the callback function pointers, the following function is required:

```
void pal_set_callbacks(pal_callbacks_t *pal_cbs);
```

Parameters:

- *pal_cbs*: Pointer to the callback structure:

```
typedef struct TPalCallbacks {
    pal_data_confirm_cb_t data_confirm;
    pal_data_indication_cb_t data_indication;
} pal_callbacks_t;
```

11.3.4 Noise Capture Function (PL360/PL460 Platform)

The PL360 device offers a functionality to capture noise in the channel. The PAL layer can access this service with the following function:

```
uint16_t pal_get_noise_capture(uint8_t *puc_dst, uint8_t uc_mode, uint32_t ul_time_start,
uint32_t ul_duration);
```

Parameters:

- *puc_dst*: Pointer to destination buffer to store data
- *uc_mode*: Capture mode
- *ul_time_start*: Start time in μ s based on PL360 timer reference
- *ul_duration*: Duration time in μ s

Return value: Size in bytes of noise capture.



Important: During the noise capture, any ongoing transmission or reception is cancelled.

11.4 PAL Primitives

The PAL primitives are described in header file *pal.h*.

11.4.1 Data Primitives

The PAL implements the data interface to the PHY layer based on the PHY primitives from the PRIME specification.

Table 11-1. PHY Data Plane Primitives

PHY primitive	Function
PHY_DATA.request	<code>uint8_t pal_data_request(x_pal_msg_tx_t *px_msg);</code>
PHY_DATA.confirm	<code>typedef void (*pal_data_confirm_cb_t)(x_pal_data_cfm_t *px_data_cfm);</code>
PHY_DATA.indication	<code>typedef void (*pal_data_indication_cb_t)(x_pal_data_ind_t *px_data_ind);</code>

11.4.2 Management Primitives

The PAL implements the control interface to the PHY layer based on the PHY primitives from the PRIME specification.

Table 11-2. PHY Control Plane Primitives

PHY primitive	Function	PHY layer
PHY_AGC.set ¹	<code>uint8_t pal_agc_set(uint8_t uc_mode, uint8_t uc_gain, uint16_t us_pch);</code>	Only in PLC PHY
PHY_AGC.get	<code>uint8_t pal_agc_get(uint16_t us_id, void *p_val, uint16_t uc_len, uint16_t us_pch);</code>	Only in PLC PHY
PHY_AGC.confirm	Referenced parameters of <i>pal_agc_get</i> .	Only in PLC PHY
PHY_TIMER.get	<code>uint8_t pal_timer_get(uint32_t *pul_timer, uint16_t us_pch);</code>	All
PHY_TIMER.confirm	Referenced parameters of <i>pal_timer_get</i> .	All
PHY_CD.get	<code>uint8_t pal_cd_get(uint8_t *puc_cd, uint8_t *puc_rssi, uint32_t *pul_time, uint8_t *puc_header, uint16_t us_pch);</code>	Only in PLC PHY
PHY_CD.confirm	Referenced parameters of <i>pal_cd_get</i> .	Only in PLC PHY
PHY_NL.get	<code>uint8_t pal_nl_get(uint8_t *puc_noise, uint16_t us_pch);</code>	All except Serial PHY
PHY_NL.confirm	Referenced parameters of <i>pal_nl_get</i> .	All except Serial PHY
PHY_SNR.get	<code>uint8_t pal_snr_get(uint8_t *puc_snr, uint8_t uc_qt, uint16_t us_pch);</code>	All except Serial PHY
PHY_SNR.confirm	Referenced parameters of <i>pal_snr_get</i> .	All except Serial PHY
PHY_ZCT.get	<code>uint8_t pal_zct_get(uint32_t *pul_zct, uint16_t us_pch);</code>	Only in PLC PHY

.....continued

PHY primitive	Function	PHY layer
PHY_ZCT.confirm	Referenced parameters of <i>pal_zct_get</i> .	Only in PLC PHY

Note:

1. Manual management of AGC is not supported in the current implementation.

12. API of PRIME FW Stack

The API of the PRIME FW stack defines the functions as macros in the header file *prime_api.h*.

Note that there might be other functions not described in this document, but available in the *prime_api.h* file. Those functions are irrelevant in this document.

Likewise, there might be other parameters belonging to a different specification. Such parameters must be set to 0 when invoking the function.

12.1 MAC Primitives

Refer to the PRIME specification for more information about MAC primitives and their functionalities.

12.1.1 Signalling Primitives

Table 12-1. Signalling Primitives

Signalling primitive	Function
MAC_ESTABLISH.request	prime_cl_null_establish_request(mac_establish_request_t) 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);
MAC_ESTABLISH.indication	typedef void (*mac_establish_ind_cb_t)(uint16_t us_con_handle, uint8_t *puc_eui48, uint8_t uc_type, uint8_t *puc_data, uint16_t us_data_len, uint8_t uc_cfbytes);
MAC_ESTABLISH.response	prime_cl_null_establish_response(mac_establish_response_t) typedef void (*mac_establish_response_t)(uint16_t us_con_handle, mac_establish_response_answer_t uc_answer, uint8_t *puc_data, uint16_t us_data_len);
MAC_ESTABLISH.confirm	typedef void (*mac_establish_cfm_cb_t)(uint16_t us_con_handle, mac_establish_confirm_result_t uc_result, uint8_t *puc_eui48, uint8_t uc_type, uint8_t *puc_data, uint16_t us_data_len);
MAC_RELEASE.request	prime_cl_null_release_request(mac_release_request_t) typedef void (*mac_release_request_t)(uint16_t us_con_handle);
MAC_RELEASE.indication	typedef void (*mac_release_ind_cb_t)(uint16_t us_con_handle, mac_release_indication_reason_t uc_reason);
MAC_RELEASE.response	prime_cl_null_release_response(mac_release_response_t) typedef void (*mac_release_response_t)(uint16_t us_con_handle, mac_release_response_answer_t uc_answer);
MAC_RELEASE.confirm	typedef void (*mac_release_cfm_cb_t)(uint16_t us_con_handle, mac_release_confirm_result_t uc_result);
MAC_JOIN.request	prime_cl_null_join_request(mac_join_request_t) typedef void (*mac_join_request_t)(mac_join_mode_t us_broadcast, uint16_t us_con_handle, uint8_t *puc_eui48, connection_type_t uc_con_type, uint8_t *puc_data, uint16_t us_data_len);
MAC_JOIN.indication	typedef void (*mac_join_ind_cb_t)(uint16_t us_con_handle, uint8_t *puc_eui48, uint8_t uc_con_type, uint8_t *puc_data, uint16_t us_data_len);
MAC_JOIN.response	prime_cl_null_join_response(mac_join_response_t) typedef void (*mac_join_response_t)(uint16_t us_con_handle, uint8_t *puc_eui48, mac_join_response_answer_t uc_answer);
MAC_JOIN.confirm	typedef void (*mac_join_cfm_cb_t)(uint16_t us_con_handle, mac_join_confirm_result_t uc_result);
MAC_LEAVE.request	prime_cl_null_leave_request(mac_leave_request_t) typedef void (*mac_leave_request_t)(uint16_t us_con_handle, uint8_t *puc_eui48);

.....continued

Signalling primitive	Function
MAC_LEAVE.indication	typedef void (*mac_leave_ind_cb_t)(uint16_t us_con_handle, uint8_t *puc_eui48);
MAC_LEAVE.confirm	typedef void (*mac_leave_cfm_cb_t)(uint16_t us_con_handle, mac_leave_confirm_result_t uc_result);

12.1.2 Data Primitives

Table 12-2. Data Primitives

Data primitive	Function
MAC_DATA.request	prime_cl_null_data_request(mac_data_request_t) typedef void (*mac_data_request_t)(uint16_t us_con_handle, uint8_t *puc_data, uint16_t us_data_len, uint8_t uc_prio);
MAC_DATA.confirm	typedef void (*mac_data_cfm_cb_t)(uint16_t us_con_handle, uint8_t *puc_data, mac_data_result_t drt_result);
MAC_DATA.indication	typedef void (*mac_data_ind_cb_t)(uint16_t us_con_handle, uint8_t *puc_data, uint16_t us_data_len);

12.1.3 PLME Primitives

Table 12-3. PLME Primitives

PLME primitive	Function
PLME_RESET.request	prime_cl_null_plme_reset_request(plme_reset_request_t) typedef void (*plme_reset_request_t)(void);
PLME_RESET.confirm	typedef void (*plme_reset_cfm_cb_t)(plme_result_t x_result);
PLME_SLEEP.request ¹	prime_cl_null_plme_sleep_request(plme_sleep_request_t) typedef void (*plme_sleep_request_t)(void);
PLME_SLEEP.confirm	typedef void (*plme_sleep_cfm_cb_t)(plme_result_t x_result);
PLME_RESUME.request ¹	prime_cl_null_plme_resume_request(plme_resume_request_t) typedef void (*plme_resume_request_t)(void);
PLME_RESUME.confirm	typedef void (*plme_resume_cfm_cb_t)(plme_result_t x_result);
PLME_TESTMODE.request ¹	prime_cl_null_plme_testmode_request(plme_testmode_request_t) typedef void (*plme_testmode_request_t)(uint8_t uc_enable, uint8_t uc_mode, uint8_t uc_modulation, uint8_t uc_pwr_level);
PLME_TESTMODE.confirm	typedef void (*plme_testmode_cfm_cb_t)(plme_result_t x_result);
PLME_GET.request	prime_cl_null_plme_get_request(plme_get_request_t) typedef void (*plme_get_request_t)(uint16_t us_pib_attr);
PLME_GET.confirm	typedef void (*plme_get_cfm_cb_t)(plme_result_t x_status, uint16_t us_pib_attr, void *pv_pib_value, uint8_t uc_pib_size);
PLME_SET.request	prime_cl_null_plme_set_request(plme_set_request_t) typedef void (*plme_set_request_t)(uint16_t us_pib_attr, void *pv_pib_value, uint8_t uc_pib_size);
PLME_SET.confirm	typedef void (*plme_set_cfm_cb_t)(plme_result_t x_result);

Note:

1. The marked primitives invoke optional functionality that is currently not available.

12.1.4 MLME Primitives

Table 12-4. MLME Primitives in the Base Node

MLME primitive	Function
MLME_PROMOTE.request	prime_cl_null_mlme_promote_request(mlme_promote_request_t) typedef void (*mlme_promote_request_t) (uint8_t *puc_eui48);
MLME_PROMOTE.confirm	typedef void (*mlme_promote_cfm_cb_t) (mlme_result_t x_result);
MLME_RESET.request	prime_cl_null_mlme_reset_request(mlme_reset_request_t) typedef void (*mlme_reset_request_t) (void);
MLME_RESET.confirm	typedef void (*mlme_reset_cfm_cb_t) (mlme_result_t x_result);
MLME_GET.request	prime_cl_null_mlme_get_request(mlme_get_request_t) typedef void (*mlme_get_request_t) (uint16_t us_pib_attr);
MLME_GET.confirm	typedef void (*mlme_get_cfm_cb_t) (mlme_result_t x_status, uint16_t us_pib_attr, void *pv_pib_value, uint8_t uc_pib_size);
MLME_LIST_GET.request	prime_cl_null_mlme_list_get_request(mlme_list_get_request_t) typedef void (*mlme_list_get_request_t) (uint16_t us_pib_attr);
MLME_LIST_GET.confirm	typedef void (*mlme_list_get_cfm_cb_t) (mlme_result_t x_status, uint16_t us_pib_attr, uint8_t *puc_pib_buff, uint16_t us_pib_len);
MLME_SET.request	prime_cl_null_mlme_set_request(mlme_set_request_t) typedef void (*mlme_set_request_t) (uint16_t us_pib_attr, uint32_t ul_pib_value);
MLME_SET.confirm	typedef void (*mlme_set_cfm_cb_t) (mlme_result_t x_result);

12.1.5 Retrieval of Lists

The MLME_LIST_GET.confirm primitive returns a buffer `puc_pib_buff` where the requested list `us_pib_attr` is contained. Valid records are found one after the other, with their fields ordered as described in the PRIME specification. From the provided buffer length `us_pib_len` and the record size, users can calculate the number of records contained in the returned buffer and extract them.

Since the lists in the Base Node can contain a lot of records and the returned buffer size is limited to 1024 bytes, the confirm primitive might be invoked several times until all records have been returned. Records are not split between two confirm primitives. The last confirm invocation is marked with a buffer length of zero to indicate that there are no more valid records.

12.1.6 Callback Functions

The result of confirm and indication primitives is returned by the MAC layer invoking the corresponding callback function. To set up the callback function pointers, the following function is required:

```
typedef void (*mac_set_callbacks_t) (mac_callbacks_t *px_prime_cbs);
```

Parameters:

- `px_prime_cbs`: Pointer to the callback structure:

```
typedef struct {
    mac_establish_ind_cb_t mac_establish_ind_cb;
    mac_establish_cfm_cb_t mac_establish_cfm_cb;
    mac_release_ind_cb_t mac_release_ind_cb;
    mac_release_cfm_cb_t mac_release_cfm_cb;
    mac_join_ind_cb_t mac_join_ind_cb;
    mac_join_cfm_cb_t mac_join_cfm_cb;
    mac_leave_ind_cb_t mac_leave_ind_cb;
    mac_leave_cfm_cb_t mac_leave_cfm_cb;
    mac_data_ind_cb_t mac_data_ind_cb;
    mac_data_cfm_cb_t mac_data_cfm_cb;
    plme_reset_cfm_cb_t plme_reset_cfm_cb;
}
```

```

plme_sleep_cfm_cb_t plme_sleep_cfm_cb;
plme_resume_cfm_cb_t plme_resume_cfm_cb;
plme_testmode_cfm_cb_t plme_testmode_cfm_cb;
plme_get_cfm_cb_t plme_get_cfm_cb;
plme_set_cfm_cb_t plme_set_cfm_cb;
mlme_register_ind_cb_t mlme_register_ind_cb1;
mlme_register_cfm_cb_t mlme_register_cfm_cb1;
mlme_unregister_ind_cb_t mlme_unregister_ind_cb1;
mlme_unregister_cfm_cb_t mlme_unregister_cfm_cb1;
mlme_promote_ind_cb_t mlme_promote_ind_cb1;
mlme_promote_cfm_cb_t mlme_promote_cfm_cb;
mlme_demote_ind_cb_t mlme_demote_ind_cb1;
mlme_demote_cfm_cb_t mlme_demote_cfm_cb1;
mlme_reset_cfm_cb_t mlme_reset_cfm_cb;
mlme_get_cfm_cb_t mlme_get_cfm_cb;
mlme_list_get_cfm_cb_t mlme_list_get_cfm_cb;
mlme_set_cfm_cb_t mlme_set_cfm_cb;
} mac_callbacks_t;

```

Note:

- a. This function is not applicable in a Base Node.



Important: Unused callback functions must be set to NULL.

12.2 IEC 61334-4-32 Primitives

Refer to the PRIME specification for more information about IEC 61334-4-32 primitives and their functionalities.

Table 12-5. IEC 61334-4-32 Primitives in the Base Node

IEC 61334-4-32 primitive	Function
CL_432_RELEASE.request	prime_cl_432_release_request(cl_432_release_request_t) typedef void (*cl_432_release_request_t)(uint16_t us_dst_address);
CL_432_JOIN.indication	typedef void (*cl_432_join_ind_cb_t)(uint8_t *puc_device_id, uint8_t uc_device_id_len, uint16_t us_dst_address, uint8_t *puc_mac);
CL_432_LEAVE.indication	typedef void (*cl_432_leave_ind_cb_t)(uint16_t us_dst_address);
DL_DATA.request	prime_cl_432_dl_data_request(cl_432_dl_data_request_t) typedef void (*cl_432_dl_data_request_t)(uint8_t uc_dst_lsap, uint8_t uc_src_lsap, uint16_t us_dst_address, dl_432_buffer_t *px_buff, uint16_t uc_lsdu_len, uint8_t uc_link_class);
DL_DATA.confirm	typedef void (*cl_432_dl_data_cfm_cb_t)(uint8_t uc_dst_lsap, uint8_t uc_src_lsap, uint16_t us_dst_address, dl_432_tx_status_t uc_tx_status);
DL_DATA.indication	typedef void (*cl_432_dl_data_ind_cb_t)(uint8_t uc_dst_lsap, uint8_t uc_src_lsap, uint16_t us_dst_address, uint16_t us_src_address, uint8_t *puc_data, uint16_t uc_lsdu_len, uint8_t uc_link_class);

The result of confirm and indication primitives is returned by the SSCS invoking the corresponding callback function. To set up the callback function pointers, the following function is required:

```
typedef void (*cl_432_set_callbacks_t)(cl_432_callbacks_t *px_cl_432_cbs);
```

Parameters:

- **px_cl_432_cbs:** Pointer to the callback structure:

```

typedef struct {
    cl_432_dl_data_ind_cb_t cl_432_dl_data_ind_cb;
    cl_432_dl_data_cfm_cb_t cl_432_dl_data_cfm_cb;
    cl_432_join_ind_cb_t cl_432_join_ind_cb;
}

```

```

    cl_432_leave_ind_cb_t cl_432_leave_ind_cb;
} cl_432_callbacks_t;

```



Important: Unused callback functions must be set to NULL.

12.3 Base Management Primitives

12.3.1 Firmware Upgrade Protocol Primitives

Table 12-6. FUP Primitives in the Base Node

FUP primitive	Function
BMNG_FUP_CLEAR_TARGET_LIST.request	<code>void bmng_fup_clear_target_list_request(uint8_t uc_cmd);</code>
BMNG_FUP_ADD_TARGET.request	<code>void bmng_fup_add_target_request(uint8_t uc_cmd, uint8_t *puc_eui48);</code>
BMNG_FUP_SET_FW_DATA.request	<code>void bmng_fup_set_fw_data_request(uint8_t uc_cmd, uint8_t uc_vendor_len, char *pch_vendor, uint8_t uc_model_len, char *pch_model, uint8_t uc_version_len, char *pch_version);</code>
BMNG_FUP_SET_UPG_OPTIONS.request	<code>void bmng_fup_set_upg_options_request(uint8_t uc_cmd, uint8_t uc_arq_en, fup_page_size_t x_page_size, uint8_t uc_multicast_en, uint32_t ul_delay_restart, uint32_t ul_safety_timer);</code>
BMNG_FUP_INIT_FILE_TX.request	<code>void bmng_fup_init_file_tx_request(uint8_t uc_cmd, uint16_t us_frame_number, uint32_t ul_file_size, uint16_t us_frame_size, uint32_t ul_crc);</code>
BMNG_FUP_DATA_FRAME.request	<code>void bmng_fup_data_frame_request(uint8_t uc_cmd, uint16_t us_frame_number, uint16_t us_data_len, uint8_t *puc_data);</code>
BMNG_FUP_CHECK_CRC.request	<code>void bmng_fup_check_crc_request(uint8_t uc_cmd);</code>
BMNG_FUP_ABORT_FU.request	<code>void bmng_fup_abort_fu_request(uint8_t uc_cmd, uint8_t *puc_eui48);</code>
BMNG_FUP_START_FU.request	<code>void bmng_fup_start_fu_request(uint8_t uc_cmd, uint8_t uc_enable);</code>
BMNG_FUP_SET_MATCH_RULE.request	<code>void bmng_fup_set_match_rule_request(uint8_t uc_cmd, uint8_t uc_rules);</code>
BMNG_FUP_GET_VERSION.request	<code>void bmng_fup_get_version_request(uint8_t uc_cmd, uint8_t *puc_eui48);</code>
BMNG_FUP_GET_STATE.request	<code>void bmng_fup_get_state_request(uint8_t uc_cmd, uint8_t *puc_eui48);</code>
BMNG_FUP_SET_SIGNATURE_DATA.request	Not applicable in 1.3
BMNG_FUP_ACK.indication	<code>typedef void (*bmng_fup_ack_cb_t)(uint8_t uc_cmd, fup_ack_code_t x_ack_code, uint16_t us_extra_info);</code>
BMNG_FUP_STATUS.indication	<code>typedef void (*bmng_fup_status_ind_cb_t)(fup_node_state_t x_state, uint16_t ul_pages, uint8_t *puc_eui48);</code>
BMNG_FUP_ERROR.indication	<code>typedef void (*bmng_fup_error_ind_cb_t)(fup_error_code_t x_error_code, uint8_t *puc_eui48);</code>
BMNG_FUP_VERSION.indication	<code>typedef void (*bmng_fup_version_ind_cb_t)(uint8_t *puc_eui48, uint8_t uc_vendor_len, char *pch_vendor, uint8_t uc_model_len, char *pch_model, uint8_t uc_version_len, char *pch_version);</code>

.....continued

FUP primitive	Function
BMNG_FUP_KILL.indication	typedef void (*bmng_fup_kill_ind_cb_t) (uint8_t *puc_eui48);

12.3.2 Network Event Primitives

Table 12-7. Network Event Primitives

Network Event primitive	Function
BMNG_NETWORK_EVENT.indication	typedef void (*bmng_network_event_ind_cb_t) (bmng_net_event_t *px_net_event);

12.3.3 PRIME Profile Primitives

Table 12-8. PRIME Profile Primitives

PRIME Profile primitive	Function
BMNG_PPROF_GET.request	void bmng_pprof_get_request(uint8_t uc_cmd, uint8_t *puc_eui48, uint16_t us_data_len, uint8_t *puc_data);
BMNG_PPROF_SET.request	void bmng_pprof_set_request(uint8_t uc_cmd, uint8_t *puc_eui48, uint16_t us_data_len, uint8_t *puc_data);
BMNG_PPROF_RESET.request	void bmng_pprof_reset_request(uint8_t uc_cmd, uint8_t *puc_eui48);
BMNG_PPROF_REBOOT.request	void bmng_pprof_reboot_request(uint8_t uc_cmd, uint8_t *puc_eui48);
BMNG_PPROF_GET_ENHANCED.request	Not applicable in 1.3
BMNG_PPROF_ZC_DIFF.request ¹	void bmng_pprof_zc_diff_request(uint8_t uc_cmd, uint8_t *puc_eui48);
BMNG_PPROF_ACK.indication	typedef void (*bmng_pprof_ack_cb_t) (uint8_t uc_cmd, pprof_ack_code_t x_ack_code);
BMNG_PPROF_GET_RESPONSE.indication	typedef void (*bmng_pprof_get_response_cb_t) (uint8_t *puc_eui48, uint16_t us_data_len, uint8_t *puc_data);
BMNG_PPROF_GET_ENHANCED_RESPONSE.indication	Not applicable in 1.3
BMNG_PPROF_GET_ZC_RESPONSE.indication ²	typedef void (*bmng_pprof_get_zc_response_cb_t) (uint8_t *puc_eui48, uint8_t uc_zc_status, uint32_t ul_zc_time ²);
BMNG_PPROF_ZC_DIFF_RESPONSE.indication	typedef void (*bmng_pprof_zc_diff_response_cb_t) (uint8_t *puc_eui48, uint32_t ul_time_freq, uint32_t ul_time_diff ³);

Notes:

1. The response to ZC_DIFF is ZC_DIFF_RESPONSE followed by GET_ZC_RESPONSE, if its callback has been set up.
2. The standard PIB 0x0078 (PIB_MAC_APP_LIST_ZC_STATUS) is special as it returns only one element of the list together with the zero crossing time. Therefore, even if the PIB is requested using a GET request or a GET_ENHANCED request, the invoked callback function with the response is GET_ZC_RESPONSE instead of GET_RESPONSE or GET_ENHANCED_RESPONSE.
3. In tenths of microsecond.

12.3.4 Whitelist Management Primitives

Table 12-9. Whitelist Management Primitives

Whitelist Management primitive	Function
BMNG_WHITELIST_ADD.request	<code>void bmng_whitelist_add_request(uint8_t uc_cmd, uint8_t *puc_eui48);</code>
BMNG_WHITELIST_REMOVE.request	<code>void bmng_whitelist_remove_request(uint8_t uc_cmd, uint8_t *puc_eui48);</code>
BMNG_WHITELIST_ACK.indication	<code>typedef void (*bmng_whitelist_ack_cb_t)(uint8_t uc_cmd, whitelist_ack_code_t x_ack_code);</code>

12.3.5 Callback Functions

The result of indication primitives is returned by the Base Management layer invoking the corresponding callback function. To set up the callback function pointers, the following function is required:

```
void prime_bmng_set_callbacks(bmng_callbacks_t *px_base_mng_cbs);
```

Parameters:

- *px_base_mng_cbs*: Pointer to the callback structure:

```
typedef struct {
    bmng_fup_ack_cb_t fup_ack_cb;
    bmng_fup_status_ind_cb_t fup_status_ind_cb;
    bmng_fup_error_ind_cb_t fup_error_ind_cb;
    bmng_fup_version_ind_cb_t fup_version_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_cb_t pprof_ack_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_get_zc_response_cb_t pprof_get_zc_response_cb;
    bmng_pprof_zc_diff_response_cb_t pprof_zc_diff_response_cb;
    bmng_whitelist_ack_cb_t whitelist_ack_cb;
} bmng_callbacks_t;
```



Important: Unused callback functions must be set to NULL.

12.4 PIB Objects Specification and Access

The PRIME FW stack supports all the mandatory attributes of the PLC Information Base (PIB) defined in the PRIME specification. In addition, Microchip has added several proprietary PIB attributes to support extra functionalities. These attributes are described in the next sections. The list of all available PIB attributes can be found in file *mac_pib.h*.

12.4.1 Proprietary PIB Attributes in the PHY Layer

Table 12-10. Proprietary PIB Attributes in the PHY Layer

Attribute	ID	Size	Range	Description
PIB_PHY_SW_VERSION	0x8080	4 bytes	-	PLC PHY layer software version.
PIB_PHY_ZCT	0x8081	4 bytes	-	Time in microseconds between the zero cross of the mains and the end of the last transmission or reception. Only applicable in PLC.
PIB_PHY_HOST_VERSION	0x8082	4 bytes	-	PL360 Host Controller version.

.....continued

Attribute	ID	Size	Range	Description
PIB_MTP_PHY_TX_TIME	0x8085	4 bytes	-	Transmission time of the last frame transmitted in PLC in tenths of μ s.
PIB_MTP_PHY_RMS_CALC_CORRECTED	0x8086	4 bytes	-	RMS value of the last PLC signal.
PIB_MTP_PHY_EXECUTE_CALIBRATION	0x8087	2 bytes	-	Executes calibration process [0: stop process, 1: start process]. It returns threshold level to configure board for PLC.
PIB_MTP_PHY_RX_PARAMS	0x8088	22 bytes	-	RX PLC PHY structure ¹ : <ul style="list-style-type: none"> RxParam[0]. Modulation scheme: <ul style="list-style-type: none"> DBPSK = 0 DQPSK = 1 D8PSK = 2 DBPSK_CC = 4 DQPSK_CC = 5 D8PSK_CC = 6 DBPSK_RB = 12 DQPSK_RB = 13 RxParam[1]. RSSI. RxParam[2-3]. Reception data length in bytes. RxParam[4-5]. EvmHeader. RxParam[6-7]. EvmPayload. RxParam[8-11]. EvmHeaderAcum. RxParam[12-15]. EvmPayloadAcum. RxParam[16-19]. Reception time in microseconds. RxParam[20]. Noise. RxParam[21]. SNR in dB (0 - 21).

.....continued

Attribute	ID	Size	Range	Description
PIB_MTP_PHY_TX_PARAMS	0x8089	7 bytes	-	<p>TX PLC PHY structure¹:</p> <ul style="list-style-type: none"> TxParam[0]. Modulation scheme: <ul style="list-style-type: none"> DBPSK = 0 DQPSK = 1 D8PSK = 2 DBPSK_CC = 4 DQPSK_CC = 5 D8PSK_CC = 6 DBPSK_RB = 12 DQPSK_RB = 13 TxParam[1]. Attenuation level (0 - 21 dB). When set to 255, attenuation and gain are set to 0. TxParam[2-3]. Transmission data length in bytes. TxParam[4]. Inter-frame time in tenths of ms (0 - 2550 ms). TxParam[5]. Random seed data generation. (0: send the same constant payload in each message, 1 - 255 seed used to generate different ranges of random values in the data payload). TxParam[6]. Number of messages (1 - 0xFF).
PIB_MTP_PHY_CONTINUOUS_TX	0x808A	1 byte	0 - 1	Set the PLC PHY layer to transmit continuously [0: disabled, 1: enabled].
PIB_MTP_PHY_ENABLE	0x808E	1 byte	0 - 1	Manufacturing test procedure for PLC [0: disabled, 1: enabled]. Only when the MTP is enabled, related PIBs can be set.
PIB_PHY_TX_CHANNEL	0x8090	1 byte	1 - 8	Transmission/Reception PLC channel, only when hardware permits multichannel. The channel depends on the selected PLC coupling. It can only be set when MTP is enabled.
PIB_PHY_SNIFFER_ENABLED	0x8106	1 byte	0 - 1	Enable/disable the sniffer [0: disabled, 1: enabled]. Bit 0 corresponds to the PLC sniffer. Bit 3 corresponds to the Serial sniffer.
PIB_PHY_DRV_AUTO	0x8301	1 byte	0 - 1	Enable/disable automatic selection of PLC transmission mode [0: disabled, 1: enabled].
PIB_PHY_DRV_IMPEDANCE	0x8302	1 byte	0 - 2	Enable/disable the PLC high impedance branch [0: high, 1: low, 2: very low].
PIB_PHY_DRV_ATTENUATION	0x8303	1 byte	0 - 255	Additional attenuation with which a PLC message must be transmitted.

Note:

1. See Application Note *Guidelines for PLC performance verification*.

12.4.2 Proprietary PIB Attributes in the MAC Layer

Table 12-11. Proprietary PIB Attributes in the MAC Layer

Attribute	ID	Size	Range	Description
PIB_MTP_MAC_EUI_48	0x8100	6 bytes	-	MAC address.
PIB_MAC_PLC_STATE	0x8101	1 byte	-	PRIME PLC state: <ul style="list-style-type: none"> • 0: SN disconnected • 1: SN detection • 2: SN registering • 3: SN operative • 4: BN
PIB_MAC_ACTION_CFP_LENGTH	0x810D	2 bytes	0 - 128	Set CFP length in symbols in MAC certification mode.
PIB_MAC_ACTION_BCN_SLOT_COUNT	0x810E	1 byte	1 - 4	Change the number of beacon slots per frame in MAC certification mode.
PIB_CERTIFICATION_MODE	0x8120	1 byte	0 - 3	Certification mode: <ul style="list-style-type: none"> • 0: no certification mode • 1: PHY certification for 1.3 • 2: MAC certification • 3: PHY certification for 1.4
PIB_CERTIFICATION_SEND_MSG	0x8121	Array	-	Send message in PHY certification mode. See Table 12-12 .
PIB_MTP_MAC_WRITE_SNA	0x8123	6 bytes	-	Write SNA in MTP mode.
PIB_MAC_ACTION_ARQ_WIN_SIZE	0x8124	1 byte	1 - 32	Change the ARQ window size in MAC certification mode.
PIB_MAC_INTERNAL_SW_VERSION	0x8126	4 bytes	-	MAC internal software version.
PIB_BCN_SLOTS_BUSY	0x8131	1 byte	0 - 128	Number of used beacon slots.
PIB_MAC_WHITELIST	0x8150	-	-	List of MAC addresses of nodes which are allowed to register.
PIB_MAC_WHITELIST_ENABLED	0x8151	1 byte	0 - 1	Whitelist status [0: disabled, 1: enabled].

Table 12-12. Data in PIB_CERTIFICATION_SEND_MSG

Element	Size	Range	Description
MsgCount	2 bytes	1 - 2000	Number of messages to transmit
Modulation	1 byte	0 - 7	Modulation scheme (as in PIB_MTP_PHY_TX_PARAMS)
SignalAtt	1 byte	0 - 21	Signal attenuation (in dB)
DutyCycle	1 byte	1 - 100	Duty cycle
PrimeFrame	1 byte	0 - 3	PRIME frame: <ul style="list-style-type: none"> • 1.3 frame • 1.4 type A frame • 1.4 type B frame • 1.4 type BC frame



Important: Only when the certification mode is set to MAC certification, the following standard PIB attributes are enabled and operative:

- 0x0067 (PIB_MAC_ACTION_REJECT)
- 0x0068 (PIB_MAC_ACTION_ALIVE_TIME)
- 0x006A (PIB_MAC_ACTION_BROADCAST_DATA_BURST)
- 0x006B (PIB_MAC_ACTION_MGMT_CON)
- 0x006C (PIB_MAC_ACTION_MGMT_MUL)
- 0x006F (PIB_MAC_ACTION_SEGMENTED_432)
- 0x0080 (PIB_MAC_ACTION_APPEMU_DATA_BURST)

12.4.3 Proprietary PIB Attributes in the IEC 61334-4-32 SCS Layer

Table 12-13. Proprietary PIB Attributes in the IEC 61334-4-32 SCS Layer

Attribute	ID	Size	Range	Description
PIB_CL_INTERNAL_SW_VERSION	0x8201	4 bytes	-	Internal software version of the SCS layer.
PIB_432_LIST_NODES	0x8250	-	-	List of nodes registered in IEC 61334-4-32 layer. See Data in PIB_432_LIST_NODES .

Table 12-14. Data in PIB_432_LIST_NODES

Element	Size	Range	Description
Address	2 bytes	0 - 0xFFFF	IEC 61334-4-32 address
SerialNumber	16 bytes	-	Serial number assigned in IEC 61334-4-32
LenSerial	1 byte	0 - 16	Length of serial number
MAC	6 bytes	-	MAC address of the node

12.4.4 Proprietary PIB Attributes in the Management Plane

Table 12-15. Proprietary PIB Attributes in the Management Plane

Attribute	ID	Size	Range	Description
PIB_FU_LIST	0x8350	-	-	List of nodes in the FU target list. See Data in PIB_FU_LIST .

Table 12-16. Data in PIB_FU_LIST

Element	Size	Range	Description
FUNodeState	1 byte	0 - 0x7F	FU node state: <ul style="list-style-type: none"> • 0: idle • 1: receiving • 2: complete • 3: countdown • 4: upgrade • 5: exception • 0x7F: unknown
PagesCompleted	4 bytes	-	Array that indicates which pages are completed

.....continued

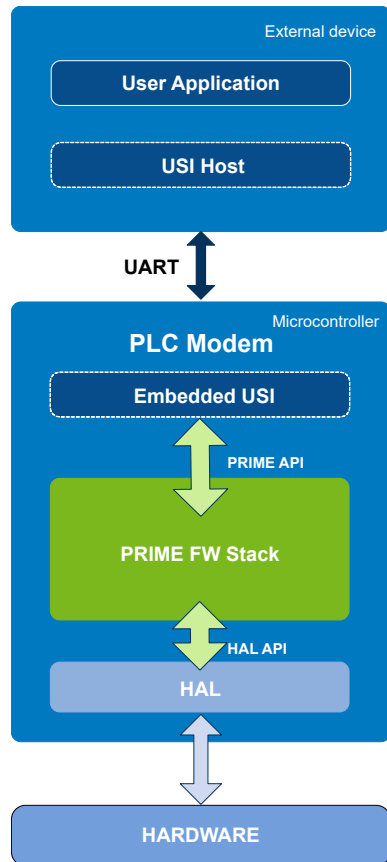
Element	Size	Range	Description
MAC	6 bytes	-	MAC address of the node

13. Example Applications

13.1 PRIME Base Modem

The Base Modem is an application example that shows how to serialize the PRIME API when the user application and the PRIME FW stack are running in different devices. This example serializes the PRIME API through the USI using the UART. The overall architecture of this functionality is shown in the next figure.

Figure 13-1. Layer Diagram for a Serialized PRIME FW Stack

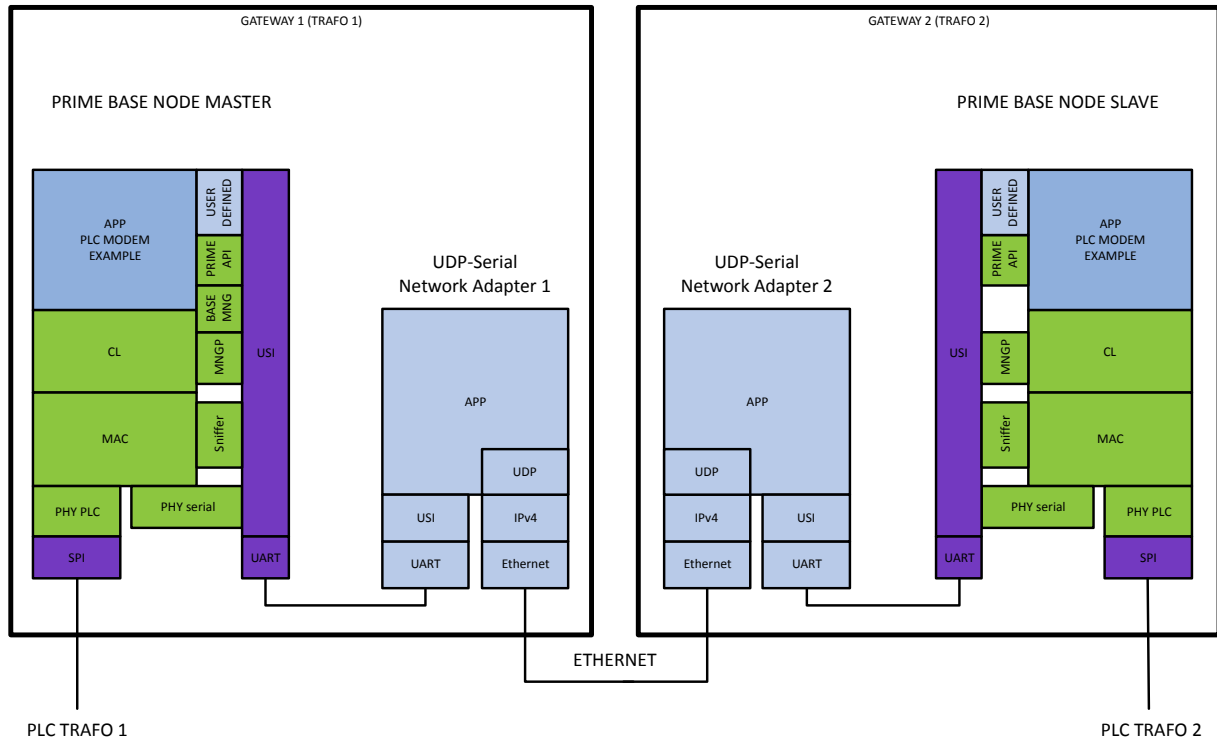


In this architecture, files *modem.h* and *modem_base.c* in the embedded user application are in charge of coding and decoding the PRIME API primitives, whereas the Embedded USI (included in the HAL) is responsible for the transmission and reception of serial messages. Similarly, the USI Host in the external application is responsible for coding and decoding the PRIME API primitives and also for transmission and reception of serial messages.

The configuration required to support this serialization and the format of the serial messages in the USI is described in chapter [Serialization with Embedded USI](#). The integration of the USI Host in the external application is explained in the Application Note *PLC Universal Serial Interface*.

13.2 Base Slave Modem

The Base Slave Modem is an application example that, together with the PRIME Base Modem example running in a Base Node, allows the connection of auxiliary nodes in a multi-transformer substation. The overall architecture of this functionality is shown in the figure below.

Figure 13-2. Layer diagram for a multi-transformer system

The Base Slave Node is a customized Service Node with its own library running over the PLC Modem application. The communication between the Base Node and the Base Slave Node takes place using the serial PHY layer, which is serialized through the USI and then transmitted over UDP by external devices. The Base Slave Node is promoted to switch by the Base Node as soon as it registers so that it can forward PRIME messages using the PLC PHY layer. In this situation, the MAC sniffer provides information about all the traffic in the network being it from the PLC PHY layer or the serial PHY layer.



Important: Microchip does not provide the Network Adapters or the UDP encapsulation.

The configuration required to support this functionality and the format of the serial messages in the USI is described in chapter [Serialization with Embedded USI](#).

For more information about this functionality, see chapter [Auxiliary Nodes Connection in Multi-transformer Substations](#).

13.3 PHY Applications

Microchip also provides several applications that demonstrate the complete performance of the PRIME PHY layer. Except for the PHY sniffer, such applications only offer the ability to send and receive PLC data through the PLC PHY layer without any MAC layer as shown in [Figure 13-3](#). The API of the PHY is described in chapter [API of PHY and PAL Layers](#).

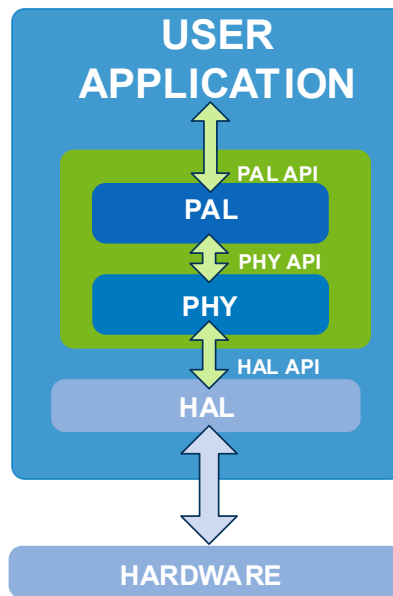
The available PLC PHY applications are the following:

- **PHY Tester:** This PHY application example shows the capabilities of the PL360 device in a point-to-point connection, using the USI to serialize the API of the PLC PHY layer.

- PHY TX Test Console: This PHY application example uses a terminal console to configure the PLC PHY layer and perform several board tests. Parameters that are configured include transmission parameters such as modulation, frame data length and time interval between frames.
- PHY PLC&Go: This PHY application example shows the basic code required to work with the PLC PHY layer.

There is also the PHY Sniffer application, which allows monitoring of data traffic on a PRIME network by serializing PHY frames (see the User Manual of the *Hybrid Sniffer*). The PHY Sniffer can detect traffic coming from any PHY layer.

Figure 13-3. Layer Diagram of a PHY-only Application



14. Serialization with Embedded USI

The Embedded USI is a wrapper part of the HAL that provides the interface between the PRIME FW stack and the serial communications channel.

For serial transmissions from the PRIME stack, the Embedded USI provides a function that packs and sends each message via the serial link to the external application. For serial receptions from the serial link, the Embedded USI provides a function that unpacks the received message and passes it to the PRIME FW stack.

The equivalent wrapper in the external application is the provided USI Host, which is also in charge of coding and decoding the messages. If users want to develop their own USI Host application, they will have to take into account the following operation of the Embedded USI to make it compatible:

- USI frame format
- USI PRIME protocols
- Embedded USI Configuration

For more information about available services and the provided USI Host, see Application Note *PLC Universal Serial Interface* and the *USI Host User Guide*.

14.1 USI Frame Format

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

Figure 14-1. USI Frame Format

7E (1 byte)	MSG LENGTH (10 bits)	PROTOCOL ID (6 bits)	MESSAGE DATA	CRC (variable)	7E (1 byte)
-----------------------	-------------------------	-------------------------	-----------------	-------------------	-----------------------

The frame starts and ends with 0x7E. The following is the description of each field:

- MSG LENGTH: Command length in bytes (protocol command byte plus message data bytes).
- PROTOCOL ID: Protocol in the frame (see [Table 14-1](#)).
- MESSAGE DATA: Variable field with the data of the exchanged message.
- CRC: Error correction code for the message. The CRC field can have a different length depending on the protocol (see [Table 14-1](#)).

Table 14-1. USI Protocols and Associated CRC Size

Protocol	Protocol ID	CRC size (bits)
PROTOCOLS_MNGP_PRIME ¹	0x00 - 0x07	32
PROTOCOL_SNIF_PRIME	0x13	16
PROTOCOL_PHY_SERIAL	0x1F	16
PROTOCOL_PHY_TESTER ²	0x22	16
PROTOCOL_PRIME_API	0x30	8
PROTOCOL_INTERNAL	0x3F	16
PROTOCOL_USER_DEFINED ³	0xFE	Defined by the user. By default, 16.

Notes:

1. These protocols are described in the Serial Communications Profile of the Management Plane defined in the PRIME specification.
2. This protocol is only used by the PLC PHY Tester PC tool that Microchip provides with the evaluation kit in order to serialize the API of the PLC PHY layer.
3. Defined by the user for their own proprietary protocol, if necessary.

14.2 USI PRIME Protocols

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

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

14.2.1 PRIME Management Plane Frame Format

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 shown in the following figure.

Figure 14-2. MNGP PRIME USI Frame Format

7E (1 byte)	MSG LENGTH (10 bits)	PROTOCOL ID (6 bits)	MESSAGE DATA	CRC (4 bytes)	7E (1 byte)
-----------------------	--------------------------------	--------------------------------	---------------------	-------------------------	-----------------------

The available management functions are described in the PRIME specification and shown in the following table.

Table 14-2. USI MNGP PRIME Protocols

Protocols MNGP PRIME	Protocol 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.

14.2.2 PRIME Sniffer Frame Format

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

use. The embedded sniffer is only available when it has been enabled and properly configured (see chapter [PRIME Stack User Configuration Parameters](#)).

[Figure 14-3](#) and [Figure 14-4](#) show the USI frame format of the frames generated in the node. In this case, the field MESSAGE DATA that appears in the USI frame format (see [Figure 14-1](#)) is divided into two different fields: header and PDU sniffer message.

Figure 14-3. PRIME Sniffer USI Frame Format

7E (1 byte)	MSG LENGTH (10 bits)	PROTOCOL ID 0x13 (6 bits)	MESSAGE DATA		CRC (2 bytes)	7E (1 byte)
			HEADER (32 bytes)	PDU SNIFFER MSG		

Figure 14-4. PRIME Sniffer USI Header Field

FRA T 1 byte	SNIF F 1 byte	SNIF T 1 byte	MODUL 1 byte	SYM PDU 1 byte	SNR 1 byte	EX SNR 1 byte	CHN 1 byte	CINR 1 byte	BERSOFT 1 byte	BERS MAX 1 byte	0x00...0x00 8 bytes
Time Start 4 bytes		Time End 4 bytes		RSSI 2 bytes	0x00 1 byte	PDU LEN 2 bytes					

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

For details about the PHY information, see the datasheets for *PL360* and *PL460*.

14.2.3 PRIME API Frame Format

This protocol consists of the serialization of the PRIME API primitives. The PRIME FW stack can provide the PRIME API through a serial interface as an independent protocol of USI. It is only available when the user application contains the modem example.

The frame format is shown in the following figure.

Figure 14-5. 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		

There are three fields in this frame inside the general MESSAGE DATA field:

- **LENGTH Extended:** Since 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 total message length size. In this field the most significant bit of the message length is codified
- **PRIME API COMMAND:** This field refers directly to the primitive included in the message, using the same primitives described in the PRIME API interface description (see chapter [API of PRIME FW Stack](#)). The values for those primitives are defined in enumerator `prime_api_cmd_t` in file `modem.h`
- **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 more easily. This is applicable to all primitives in the PRIME API. [Figure 14-6](#) 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 14-6. 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: Note that the Base Management primitives already include the PRIME API command as the first function parameter, so exceptionally sending it twice is not needed.

14.2.4 PRIME PHY Serial Frame Format

This protocol consists of the serialization of the PHY layer. The serial PHY layer is used in applications where a Base Node communicates with an auxiliary node without using the PLC channel. The serialized PRIME PDU is passed to an external device which will transfer it to the USI in the other node. The serial PHY layer is only available when the PRIME FW stack is configured to use the serial PHY layer (see chapter [Enabling Serial PHY Layer](#)).

The frame format is shown in the following figure.

Figure 14-7. PRIME PHY layer USI frame format

7E (1 byte)	MSG LENGTH (10 bits)	PROTOCOL ID 0x1F (6 bits)	PRIME PDU	CRC (4 bytes)	7E (1 byte)
----------------	-------------------------	------------------------------	-----------	------------------	----------------

14.3 Embedded USI Configuration

The Embedded USI must be configured according to the user requirements. This configuration consists of indicating the protocols to be serialized and which port will be used by each protocol.

14.3.1 Definition and Configuration of USI Ports

Users can define the ports to be used and their configurations in the *conf_usi.h* file.

```

/* Port Communications configuration */
#define NUM_PORTS 2
#define PORT_0 CONF_PORT(UART_TYPE, 0, 115200, HAL_TX_UART_BUF0_SIZE, HAL_RX_UART_BUF0_SIZE)
#define PORT_1 CONF_PORT(USART_TYPE, 4, 57600, HAL_TX_USART_BUF1_SIZE, HAL_RX_USART_BUF1_SIZE)
#define PORT_2 CONF_PORT(USB_TYPE, 0, 115200, 1024, 1024)

```

NUM_PORTS defines the number of ports to be used. After that, every PORT_x must be configured following a sequential order (PORT_0, PORT_1, etc.). The input parameters of the port configuration are shown in the following table.

Table 14-3. USI Port Configuration Parameters

Parameter	Description	Valid Values
Type	Type of link ¹	UART_TYPE for UART USART_TYPE for USART USB_TYPE for USB
Channel	Instance	0: UART0/USART0/USB 1: UART1/USART1 2: UART2/USART2 4: USART4
Speed	Baudrate	9600, 19200, 38400, 57600, 115200, 230400, 256000, 921600
TX_size	Size of transmission buffer	Size of buffer must be (at least) twice the size of the bigger message payload linked to this port. ²
RX_size	Size of reception buffer	Size of buffer must be (at least) twice the size of the bigger message payload linked to this port. ²

Notes:

- Only UART, USART and USB are currently supported.
- In case of IEC61334-4-32 messages, please note that protocol headers are part of the USI message payload.



Important: Both Embedded USI and the external application must use the same baud rate and protocols. Messages from protocols not serialized in both sides of the serial communications channel are discarded.

14.3.2 Linking of Serial Communication Profile of PRIME Management Plane

To link the Serial Communication Profile of the PRIME Management Plane to a USI port defined in the Embedded USI, users must use file *conf_mngp.h*.

By default, the link is set to PORT_0.

```

/* Select PORT for Serial Profile */
#define MNGP_SPROF_USI_PORT 0

```

14.3.3 Linking of PRIME Sniffer

To link the embedded PRIME sniffer to a USI port defined in the Embedded USI, users must use file *conf_pal.h*.

By default, the link is set to `PORT_0`.

```
/* Select PORT to serialize PHY sniffer */  
#define PHY_SNIFFER_USI_PORT 0
```

14.3.4 Linking of PRIME API

To link the PRIME API to a USI port defined in the Embedded USI, users must use file *conf_app_example.h* (assuming they are using the modem example). By default, the link is set to `PORT_0`.

```
/** USI PORT */  
#define MODEM_USI_PORT 0
```

14.3.5 Linking of PRIME PHY Serial

To link the PRIME PHY Serial to a USI port defined in the Embedded USI, users must use file *conf_phy_serial.h*.

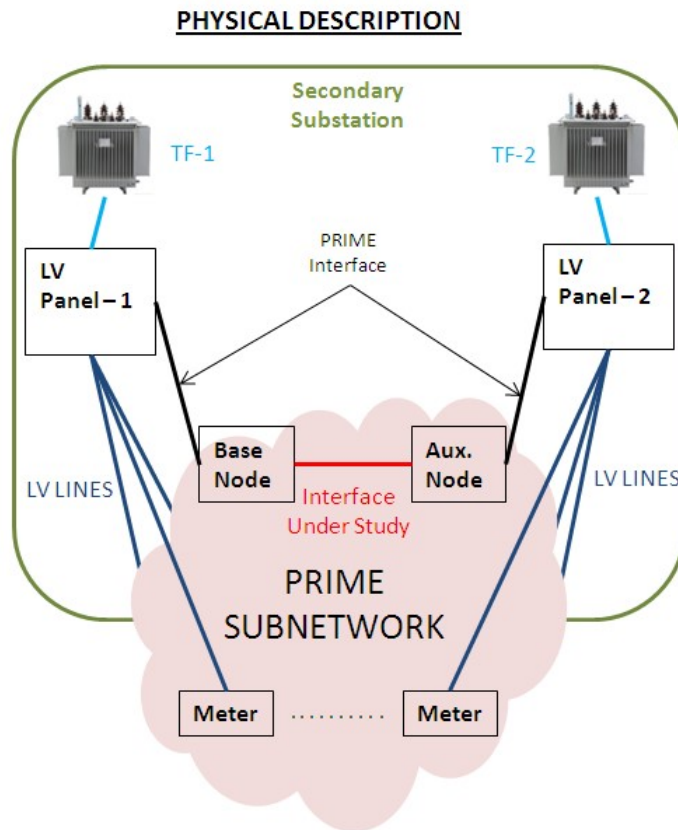
By default, the link is set to `PORT_0`.

```
/** Select PORT to serialize PHY */  
#define PHY_SERIAL_USI_PORT 0
```

15. Auxiliary Nodes Connection in Multi-transformer Substations

In a secondary substation with multiple transformers whose respective panels are independent from each other, the PRIME PLC signal does not have enough power transfer between the two panels. Therefore, the Base Node is not able to communicate appropriately with Service Nodes connected to a different panel. To solve this, an Auxiliary Node is required in each panel. The function of this Auxiliary Node is to repeat Base Node messages (as a plain PRIME switch) addressed to any meter in its panel.

Figure 15-1. Substation with Auxiliary Node



Considering the elements in the figure above, the interface between the Base Node and the Auxiliary Node becomes important, as PRIME communications need to be transferred over it with as fewer losses as possible. The PRIME group proposes to use an Ethernet link where PRIME PDUs are encapsulated in UDP packets.

More information about this problem can be found in the Ticket #65 of the MAC Task Force in the PRIME Alliance.

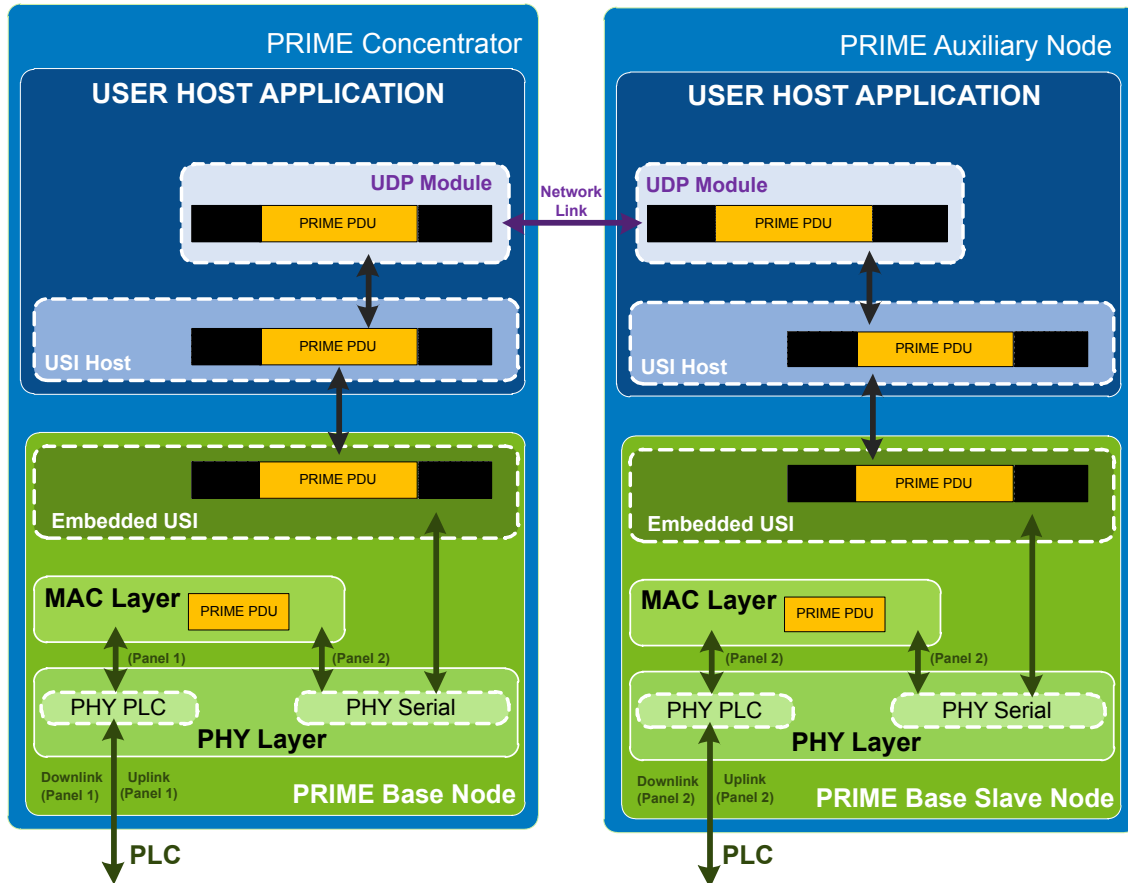
Microchip provides a solution consisting of a Base Node and a Base Slave Node which both have two PHY layers: the PLC PHY layer and a serial PHY layer (for configuration, see chapter [Enabling Serial PHY Layer](#)). The traffic from the serial PHY layer is serialized through the USI of the HAL (see chapter [Serialization with Embedded USI](#)). Users are responsible for converting the serial frames into UDP frames (and vice versa) and exchange them between the Base Node and the Base Slave Node.

The figure below shows the block architecture of this system and how the PRIME PDU is exchanged between them. In the Base Node, the PRIME PDU is generated in the MAC layer, which determines if it must be sent via PLC or serial link. If the PDU must be sent via serial link, it is passed to the

serial PHY layer, which then passes it to the Embedded USI. The USI Host in the external application extracts the PRIME PDU and sends it via UDP to the Auxiliary Node. The USI Host in the Auxiliary Node extracts the PDU and serializes it so that it is received in the Embedded USI, which passes it to the serial PHY layer and from there to the MAC layer. The MAC layer in the Base Slave Node forwards messages from the serial PHY layer to the PLC PHY layer. Messages received in the PLC PHY layer of the Base Slave Node are forwarded to the serial PHY layer, which passes them to the Embedded USI. They are received in the MAC layer of the Base Node following the inverse way.

This solution is only available in the SAME70Q21 device, in the mini and the full PRIME libraries of the Base Node. The Base Slave Node has its own library.

Figure 15-2. Block diagram of connected PRIME nodes



16. Abbreviations

AES	Advanced Encryption Standard
AGC	Automatic Gain Mode
API	Application Programming Interface
ARQ	Automatic Repeat Request
BER	Bit Error Rate
BN	Base Node
CC	Convolutional Code
CFP	Contention Free Period
CINR	Carrier to Interference Noise Ratio
CL	Convergence Layer
CPCS	Common Part Convergence Sublayer
CRC	Cyclic Redundancy Check
D8PSK	Differential Eight-Phase Shift Keying
DBPSK	Differential Binary Phase Shift Keying
DCU	Data Concentrator Unit
DLMS	Device Language Message Specification
DMA	Direct Memory Access
DQPSK	Differential Quaternary Phase Shift Keying
FU	Firmware Upgrade
FUP	Firmware Upgrade Protocol
FW	Firmware
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
IEC	International Electrotechnical Committee
IP	Internet Protocol
LNID	Local Node Identifier
MAC	Medium Access Control
MCU	Microcontroller Unit
MLME	MAC Layer Management Entity
MNGP	Management Plane
MPAL	MultiPhy Abstraction Layer
MPDU	MAC Protocol Data Unit
MTP	Manufacturing Test Procedure
PAL	Physical Abstraction Layer
PDU	Protocol Data Unit
PHY	Physical
PIB	PLC Information Base
PICS	Protocol Implementation Conformance Statement
PLC	Power Line Communications
PLME	PHY Layer Management Entity
PMC	Power Management Controller
PRIME	Powerline Intelligent Metering Evolution
RB	Robust
RMS	Root Mean Square
RSSI	Received Signal Strength Indicator
SAP	Service Access Point

SDRAM	Synchronous Dynamic Random-Access Memory
SN	Service Node
SNA	Sub-Network Address
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
SSCS	Service Specific Convergence Sublayer
TC	Timer/Counter
TFTP	Trivial File Transfer Protocol
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USI	Universal Serial Interface

17. References

[Microchip Smart Energy](#)

[Microchip Power Line Communications](#)

[Microchip Design Support](#)

[PRIME Alliance](#)

[Base Node Evaluation Kit](#)

[PL460 Evaluation Kit](#)

[PIC32CXMTSH Demo Board](#)

[PIC32CXMTG Evaluation Kit](#)

www.microchip.com/en-us/development-tool/ATSAME70-XPLD

[SAMG55 Xplained PRO Evaluation Kit](#)

[Documents for supported families and boards](#)

18. Microchip Revision History

18.1 Revision C - 03/2024

No content changes compared with the previous revision, cleared DS# related issues.

18.2 Revision B - 02/2024

The following changes were incorporated in this revision:

Document	Removed all references to ATPL230, ASF, Atmel Studio and Doxygen. Added and removed PLC where necessary to make it general for all available PHY layers. Added PL460 whenever there was a reference to PL360 platform. Other editorials.
Features	Updated platforms and modems.
PRIME FW Stack, Event-Driven Operation	Updated description.
PRIME FW Stack Library	Updated libraries according to the new devices and boards.
Project Examples	Updated paths and image.
Introduction	Removed references to old platforms. Corrected PIB name.
Physical Abstraction Layer (PAL)	Added MPAL.
Hardware Resources Usage	Updated mandatory timers.
HAL API	Updated HAL functions.
4.3.1. Task Manager, Priorities and Preemption	Updated timing.
Other Coding Requirements	Removed priority setting for TC0.
PRIME Callback Functions	Added zero cross functions.
Firmware Version Information	Updated model and version.
Communication Channel and other PHY Parameters	Added PLC couplings for PL460. Corrected configuration key. Removed thresholds.
PLC Coupling (PL460 platform), Zero Cross Detection in PLC, Serial PHY SAP	New chapters.
Band Plan in PLC	Renamed chapter. Forced channel 1 in band plan.
Enabling PLC PHY Layer, Enabling Serial PHY Layer, Sniffer Serialization	Renamed chapter. Updated setting.
Base Node Parameters	Corrected configuration key.
File Transfer Stage	Added note to indicate that signature is not supported.
API for PRIME Profile in Management Plane	Added comment about enhanced API for phase detection.
General Prerequisites	Updated IAR version. Removed Atmel Studio.
Supported MCU Families, Supported Devices and Supported boards	Updated devices and boards.
PHY Frames	Removed date of specification.
PLC PHY SAP, Linking of PRIME Sniffer	Renamed chapters.
Callback Functions	Updated callback functions.
Data Primitives	Updated request function input and output parameters.
Management Primitives	Added column to indicate in which PHY platforms the primitives are available. Added PCH to primitives.
API of PRIME FW Stack	Added comment about parameters from other specification versions.
PRIME Profile Primitives	Added zero cross functions.

Proprietary PIB Attributes in the PHY Layer	Removed obsolete PIBs. Updated MTP PIBs. Added new PIBs. Added specific enabling/disabling of sniffer for different PHY layers.
Proprietary PIB Attributes in the IEC 61334-4-32 SSSC Layer	Corrected PIB name.
PLC PHY Applications	Renamed chapter. Updated name of last application.
USI Frame Format	Corrected name of protocol and added missing one.
Definition and Configuration of USI Ports	Removed note about SAM4C.
Linking of PRIME PHY Serial	Corrected setting.
Abbreviations	Updated abbreviations.
References	Updated links.

18.3 Rev A - 07/2020

The following changes were incorporated in this revision:

Document	Editorials.
Example Configuration - conf_app_example.h and Linking of PRIME API	Updated configuration of modem port.
HAL API	Added HAL functions for network recovery. Changed structure for notes.
HAL API and PRIME Management Plane	Added user specific PIBs.
Main Function Structure	Improved hardware initialization.
Other Coding Requirements	Added effect on increased call stack size. Removed optimizations in Atmel Studio projects. Added clarifications about interrupt handling.
Data Storage	Removed allocation of PHY layer in the PL360 platform.
Proprietary PIB Attributes in the PHY Layer	Added new proprietary PIBs and updated MTP PIB.
Get Function and Set Function	Aligned function parameters with code.
Proprietary PIB Attributes in the PHY Layer and PRIME Sniffer Frame Format	Added clarifications about time units.
Firmware Version Information	Updated vendor and version.
Communication Channel and other PHY Parameters	Added ATPLCOUP011. Corrected configuration key.
Hardware Resources Usage	Added remark about platform for mandatory timers.
Automatic Repeat Request (ARQ)	Replaced frame by fragment.
General Prerequisites	Updated IAR version.
Definition and Configuration of USI Ports	Added USB type as USI port.
Features	Added missing modem PL360B.
PLC Coupling (PL360 Platform)	Added coupling configuration in PL360 platform.
Definition and Configuration of USI Ports, Linking of Serial Communications Profile of PRIME Management Plane, Linking of PRIME Sniffer (PL360 Platform), Linking of PRIME PHY Serial, 14.3.3. Linking of PRIME Sniffer	Clarified usage of USI ports.
PHY Applications	Added another PHY example.
PRIME Sniffer Frame Format	Added sniffer type version for PL360.
PRIME FW Stack Library, Supported MCU Families, Supported Devices and Supported boards	Added new devices and boards.
PRIME Management Plane Frame Format	Corrected name of MNG protocol.
USI Host Example	Added meaning of abbreviation.
Proprietary PIB Attributes in the IEC 61334-4-32 SSSC Layer	Removed PIB that belonged to SN only. Corrected size of serial number.

Proprietary PIB Attributes in the Management Plane	Removed LNID from PIB_FU_LIST.
References	Updated links.

18.4 Old Revision History

The following revision histories correspond to the document updates prior moving to MCU32.

18.4.1 Rev A - 06/2016

Document	Initial document release.
----------	---------------------------

18.4.2 Rev B - 07/2016

Document	Minor corrections and editorials.
Proprietary PIB Attributes in the MAC Layer	Added proprietary PIB.
References	Corrected references.
Understanding the Firmware Package, Base Slave Modem, Auxiliary Nodes Connection in Multi-transformer Substations	Added updates related to ticket 65 (auxiliary nodes in multi-transformer substations).

18.4.3 Rev C - 09/2016

Other Coding Requirements	Added more details about confirm requirements.
Retrieval of Lists	Added information about list retrieval.
Serialization with Embedded USI	Added embedded USI.
Base Slave Modem, Auxiliary Nodes Connection in Multi-transformer Substations	Improved ticket 65.

18.4.4 Rev D - 12/2016

PRIME API Frame Format	Added note to BMNG commands in USI.
HAL API	Modified HAL API function.
Proprietary PIB Attributes in the MAC Layer	Added PIB for ticket 65.
PLC Firmware Upgrade Stage	Corrected confirmed state of FUP.
General Prerequisites	Updated tool versions.
Other Coding Requirements	Added warning about code optimizations.

18.4.5 Rev E - 05/2017

Other coding requirements and Main Function Structure	Added supply monitor control.
HAL API	Added HAL functions.
Whitelist Management and Whitelist Management Primitives	Added support of whitelist.
Firmware Version Information	Improved Firmware Version Information chapter.
MAC primitives and IEC 61334-4-32 Primitives	Corrected MAC and 4.32 primitives.

18.4.6 Rev F - 05/2017

Whitelist Management	Added comment about removing node from whitelist.
Firmware Upgrade Protocol	Corrected macro names for FUP states. Added error codes for FUP ACK.
Enabling Serial PHY Layer	Corrected configuration of Serial PHY layer.
General Prerequisites	Updated version of Atmel Studio.

18.4.7 Rev G - 10/2017

Document	Editorials.
----------	-------------

HAL API	Added warning about interruptions in hal_cfg.
---------	---

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, SQL, 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.

All other trademarks mentioned herein are property of their respective companies.

© 2024, Microchip Technology Incorporated and its subsidiaries. All Rights Reserved.

ISBN: 978-1-6683-4114-8

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 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com	Australia - Sydney Tel: 61-2-9868-6733 China - Beijing Tel: 86-10-8569-7000 China - Chengdu Tel: 86-28-8665-5511 China - Chongqing Tel: 86-23-8980-9588 China - Dongguan Tel: 86-769-8702-9880 China - Guangzhou Tel: 86-20-8755-8029 China - Hangzhou Tel: 86-571-8792-8115 China - Hong Kong SAR Tel: 852-2943-5100 China - Nanjing Tel: 86-25-8473-2460 China - Qingdao Tel: 86-532-8502-7355 China - Shanghai Tel: 86-21-3326-8000 China - Shenyang Tel: 86-24-2334-2829 China - Shenzhen Tel: 86-755-8864-2200 China - Suzhou Tel: 86-186-6233-1526 China - Wuhan Tel: 86-27-5980-5300 China - Xian Tel: 86-29-8833-7252 China - Xiamen Tel: 86-592-2388138 China - Zhuhai Tel: 86-756-3210040	India - Bangalore Tel: 91-80-3090-4444 India - New Delhi Tel: 91-11-4160-8631 India - Pune Tel: 91-20-4121-0141 Japan - Osaka Tel: 81-6-6152-7160 Japan - Tokyo Tel: 81-3-6880-3770 Korea - Daegu Tel: 82-53-744-4301 Korea - Seoul Tel: 82-2-554-7200 Malaysia - Kuala Lumpur Tel: 60-3-7651-7906 Malaysia - Penang Tel: 60-4-227-8870 Philippines - Manila Tel: 63-2-634-9065 Singapore Tel: 65-6334-8870 Taiwan - Hsin Chu Tel: 886-3-577-8366 Taiwan - Kaohsiung Tel: 886-7-213-7830 Taiwan - Taipei Tel: 886-2-2508-8600 Thailand - Bangkok Tel: 66-2-694-1351 Vietnam - Ho Chi Minh Tel: 84-28-5448-2100	Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 Denmark - Copenhagen Tel: 45-4485-5910 Fax: 45-4485-2829 Finland - Espoo Tel: 358-9-4520-820 France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 Germany - Garching Tel: 49-8931-9700 Germany - Haan Tel: 49-2129-3766400 Germany - Heilbronn Tel: 49-7131-72400 Germany - Karlsruhe Tel: 49-721-625370 Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 Germany - Rosenheim Tel: 49-8031-354-560 Israel - Ra'anana Tel: 972-9-744-7705 Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781 Italy - Padova Tel: 39-049-7625286 Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340 Norway - Trondheim Tel: 47-72884388 Poland - Warsaw Tel: 48-22-3325737 Romania - Bucharest Tel: 40-21-407-87-50 Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 Sweden - Gothenberg Tel: 46-31-704-60-40 Sweden - Stockholm Tel: 46-8-5090-4654 UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820