

---

## Introduction

This document is the user guide for the implementation of the PRIME 1.3 Stack for Service Nodes 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 Service Nodes
- Support of several microcontroller families including PIC32CXMTx, SAM4C, SAM4CMx, and SAMG55
- 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 Project.....	12
3.1. Introduction.....	12
3.2. PRIME Stack User Configuration Parameters.....	13
4. Integrating the PRIME FW Stack.....	19
4.1. Memory Allocation.....	19
4.2. Hardware Resources Usage.....	22
4.3. PRIME Interfaces.....	23
4.4. PRIME Integration Requirements.....	27
4.5. Main Function Structure.....	28
4.6. Application Configuration Parameters.....	31
5. Data Exchange.....	33
5.1. Null SSCS.....	33
5.2. IEC 61334-4-32.....	33
5.3. Automatic Repeat Request (ARQ).....	33
6. File Transfer Service for Firmware Upgrade.....	37
6.1. Introduction.....	37
6.2. CRC Result Callback.....	40
6.3. FU Configuration Data.....	40
6.4. Starting FU.....	41
6.5. Providing the Bitmap.....	41
6.6. Writing FU Data.....	41
6.7. CRC Calculation.....	42
6.8. Finishing FU.....	42
6.9. Managing the Received File.....	42
7. PRIME Management Plane.....	44
8. Toolchain.....	45
8.1. General Prerequisites.....	45
8.2. Building the Applications.....	45
9. Supported Platforms.....	46
9.1. Supported MCU Families.....	46
9.2. Supported Devices.....	46
9.3. Supported Boards.....	46

10. PICS.....	47
10.1. Major Roles for Devices Compliant with PRIME.....	47
11. API of PHY and PAL Layers.....	49
11.1. PLC PHY SAP.....	49
11.2. PAL SAP.....	49
11.3. PAL Primitives.....	50
12. API of PRIME FW Stack.....	51
12.1. MAC Primitives.....	51
12.2. IEC 61334-4-32 Primitives.....	54
12.3. PIB Objects Specification and Access.....	55
13. Example Applications.....	59
13.1. Usage of the Firmware.....	59
13.2. PRIME Service DLMS + Metrology Application.....	59
13.3. PRIME Service DLMS Application.....	60
13.4. PRIME Service Modem.....	61
13.5. PHY Applications.....	62
14. Serialization with Embedded USI.....	64
14.1. USI Frame Format.....	64
14.2. USI PRIME Protocols.....	65
14.3. Embedded USI Configuration.....	67
15. Abbreviations.....	69
16. References.....	71
17. Revision History.....	72
17.1. Rev A - 07/2018.....	72
17.2. Rev B - 12/2018.....	72
17.3. Rev C - 07/2020.....	72
17.4. Revision D - 02/2024.....	73
Microchip Information.....	75
The Microchip Website.....	75
Product Change Notification Service.....	75
Customer Support.....	75
Microchip Devices Code Protection Feature.....	75
Legal Notice.....	75
Trademarks.....	76
Quality Management System.....	77
Worldwide Sales and Service.....	78

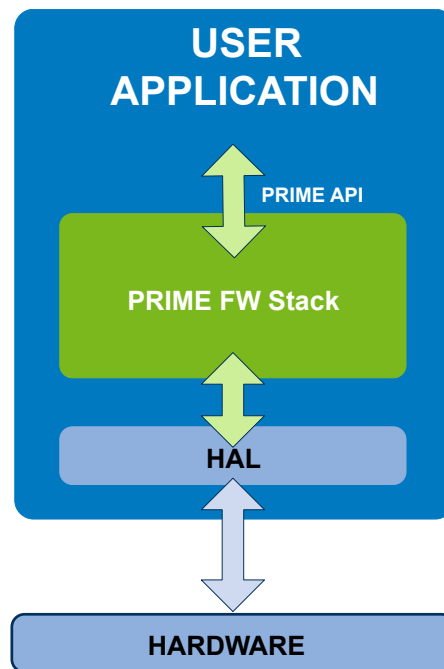
# 1. Overview

## 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.

**Figure 1-1.** Block Diagram of the PRIME General Architecture



The PRIME general architecture has been implemented with separated software modules as a way to optimize hardware resources and provide flexibility and versatility to the final user system. Following this concept, the user application and the PRIME FW stack have separated binary images. Additionally, in the PL360/PL460 platform, there is a third binary image corresponding to the PL360 firmware to be loaded into the PL360 device.



**Important:** Device memory and HAL must be managed by the users in order to allocate all binary images of the system and to handle the hardware resources according to their needs (see chapter [Integrating the PRIME FW Stack](#)).

Microchip provides an example project of the PRIME FW stack. Structure and features of this project are described in chapter [PRIME FW Stack Project](#). The API of the PRIME FW stack is described in chapter [API of PRIME FW Stack](#).

Structure, requirements and features of the user application are described in chapter [Integrating the PRIME FW Stack](#). 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 system. 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](#)).

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 and users can allocate it at any address within their region. 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 Service 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/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.

One of the most important features of the PRIME implementation is its ability to manage the PRIME FW stack and the user application as separate software modules. The corresponding binary images (together with the PL360 firmware in the PL360/PL460 platform) can be allocated to any memory addresses configured by the end user, thus providing a highly flexible and versatile implementation.

### 2.1 PRIME Firmware Package Contents

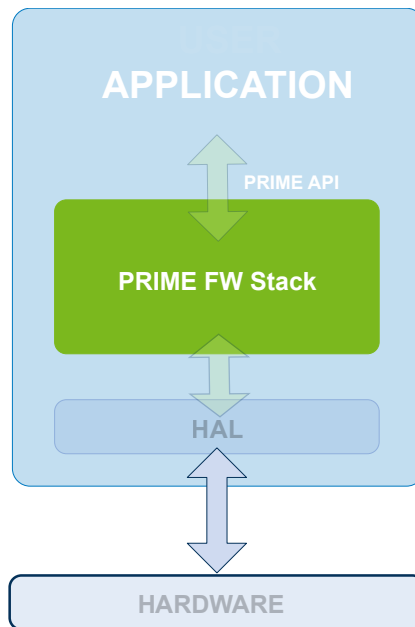
Microchip provides PRIME packages for every platform that supports PRIME. These packages include PRIME applications that communicate using the PRIME network and applications that only make use of the PRIME PHY layer, so that users can easily access all available user applications and their related files. 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 Project Example

The following figure shows where the PRIME FW stack is located inside the complete general architecture.

**Figure 2-1.** PRIME FW Stack in the PRIME General Architecture



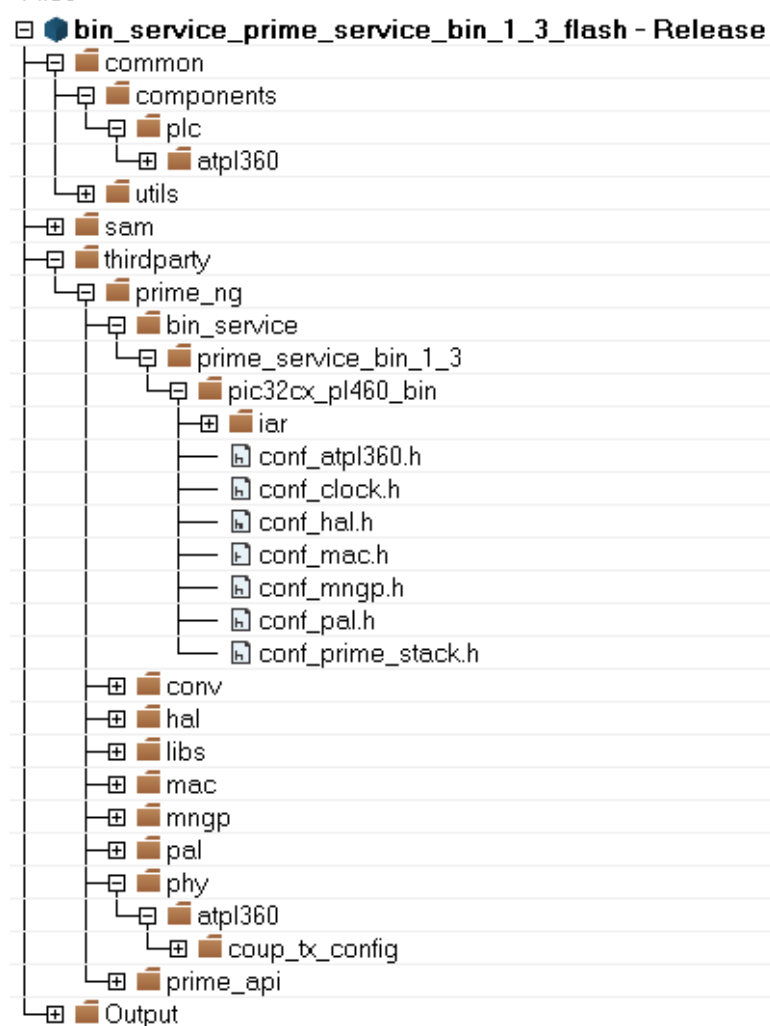
The PRIME FW stack project example contains the PRIME library (with the PL360 Host Controller) and the configuration files described in chapter [PRIME Stack User Configuration Parameters](#). This project generates the mentioned binary image corresponding to the PRIME FW stack.

The PRIME FW stack project can be found in the package following this path:

```
..\thirdparty\prime_ng\bin_service\prime_service_bin_1_3
```

The structure of this PRIME FW stack project in IAR Embedded Workbench® is shown in the following figure.

Figure 2-2. Folder Structure of the PRIME FW Stack Project

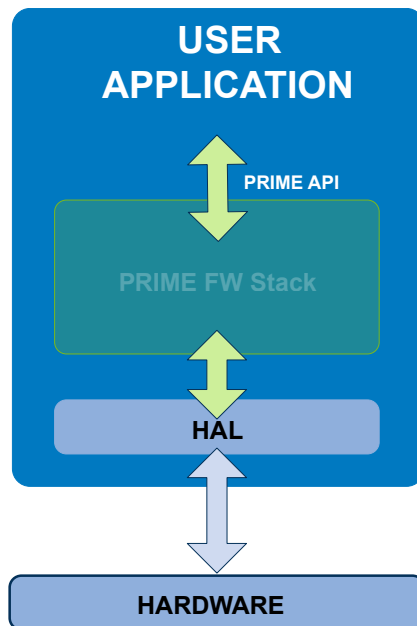


**Important:** It is recommended that users modify only the available configuration files according to their application needs (see chapter [PRIME Stack User Configuration Parameters](#)).

### 2.1.2 User Application Project Examples

The following figure shows where the user application is located inside the complete general architecture.

**Figure 2-3.** User Application in the PRIME General Architecture



Microchip provides several user application projects that show how to interact with the PRIME FW stack:

- DLMS (Device Language Message Specification) application: this application shows how to interact with the PRIME FW stack to simulate the exchange of DLMS data in a PRIME network.
- DLMS+Metrology application: this application shows how to interact with the PRIME FW stack to exchange real DLMS data in a PRIME network (only available in the SAM4C platform).
- Modem application: this application shows how to manage the PRIME FW stack from an external application by serializing the PRIME API.

The user application projects can be found in the package following this path:

```
..\thirdparty\prime_ng\apps\
```

All of these user application projects are provided in dual mode, i.e., they can be used in any Service Node regardless of the PRIME FW stack version that is running.



**Important:** Dual mode applications are not provided in the SAMG55 platform due to the lack of internal memory space. See Chapter [Memory Allocation](#) for more information.

Additionally, Microchip also provides several PLC PHY applications that show how to manage the PRIME PLC PHY layer exclusively:

- 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.
- PHY Sniffer: this PHY application allows monitoring of data traffic on a PRIME network by serializing PHY frames. Depending on the platform, this PHY sniffer might be located in the `apps` folder instead of the `phy apps` folder.



- PHY PLC&Go: this PHY application shows the basic code required to work with the PLC PHY layer.

The PHY application projects can be found in the package following this path:

```
..\thirdparty\prime_ng\phy\atpl360\apps
```

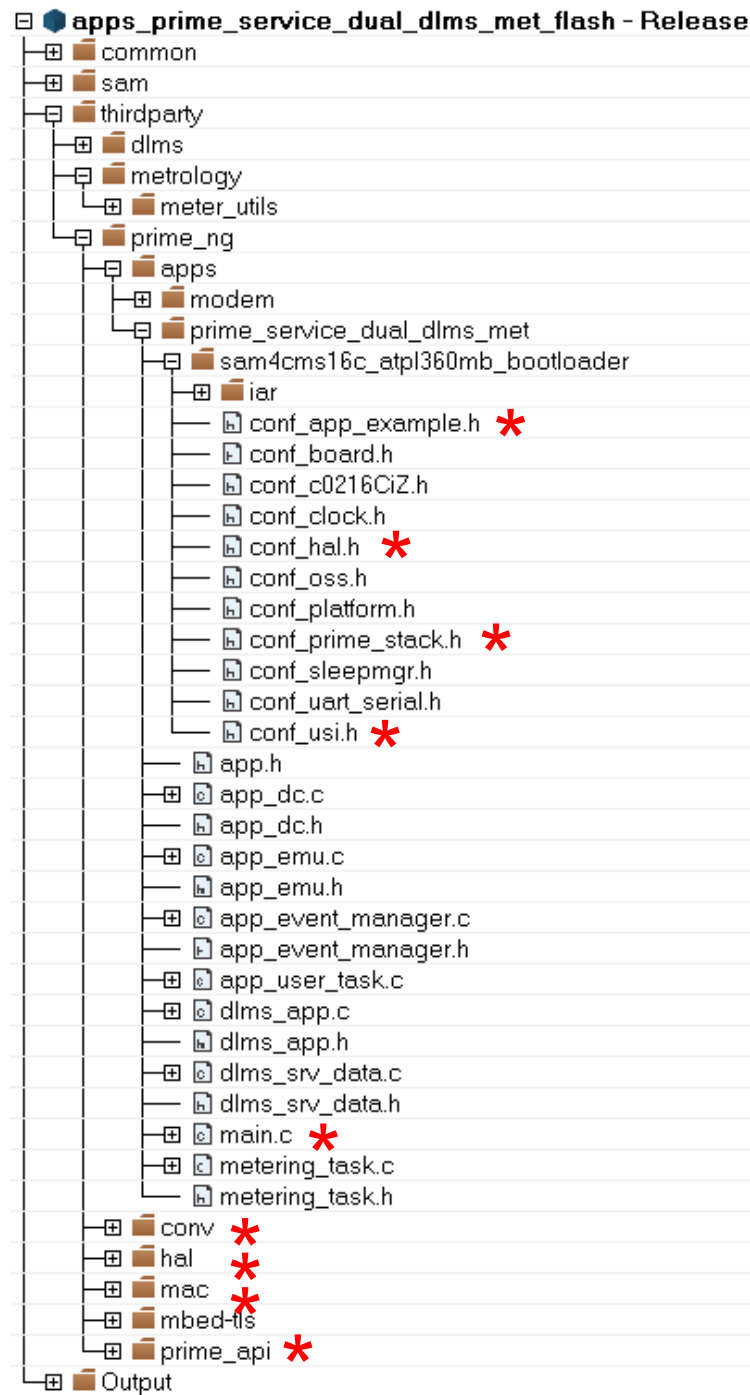
See Chapter [Example Applications](#) for detailed information about the operation of every example.

For example, a DLMS+Metrology application project can be found in the package following this path:

```
..\thirdparty\prime_ng\apps\prime_service_dual_dlms_met\sam4cms16c_atpl360mb_  
bootloader\iar\apps_prime_service_dual_dlms_met.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 2-4. Folder Structure of the DLMS+Metrology Application 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 `lsam\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.

### 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*.

### 2.1.5 Bootloader Example

Microchip provides a bootloader example in order to allow the upgrade not only of the PRIME FW stack but also of the user application and of the PL360 firmware running in the PL360 device. Details about the firmware upgrade can be found in chapter [File Transfer Service for Firmware Upgrade](#).



**Important:** All provided user application project examples that interact with the PRIME FW stack are configured to be used together with this bootloader example.

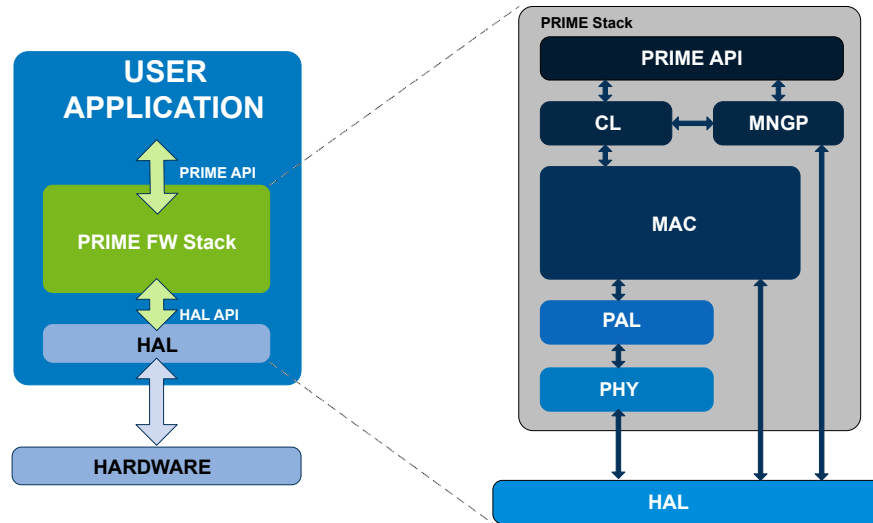
---

## 3. PRIME FW Stack Project

### 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. The binary file generated with the PRIME FW stack project example contains the complete PRIME FW stack with all the layers shown in the previous figure. The PHY is not the PHY layer as such but the host controller component.

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 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 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 12-6](#)).

### 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](#).

### 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 12-7](#)).

### 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.

## 3.2 PRIME Stack User 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).

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 4.2.1. [Data Storage](#) for more information.

### 3.2.1 Firmware Version Information

In the provided PRIME FW stack binary project, 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 or upgraded. Users can edit this information in file *conf\_prime\_stack.h*.

```
/* Firmware Information */
#define PRIME_FW_VENDOR    "MCHP"
#define PRIME_FW_MODEL     "PL360MB"
#define PRIME_FW_VERSION   "S13.10.05\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    0x3937
```

The modifiable attributes are the following:

**Table 3-1. 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. "PL360MB".	ASCII. Maximum 16 bytes.
PRIME_FW_VERSION	Version identifier as defined in PIB 0x0075 (PIB_MAC_APP_FW_VERSION), e.g. "S13.10.05\0\0\0\0\0\0\0".	Numerical, ASCII recommended. <b>It must be 16 bytes long.</b>
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. "97".	Numerical or ASCII, two bytes size.

### 3.2.2 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 3-2. 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 12-6](#). 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.

### 3.2.3 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
```

### 3.2.4 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
```

### 3.2.5 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 3-2](#). This is set in `USER_BAND_PLAN` in file `conf_pal.h`.

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

### 3.2.6 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 12-7](#)). 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.

### 3.2.7 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
```

### 3.2.8 PL360 Firmware Information (PL360/PL460 Platform)

Users must configure information related to the PL360 firmware that is loaded into the PL360 device. This information includes the address where the binary image is stored (`ATPL360_BINARY_ADDRESS`) as well as its size (`ATPL360_BINARY_LEN`). These values must be



taken into account when assigning memory resources. See chapter [Memory Allocation](#) for more information.

```
/* Component ATPL360 Binary Information */
#define ATPL360_BINARY_ADDRESS      0x010C0000
#define ATPL360_BINARY_LEN         0x10000
```

### 3.2.9 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
```



**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 12-6](#)).

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

### 3.2.10 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.

### 3.2.11 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.

### 3.2.12 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".

### 3.2.13 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 3-3.** Configured PIB Values in the PRIME FW Stack

Attribute	ID	PRIME default value	Implementation value
PIB_MAC_MIN_SWITCH_SEARCH_TIME	0x0010	24	32
PIB_MAC_MAX_PROMOTION_PDU	0x0011	2	2
PIB_MAC_PROMOTION_PDU_TX_PERIOD	0x0012	5	5
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. Integrating the PRIME FW Stack

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

The following issues must be taken into account:

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

### 4.1 Memory Allocation

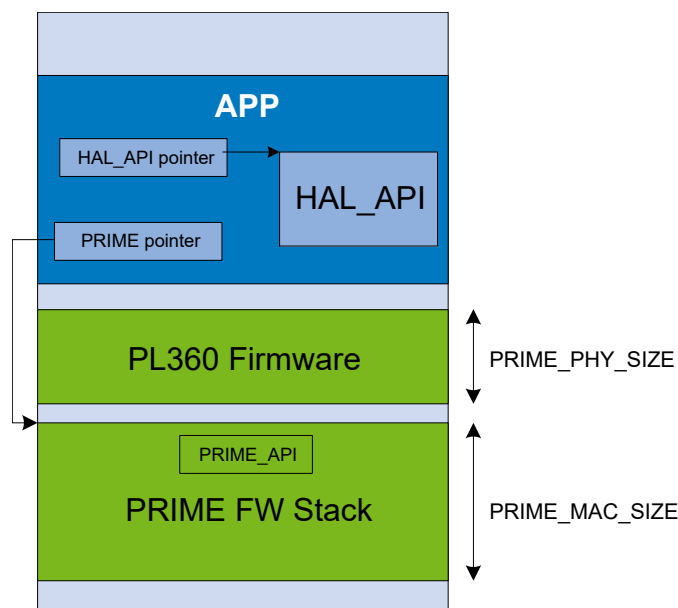
One of the most important features of the PRIME implementation is its ability to manage the PRIME FW stack and the user application as separate software modules. The corresponding binary images can be allocated to any memory addresses configured by the end user, thus providing a highly flexible and versatile implementation. The PL360 firmware also requires a separate memory section.



**Important:** The PRIME FW stack requires a minimum size of 128kB for its memory sector in ROM and a minimum size of 32kB of RAM. Additionally, the PL360 firmware requires a minimum size of 64kB for its memory sector in ROM.

The allocation address of the PRIME FW stack module is defined and managed by the user application. All the PRIME functions in the PRIME API are referenced relatively to this address. The pointer to the HAL API is also needed, so that the PRIME FW stack can invoke the HAL API functions as shown in the figure below.

**Figure 4-1.** Pointers in the User Application





**Important:** Users can reserve and manage as many memory regions as they wish provided they keep at least one area for a copy of the last upgraded binary file. The PRIME specification requires this in case the firmware upgrade process must revert to a previous version.

#### 4.1.1 Default Memory Allocation Example

The figures below depict the memory allocation configured in the provided application project examples with bootloader. As it can be seen, there are several regions reserved for all the binaries as well as for the firmware upgrade management, whose region must be at least as big as the biggest binary region (according to the example provided by Microchip).

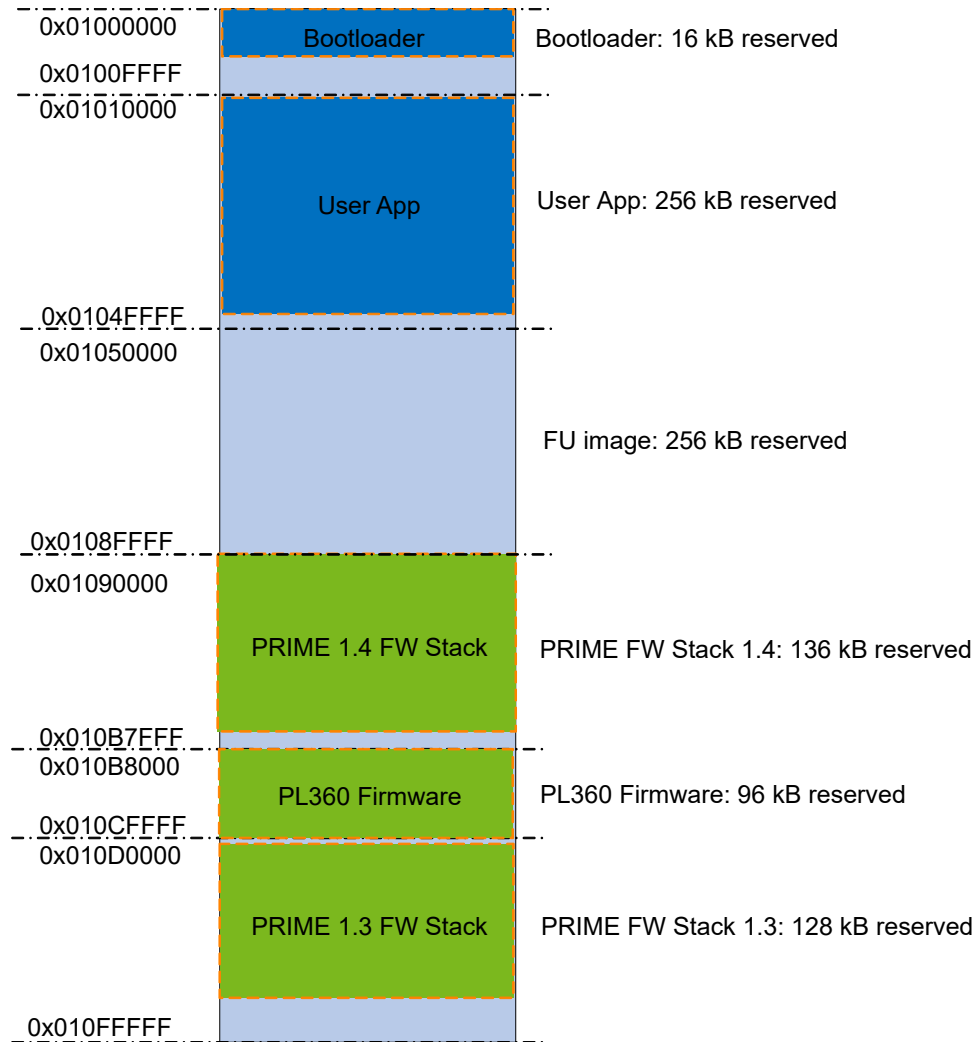
This memory distribution has been chosen to ease the firmware upgrade process. The memory addresses can be configured in file *conf\_app\_example.h*. For example, for a Flash memory of 1MB in a dual mode application:

```
/* Define PRIME location in flash */
#define PRIME_APP_SIZE           0x00040000
#define PRIME_MAC13_SIZE        0x00020000
#define PRIME_MAC14_SIZE        0x00022000
#define PRIME_PHY_SIZE          0x00018000
#define PRIME_IMAGE_SIZE        0x00040000
#define PRIME_APP_FLASH_LOCATION 0x01010000
#define PRIME_IMAGE_FLASH_LOCATION 0x01050000
#define PRIME_MAC14_FLASH_LOCATION 0x01090000
#define PRIME_PHY_FLASH_LOCATION 0x010B8000
#define PRIME_MAC13_FLASH_LOCATION 0x010D0000
```



**Important:** The address of the PLC PHY layer configured in file *conf\_app\_example.h* must have the same value of the address of the PL360 firmware that is configured in file *conf\_pal.h* (see chapter [PL360 Firmware Information \(PL360/PL460 Platform\)](#)).

**Figure 4-2.** Flash Example for PRIME with Bootloader (Compilation with 1 MB Flash Memory in Dual Mode Application)



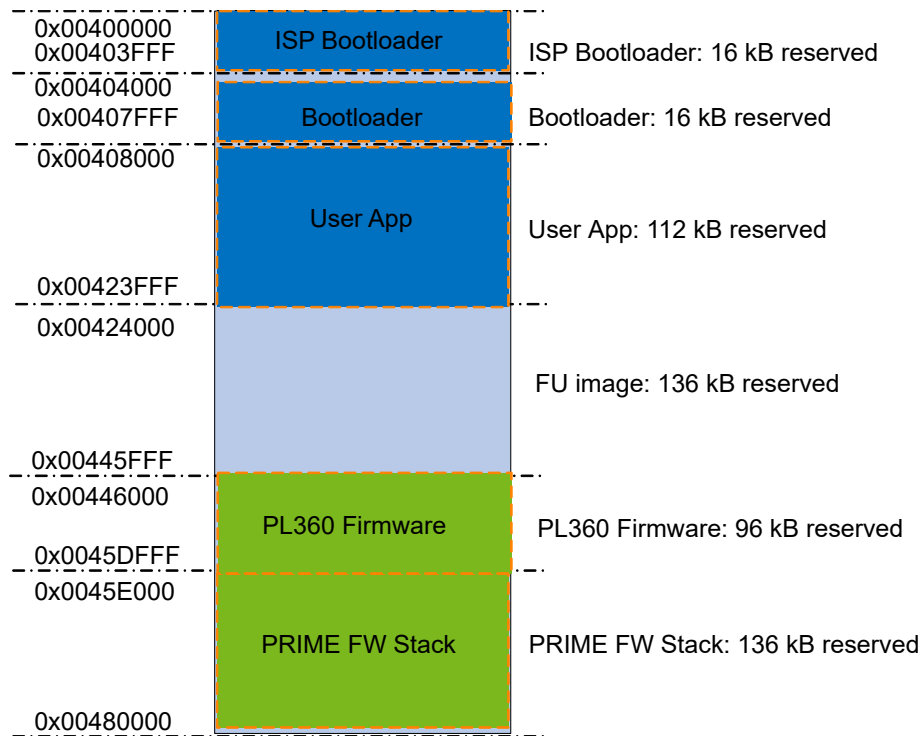
And this is an example for the SAMG55 platform, which does not support dual mode:

```
/* Define PRIME location in flash */
#define PRIME_APP_SIZE 0x0001C000
#define PRIME_MAC13_SIZE 0x00020000
#define PRIME_MAC14_SIZE 0x00022000
#define PRIME_PHY_SIZE 0x00018000
#define PRIME_IMAGE_SIZE 0x00022000
#define PRIME_APP_FLASH_LOCATION 0x00408000
#define PRIME_IMAGE_FLASH_LOCATION 0x00424000
#define PRIME_PHY_FLASH_LOCATION 0x00446000
#define PRIME_MAC14_FLASH_LOCATION 0x0045E000
#define PRIME_MAC13_FLASH_LOCATION 0x0045E000
```



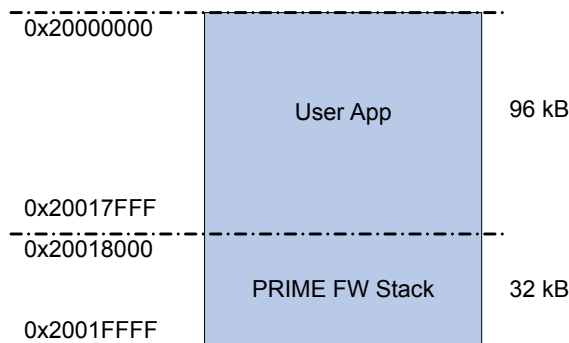
**Important:** The address of the PLC PHY layer configured in file *conf\_app\_example.h* must have the same value of the address of the PL360 firmware that is configured in file *conf\_pal.h* (see chapter [PL360 Firmware Information \(PL360/PL460 Platform\)](#)).

**Figure 4-3.** Flash Example for PRIME with Bootloader (Compilation for SAMG55)



Additionally, the PRIME FW stack imposes memory requirements to the user application regarding the RAM. This is shown in the next figure. The memory addresses and sizes are defined in the linker scripts that go with the application examples.

**Figure 4-4.** 128 kB SRAM Memory Map



## 4.2 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 $\mu$ 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).



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

#### 4.2.1 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:

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

### 4.3 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.

#### 4.3.1 PRIME API

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

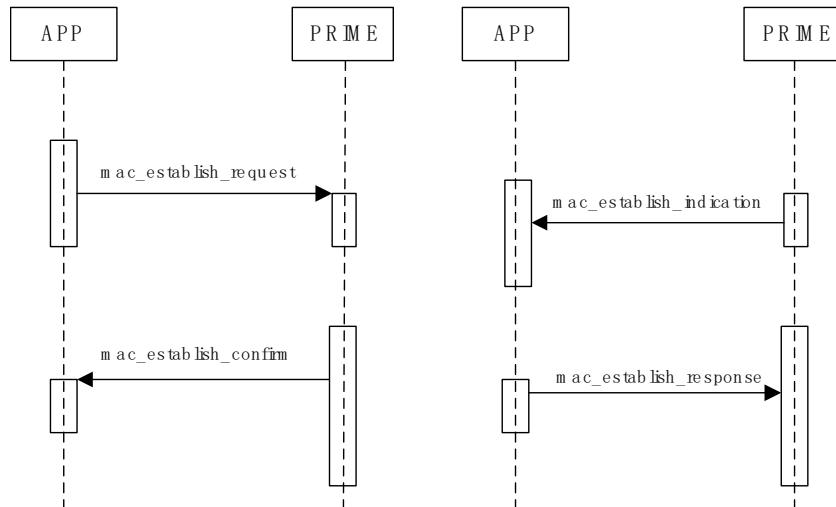
The PRIME API is based on a request/confirm mechanism, which is a particular instance of an event-driven programming model. A *request* is an asynchronous call to the underlying stack to perform some action on behalf of the user application; a *confirm* is the callback that executes when that action 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 4-5. 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](#).

#### 4.3.1.1 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.



### 4.3.1.2 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 may or may 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.

### 4.3.2 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	<i>hal.c</i>	Restart the complete system.
hal_pcrc_calc	<i>hal_pcrc.c</i>	Calculate the CRC of the input buffer according to the PRIME specification.

.....continued

Function	File	Description
hal_pcrc_config_sna	hal_pcrc.c	Configure the SNA for the CRC.
hal_fu_data_write	hal_fu.c	Write data in memory during FU.
hal_fu_data_cfg_read	hal_fu.c	Read FU configuration data.
hal_fu_data_cfg_write	hal_fu.c	Write FU configuration data.
hal_fu_start	hal_fu.c	Initialize and unlock memory for a FU process.
hal_fu_end	hal_fu.c	Finish the FU process.
hal_fu_revert	hal_fu.c	Revert to the old firmware.
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_fu_get_bitmap	hal_fu.c	Get bitmap with the status of each page.
hal_plc_init <sup>1</sup>	hal_plc.c	Initialize the PLC interface.
hal_plc_reset <sup>1</sup>	hal_plc.c	Reset the internal PLC modem.
hal_plc_set_handler <sup>1</sup>	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 <sup>2</sup>	hal_cfg.c	Read configuration parameters. <b>This function disables all interrupts when accessing the User Signature.</b>
hal_set_config_info <sup>2</sup>	hal_cfg.c	Write configuration parameters. <b>This function disables all interrupts when accessing the User Signature.</b>
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 <sup>3</sup>	hal_plc.c	Send a boot command. <b>This function disables all interrupts.</b>
hal_plc_send_wrrd_cmd <sup>3</sup>	hal_plc.c	Send a write/read command. <b>This function disables all interrupts.</b>
hal_plc_enable_interrupt <sup>3</sup>	hal_plc.c	Enable or disable the PLC interrupt.
hal_plc_delay <sup>3</sup>	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_swap_stack	hal.c	Request to swap the PRIME FW stack.
hal_plc_set_stby_mode <sup>3</sup>	hal_plc.c	Set the stand-by mode.
hal_plc_get_thermal_warning <sup>3</sup>	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.

## 4.4 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.

### 4.4.1 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.

Note that the maximum execution time of the PRIME FW stack process is less than 1ms.

### 4.4.2 Stack Size

Users must take into account their own user tasks. User tasks need to have an appropriate stack size to guarantee that no heap task overflow occurs. The optimal number of tasks and memory task stack sizes need to be configured by users to avoid unexpected behaviors. This also applies to the PRIME task stack size because it depends on the callback functions implemented by users.



**Important:** Users must calculate the stack size considering the PRIME FW stack functions, the HAL functions and the application functions in order to obtain the worst case. Then, they should add a safety margin.

#### 4.4.3 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.
- Set to NULL all unused callback function pointers:

```
memset(&mac_cbs, NULL, sizeof(mac_cbs));
memset(&cl432_cbs, NULL, sizeof(cl432_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.
- Do not reset the software in the function that receives the result of the FU process. The reset must take place in a user task.
- Do not reset the software or change the PRIME API pointer in the function that requests a swap of the PRIME FW stack versions. The reset or pointer change must take place in a user task.
- Define an exclusive serial port number for the Serial Communication Profile of the Management Plane (only required for PRIME certification). The baud rate must be set to 57600 bauds.
- 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.

#### 4.4.4 CPU Usage

The CPU usage due to the PRIME FW stacks varies depending on the task that is being executed.

### 4.5 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 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 hal layer */
hal_init();

/* Init HAL callback for stack swap request */
hal_swap_stack_set_callback(_prime_swap_stack_request);

/* Read PRIME ptr address */
prime_api = PRIME_MAC_FLASH_LOCATION;

/* 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. Set the `prime_api` pointer
2. Invoke the `prime_init()` function to initialize the PRIME FW stack
3. 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 4-5](#).



**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.

#### 4.5.1 Pointer to the PRIME FW Stack

In order to call the functions of the PRIME API, the `prime_api` pointer must be available. The user is responsible for setting this pointer according to the required memory allocation.

#### 4.5.2 PRIME FW Stack Swap

When a node detects traffic from a different stack version, it notifies it to the HAL with function `hal_swap_stack` (see [HAL API](#)) indicating the type of the detected traffic. If users want to be notified about this in their application, the corresponding callback function must be provided to the HAL with the following function:

```
void hal_swap_stack_set_callback(void (*p_handler)(uint8_t uc_traffic));
```

Parameters:

- *p\_handler*: Pointer address to the callback function

#### 4.5.3 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.

##### 4.5.3.1 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

##### 4.5.3.2 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

#### 4.5.4 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](#).

## 4.5.5 PRIME Callback Functions

In order to be informed about confirm and indication primitives from the MAC layer and the CL, when the user application is initialized, it must set up required callback function pointers. 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)

This is an example of callback function pointer setup:

```
mac_callbacks_t mac_cbs;
cl_432_callbacks_t cl432_cbs;

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

/* Initialize callback pointers for MAC */
mac_cbs.mlme_register_ind_cb = _user_app_mlme_register_ind_cb;
mac_cbs.mlme_unregister_ind_cb = _user_app_mlme_unregister_ind_cb;
mac_cbs.mac_establish_cfm_cb = _user_app_establish_confirm_cb;
mac_cbs.mac_release_ind_cb = _user_app_release_ind_cb;
mac_cbs.mac_data_ind_cb = _user_app_data_indication_cb;

prime_cl_null_set_callbacks(&mac_cbs);

/* Initialize callback pointers for 4.32 */
cl432_cbs.cl_432_establish_cfm_cb = _user_app_cl_432_establish_cfm_cb;
cl432_cbs.cl_432_release_cfm_cb = _user_app_cl_432_release_cfm_cb;
cl432_cbs.cl_432_dl_data_ind_cb = _user_app_cl_432_dl_data_ind_cb;
cl432_cbs.cl_432_dl_data_cfm_cb = _user_app_cl_432_dl_data_cfm_cb;

prime_cl_432_set_callbacks(&cl432_cbs);
```

## 4.5.6 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.

## 4.6 Application Configuration Parameters

### 4.6.1 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.

### 4.6.2 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](#)).

### 4.6.3 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.



**Important:** Currently, Microchip only provides support for separated applications.

---

By commenting this parameter, users could also implement a monolithic architecture, that is, one project contains both the PRIME FW stack library as well as the user application and only one binary file is generated (including the PL360 firmware if necessary). However, a firmware upgrade of this kind of architecture is not supported by the provided example applications. The main purpose of this action would be to debug.

### 4.6.4 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 the MAC layer directly to establish a MAC connection to exchange data between application peers. Such connections can be made between a Base Node and a Service Node or between two Service Nodes (i.e. a direct connection).

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.

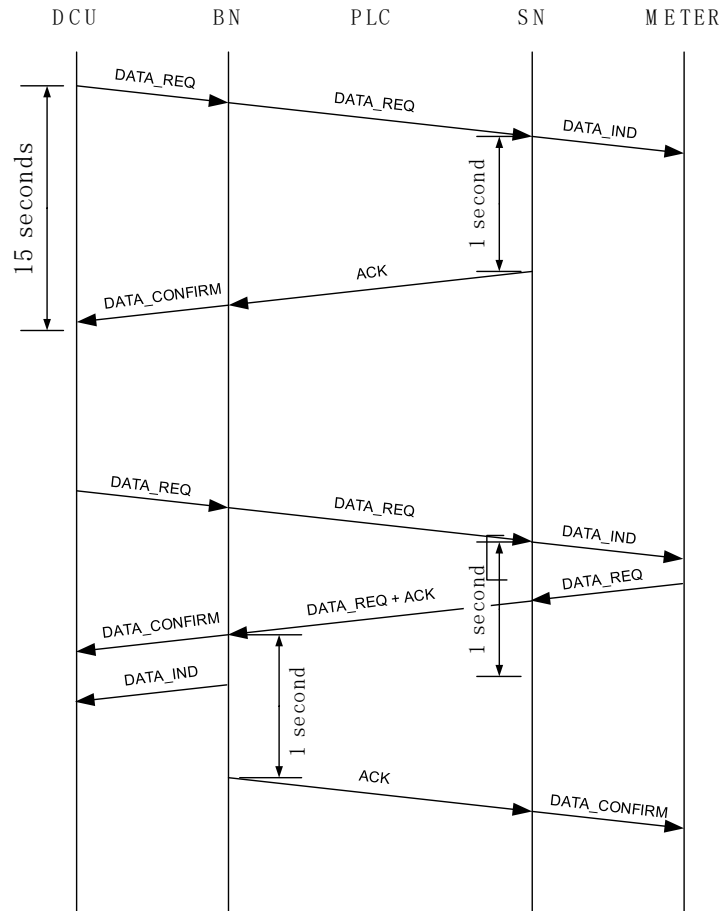
Available IEC 61334-4-34 primitives can be found in [Table 12-5](#).

### 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.

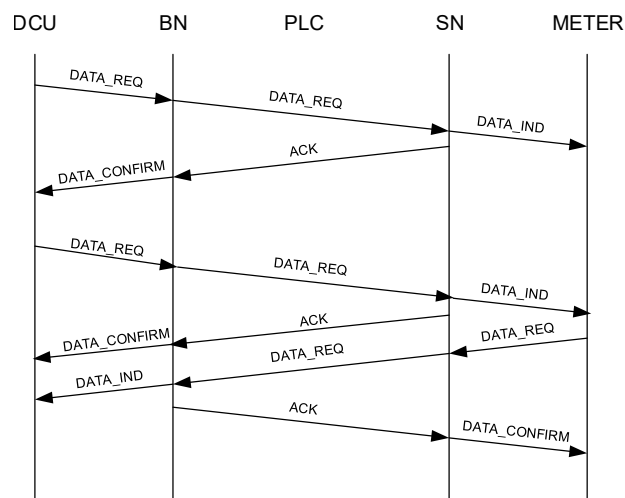
The following figure shows how ARQ works. If the application needs more than one second to send the reply, the Service Node sends the ACK after that time. If the response is received within one second, the PRIME FW stack piggybacks the ACK in the response.

**Figure 5-1. ARQ with One Fragment and Piggyback**



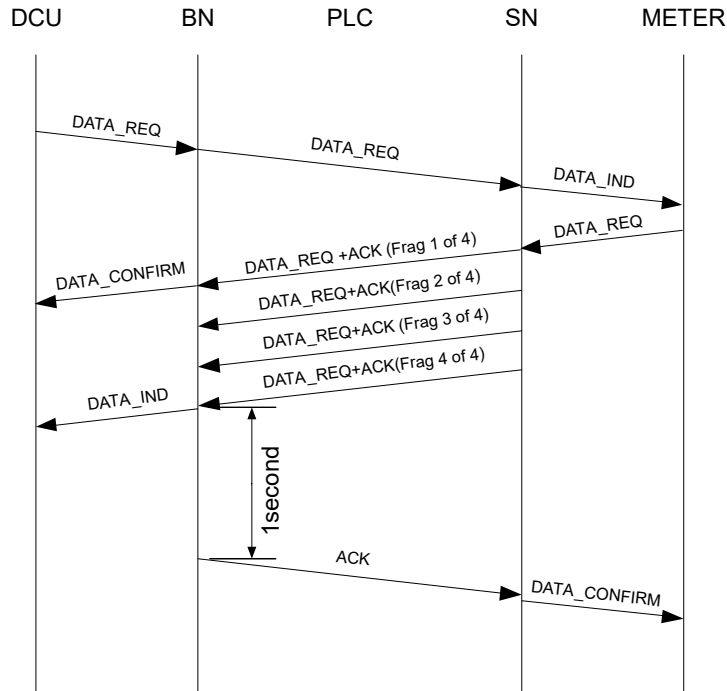
In certain situations, it might be possible that the ACK is sent as soon as the request is received, i.e. it is never piggybacked. That is shown in the next figure.

**Figure 5-2. ARQ with One Fragment**



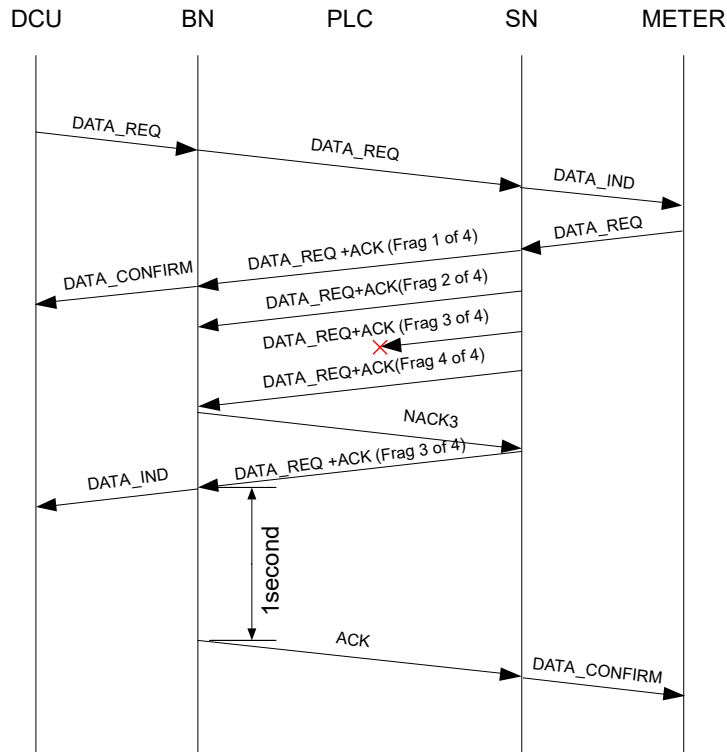
For long fragmented responses, there is an ACK piggybacked to each fragment and the BN sends the ACK after receiving the last fragment, as shown in the following figure.

**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.

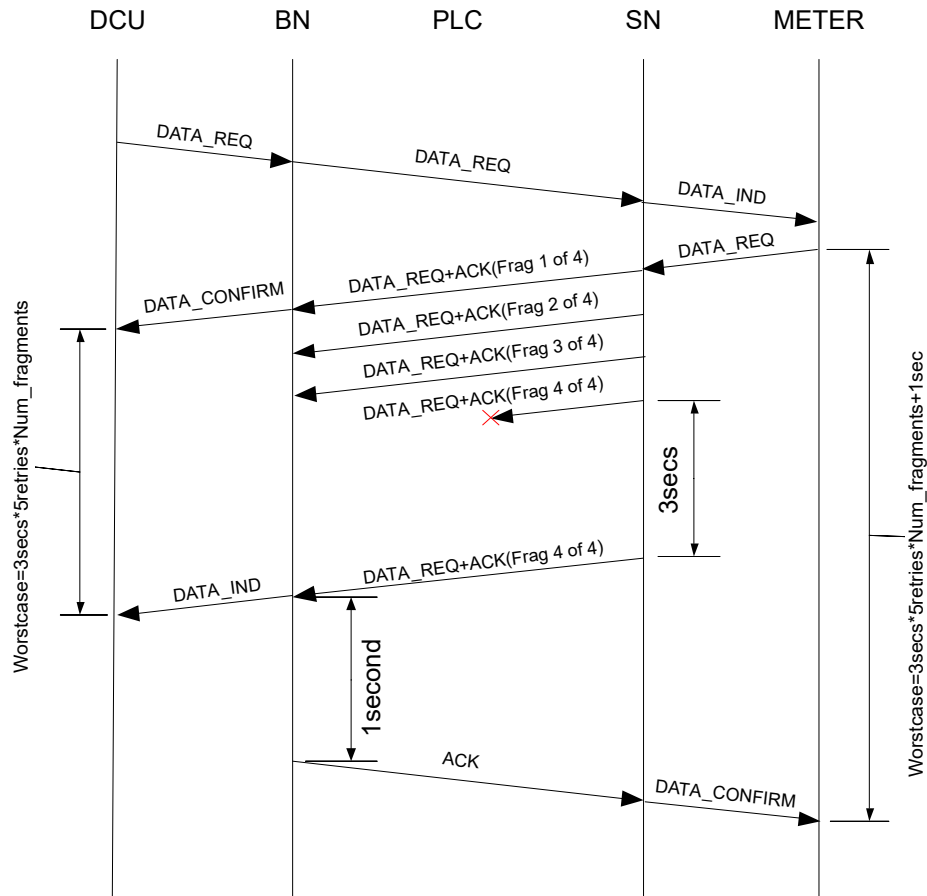
**Figure 5-4. ARQ with Four Fragments and Retransmission**



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. File Transfer Service for Firmware Upgrade

### 6.1 Introduction

The PRIME FW stack implements the PRIME FU process as described in the PRIME specification. Please refer to the PRIME specification for details.

In order to upgrade any binary, Microchip provides an example where it is only needed to reserve one memory region, which will first store the new image and, after the restart, the previous binary (see chapter [Memory Allocation](#)). The PRIME FW stack is only responsible for managing the FU process as described in the PRIME specification, whereas the user application is responsible for handling the pointers to these regions and managing the indications received in the HAL. Additionally, a complete system restart is required when the FU process finishes so that the bootloader application moves the received binary to its right location and keeps the previous binary for a possible revert.



**Important:** The bootloader application as well as the FU handling in the user application and in the HAL are only an example provided by Microchip. Users can develop any other strategy that fits their system requirements and resources in a more optimal way.

Moreover, since the PRIME FW stack does not control what kind of file is being received, the user application can also use the PRIME FU process to transfer files. In this case, users need to define a memory area to store the received file. Later, when the FU process finishes, the user application decides what to do with this file.

According to the PRIME specification, where it is assumed that the FU process is only used to upgrade the PRIME stack, the FU process does not end until the sent file is confirmed (result `HAL_FU_FW_CONFIRM` in the state diagram in [Figure 6-2](#)), i.e. regardless of the type of received file, the PRIME FW stack must always be restarted so that the FU process finishes properly.

The user should note that the PRIME FW stack is not able to access the hardware by itself, i.e. it is not directly reading from or writing to the memory, and it is not even changing the running PRIME FW stack. The user application manages and controls the HAL, which defines and handles the hardware accesses, including memories, and decides what to do with the received file.

At the beginning of an FU process, the PRIME FW stack starts the HAL FU module so that the HAL prepares the memory area to use when the stack requests to access the memory. The received file will be stored in that location.

At the end of the FU process, the PRIME FW stack indicates the result of the FU process to the HAL and then the user application manages the received file accordingly. This file can be a new image of the PRIME FW stack, a new image of the PL360 firmware, a new image of the user application or any other user file.



**Important:** Note that the FU process in the PRIME FW stack only checks the integrity and authentication of the received file using the file CRC. It does not perform any check on the validity of this file for a potential upgrade. It is recommended that the application performs its own validation by means of checking any metadata contents in the received file. It is up to the users to define how to check them.

For instance, in the PRIME FW stack binary file, the first 20 bytes contain the Vendor, Model and Version information. It is recommended to only accept binary files with matching Vendor and Model (as set in `PRIME_FW_VENDOR` and `PRIME_FW_MODEL`). The HAL is performing such check in file `hal_fu.c`.

Furthermore, in the provided examples, the last 16 bytes of any binary file are expected to include an identifier of the file being upgraded or transferred (the expected identifiers in file `hal_fu.c` are defined in file `conf_app_example.h`).

Figure 6-1 shows a flow diagram for a successful FU process with associated states and actions. Figure 6-2 shows the FU state diagram of the PRIME specification with the results provided by the HAL during a FU process. Both diagrams apply to any kind of file transfer.

**Figure 6-1. FU Flow Diagram with States and Actions**

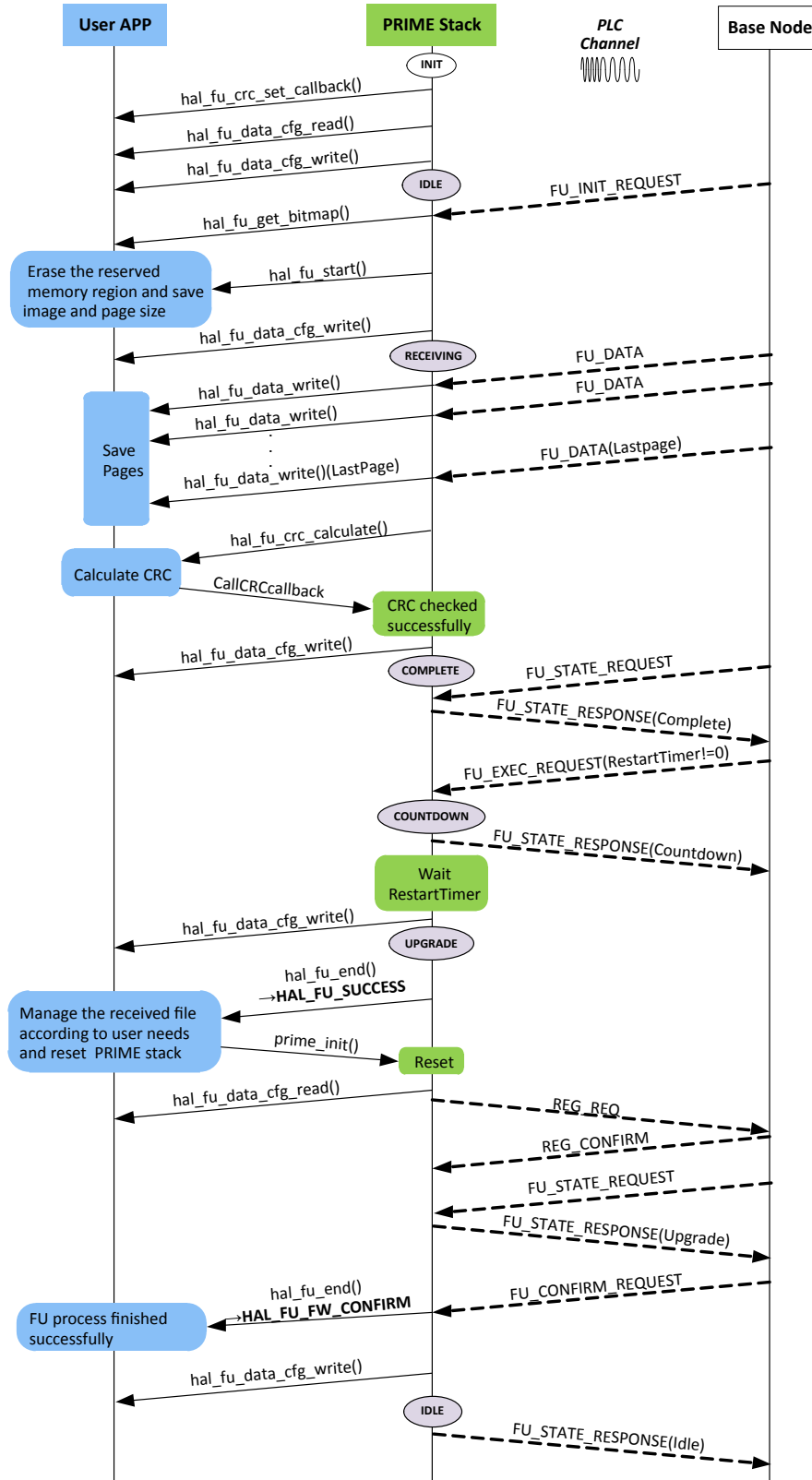
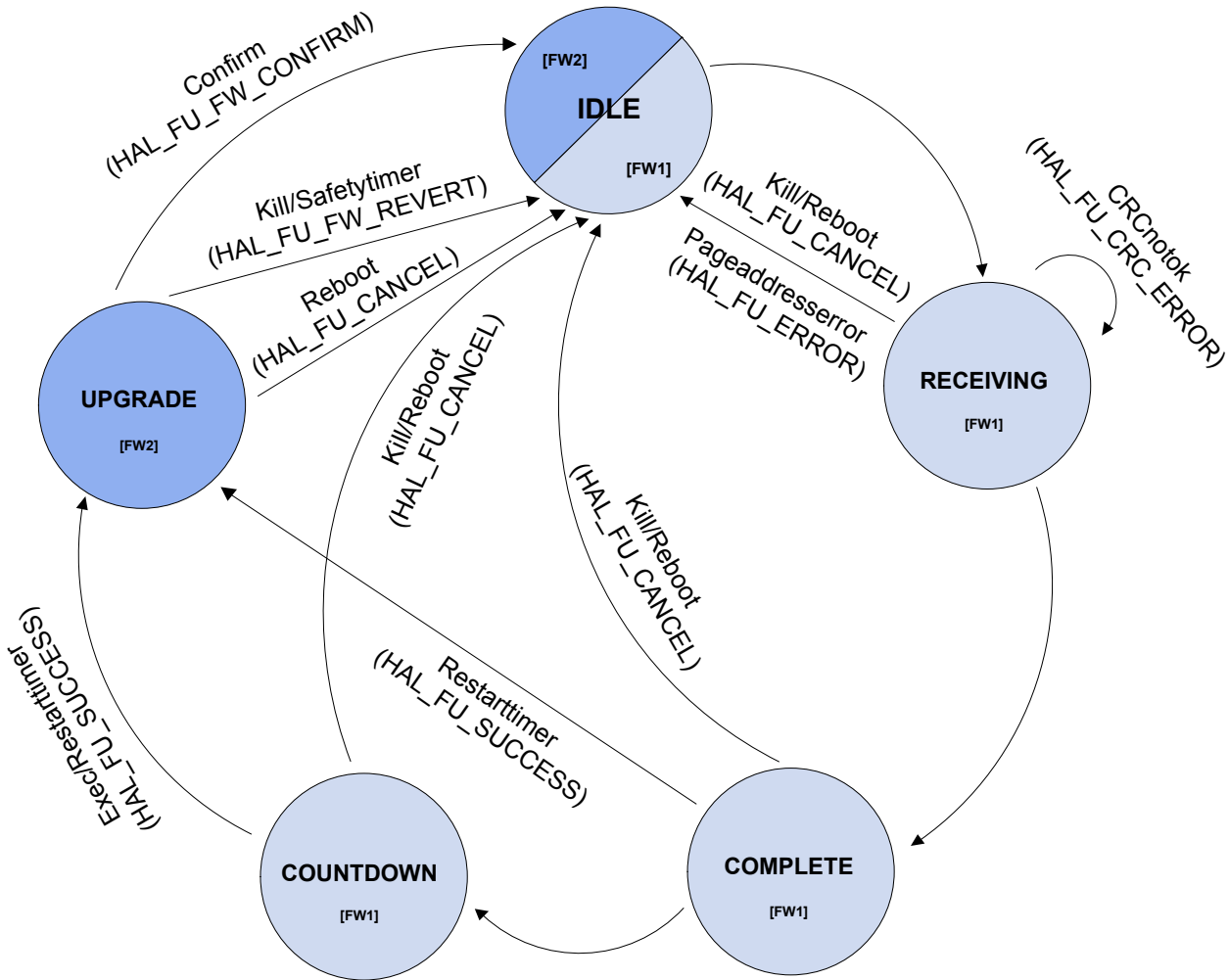


Figure 6-2. FU State Diagram with HAL FU Results



## 6.2 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

## 6.3 FU Configuration Data

The PRIME FW stack needs to save some configuration data that cannot be lost during the FU process. The provided HAL stores this data in four 32-bit General Purpose Backup Registers (GPBR0 to GPBR3) but users can determine to use a non-volatile storage area.

The PRIME FW stack requests to read the FU configuration data with the following function:

```
void hal_fu_data_cfg_read(void *pv_dst, uint16_t us_size);
```

Parameters:

- *pv\_dst*: Pointer where the read data must be stored



- *us\_size*: Length of data to be read

The PRIME FW stack requests to save the FU configuration data with the following function:

```
uint8_t hal_fu_data_cfg_write(void *pv_src, uint16_t us_size);
```

Parameters:

- *pv\_src*: Pointer to the data to be saved
- *us\_size*: Length of data to be saved

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

## 6.4 Starting FU

When a FU process starts, 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;    /* in bytes */
    uint8_t page_size;     /* in bytes */
} hal_fu_info_t;
```

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

## 6.5 Providing the Bitmap

The bitmap is used to keep track of received pages during a FU process. The PRIME FW stack does that internally, but the user application can do the same if desired. Then, when a new FU process starts or if there is a reset and the FU process restarts, the PRIME FW stack requests this information to use it if available. The function to request this from the HAL is the following:

```
uint16_t hal_fu_get_bitmap(uint8_t *puc_bitmap, uint32_t *pus_num_rcv_pages)
```

Parameters:

- *puc\_bitmap*: Pointer to the bitmap
- *pus\_num\_rcv\_pages*: Pointer to the number of received pages

Result: Bitmap size.

## 6.6 Writing FU Data

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

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

Parameters:

- *ul\_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.

## 6.7 CRC Calculation

When all the pages have been successfully 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 in chapter [CRC Result Callback](#).

## 6.8 Finishing FU

The PRIME FW stack can finish the FU process at any time, as shown in [Figure 6-2](#). This is indicated to the HAL with the following function:

```
void hal_fu_end(hal_fu_result_t uc_hal_res);
```

Parameters:

- *uc\_hal\_res*: Result of the FU process

The user application is responsible for storing this result at this point and later, during its process time, deciding what to do. The following results are possible:

```
typedef enum {
    HAL_FU_SUCCESS,           /* Request to restart with new image */
    HAL_FU_CANCEL,           /* The FU has been killed */
    HAL_FU_CRC_ERROR,        /* CRC error */
    HAL_FU_FW_ERROR,         /* (Deprecated) */
    HAL_FU_FW_REVERT,        /* Request to restart with old image */
    HAL_FU_FW_CONFIRM,       /* The FU has been confirmed */
    HAL_FU_ERROR             /* Error during FU */
} hal_fu_result_t;
```



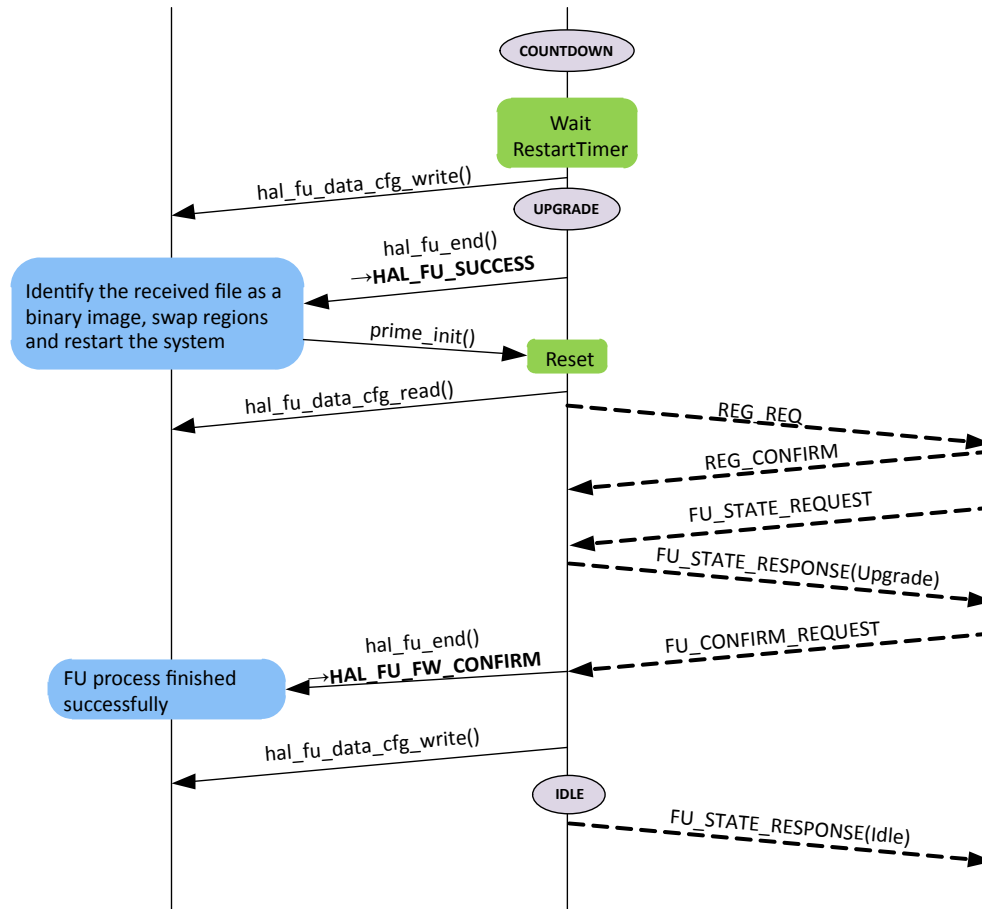
**Important:** Both results `HAL_FU_SUCCESS` and `HAL_FU_FW_REVERT` require a restart of the PRIME FW stack so that the PRIME FU process finishes properly. The rest of the results do not have any specific action associated.

## 6.9 Managing the Received File

Once the file has been successfully received, the user application can check what type of file it is. But in all cases the PRIME FW stack must always be restarted so that the FU process continues with confirmation and goes to idle again, as shown in [Figure 6-1](#) and [Figure 6-2](#).

In case of receiving a new image, the decision to swap regions in order to execute the new version (or to revert to the old one) is made in the user application. This triggers a software reset that restarts the whole system by executing the bootloader. The specific actions to complete a PRIME firmware upgrade are shown in the next figure.

Figure 6-3. Finishing a PRIME FU Process



**Important:** The system must not be restarted in the HAL function that receives the result of the FU process. The restart must take place in a user task.

## 7. 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.

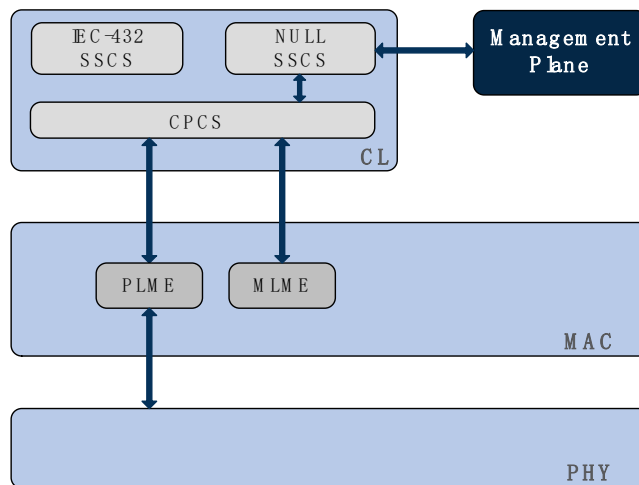
Node management in the PRIME FW stack is accomplished through a set of attributes. Attributes are defined for both PHY and MAC layers. The set of these management attributes is called PLC Information Base (PIB). Some attributes are read-only, while others are read-write. For details about implementation 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 control entity can access the Management Plane by using the two communication profiles defined in the PRIME specification:

- **PRIME Profile.** A remote control entity, usually the Base Node, uses the PLC to send requests to the node. These requests are received in the Management Plane from the Null SSCS in the CL (see chapter [Null SSCS](#))
- **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 7-1.** 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](#)).

## 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

As explained in chapter [General Architecture](#), for a system to be operative with the user application and the PRIME FW stack, two different projects must be built and loaded into the board as well as the bootloader. All provided projects can be built using the IAR Embedded Workbench directly.

For the project generating the binary file with the bootloader, follow the procedure as described:

1. Open the project *addons/bootloader*
2. Build the project
3. Program the board

For the project generating the binary file with the PRIME FW stack, follow the procedure as described:

1. Open the project *prime\_service\_bin*
2. Configure the library as required (see chapter [PRIME Stack User Configuration Parameters](#))
3. Build the project
4. Program the board so that the binary file is stored at the selected memory location (see chapter [Memory Allocation](#))

For each example application described in chapter [Example Applications](#), follow the procedure as described:

1. Open the corresponding project
2. Configure the project as required (see chapter [Application Configuration Parameters](#))
3. Build the project
4. Program the board so that the binary file is stored at the selected memory location (see chapter [Memory Allocation](#))

In the PL360/PL460 platform, the PL360 firmware is also required for a system to be operative but it is directly provided as a binary file. Therefore, it is only needed to store it in the board at the selected memory location (see chapter [Memory Allocation](#)).



**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 families are PIC32CXMTx, SAM4C and SAMG55 platforms.

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:

- With PL360: SAM4CMS16C, SAMG55J19
- With PL460: PIC32CX2051MTSH128, PIC32CX2051MTG128, SAMG55J19

### 9.3 Supported Boards

The currently supported boards and combinations are given below:

- ATPL360-EK
- PIC32CXMTSH-DB with PL460-EK on Xplained port
- PIC32CXMTG-EK with PL460-EK on Xplained port
- PL360G55CF-EK
- SAMG55 Xplained with PL460-EK on Xplained port

## 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 Service 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 <sup>1</sup>
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 Service 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 Service 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 Service 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.

#### 10.1.4.3 Firmware Upgrade

According to the PRIME specification, when the Base Node requests information about the missing pages during a FU process with packet FU\_MISS\_REQ, the Service Node can decide to reply with either FU\_MISS\_BITMAP or FU\_MISS\_LIST. The current implementation of PRIME specification version 1.3.6 usually replies with packet FU\_MISS\_BITMAP and in some cases with FU\_MISS\_LIST. This cannot be changed by users.



## 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 PAL SAP

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

#### 11.2.1 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.

#### 11.2.2 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.

#### 11.2.3 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;
```

#### 11.2.4 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  $\mu$ s based on PL360 timer reference
- *ul\_duration*: Duration time in  $\mu$ s

Return value: Size in bytes of noise capture.



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

## 11.3 PAL Primitives

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

### 11.3.1 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>

### 11.3.2 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_AGC.set <sup>1</sup>	<code>uint8_t pal_agc_set(uint8_t uc_mode, uint8_t uc_gain, uint16_t us_pch);</code>
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>
PHY_AGC.confirm	Referenced parameters of <i>pal_agc_get</i> .
PHY_TIMER.get	<code>uint8_t pal_timer_get(uint32_t *pul_timer, uint16_t us_pch);</code>
PHY_TIMER.confirm	Referenced parameters of <i>pal_timer_get</i> .
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>
PHY_CD.confirm	Referenced parameters of <i>pal_cd_get</i> .
PHY_NL.get	<code>uint8_t pal_nl_get(uint8_t *puc_noise, uint16_t us_pch);</code>
PHY_NL.confirm	Referenced parameters of <i>pal_nl_get</i> .
PHY_SNR.get	<code>uint8_t pal_snr_get(uint8_t *puc_snr, uint8_t uc_qt, uint16_t us_pch);</code>
PHY_SNR.confirm	Referenced parameters of <i>pal_snr_get</i> .
PHY_ZCT.get	<code>uint8_t pal_zct_get(uint32_t *pul_zct, uint16_t us_pch);</code>
PHY_ZCT.confirm	Referenced parameters of <i>pal_zct_get</i> .

**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	<code>prime_cl_null_establish_request(mac_establish_request_t)</code> <pre>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);</pre>
MAC_ESTABLISH.indication	<pre>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);</pre>
MAC_ESTABLISH.response	<code>prime_cl_null_establish_response(mac_establish_response_t)</code> <pre>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);</pre>
MAC_ESTABLISH.confirm	<pre>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);</pre>
MAC_RELEASE.request	<code>prime_cl_null_release_request(mac_release_request_t)</code> <pre>typedef void (*mac_release_request_t)(uint16_t us_con_handle);</pre>
MAC_RELEASE.indication	<pre>typedef void (*mac_release_ind_cb_t)(uint16_t us_con_handle, mac_release_indication_reason_t uc_reason);</pre>
MAC_RELEASE.response	<code>prime_cl_null_release_response(mac_release_response_t)</code> <pre>typedef void (*mac_release_response_t)(uint16_t us_con_handle, mac_release_response_answer_t uc_answer);</pre>
MAC_RELEASE.confirm	<pre>typedef void (*mac_release_cfm_cb_t)(uint16_t us_con_handle, mac_release_confirm_result_t uc_result);</pre>
MAC_JOIN.request	<code>prime_cl_null_join_request(mac_join_request_t)</code> <pre>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);</pre>
MAC_JOIN.indication	<pre>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);</pre>
MAC_JOIN.response	<code>prime_cl_null_join_response(mac_join_response_t)</code> <pre>typedef void (*mac_join_response_t)(uint16_t us_con_handle, uint8_t *puc_eui48, mac_join_response_answer_t uc_answer);</pre>
MAC_JOIN.confirm	<pre>typedef void (*mac_join_cfm_cb_t)(uint16_t us_con_handle, mac_join_confirm_result_t uc_result);</pre>
MAC_LEAVE.request	<code>prime_cl_null_leave_request(mac_leave_request_t)</code> <pre>typedef void (*mac_leave_request_t)(uint16_t us_con_handle, uint8_t *puc_eui48);</pre>

.....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 <sup>1</sup>	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 <sup>1</sup>	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 <sup>1</sup>	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 Service Node

MLME primitive	Function
MLME_REGISTER.request	prime_cl_null_mlme_register_request(mlme_register_request_t) typedef void (*mlme_register_request_t) (uint8_t *puc_sna, uint8_t uc_sid);
MLME_REGISTER.confirm	typedef void (*mlme_register_cfm_cb_t) (mlme_result_t x_result, uint8_t *puc_sna, uint8_t uc_sid);
MLME_REGISTER.indication	typedef void (*mlme_register_ind_cb_t) (uint8_t *puc_sna, uint8_t uc_sid);
MLME_UNREGISTER.request	prime_cl_null_mlme_unregister_request (mlme_unregister_request_t) typedef void (*mlme_unregister_request_t) (void);
MLME_UNREGISTER.confirm	typedef void (*mlme_unregister_cfm_cb_t) (mlme_result_t x_result);
MLME_UNREGISTER.indication	typedef void (*mlme_unregister_ind_cb_t) (void);
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_PROMOTE.indication	typedef void (*mlme_promote_ind_cb_t) (void);
MLME_DEMOTE.request	prime_cl_null_mlme_demote_request(mlme_demote_request_t) typedef void (*mlme_demote_request_t) (void);
MLME_DEMOTE.confirm	typedef void (*mlme_demote_cfm_cb_t) (mlme_result_t x_result);
MLME_DEMOTE.indication	typedef void (*mlme_demote_ind_cb_t) (void);
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.

If there are valid records in the list, the confirm primitive is invoked twice: the first time with a buffer containing the records and the second with a buffer length of zero to indicate that there are no more valid records.

If there are no valid records in the list, the confirm primitive is invoked with a buffer length of zero.

### 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_cb;
    mlme_register_cfm_cb_t mlme_register_cfm_cb;
    mlme_unregister_ind_cb_t mlme_unregister_ind_cb;
    mlme_unregister_cfm_cb_t mlme_unregister_cfm_cb;
    mlme_promote_ind_cb_t mlme_promote_ind_cb;
    mlme_promote_cfm_cb_t mlme_promote_cfm_cb;
    mlme_demote_ind_cb_t mlme_demote_ind_cb;
    mlme_demote_cfm_cb_t mlme_demote_cfm_cb;
    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;
```



**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 Service Node

IEC 61334-4-32 primitive	Function
CL_432_ESTABLISH.request	prime_cl_432_establish_request(cl_432_establish_request_t) typedef void (*cl_432_establish_request_t)(uint8_t *puc_device_id, uint8_t uc_device_id_len);
CL_432_ESTABLISH.confirm	typedef void (*cl_432_establish_cfm_cb_t)(uint8_t *puc_device_id, uint8_t uc_device_id_len, uint16_t us_dst_address, uint16_t us_base_address);
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_RELEASE.confirm	typedef void (*cl_432_release_cfm_cb_t)(uint16_t us_dst_address, dl_432_result_t uc_result);

.....continued

IEC 61334-4-32 primitive	Function
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_establish_cfm_cb_t cl_432_establish_cfm_cb;
    cl_432_release_cfm_cb_t cl_432_release_cfm_cb;
} cl_432_callbacks_t;
```



**Important:** Unused callback functions must be set to NULL.

## 12.3 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*.

### 12.3.1 Proprietary PIB Attributes in the PHY Layer

**Table 12-6.** 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.
PIB_MTP_PHY_TX_TIME	0x8085	4 bytes	-	Transmission time of the last frame transmitted in PLC in tenths of $\mu$ 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.

.....continued

Attribute	ID	Size	Range	Description
PIB_MTP_PHY_RX_PARAMS	0x8088	22 bytes	-	RX PLC PHY structure <sup>1</sup> : <ul style="list-style-type: none"> <li>RxParam[0]. Modulation scheme:               <ul style="list-style-type: none"> <li>DBPSK = 0</li> <li>DQPSK = 1</li> <li>D8PSK = 2</li> <li>DBPSK_CC = 4</li> <li>DQPSK_CC = 5</li> <li>D8PSK_CC = 6</li> <li>DBPSK_RB = 12</li> <li>DQPSK_RB = 13</li> </ul> </li> <li>RxParam[1]. RSSI.</li> <li>RxParam[2-3]. Reception data length in bytes.</li> <li>RxParam[4-5]. EvmHeader.</li> <li>RxParam[6-7]. EvmPayload.</li> <li>RxParam[8-11]. EvmHeaderAcum.</li> <li>RxParam[12-15]. EvmPayloadAcum.</li> <li>RxParam[16-19]. Reception time in microseconds.</li> <li>RxParam[20]. Noise.</li> <li>RxParam[21]. SNR in dB (0 - 21).</li> </ul>
PIB_MTP_PHY_TX_PARAMS	0x8089	7 bytes	-	TX PLC PHY structure <sup>1</sup> : <ul style="list-style-type: none"> <li>TxParam[0]. Modulation scheme:               <ul style="list-style-type: none"> <li>DBPSK = 0</li> <li>DQPSK = 1</li> <li>D8PSK = 2</li> <li>DBPSK_CC = 4</li> <li>DQPSK_CC = 5</li> <li>D8PSK_CC = 6</li> <li>DBPSK_RB = 12</li> <li>DQPSK_RB = 13</li> </ul> </li> <li>TxParam[1]. Attenuation level (0 - 21 dB). When set to 255, attenuation and gain are set to 0.</li> <li>TxParam[2-3]. Transmission data length in bytes.</li> <li>TxParam[4]. Inter-frame time in tenths of ms (0 - 2550 ms).</li> <li>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).</li> <li>TxParam[6]. Number of messages (1 - 0xFF).</li> </ul>
PIB_MTP_PHY_CONTINUOUS_TX	0x808A	1 byte	0 - 1	Set the PLC PHY layer to transmit continuously [0: disabled, 1: enabled].



.....continued

Attribute	ID	Size	Range	Description
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].
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*.

### 12.3.2 Proprietary PIB Attributes in the MAC Layer

**Table 12-7.** 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"> <li>• 0: SN disconnected</li> <li>• 1: SN detection</li> <li>• 2: SN registering</li> <li>• 3: SN operative</li> <li>• 4: BN</li> </ul>
PIB_MAC_SERVICE_STATE	0x8102	1 byte	-	Service Node state: <ul style="list-style-type: none"> <li>• 0: SN disconnected</li> <li>• 1: SN registering</li> <li>• 2: SN terminal</li> <li>• 3: SN unregistering</li> <li>• 4: SN promoting</li> <li>• 5: SN switch</li> <li>• 6: SN demoting</li> <li>• 7: SN roaming</li> </ul>
PIB_MAC_REG_RSS	0x8103	1 byte	0 - 255	Attenuation level to send REG and PNPDU messages.

.....continued

Attribute	ID	Size	Range	Description
PIB_CERTIFICATION_MODE	0x8120	1 byte	0 - 3	Certification mode: <ul style="list-style-type: none"> <li>0: no certification mode</li> <li>1: PHY certification for 1.3</li> <li>2: MAC certification</li> <li>3: PHY certification for 1.4</li> </ul>
PIB_CERTIFICATION_SEND_MSG	0x8121	Array	-	Send message in PHY certification mode. See <a href="#">Table 12-8</a> .
PIB_MAC_INTERNAL_SW_VERSION	0x8126	4 bytes	-	MAC internal software version.

**Table 12-8.** 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"> <li>1.3 frame</li> <li>1.4 type A frame</li> <li>1.4 type B frame</li> <li>1.4 type BC frame</li> </ul>

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

**Table 12-9.** Proprietary PIB Attributes in the IEC 61334-4-32 SCS Layer

Attribute	ID	Size	Range	Description
PIB_432_CON_STATE	0x8200	1 byte	0 - 3	State of IEC 61334-4-32 SCS layer: <ul style="list-style-type: none"> <li>0: closed</li> <li>1: connecting</li> <li>2: disconnecting</li> <li>3: open</li> </ul>
PIB_CL_INTERNAL_SW_VERSION	0x8201	4 bytes	-	Internal software version of the SCS layer.

## 13. Example Applications

### 13.1 Usage of the Firmware

The user application can use the complete PRIME FW stack for any PRIME related functionality even if network capabilities offered by any specific protocol of the CL are not required. In that case, the primitives to use are those of the Null SSCS (see chapter [Null SSCS](#)), which transparently match the MAC primitives.

When the user application is using the complete PRIME FW stack in the same device, the system structure is the same as depicted in [Figure 1-1](#).

The user application may also be running in a different device from the PRIME FW stack. In this case, the PRIME API is serialized using a UART or a USB port. The Modem example in chapter [PRIME Service Modem](#) shows an application with the serialized PRIME API.

The user application can also be dual, i.e., it can allocate different versions of the PRIME FW stack and then swap to any of them. The example applications provided by Microchip are dual, allocating PRIME FW stack versions 1.3 and 1.4., if such allocation is possible in the selected platform. For more information, see chapter [Memory Allocation](#).

It is also possible that an application only requires the ability of sending and receiving data through the PLC PHY layer without any MAC layer. Then, it is possible to remove the MAC layer and access the PLC PHY layer directly from the PAL. Microchip provides several examples of applications that only require the PRIME PLC PHY layer (see chapter [13.5. PHY Applications](#)).

### 13.2 PRIME Service DLMS + Metrology Application

The Service DLMS+Metrology Application is an application example that shows how the PRIME API should be used by integrating an application with a DLMS server. This application configures the board as a Service Node with DLMS capabilities and exchanges real data between the PRIME Base Node and the Service Node. DLMS is linked to the IEC 61334-4-32 SSCS and answers a minimum set of objects requested from a commercial PRIME Data Concentrator. Therefore, for this example, a PRIME Data Concentrator is required.



**Important:** Microchip does not provide a PRIME Data Concentrator. Note that the Microchip Gateway example can be connected to a PRIME Data Concentrator by means of TCP (DLMS over TCP). Note also that the Microchip PRIME Manager tool can act as a gateway placed between any Microchip BN and a PRIME Data Concentrator (see the User Manual of the PRIME Manager).

The following list provides the set of example OBIS implemented:

1-0:0.2.0.255 Class id 1	1-0:1.8.25.255 Class id 3
0-0:96.1.0.255 Class id 1	1-0:1.8.26.255 Class id 3
0-0:96.1.1.255 Class id 1	1-0:1.8.30.255 Class id 3
0-0:96.1.2.255 Class id 1	1-0:1.8.31.255 Class id 3
1-0:1.8.0.255 Class id 3	1-0:1.8.32.255 Class id 3
1-0:1.8.10.255 Class id 3	1-0:1.8.33.255 Class id 3
1-0:1.8.11.255 Class id 3	1-0:1.8.34.255 Class id 3
1-0:1.8.12.255 Class id 3	1-0:1.8.35.255 Class id 3
1-0:1.8.13.255 Class id 3	1-0:1.8.36.255 Class id 3
1-0:1.8.14.255 Class id 3	1-0:1.8.255.255 Class id 3
1-0:1.8.15.255 Class id 3	1-0:99.1.0.255 Class id 7
1-0:1.8.16.255 Class id 3	0-0:21.0.5.255 Class id 7
1-0:1.8.20.255 Class id 3	0-0:21.0.6.255 Class id 7
1-0:1.8.21.255 Class id 3	0-0:1.0.0.255 Class id 8
1-0:1.8.22.255 Class id 3	0-0:28.7.0.255 Class id 86
1-0:1.8.23.255 Class id 3	0-0:40.0.0.255 Class id 15
1-0:1.8.24.255 Class id 3	



**Important:** Note that this project example is configured to be used with a bootloader, thus allowing its upgrade using the file transfer over PRIME. It is also a dual mode application, but only supported in the SAM4C platform.

### 13.3 PRIME Service DLMS Application

The Service DLMS Application is an application example that shows how the PRIME API should be used by integrating an application with a DLMS server. This application configures the board as a Service Node with DLMS capabilities and simulates the data exchange between the PRIME Base Node and the Service Node. DLMS is linked to the IEC 61334-4-32 SSCS and answers a minimum set of objects requested from a commercial PRIME Data Concentrator. Therefore, for this example, a PRIME Data Concentrator is required.



**Important:** Microchip does not provide a PRIME Data Concentrator. Note that the Microchip Gateway example can be connected to a PRIME Data Concentrator by means of TCP (DLMS over TCP). Note also that the Microchip PRIME Manager tool can act as a gateway placed between any Microchip BN and a PRIME Data Concentrator (see the User Manual of the PRIME Manager).

The following list provides the set of example OBIS implemented:

1-0:0.2.0.255 Class id 1	1-0:1.8.25.255 Class id 3
0-0:96.1.0.255 Class id 1	1-0:1.8.26.255 Class id 3
0-0:96.1.1.255 Class id 1	1-0:1.8.30.255 Class id 3
0-0:96.1.2.255 Class id 1	1-0:1.8.31.255 Class id 3
1-0:1.8.0.255 Class id 3	1-0:1.8.32.255 Class id 3
1-0:1.8.10.255 Class id 3	1-0:1.8.33.255 Class id 3
1-0:1.8.11.255 Class id 3	1-0:1.8.34.255 Class id 3
1-0:1.8.12.255 Class id 3	1-0:1.8.35.255 Class id 3
1-0:1.8.13.255 Class id 3	1-0:1.8.36.255 Class id 3
1-0:1.8.14.255 Class id 3	1-0:1.8.255.255 Class id 3
1-0:1.8.15.255 Class id 3	1-0:99.1.0.255 Class id 7
1-0:1.8.16.255 Class id 3	0-0:21.0.5.255 Class id 7
1-0:1.8.20.255 Class id 3	0-0:21.0.6.255 Class id 7
1-0:1.8.21.255 Class id 3	0-0:1.0.0.255 Class id 8
1-0:1.8.22.255 Class id 3	0-0:28.7.0.255 Class id 86
1-0:1.8.23.255 Class id 3	0-0:40.0.0.255 Class id 15
1-0:1.8.24.255 Class id 3	

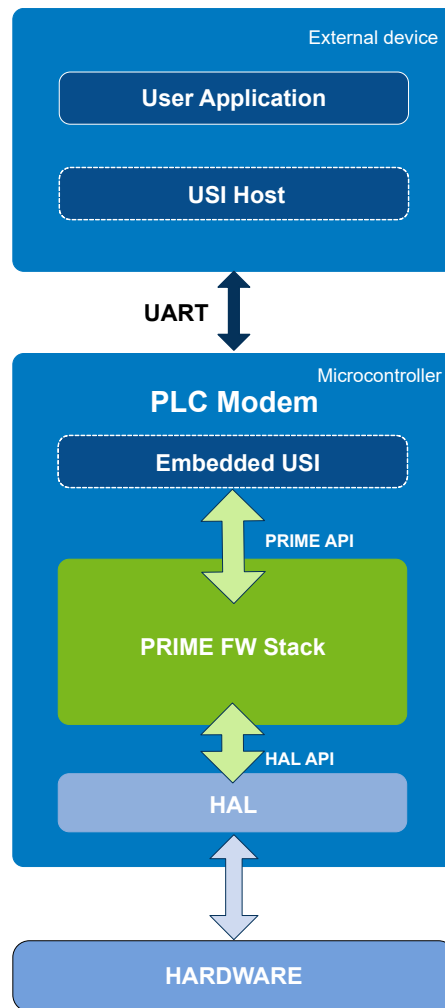


**Important:** Note that this project example is configured to be used with a bootloader, thus allowing its upgrade using the file transfer over PRIME. It is also a dual mode application, except in the SAMG55 platform.

## 13.4 PRIME Service Modem

The Service 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\_service.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 Application Note *PLC Universal Serial Interface*.



**Important:** Note that this project example is configured to be used with a bootloader, thus allowing its upgrade using the file transfer over PRIME. It is also a dual mode application, except in the SAMG55 platform.

## 13.5 PHY Applications

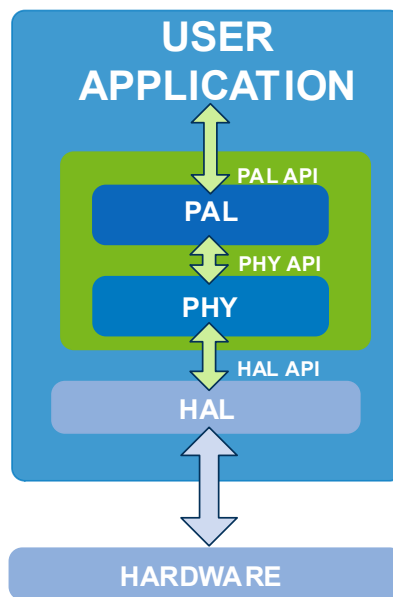
Microchip also provides several applications that demonstrate the complete performance of the PRIME PHY layer. Except for the PHY sniffer, such applications only offer the ability to send and

receive PLC data through the PLC PHY layer without any MAC layer as shown in [Figure 13-2](#). 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 Sniffer:** This PHY application allows monitoring of data traffic on a PRIME network by serializing PHY frames (see the User Manual of the *Hybrid Sniffer*).
- **PHY PLC&Go:** This PHY application example shows the basic code required to work with the PLC PHY layer.

**Figure 13-2.** 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

<b>7E</b> (1 byte)	<b>MSG LENGTH</b> (10 bits)	<b>PROTOCOL ID</b> (6 bits)	<b>MESSAGE DATA</b>	<b>CRC</b> (variable)	<b>7E</b> (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 <sup>1</sup>	0x00 - 0x07	32
PROTOCOL_SNIF_PRIME	0x13	16
PROTOCOL_PHY_TESTER <sup>2</sup>	0x22	16
PROTOCOL_PRIME_API	0x30	8
PROTOCOL_INTERNAL	0x3F	16
PROTOCOL_USER_DEFINED <sup>3</sup>	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
- 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

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

## 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 <sup>1</sup>	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. <sup>2</sup>
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. <sup>2</sup>

**Notes:**

1. Only UART, USART and USB are currently supported.
2. 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.



**Important:** In the SAM4C microcontroller family, the UART1 has no access to the DMA so it cannot be chosen for the USI operation.

### 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
```

## 15. 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
FW	Firmware
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
IEC	International Electrotechnical Committee
IP	Internet Protocol
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
OBIS	Object Identification System
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
PRIME	Powerline Intelligent Metering Evolution
RB	Robust
RMS	Root Mean Square
RSSI	Received Signal Strength Indicator
SAP	Service Access Point
SN	Service Node
SNA	Sub-Network Address

SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
SSCS	Service Specific Convergence Sublayer
TC	Timer/Counter
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USI	Universal Serial Interface

## 16. References

[Microchip Smart Energy](#)

[Microchip Power Line Communications](#)

[Microchip Design Support](#)

[PRIME Alliance](#)

[PL360 Evaluation Kit](#)

[PL460 Evaluation Kit](#)

[PIC32CXMTSH Demo Board](#)

[PIC32CXMTG Evaluation Kit](#)

[SAMG55 Xplained PRO Evaluation Kit](#)

[Documents for supported families and boards](#)

## 17. Revision History

### 17.1 Rev A - 07/2018

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

### 17.2 Rev B - 12/2018

<a href="#">HAL API</a> and <a href="#">PRIME Management Plane</a>	Added user specific PIBs.
<a href="#">Main Function Structure</a>	Improved hardware initialization.
<a href="#">HAL API</a>	Removed non-applicable function. Changed structure for notes.

### 17.3 Rev C - 07/2020

Document	Editorials.
<a href="#">Proprietary PIB Attributes in the PHY Layer</a>	Added new proprietary PIBs and updated MTP PIB.
Get Function and Set Function	Aligned function parameters with code.
<a href="#">Proprietary PIB Attributes in the PHY Layer</a> and <a href="#">PRIME Sniffer Frame Format</a>	Added clarifications about time units.
<a href="#">Default Memory Allocation Example</a>	Added warning about default values in Atmel Studio.
<a href="#">Other Coding Requirements</a>	Added effect on increased call stack size. Removed optimizations in Atmel Studio projects. Added clarifications about interrupt handling.
Brief about ASF, Using Atmel Studio, <a href="#">Usage of the Firmware</a> , <a href="#">PRIME Service DLMS + Metrology Application (PL360 Platform)</a> , <a href="#">PRIME Service DLMS Application</a> and <a href="#">PRIME Service Modem</a>	Added that Atmel Studio projects are only for evaluation purposes and cannot be upgraded.
<a href="#">Firmware Version Information</a>	Updated vendor and version.
<a href="#">Communication Channel and other PHY Parameters</a>	Added ATPLCOUP011. Corrected configuration key.
<a href="#">Hardware Resources Usage</a>	Added remark about platform for mandatory timers.
<a href="#">Automatic Repeat Request (ARQ)</a>	Replaced frame by fragment.
<a href="#">General Prerequisites</a>	Updated IAR version.
<a href="#">Definition and Configuration of USI Ports</a>	Added USB type as USI port.
<a href="#">PLC Coupling (PL360 Platform)</a>	Added coupling configuration in PL360 platform.
<a href="#">FU Configuration Data</a>	Added more information about the usage of GPBR during FU.
<a href="#">Definition and Configuration of USI Ports, Linking of Serial Communications Profile of PRIME Management Plane, Linking of PRIME Sniffer (PL360 Platform) and Linking of PRIME Sniffer (ATPL230A Platform)</a>	Clarified usage of USI ports.
<a href="#">PHY Applications</a>	Added another PHY example.
<a href="#">PRIME Sniffer Frame Format</a>	Added sniffer type version for PL360.
<a href="#">Features, Supported Devices</a> and <a href="#">Supported Boards</a>	Added new devices and boards.
<a href="#">PRIME Management Plan</a>	Corrected name of MNG protocol.
<a href="#">USI Host Example</a>	Added meaning of abbreviation.
<a href="#">References</a>	Updated links.
<a href="#">Usage of the Firmware</a>	Added USB as possible port for serialization.



## 17.4 Revision D - 02/2024

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.
<a href="#">PRIME FW Stack, Event-Driven Operation</a>	Updated description.
<a href="#">PRIME FW Stack Project Example</a>	Updated paths and images.
<a href="#">User Application Project Examples</a>	Updated paths and images. Added information about dual mode applications.
<a href="#">Bootloader Example</a>	Added that PL360 firmware can also be upgraded.
<a href="#">Introduction</a>	Removed references to old platforms. Corrected PIB name.
<a href="#">Firmware Version Information</a>	Updated model and version.
<a href="#">Communication Channel and other PHY Parameters</a>	Added PLC couplings for PL460. Corrected configuration key. Removed thresholds.
<a href="#">PLC Coupling (PL460 platform), PRIME FW Stack Swap</a>	New chapters.
<a href="#">Band Plan in PLC</a>	Renamed chapter. Forced channel 1 in band plan.
<a href="#">Enabling PLC PHY Layer, Sniffer Serialization</a>	Renamed chapter. Updated setting.
<a href="#">PL360 Firmware Information (PL360/PL460 Platform)</a>	Renamed chapter. Added link to memory allocation.
<a href="#">Memory Allocation</a>	Updated image.
<a href="#">Default Memory Allocation Example</a>	Updates sizes, locations and images due to dual mode.
<a href="#">Hardware Resources Usage</a>	Updated mandatory timers and SPI.
<a href="#">HAL API</a>	Updated HAL functions.
<a href="#">4.4.1. Task Manager, Priorities and Preemption</a>	Updated timing.
<a href="#">Other Coding Requirements</a>	Removed priority setting for TC0. Added requirement when swapping stacks.
<a href="#">Main Function Structure</a>	Added swap function.
<a href="#">General Prerequisites</a>	Updated IAR version. Removed Atmel Studio.
<a href="#">Supported MCU Families, Supported Devices and Supported boards</a>	Updated devices and boards.
<a href="#">PHY Frames, Major Capabilities of the MAC Layer</a>	Removed date of specification.
<a href="#">PLC PHY SAP, Linking of PRIME Sniffer</a>	Renamed chapters.
<a href="#">Callback Functions</a>	Updated callback functions.
<a href="#">Data Primitives</a>	Updated request function input and output parameters.
<a href="#">Management Primitives</a>	Added PCH to primitives.
<a href="#">API of PRIME FW Stack</a>	Added comment about parameters from other specification versions.
<a href="#">Proprietary PIB Attributes in the PHY Layer</a>	Removed obsolete PIBs. Updated MTP PIBs. Added new PIBs.
<a href="#">Proprietary PIB Attributes in the MAC Layer</a>	Removed PIB to write SNA.
<a href="#">Proprietary PIB Attributes in the IEC 61334-4-32 SCS Layer</a>	Corrected PIB name.
<a href="#">Usage of the Firmware, PRIME Service DLMS + Metrology Application, PRIME Service DLMS Application, PRIME Service Modem</a>	Added comment about dual mode applications.
<a href="#">PLC PHY Applications</a>	Renamed chapter. Updated name of last application.
<a href="#">USI Frame Format</a>	Corrected name of protocol and added missing one.
<a href="#">Abbreviations</a>	Updated abbreviations.

<a href="#">References</a>	Updated links.
----------------------------	----------------

## Microchip Information

### The Microchip Website

Microchip provides online support via our website at [www.microchip.com/](http://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](http://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](http://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](http://www.microchip.com/en-us/support/design-help/client-support-services).

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

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

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

## Trademarks

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

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

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

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

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

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

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

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

ISBN: 978-1-6683-3998-5

## **Quality Management System**

For information regarding Microchip's Quality Management Systems, please visit [www.microchip.com/quality](http://www.microchip.com/quality).

## Worldwide Sales and Service

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