G3-PLC Firmware Stack including Hybrid Profile

Introduction

This document is the user guide for the Microchip implementation of the G3-PLC firmware stack.

The mechanisms and functionality defined in the G3-PLC specification are the basis of the entire firmware stack implementation. Therefore, it is highly recommended to use it as a reference. Basic concepts that are introduced by this G3-PLC specification are assumed to be known within this document.

In addition, the Microchip G3-PLC stack includes the G3 Hybrid Profile implementation, which is based on the Hybrid G3-PLC and RF Profile Annex to the G3-PLC specification.

Table of Contents

1. General Architecture

The provided G3-PLC Firmware Stack follows a layered approach based on the G3-PLC specification.

The stack is a set of libraries and source code blocks implementing the different parts of G3. Blocks are separated by "wrappers", which provide a clean separation between modules, a way to change stack table sizes, and even allow replacing a module with a different implementation of it.

The following figures show the stack architecture and its variants depending on the platform used:

Figure 1-1. Block Diagram of Microchip G3-PLC Stack (ATPL250)

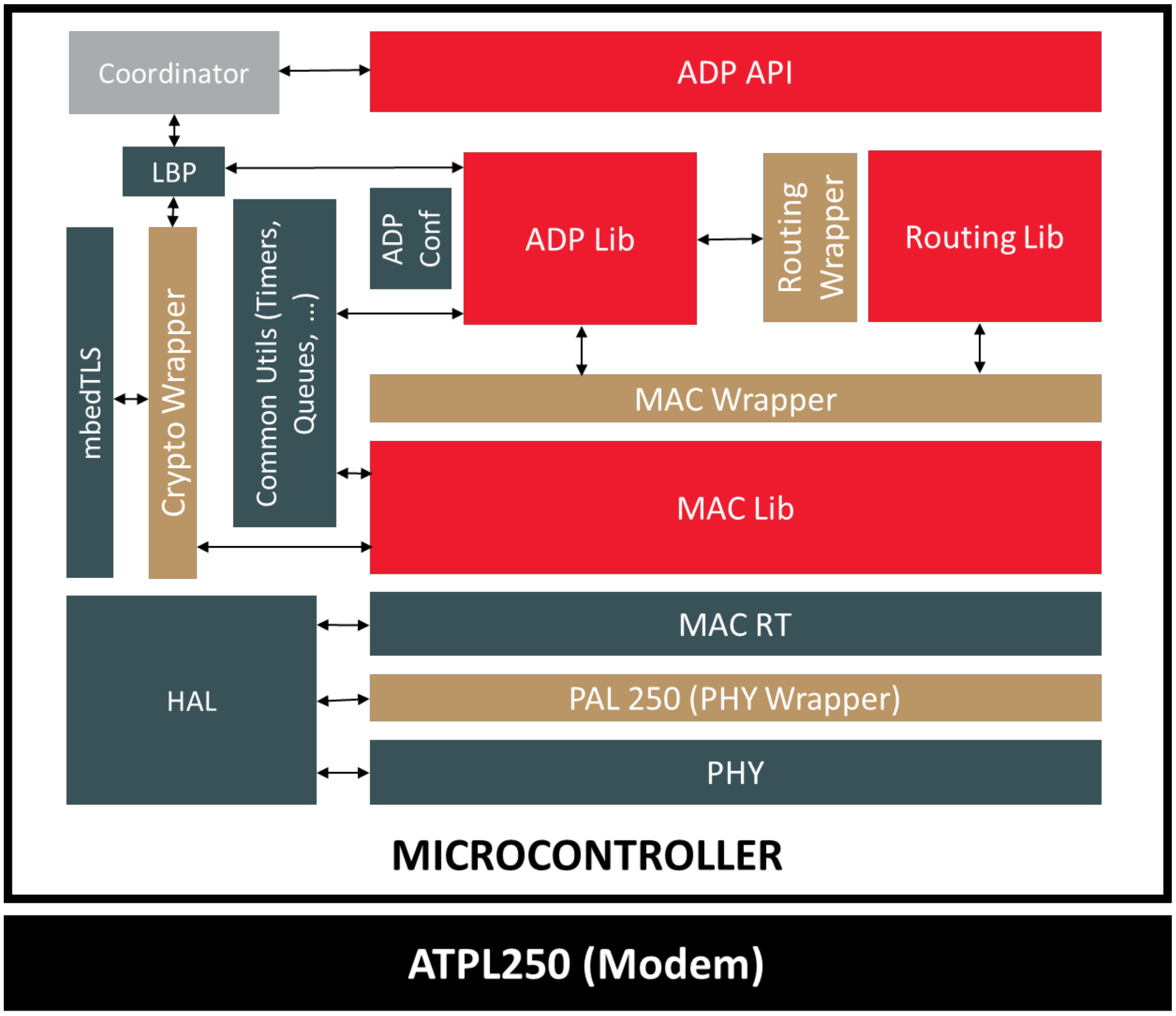


Figure 1-2. Block Diagram of Microchip G3-PLC Stack (PL360 MAC RT)

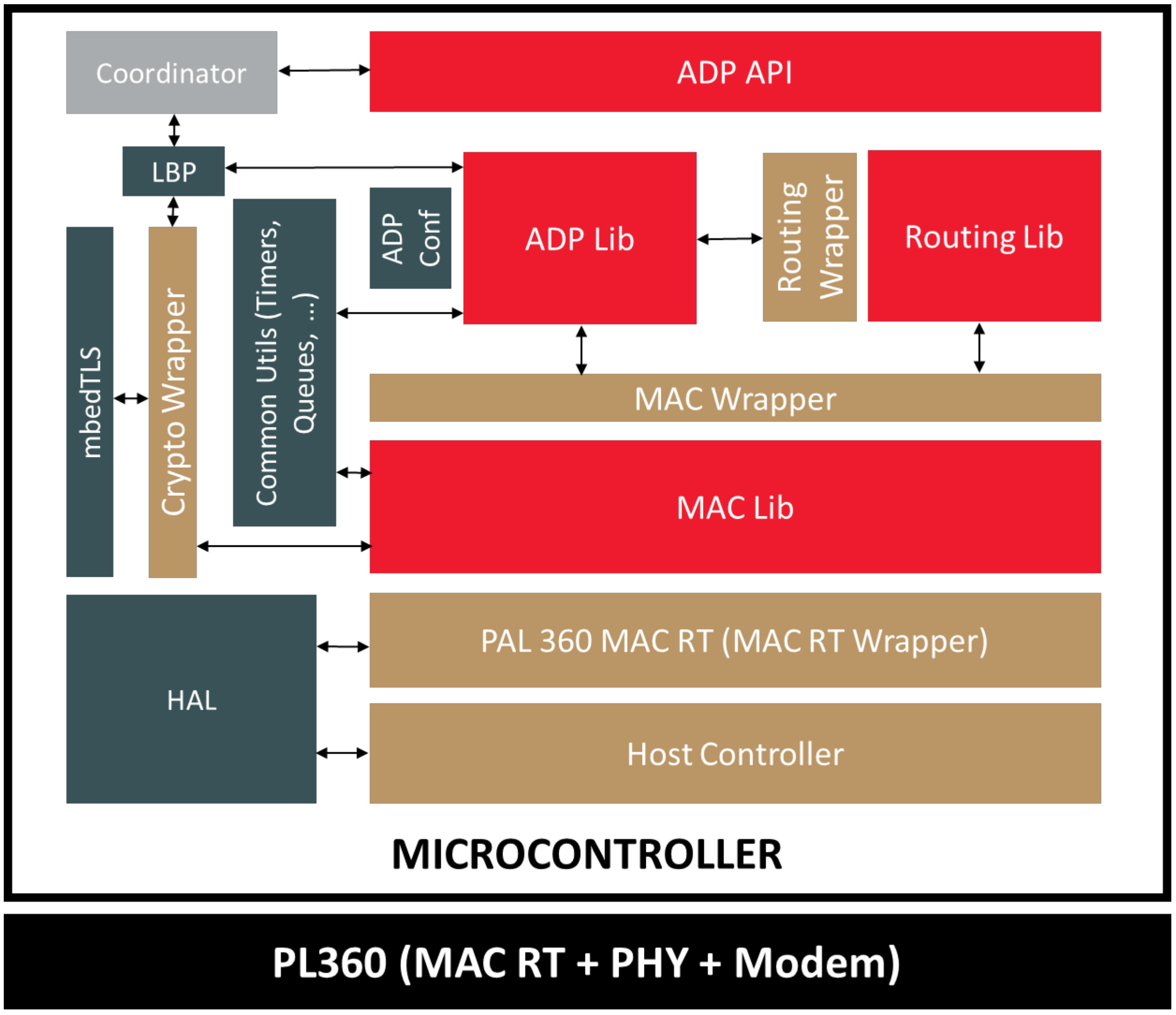
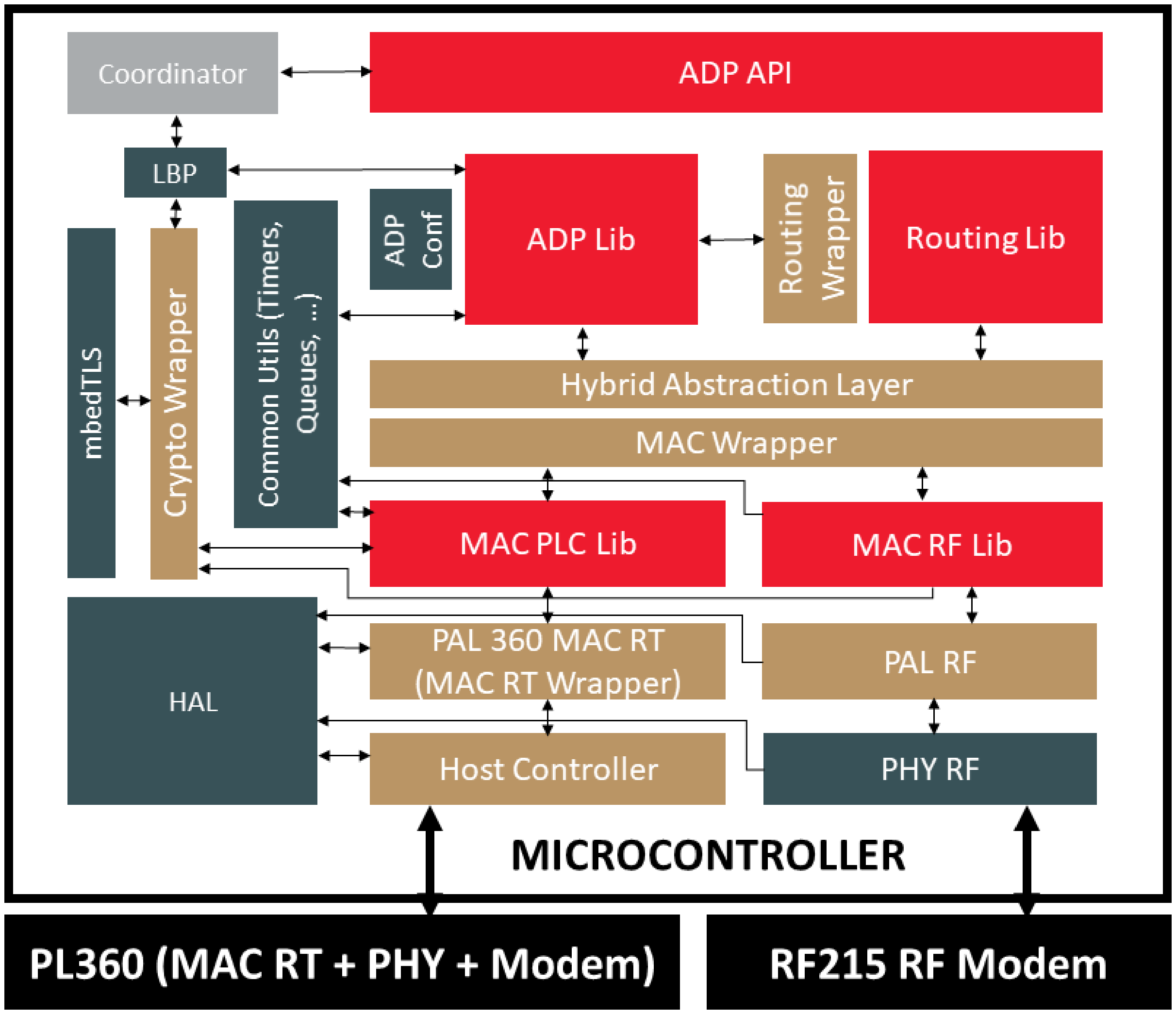


Figure 1-3. Block Diagram of Microchip G3-Hybrid PLC&RF Stack



1.1 PLC PHY Layer

The PLC PHY layer is in charge of frame transmission and reception. This layer is mainly interrupt-driven with events coming from the PLC modem.

In ATPL250 architecture, the PHY layer runs in the host microcontroller. The ATPL250 device generates interrupt events while transmitting and receiving frames. There are two special events that trigger data indication and data confirm primitives to be handled by upper layers: a frame reception and the end of a frame transmission.

In the case of PL360 architecture, PHY layer firmware is managed by the PL360 device; thus, data indication and data confirm events are managed by the Host Controller module in case of PHY layer applications, or internally for G3 stack applications, where 1.6. PLC MAC RT (MAC Real Time) runs inside the PL360.

Apart from these events, the PHY layer API (either in the host microcontroller or through the Host Controller module) implements entry functions to transmit a frame using the PLC modem, to perform periodic tasks and to access the PHY Information Base (PIB) to read, write or modify parameters.

PL360 is a Flashless device, so its firmware has to be stored in the host microcontroller memory and downloaded at start-up. In the project examples provided, it is the linker and compiler that are configured to perform the allocation automatically. For further details, refer to the *PL360 Host Controller* document provided by Microchip.

1.2 RF PHY Layer

The RF PHY layer is in charge of frame transmission and reception over the air. This layer is mainly interrupt-driven with events coming from the RF transceiver.

The supported RF transceiver is AT86RF215, the PHY layer runs in the host microcontroller. The AT86RF215 device generates interrupt events while transmitting and receiving frames. There are two special events that trigger data indication and data confirm primitives to be handled by upper layers: a frame reception and the end of a frame transmission.

The PHY layer API implements entry functions to transmit a frame using the RF transceiver, request channel sensing, perform periodic tasks and to access the PHY Information Base (PIB) to read or write parameters.

1.3 PLC Host Controller

This block is only present in the PL360 platform.

It is in charge of controlling and monitoring all PL360 activities, as well as loading the binary at start-up and managing any unexpected behavior.

It is a complex module with its own documentation. Refer to the *PL360 Host Controller* document provided by Microchip.

1.4 PLC PAL (PHY Wrapper)

This layer abstracts the interface of the 1.1. PLC PHY Layer developed by Microchip and provides an interface compliant with the G3-PLC PHY layer. For more details, refer to the G3 PLC Spec [www.g3-plc.com/]. In other words, this layer acts as the interface between the currently developed PHY and MAC layers, and is in charge of communicating both layers properly.

It implements primitives to inform the MAC layer of events coming from the PLC, namely PhyDataConfirm, PhyAckConfirm, PhyDataIndication and PhyAckIndication.

Apart from these tasks, PHY API will directly address function calls from MAC to transmit a frame, PhyDataRequest and PhyAckRequest to PHY layer. The same will occur with MAC calls to PLME functions (PhyPlmeSetRequest, PhyPlmeSetConfirm, PhyPlmeGetRequest, PhyPlmeGetConfirm, PhyPlmeSetTrxStateRequest, PhyPlmeSetTrxStateConfirm, PhyPlmeCsRequest, PhyPlmeCsConfirm, PhyPlmeResetRequest), which will be directly addressed to PHY functions related to PIB access.

Please note that the semantics of the PHY primitives included in this firmware stack cover the requirements of the G3-PLC PHY specs and provide more information:

- PhyInitialize has to be called to initialize the PHY layer
- PhyEventHandler has to be called to perform periodic tasks
- PhyGetTime retrieves the value of the internal PHY timer used to control RX and TX times
- PhyGetToneMapResponseData is used by the MAC layer to retrieve information from the PHY layer to fill a Tone Map Response frame
- PhySetToneMask sets the Tone Mask used by the Phy layer
- PhyGetLegacyMode retrieves whether PHY layer is running in legacy mode or not
- PhyGetParam and PhySetParam are used to access PIB objects not present in the standard PhyPlmeGetRequest and PhyPlmeSetRequest primitives

For further details, refer to the *pal.h* header file.

In case of the PL360 platform, this interface is internal to the PL360 device, as a link between the host and the PL360 is managed at the MacRT level. See 1.7. PLC PAL 360 MAC RT (PLC MAC RT Wrapper) for further information.

1.5 RF PAL (PHY Wrapper)

This layer abstracts the interface of the 1.2. RF PHY Layer developed by Microchip and provides an interface similar to the one specified for the G3-PLC PHY layer. This layer acts as the interface between the currently developed PHY and MAC layers, and is in charge of communicating both layers properly.

It implements primitives to inform the RF MAC layer of events coming from the RF, namely PalRFDataConfirm and PalRFDataIndication.

Apart from these tasks, RF PAL will directly address function calls from MAC to transmit a frame (PalRfTxRequest) or cancel a programmed transmission (PalRfTxCancel) to the PHY layer.

Other RF PAL API functions include:

- PalRfInitialize initializes the RF PHY layer
- PalRfEventHandler calls the RF PHY Event handler, which checks for events to be notified to the upper layers
- PalRfGetPhyTime retrieves the value of the internal RF PHY timer used to control the RX and TX times
- PalRfResetRequest resets the RF PAL and PHY layers

Important: The PalRfInitialize function is where the PHY layer is initialized according to chosen modulation (FSK or OFDM), frequency band, channel number, operation mode and so on. Refer to the files: *ieee_15_4_sun_fsk.h* and *ieee_15_4_sun_ofdm.h*, located under the */common/components/rf/at86rf215/* folder to check for different configuration modes available.

For further details, refer to the *pal_rf.h* header file.

1.6 PLC MAC RT (MAC Real Time)

This block is the lowest part of the G3 PLC MAC layer and, as its name indicates, contains all the tasks that require a low latency attention.

It is provided as source code for the ATPL250 platform, out of the G3 MAC library.

In the PL360 platform, it is integrated in the binary. This significantly relaxes the time requirements on the Host MCU because, as stated above, all the low latency requirements are in this layer and, thus, are managed by the PL360 device.

The functionalities included in this block are:

- CRC check
- ACK generation if requested
- CSMA
- ARQ
- Segmentation and reassembly

1.7 PLC PAL 360 MAC RT (PLC MAC RT Wrapper)

This module is the equivalent to the 1.4. PLC PAL (PHY Wrapper) layer when the 1.6. PLC MAC RT (MAC Real Time) runs inside the PL360 device.

It is a wrapper to the PLC MAC RT, which offers the same API as that module, but makes use of the 1.3. PLC Host Controller to take control of the MAC RT running inside PL360.

1.8 MAC Layer and MAC Wrapper

The PLC MAC layer is provided by Microchip as a compiled library.

Taking into account that PLC MAC RT is already a separated module, the PLC MAC layer has the following functions:

- Network scan. Generation and processing of BCN and BREQ frames
- Transmission robustness management. Generation and processing of TMR frames
- Security. Frame Encryption and Frame Decryption/Validation
- MAC IB (Information Base) management

The RF MAC layer is provided by Microchip as a compiled library. This layer has the following functions:

- Channel access arbitration
- Network scan. Generation and processing of BCN and BREQ frames
- Frame Acknowledgment and re-transmission management
- Security. Frame Encryption and Frame Decryption/Validation
- POS (Personal Operating Space) management by means of Information Elements
- MAC IB (Information Base) management

The MAC Wrapper is a source code block that acts as an interface with both the PLC and RF MAC layers. The reasons for having such a wrapper include:

- Its interface offers the same functionality as PLC and RF MAC APIs, so upper layers must access any of the MAC layers through the Wrapper, removing any reference to the MAC header files and, thus, isolating the MAC libraries from the upper layers.
- Table sizes and G3 Specification compliance are defined in this wrapper and are later initialized at run time in the MAC libraries. This allows unique PLC and RF MAC libraries for any Device type and target Specification compliance.

The MAC Wrapper interface is defined in 9.6. MAC Wrapper.

1.9 HyAL (Hybrid Abstraction Layer)

The Hybrid Abstraction Layer is a source code block that acts as interface between the Adaptation Layer (1.10. ADP Layer, ADP Conf and ADP API) and both the PLC and RF MAC layers, through the MAC Wrapper module.

The role of this layer is to act as a common input/output area for both the PLC and RF MAC Layers, making the Hybrid Profile as transparent as possible to the ADP and the upper layers.

It manages the Requests coming from ADP, Routing and LBP modules, and forwards them to one or both of the MAC layers. In the opposite direction, manages the Confirm and Indication primitives coming from any of the MAC layers and forwards them to the ADP, Routing or LBP module, indicating the MAC layer(s) that generated the event.

The HyAL interface and detailed functionality is defined in 9.8. Hybrid Abstraction Layer.

1.10 ADP Layer, ADP Conf and ADP API

The Adaptation Sublayer (ADP) is provided by Microchip as a compiled library. It has the following functions:

- Handles the device authentication and encryption key distribution
- IPv6 headers compression / decompression
- Fragmentation / reassembly of the IPv6 packets
- Controlling broadcast / multicast propagation
- Routing a message into the network, with the help of the Routing Module
- ADP IB (Information Base) management

The ADP Conf is a source code module where some Tables and Data Buffers are declared and are later referenced at run time in the ADP library. This allows a unique ADP library for any device type, by changing the size and number of such Tables and Buffers.

The ADP API is the entry point for upper layers to the G3 stack, and, thus, any access is done through this API and no other module is directly accessed.

The ADP API interface is defined in 9.9. ADP SAP, 9.10. ADP Data Primitives, 9.11. ADP Management Service Primitives and 9.12. ADPIB Objects Specification and Access.

1.11 Routing Module and Routing Wrapper

The Routing Module is provided by Microchip as a compiled library. It implements the LOADng protocol, as specified in Annex D of the G3 specification.

The purpose of this module is the discovery and maintenance of the routes into the network.

The Routing Wrapper is a source code block that acts as an interface with the Routing Module. The reasons for having such a wrapper include:

- Its interface gives access to all Routing functionality required by other modules, so these modules access the Routing Module through the Wrapper, removing any reference to the Module itself and, thus, isolating the Routing library from other modules.
- Table sizes and the G3 Specification compliance are defined in this wrapper and are later initialized at run time in the Routing library. This allows a unique Routing library for any Device type and target Specification compliance.
- In a solution where Routing is not used, the Routing library will not be included in the final system. The wrapper provides the proper interface and return values in case the Routing library is not present.

1.12 Common Utils Module

This Module contains shared resources such as Timers, Queue Management, Type Conversion, Random Number Generation and Logging capabilities.

It is used by the ADP, MAC and Routing libraries, as well as by their auxiliary modules. It is provided as source code.

1.13 Crypto Module and Crypto Wrapper

The Crypto Wrapper contains the necessary API to encrypt/decrypt at the ADP and MAC layers, independently from the specific implementation below.

Provided examples make use of mbedTLS, an ARM’s standard crypto library implementation. This library is provided as a reference, but can be changed by another crypto engine with the proper Crypto Wrapper adaptation.

The provided mbedTLS was adapted to use the HW capabilities of the microcontroller, if available.

1.14 HAL (Hardware Abstraction Layer)

Besides the G3-PLC stack, Microchip provides an open source module for the Hardware Abstraction Layer (HAL). This module provides function API-based service to the higher-level layers that allows them to perform hardware-oriented operations independent of actual hardware details.

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

There are different groups of functions, e.g., to initialize HW, write parameters in non-volatile memory, Power-Down detection, timer interrupt management, etc.

1.15 Coordinator Module

The Coordinator module is part of a G3 PAN Coordinator, and, thus, is not present in the G3 PAN Device examples.

This module is in charge of EAP-PSK on the Coordinator side, and, thus, decides whether a Device is accepted in the network or not.

Note: The provided module is a basic example, not intended to be a complete module in a real Coordinator. This module is out of the G3 spec, and it is provided for testing purposes.

1.16 LBP Module

This module is in charge of encoding / decoding the Local Bootstrap Protocol frames.

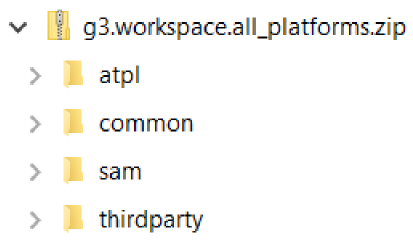
It is shared by both the Coordinator and the Device, thus, has an interaction with both the Coordinator Module and the ADP Library.

2. Understanding the Firmware Package

The Microchip G3-PLC firmware package is provided as a single zip file containing all the code and libraries needed to run G3 example applications on supported boards. This file is named after “G3 Workspace”. The package contains G3-specific code along with common ASF (Advanced Software Framework) modules and drivers that are required for this G3-PLC firmware to work. This chapter provides some general guidelines on how the different software modules are structured.

2.1 G3-PLC Workspace Structure

Figure 2-1. G3 Workspace



The workspace package is divided into four folders: *atpl*, *common*, *sam* and *thirdparty*.

This chapter locates and briefly explains G3-specific code inside the workspace. General purpose modules used by G3 are provided in the workspace but not highlighted here.

The *atpl* folder contains binaries for the PL360 device.

Common and *sam* contain generic drivers and components for Microchip microcontrollers, with special mention to the *common/components/plc* folder, where the PL360 Host Controller is found. This component is responsible for communication between the host microcontroller and the PL360 device. The *common/components/rf* folder contains the RF PHY layer and RF215 Controller. The *sam* folder contains a subfolder *sam\applications\serial_bootloader* where a Bootloader can be found, in project and binary versions, to be used in the PL360G55CB and PL360G55CF boards.

The *thirdparty* folder contains the *CMSIS* folder, where the ARM Cortex interface is located; *dlms*, where a light version of a DLMS client and a DLMS server are provided; the *freertos* folder, where the porting of FreeRTOS to Microchip microcontrollers can be found; the *cycloneTCP* folder, where an evaluation implementation of the IPv6 stack is provided; and the *g3* folder, where all the G3 stack and example applications code are located.

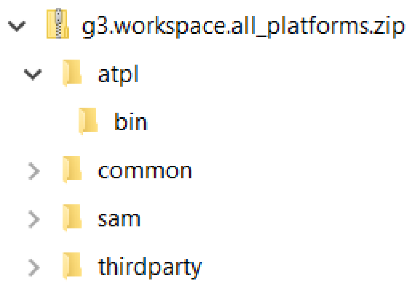
Under the *g3* folder, there is one folder for each layer of the G3 stack, plus, example applications (apart from some auxiliary folders).

The *apps* folder contains example applications involving the whole stack for every available platform. Besides, under *phy/atpl250* and *phy/atpl360*, specific PLC PHY layer applications can be found for both devices. For ATPL250, the source code of the PHY layer can also be found in this folder, as it runs in the host microcontroller.

Any of these *apps* folders contain the projects for the supported compilers, so the user can open, inspect and compile them.

2.1.1 atpl folder

Figure 2-2. atpl/bin folder

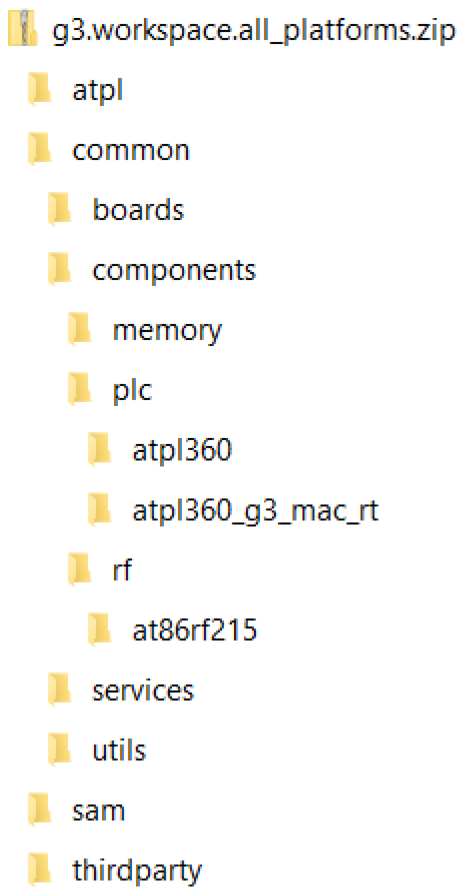


The *atpl* folder contains only one element, the *bin* sub-folder.

This folder contains the PL360 binary files. A tool for binary encryption is also provided and located in this folder.

2.1.2 common folder

Figure 2-3. common/components/plc & rf folders



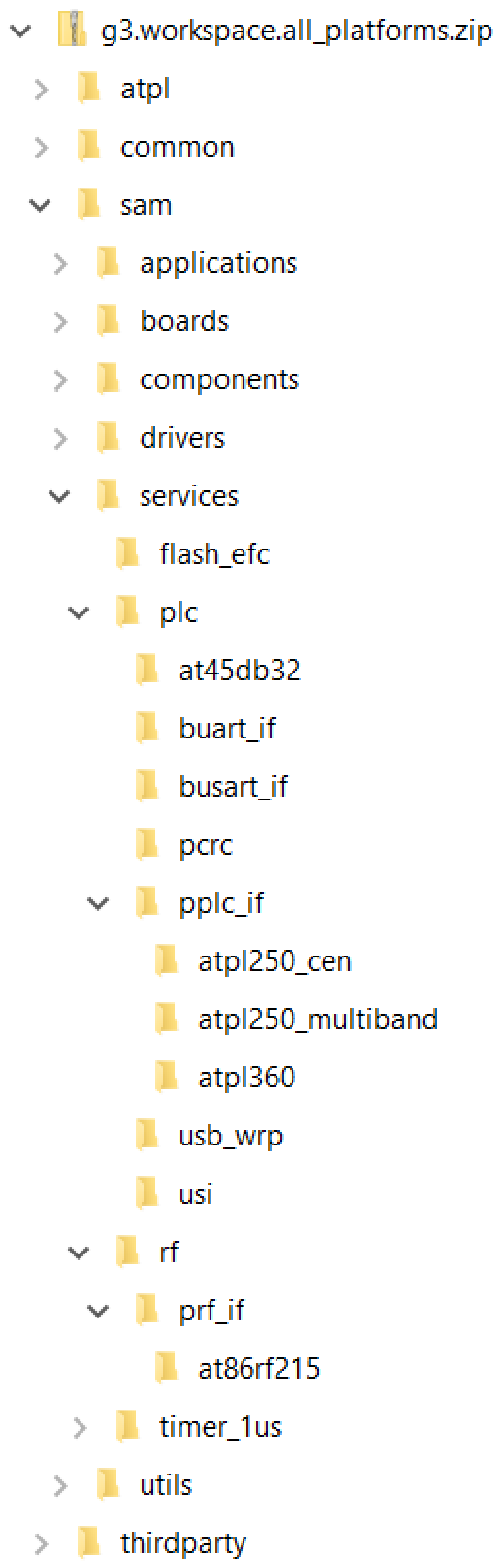
The common folder contains general purpose modules used by the G3 stack, and adds a specific one for the PL360 device, the *common/components/plc* folder.

This folder contains the 1.3. PLC Host Controller module to perform and monitor all transactions with the PL360 device. As PL360 running PHY layer (for PHY level applications) or running PHY + MAC RT layers (for G3 applications) have different Host Controllers, two sub-folders are present, one for each implementation.

The Hybrid Profile introduces the *common/components/rf* folder, where the AT86RF215 Controller is located, which includes the RF PHY layer and associated modules.

2.1.3 sam folder

Figure 2-4. sam/services/plc & rf folders


```
g3.workspace.all_platforms.zip
    atpl
    common
    sam
        applications
        boards
        components
        drivers
        services
            flash_efc
            plc
                at45db32
                buart_if
                busart_if
                pcrc
                pplc_if
                    atpl250_cen
                    atpl250_multiband
                    atpl360
                usb_wrp
                usi
            rf
                prf_if
                    at86rf215
            timer_1us
        utils
    thirdparty
```

The *sam* folder contains modules shared for all SAM devices that are used by the G3 stack, and adds a specific one containing code used by PLC stacks, the *sam/services/plc* folder.

Services are modules that are placed above drivers, and whose intent is to perform specific tasks using such drivers to add an extra abstraction layer to facilitate some operations to the users.

at45db32 service manages read/write/erase operations for the at45db32 Flash memory, instead of having to access the SPI driver directly.

buart_if and *busart_if* (named after buffered uart and buffered usart) offer buffered tx/rx access to UART and USART peripherals, handling UART/USART drivers without interrupting their users until the whole buffer is transmitted or received.

pcrc service is included in all PLC applications, but not used by G3 in particular, thus, is beyond the scope of this guide.

pplc_if service is in charge of managing all SPI transactions with the PLC transceiver, either PL360 or ATPL250, providing a simplified API to the modules that interact with the transceiver.

prf_if service is in charge of managing all SPI transactions with the RF transceiver, providing a simplified API to the modules that interact with the transceiver.

usb_wrp service is the wrapper to use the USB port as a serial interface.

usi service (Universal Serial Interface), is an HDLC-type encapsulation module used by some G3 applications where the board running G3 is connected through a Serial Link to a Host PC or micro-controller that runs its counterpart (USI Host) to interact with G3 stack, abstracting the Serial connection.

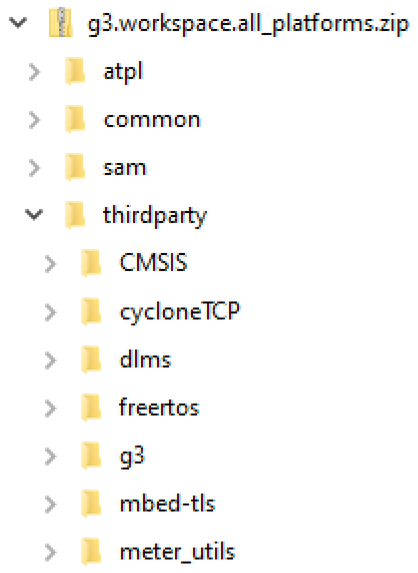
The *sam* folder contains also a Bootloader in both Project and Binary formats to be used along with PL360G55CB and PL360G55CF boards. These project and binary can be found in the folder: *\sam\applications\serial_bootloader*.

This bootloader allows the target boards to be reprogrammed without the need of additional HW programming tools. Only a connection though USB or UART and a PC tool are needed.

Note: If using the USB interface with Windows versions prior to 10, it is necessary to install a driver in the PC to create a Virtual COM port that will be used for the programming process. For this virtual port to be created, the computer will ask for a driver the first time the board is connected. The driver can be found in the folder: *\common\services\usb\class\cdc\device\atmel_devices_cdc.inf*.

2.1.4 thirdparty folder

Figure 2-5. thirdparty folder

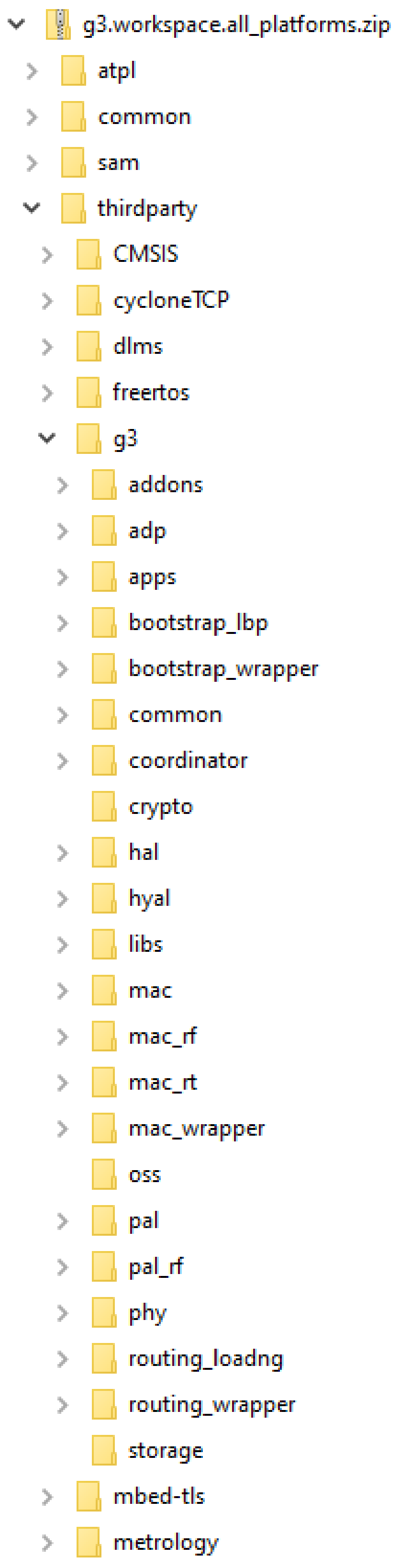


This folder is where code not related to specific modules/drivers/services is placed. In the G3 workspace, the following folders are included:

- *CMSIS*: where Cortex-M system files and libraries can be found.
- *cycloneTCP*: contains an IPv6 layer implementation from a third party, distributed under the license terms defined by the provider.
- *dlms*: contains an example implementation of a DLMS client and server, for G3 demonstration purposes.
- *freertos*: this RTOS is included as an example on how to integrate G3 stack in this kind of embedded solutions.
- *g3*: contains all G3 code and most of its auxiliary modules.
- *mbed-tls*: is the default cryptographic library that G3 relies on to perform security duties.
- *metrology/meter_utils*: metrology-related code, used in projects including metrology functionality.

2.1.4.1 g3 folder

Figure 2-6. thirdparty/g3 folder



```
g3.workspace.all_platforms.zip
  atpl
  common
  sam
  thirdparty
    CMSIS
    cycloneTCP
    dlms
    freertos
    g3
      addons
      adp
      apps
      bootstrap_lbp
      bootstrap_wrapper
      common
      coordinator
      crypto
      hal
      hyal
      libs
      mac
      mac_rf
      mac_rt
      mac_wrapper
      oss
      pal
      pal_rf
      phy
      routing_loadng
      routing_wrapper
      storage
    mbed-tls
    metrology
```

As previously stated, the *g3* folder contains all the G3 code and most of its auxiliary modules.

The folder structure reflects the G3 Architecture, as presented in 1. General Architecture. To summarize the modules already presented and present the new ones, the different sub-folders are listed here:

- *addons*: contains serialization layers to access ADP and MAC layers through a Serial Link using USI protocol.
- *adp*: contains the ADP library, configuration file and API header files, as presented in 1.10. ADP Layer, ADP Conf and ADP API and detailed in subsections of 9. API of DATA LINK Layer.
- *apps*: contains G3 stack example applications; see 5.1. ADP Examples for detailed information.
- *bootstrap_lbp*: contains the Local Bootstrap Protocol module common to the Coordinator and Device as presented in 1.16. LBP Module.
- *bootstrap_wrapper*: contains the LBP Module wrapper.
- *common*: contains auxiliary utils as presented in 1.12. Common Utils Module.
- *coordinator*: contains the Coordinator module implementation example as presented in 1.15. Coordinator Module.
- *crypto*: contains the cryptography wrapper and auxiliary files as presented in 1.13. Crypto Module and Crypto Wrapper.
- *hal*: contains Hardware Abstraction Layer as presented in 1.14. HAL (Hardware Abstraction Layer).
- *hyal*: contains the Hybrid Abstraction Layer as presented in 1.9. HyAL (Hybrid Abstraction Layer).
- *libs*: contains the G3 stack libraries, which include ADP libs, PLC MAC lib, RF MAC lib and Routing libs.
- *mac*: contains header files, which define PLC MAC API, detailed in subsections of 9. API of DATA LINK Layer.
- *mac_rf*: contains header files, which define RF MAC API, detailed in subsections of 9. API of DATA LINK Layer.
- *mac_rt*: contains MAC Real Time source code, as presented in 1.6. PLC MAC RT (MAC Real Time).
- *mac_wrapper*: contains the MAC layer Wrapper, as presented in 1.8. MAC Layer and MAC Wrapper and detailed in 9.6. MAC Wrapper.
- *oss*: contains the Operating System Support, explained in 4. Operating System Support.
- *pal*: contains PHY Abstraction Layers for both PL360 and ATPL250 devices, as presented in 1.4. PLC PAL (PHY Wrapper) and 1.7. PLC PAL 360 MAC RT (PLC MAC RT Wrapper).
- *pal_rf*: contains PHY Abstraction Layers for the RF215 device, as presented in 1.5. RF PAL (PHY Wrapper).
- *phy*: contains the PLC PHY layer source code for the ATPL250 device, as presented in 1.1. PLC PHY Layer and the PHY layer example applications for both the PL360 and ATPL250 devices, detailed in 5.2. PLC PHY Examples.
- *routing_loadng*: contains LOADng protocol API files, as presented in 1.11. Routing Module and Routing Wrapper.
- *routing_wrapper*: contains Routing module wrapper, as presented in 1.11. Routing Module and Routing Wrapper.
- *storage*: contains an auxiliary module, which handles G3 non-volatile data storage requirements; see 3.4. Persistent Data Storage for more info.

3. Understanding the G3-PLC FW Stack

The following chapter explains how the G3-PLC FW stack has to be initialized, started and called periodically. It also explains how to integrate a User Application over the G3 stack.

The basic procedure for performing this integration is:

- Initializing and starting the FW stack
- Developing user application requirements
- Modifying the configuration files according to the application requirements

3.1 Operation Modes (Operating System Support)

There are two different operation modes available:

- Microcontroller Operation mode
- RTOS Operation mode

The operation mode is defined in the *conf_oss.h* file under the following define sentence:

```
    /* OS Support */
    #define OSS_USE_FREERTOS
```

Important: For configuring the FW stack to operate in RTOS mode, define OSS_USE_FREERTOS. If *OSS_USE_FREERTOS* is not defined, the FW is executed in Microcontroller Operation mode. (See OSS Layer Configuration – conf_oss.h.)

Tip: The default RTOS included in the stack is FreeRTOS.
Also note that the RTOS mode is provided as an example of integration of the stack inside an OS for demonstration purposes only. It must not be taken as the starting point of a development, and Microchip does not ensure the same performance and stability as the examples running in Microcontroller mode.

This compilation constant changes the way code is executed in the *oss_if.c* file, which contains the code in charge of controlling the program flow. This file contains all the calls to initialization functions, calls to time control functions and calls to periodic task handler functions. All this is done in a different way depending on the operation mode, but the idea of controlling the program flow from this file is the same.

⚠ CAUTION

Regardless of the approach, it is required that the G3 periodic task is called every 1 ms when using the ATPL250 platform. This ensures that the time restrictions defined by the G3 specification are met.
In the PL360 platform, where MAC RT runs inside the device, and for PLC-only devices, this time constraint can be relaxed up to 20 ms for calling the G3 periodic task, as time-critical tasks are handled inside the PL360. For Hybrid devices, the 1-ms call rate has to be kept to ensure the PHY RF duties are correctly handled.

3.2 Initializing and Starting G3-PLC Stack and User Application

Here is an example, *dlms_app_coord*, of how the G3 PLC stack is configured and started.

The program entry point function is simple:

```
void main( void )
{
    oss_task_t x_task;
    /* Initialize OSS */
    oss_init();

    /* Register Task */
    x_task.task_init = app_init;
    x_task.task_process = app_process;
    x_task.task_1ms_timer_cb = app_timers_update;
    oss_register_task(&x_task);

    /* Register Dispatcher Task */
    x_task.task_init = dispatcher_app_init;
    x_task.task_process = dispatcher_app_process;
    x_task.task_1ms_timer_cb = dispatcher_timers_update;
    oss_register_task(&x_task);

    /* Register Ping Task */
    x_task.task_init = ping_app_init;
    x_task.task_process = ping_app_process;
    x_task.task_1ms_timer_cb = ping_app_timers_update;
    oss_register_task(&x_task);

    /* Register UDP Responder Task */
    x_task.task_init = udp_responder_app_init;
    x_task.task_process = udp_responder_app_process;
    x_task.task_1ms_timer_cb = udp_responder_app_timers_update;
    oss_register_task(&x_task);

    /* Start OSS */
    oss_start();
}
```

Three clear tasks are visible here:

- Initialization
- Task registration of different modules for further execution
- Start execution

These three steps are explained further in the following sections.

3.2.1 First Step: Initialization

```
void oss_init(void)
{
 /* Prepare the hardware */
 platform_init_hw();

 memset(&sx_oss_init_tasks, 0, sizeof(sx_oss_init_tasks));
 memset(&sx_oss_proc_tasks, 0, sizeof(sx_oss_proc_tasks));
 memset(&sx_oss_updt_tasks, 0, sizeof(sx_oss_updt_tasks));

 suc_oss_num_task_registered = 0;
}
```

First of all, the stack initializes the hardware platform according to its requirements by calling the *platform_init_hw()* function.

When using hardware different than the hardware supported by Microchip, HAL files may have to be changed according to user needs. All these files are available as source code for this purpose. In case Microchip evaluation boards are used, hardware will be correctly initialized and the user does not need to take care of this.

Then, task containers are reset and the number of registered tasks set to 0 to ensure a correct registration later.

3.2.2 Second Step: Task Registering

The OSS file (oss_if.c) is in charge of controlling all the program flow, as seen in Operation Modes (Operating System Support).

To call user application functions from this file, such functions have to be registered as tasks. The task prototype contains three fields, which the user will assign to callback functions:

- *task_init* field: to assign a callback function to perform initialization duties, such as variable initialization.

- *task_process* field: to assign a callback function to be called periodically by OSS, such as an application event handler. The rate at which this event handler will be called can be configured through the compilation constant OSS_APP_EXEC_RATE, which can be found in the *oss_if.h* file.
- *task_1ms_timer_cb* field: to assign a callback function to be called from a 1-ms timer interrupt. This is typically used for time control short duties requiring precision because it is called from timer interrupt, so the user has to avoid long processing time on it. (This is intended for simple duties such as incrementing a millisecond counter variable).

If a task does not require one of the offered fields, it can be set to *Null*, and OSS will not call any callback function for such task. For example, if a task does not need time control at all, the *task_1ms_timer_cb* field can be set to *Null* before the task registration.

3.2.3 Third Step: Start Program Execution

The program execution is started using *oss_start* function. The Microcontroller mode will be used to explain the different functions and processes involved in program execution. The OS mode can be easily followed in the provided code and it is not explained here, as functionality is exactly the same but performing tasks in an RTOS way.

```
    void oss_start(void)
    {
     /* Set up timer interrupt and user defined callback */
     platform_set_ms_callback(&_oss_1ms_timer_handler);
     platform_led_cfg_blink_rate(OSS_LED_BLINK_RATE);
     platform_cfg_call_process_rate(OSS_G3_STACK_EXEC_RATE);
     platform_cfg_call_app_process_rate(OSS_APP_EXEC_RATE);

#if defined(PLATFORM_PDD_INTERNAL_SUPPLY_MONITOR) ||
defined(PLATFORM_PDD_EXTERNAL_VOLTAGE_DIVIDER)
    /* Power down detection initialization */
    platform_init_power_down_det();
#endif

#ifdef OSS_ENABLE_IPv6_STACK_SUPPORT
    /* Initialize IPv6 stack */
    netInit();
#endif

#ifdef NUM_PORTS
    /* Initialize USI */
    usi_init();
#endif

    /* Task-registered initialization */
    _oss_execute_tasks(OSS_TASK_INIT);

    /* Timers may interfere with task initialization, init after */
    platform_init_1ms_timer();

    /* main loop */
    while (1) {
        /* Reset watchdog */
        RESET_WATCHDOG;

        if (platform_flag_call_process()) {
            /* G3 stack process */
            #ifdef OSS_G3_ADP_MAC_SUPPORT
            /* Internally calls MAC and PHY Event handlers */
            AdpEventHandler();
            #endif
        }

        if (platform_flag_call_app_process()) {
            /* Task-registered processes */
            _oss_execute_tasks(OSS_TASK_PROCESS);

            #ifdef NUM_PORTS
            usi_process();
            #endif

            /* Handle TCP/IP events */
            #ifdef OSS_ENABLE_IPv6_STACK_SUPPORT
            netTask();
            #endif
        }

        /* LED blink */
        platform_led_update();
    }
  }
```

This function performs some interesting tasks, which are explained in the next subsections.

3.2.3.1 Time Control Initialization

```
    /* Set up timer interrupt and user defined callback */
    platform_set_ms_callback(&_oss_1ms_timer_handler);
    platform_led_cfg_blink_rate(OSS_LED_BLINK_RATE);
    platform_cfg_call_process_rate(OSS_G3_STACK_EXEC_RATE);
    platform_cfg_call_app_process_rate(OSS_APP_EXEC_RATE);
```

First of all, the function platform_set_ms_callback configures a callback to be called every time an HW interrupt raises (which is defined and handled in *hal.c* file). This callback function is called every 1 ms, independently of the platform below, and it is in charge of managing the time base for the G3 stack.

The next function, *platform_led_cfg_blink_rate*, configures the rate at which an activity LED blinks in the board periodically, just to indicate that program is running. The rate is defined by the constant *OSS_LED_BLINK_RATE*, located in file *conf_oss.h*, and its value is indicated in milliseconds. This is an auxiliary function that does not affect the G3 stack functionality at all.

Finally, functions *platform_cfg_call_process_rate* and *platform_cfg_call_app_process_rate* are used to define the rate at which the G3 stack and user application event handlers are called. They are defined using the constants *OSS_G3_STACK_EXEC_RATE* and *OSS_APP_EXEC_RATE* located in the *oss_if.h* file. If running MAC RT in the PL360, the value is overridden, re-defining constants with a new value in the *conf_oss.h* file.

3.2.3.2 Power-Down Detector Initialization

```
#if defined(PLATFORM_PDD_INTERNAL_SUPPLY_MONITOR) ||
defined(PLATFORM_PDD_EXTERNAL_VOLTAGE_DIVIDER)
    /* Power down detection initialization */
    platform_init_power_down_det();
#endif
```

The G3 stack needs some information to be stored in non-volatile memory so that it can be restored after a Power-Down. For this purpose, a Power-Down detection mechanism is implemented in the HAL module. To use this Power-Down detector, either *PLATFORM_PDD_INTERNAL_SUPPLY_MONITOR* or *PLATFORM_PDD_EXTERNAL_VOLTAGE_DIVIDER* has to be defined in the *conf_hal.h* file. When it is defined, all needed hardware is initialized using function *platform_init_power_down_det.*

For further information regarding how Power-Down detection is performed, see 6.6. Power-Down Detection.

3.2.3.3 IPv6 Stack Support

```
#ifdef OSS_ENABLE_IPv6_STACK_SUPPORT
    /* Initialize IPv6 stack */
    netInit();
#endif
```

The G3 protocol is meant to use an IPv6 layer on top of it in most of the applications. IPv6 is considered out of the scope of the G3 stack; therefore, it is not part of the Microchip G3 library.

An implementation of a third party of an IPv6 stack is provided to demonstrate the use of DLMS on top of it and the link between IPv6 and the G3 stack.

The use of this IPv6 is optional so the user can build an application directly over the G3 stack without the IPv6 layer, but as mentioned before, this is not the usual case. The use of this layer depends on the definition of the constant *OSS_ENABLE_IPv6_STACK_SUPPORT*, located in file *conf_oss.h*. In the provided example, the IPv6 layer is needed, and, thus, this constant is defined by default.

3.2.3.4 USI Initialization

```
#ifdef NUM_PORTS
    /* Initialize USI */
    usi_init();
#endif
```

If a Universal Serial Interface (USI) is used, it is initialized here using the *usi_init* function. To call this function, *NUM_PORTS* must be defined in the *conf_usi.h* file.

The examples provided include an embedded sniffer functionality that requires the USI. The *conf_usi.h* file in the project can be inspected to see a configuration example.

For more information about what USI is and how to use it, refer to 12. Serialization with Embedded USI.

3.2.3.5 Registered Tasks Initialization

```
    /* Task-registered initialization */
    _oss_execute_tasks(OSS_TASK_INIT);
```

As seen in 3.2.2. Second Step: Task Registering, it is possible to set a pointer to a callback function where a registered task performs its initialization duties.

The function *_oss_execute_tasks(OSS_TASK_INIT)* executes initialization callbacks for all registered tasks.

Important: It is required that one of the registered tasks initializes the G3 stack using the function *AdpInitialize*. This is not done directly by OSS because one of the parameters to set in the G3 stack initialization is the pointer to the callback functions that the G3 stack calls when notifications are reported from the stack to the application, and such callback functions have to be implemented by the application. This is why the application must call the G3 stack initialization function.

For more information regarding the *AdpInitialize* function, refer to Initialization Function.

3.2.3.6 Timer Start-Up

```
    /* Timers may interfere with task initialization, init after */
    platform_init_1ms_timer();
```

After all of the initialization tasks are performed, the 1-ms timer interrupt is configured and enabled.

Once enabled, all callbacks (both internal and user-defined) registered to the timer interrupt will be called every 1 ms from the interrupt.

3.2.3.7 Main Loop

After all initializations seen in previous sections, the program enters in the main infinite loop. The following recursive processes are performed in this main loop.

3.2.3.7.1 Reset Watchdog

On every main loop execution cycle, the watchdog timer is reloaded. The watchdog timer is used to detect program execution anomalies and reset the processor if any occur.

To disable the watchdog, the constant *CONF_BOARD_KEEP_WATCHDOG_AT_INIT*, located in the*conf_board.h* file, has to be undefined.

3.2.3.7.2 Process Call Check

```
    if (platform_flag_call_process()) {
        ...
    }
```

```
    if (platform_flag_call_app_process()) {
        ...
    }
```

The call of periodic functions, or event handlers, of the stack and app layers is done depending on the result value of the functions *platform_flag_call_process* and *platform_flag_call_app_process*. As described in Time Control

Initialization, a call rate for event handlers is configured through the constants *OSS_G3_STACK_EXEC_RATE* and *OSS_APP_EXEC_RATE.* **Both are configured to 1-millisecond period by default, but for the G3 examples on the PL360 platform, it is overridden due to relaxed time constraints.**

In summary, these functions will return *true* when *OSS_G3_STACK_EXEC_RATE* and *OSS_APP_EXEC_RATE* are elapsed from the previous call, and, thus, event handlers of different layers will be called.

3.2.3.7.3 Registered Tasks Event Handler

```
    /* Task-registered processes */
    _oss_execute_tasks(OSS_TASK_PROCESS);
```

If a pointer is set to a function that is called periodically as an event handler (as in 3.2.2. Second Step: Task Registering), it will be called inside the *_oss_execute_tasks (OSS_TASK_PROCESS)* function.

The function will call the event handlers for all the registered tasks in the same order they were registered.

3.2.3.7.4 USI Event Handler

```
    #ifdef NUM_PORTS
        usi_process();
    #endif
```

As in USI Initialization, if the *NUM_PORTS* constant is defined, the USI event handler will be called here.

3.2.3.7.5 G3 Stack Event Handler

```
    #ifdef OSS_G3_ADP_MAC_SUPPORT
        /* Internally calls MAC and PHY Event handlers */
        AdpEventHandler();
    #endif
```

If constant *OSS_G3_ADP_MAC_SUPPORT* is defined (in *conf_oss.h*), the G3 stack event handler is called.

The function *AdpEventHandler* is the event handler for the ADP layer. This function internally manages the call to the lower layers of the stack (MAC, PAL and PHY), so the user does not have to worry about managing all these handlers.

Important: As stated previously, this function has to be called every 1ms when using ATPL250 platform to meet the time constraints defined in G3.

One reason for not defining this constant could be, for example, when an application is running directly over the PHY layer and the PHY event handler is called directly from the application event handler.

For more information regarding the *AdpEventHandler* function, refer to Event Handler Function.

3.2.3.7.6 IPv6 Event Handler

```
    #ifdef OSS_ENABLE_IPv6_STACK_SUPPORT
        netTask();
    #endif
```

As explained in IPv6 Stack Support, the provided example makes use of a third party IPv6 stack. When the *OSS_ENABLE_IPv6_STACK_SUPPORT* constant is defined, its event handler will be called here.

3.2.3.7.7 Activity LED Blinking

```
    platform_led_update();
```

Finally, the main loop checks whether the activity LED has to blink using the *platform_led_update* function. The blinking of the LED is decided depending on the *OSS_LED_BLINK_RATE* constant, as described in Time Control Initialization.

3.3 Configuration Files

The stack contains a variety of configuration files, which allow some features to be active or not, or to change some behaviors.

Each provided application contains a folder for each supported device and associated evaluation board. This folder contains the projects for the corresponding device and board, and the configuration files for those projects. Apart from a particular device or board configuration files, the following configuration files are available:

- conf_at86rf.h (Only in projects supporting the Hybrid Profile)
- conf_atpl360.h (Only in projects using PL360 device)
- conf_board.h
- conf_bs.h (Only in projects for Coordinator)
- conf_buart_if.h
- conf_busart_if.h
- conf_clock.h
- conf_oss.h
- conf_hal.h
- conf_pplc_if.h
- conf_prf_if.h (Only in projects supporting the Hybrid Profile)
- conf_project.h
- conf_tables.h
- conf_timer_1us.h (Only in projects supporting the Hybrid Profile)
- conf_uart_serial.h
- conf_usb.h
- conf_usi.h
- FreeRTOSConfig.h

The meaning of these configuration files is described in more detail in the following sections.

3.3.1 RF215 Configuration – conf_at86rf.h

Constants used for RF215 configuration. Available constants are:

- AT86RF_PART – Selects between AT86RF215 and AT86RF215M parts
- AT86RF215_DISABLE_RF24_TRX – Disables 2.4 GHz band
- AT86RF215_DISABLE_RF09_TRX – Disables SubGHz band
- AT86RF215_ENABLE_AUTO_FCS – Enables automatic calculation and append of FCS (Frame Check Sequence) in transmission and verification on reception
- AT86RF215_MAX_PSDU_LEN – Defines maximum PSDU length for buffers memory allocation
- AT86RF215_NUM_RX_BUFFERS – Number of reception buffers, useful in case RX data is collected at a low rate from upper layers
- AT86RF215_NUM_TX_PROG_BUFFERS – Number of programmed transmission buffers, useful in case more than one frame has to be programmed for a future transmission
- AT86RF_ADDONS_ENABLE – Enables RF215 add-ons, which add sniffer capabilities

3.3.2 PL360 Configuration – conf_atpl360.h

Constants used for PL360 configuration. Available constants are:

- ATPL360_ADDONS_ENABLE – Enables PL360 add-ons, which include sniffer capabilities and/or serial access to PHY API
- ATPL360_RST_WAIT_MS – Wait period before downloading PL360 binary retries if download verification fails
- ATPL360_WB – Defines work band for PL360 (ATPL360_WB_CENELEC_A, ATPL360_WB_CENELEC_B, ATPL360_WB_FCC, ATPL360_WB_ARIB). Not used on projects with Mac RT running in PL360, where band dependency is handled inside PL360 binary.

3.3.3 Board Configuration – conf_board.h

This file defines the peripherals used and their configuration, such as:

- Watchdog Timer can be enabled/disabled using constant CONF_BOARD_KEEP_WATCHDOG_AT_INIT.
- Console UART, if used, is defined here:

```
#define CONSOLE_UART       UART1
#define CONSOLE_UART_ID    ID_UART1
```

- External slow clock usage is enabled by defining constant CONF_BOARD_32K_XTAL
- Support for different peripherals (UART, SPI, TWI), and for external component control (LCD, external memory) are defined here.

3.3.4 Bootstrap Module Configuration – conf_bs.h

Configuration of Bootstrap and Coordinator. Contains:

- G3_COORDINATOR_PAN_ID – Defines PAN ID for the network to be created by the Coordinator
- MAX_LBDS – Number devices that are allowed to join the network
- LBS_INVALID_SHORT_ADDRESS – Address reserved for the Coordinator and considered invalid for devices
- INITIAL_KEY_INDEX – Initial Key Index (of the 2 available indexes) to use when setting GMK

3.3.5 Buffered UART Configuration – conf_buart_if.h

Configuration of HW timers and buffer sizes for the UARTs.

3.3.6 Buffered USART Configuration – conf_busart_if.h

Configuration of HW timers and buffer sizes for the USARTs.

3.3.7 Clock Configuration – conf_clock.h

Configuration of the clock source and clock tree of the microcontroller.

- System clock source is configured among different options:
  - External Crystal
  - Internal RC
  - PLL
- System clock prescaler is chosen
- PLL source, multiplier and divider are selected, if a PLL is used
- USB clock source is configured (on devices supporting it)
- Coprocessor clock is configured (on dual core devices)

Configuration provided on G3 examples are intended to take full advantage of MCU and Evaluation Board capabilities. For an alternative configuration, check 5.4. Clock Configuration Example for Low Requirements Module.

3.3.8 OSS Layer Configuration – conf_oss.h

On this file, the user can decide to run the program in Microcontroller mode or use an OS, which layers are used in the project, and the periodicity of some tasks.

- OSS_USE_FREERTOS – Defining this will cause the program to run in OS mode (using FreeRTOS as the supported and included RTOS). By default this is not defined, so Microcontroller mode will be used.
- OSS_LED_BLINK_RATE – Sets the blinking rate of the activity LED. This may be used to set a different rate to different applications to distinguish them when running on different boards.
- If the user wants to override the frequency at which the Application and G3 Stack Event Handlers are called, the values of *OSS_APP_EXEC_RATE / G3_APP_PROCESS_EXEC_RATE* and *OSS_G3_STACK_EXEC_RATE / G3_PROCESS_EXEC_RATE* have to be set in this file. When constants are defined in this file, the default value of 1 ms is overridden by the number of milliseconds set in these constants. The provided applications for Mac RT running in the PL360 Device use this method to relax the calling rate; refer to them for an example.
- The activation or deactivation of different layers is done by:
  - OSS_ENABLE_IPv6_STACK_SUPPORT – Adds an IPv6 layer to the running stack. See IPv6 Stack Support for more information.

– OSS_G3_ADP_MAC_SUPPORT – Makes the ADP and MAC layers of the G3 stack to run in the current application. As explained in a previous section, this may be confusing because the user may expect that these layers have to always run, but this is left configurable to be able to run applications directly over the PHY layer (which would not be strictly G3 applications).

3.3.9 HAL Configuration – conf_hal.h

This file contains HW configuration parameters related to:

- Timers
- Power-Down detection
- LCD signaling (in case LCD support is available)
- G3 data storage
- EUI64 initialization source

3.3.10 PPLC Interface Configuration – conf_ pplc_if.h

This file specifies the configuration of the hardwired interface between the microcontroller and the G3-PLC modem:

- Configuration of SPI module
- Interruption signaling
- Reset signaling
- LDO Enable signaling
- Standby control (if connection available)
- Thermal Warning Monitor pin definition (on PL460 based projects)
- TX Enable signaling (on PL460 based projects)
- ADC Channel definition for Vdd Monitor (on PL460 based projects)

The G3-PLC stack requires an external interrupt from the transceiver. For the ATPL250 device, the interrupt pin EINT is used. For the PL360, the firmware binary of the device configures GPIO3 (pin PA3 of PL360) as the external interrupt line. This file associates the external interrupt output of the transceiver with the stack running in the host microcontroller.

For further information about how the interrupt line of the PL360 works, refer to the PL360 Host Controller documentation.

3.3.11 PRF Interface Configuration – conf_prf_if.h

This file specifies the configuration of the hardwired interface between the microcontroller and the RF transceiver:

- Configuration of SPI module, including max length of transaction for buffer allocation
- Interruption signaling
- Reset signaling
- LED signaling

The G3-PLC stack requires an external interrupt from the transceiver. This file associates the external interrupt output of the transceiver with the stack running in the host microcontroller.

3.3.12 Project Configuration – conf_project.h

This file contains bandplan definition:

- CONF_BAND_CENELEC_A – 36 subcarriers [35,9375 - 90,625] kHz
- CONF_BAND_CENELEC_B – 16 subcarriers [98,4375 - 121,875] kHz
- CONF_BAND_FCC – 72 subcarriers [154,6875 - 487,5] kHz
- CONF_BAND_ARIB – 54 subcarriers [154,6875 - 403,125] kHz

On projects where multiple binaries for PL360 device are linked, so working band can be dynamically changed, constants *CONF_MULTIBAND_FCC_CENA* or *CONF_MULTIBAND_FCC_CENB* are defined here.

Particular PLC Coupling configurations, such as using a special branch configuration, are also defined on this file. For example *CONF_ENABLE_C11_CFG* has to be defined if COUP_011 is used, or in PL460 projects, *CONF_ENABLE_PL460_FCC_1_5B_CFG* can be defined for a particular FCC configuration.

On projects requiring restoring IBs after a Power-Down, constant *ENABLE_PIB_RESTORE* is defined here.

For PL360G55CB and PL360G55CF boards, this file contains some macros for easy configuration of USI and Console ports on USB and MikroBUS:

```
/* Port mapping for USI and Console */

/* USI. Select One or None of the following options */
#define USI_ON_MIKROBUS_USART
/* #define USI_ON_USB */

/* Console. Select One or None of the following options */
/* #define CONSOLE_ON_MIKROBUS_USART */
#define CONSOLE_ON_USB
```

Using these macros, *conf_usi.h* and *conf_usart_serial.h* files automatically adapt their definitions to map ports in the desired way so the user does not need to manually modify them.

Tip: This file is where the user needs to include any compilation constant related to the user application code. It is a generic file local to the project that the user may extend depending on individual needs.

3.3.13 Tables size configuration – conf_tables.h

Some of the tables and buffers used in the G3 stack were left size-configurable for user convenience.

The tables and buffers sizes that can be configured in this file are the following:

- CONF_COUNT_ADP_BUFFERS_1280 – Number of ADP TX buffers of size 1280 bytes
- CONF_COUNT_ADP_BUFFERS_400 – Number of ADP TX buffers of size 400 bytes
- CONF_COUNT_ADP_BUFFERS_100 – Number of ADP TX buffers of size 100 bytes
- CONF_ADP_FRAGMENTED_TRANSFER_TABLE_SIZE – Number of simultaneous fragmented Rx frames
- CONF_ADP_ROUTING_TABLE_SIZE – ADP Routing Table size
- CONF_ADP_ROUTING_SET_SIZE – ADP Routing Set size
- CONF_LOADNG_RREP_GENERATION_TABLE_SIZE – LOADng RREP Generation Table size
- CONF_MAC_NEIGHBOUR_TABLE_ENTRIES – PLC MAC Neighbour Table size
- CONF_MAC_POS_TABLE_ENTRIES – PLC MAC POS Table size
- CONF_MAC_POS_TABLE_ENTRIES_RF – RF MAC POS Table size
- CONF_MAC_DSN_SHORT_TABLE_ENTRIES – MAC Data Sequence Number Table size for Short Addresses
- CONF_MAC_DSN_EXTENDED_TABLE_ENTRIES – MAC Data Sequence Number Table size for Extended Addresses
- CONF_MAC_DSN_TABLE_ENTRIES_RF – RF MAC Data Sequence Number Table size
- CONF_NET_MEM_POOL_BUFFER_COUNT – IPv6 Buffer Count
- CONF_MAX_NEIGHBOUR_DLMS_SNAPSHOT_SIZE – DLMS Neighbour Table Snapshot size
- CONF_MAX_ROUTING_DLMS_SNAPSHOT_SIZE – DLMS Routing Table Snapshot size
- CONF_DLMS_MAX_DEV_NUM – DLMS Supported Devices Table size

More detailed information can be found in 6.2. Table Sizes and Buffer Count Configuration.

3.3.14 1 µs Timer Configuration – conf_timer_1us.h

HW TC peripheral selection to be used as the source for the 1 µs Timer Service.

3.3.15 Serial USART Service Configuration – conf_uart_serial.h

Definition of the console UART.

3.3.16 USB CDC Configuration – conf_usb.h

Constants used for USB Communication Device Configuration. Only used on boards where a native USB port is available.

These constants are tightly related with USB Virtual COM port configuration and are not intended to be modified.

3.3.17 Universal Serial Interface Configuration – conf_usi.h

If serialization is used for the project, this file contains the USI port and protocol definition and configuration.

ENABLE_SNIFFER constant is defined in this file to use embedded sniffer functionality.

3.3.18 Generic FreeRTOS Peripheral Control Functions Configuration – FreeRTOSConfig.h

FreeRTOS configuration that the user can modify according to their individual needs.

⚠ **CAUTION** Take into account the requirements, handling and usage of the Microchip G3-PLC FW Stack prior to changing the RTOS configuration. Carefully read Operating System Support.

3.3.19 Global Configuration Affecting all Projects – conf_global.h

Configuration files previously presented are project-scope files; therefore, there is a file for each of the available projects so configuration can be different from one project to another.

There is a special configuration file, *conf_global.h*, that is common to all projects. The idea is to store parameters here that are likely to be the same for different projects.

Currently, there are two parameters to be configured in this file:

- ENABLE_ROUTING – The routing enabling parameter has to be inline with the inclusion or not in the project of Routing Library. If Routing library is not included, this parameter has to be set to 0 in order to correctly handle Routing requests from G3 on Routing Wrapper. By default, routing is enabled in CENELEC-A, CENELEC-B and FCC G3 modes of operation.
- SPEC_COMPLIANCE – G3 specification compliance. The last G3 specification date was March 2017, and this is the version recommended for new implementations. However, there are still users that need to be compliant with the previous one (from 2015). The provided libraries implement both specifications, and the selection of which one to use is done at initialization time. By default, Spec17 is defined.

3.4 Persistent Data Storage

Requirements & Roles

The G3 stack requires that a certain number of IB values are preserved after a reset or even after a device is power cycled.

The IBs to be preserved are the following (as named in the Microchip G3 Stack API files):

- *MAC_WRP_PIB_FRAME_COUNTER*
- *MAC_WRP_PIB_FRAME_COUNTER_RF*
- *ADP_IB_MANUF_DISCOVER_SEQUENCE_NUMBER*
- *ADP_IB_MANUF_BROADCAST_SEQUENCE_NUMBER*

⚠ **CAUTION** The G3 stack is not responsible for this task. It is an external module that has to be aware of a reset or power-down, to perform the read/storage/recover/write sequence of the IBs.

The G3 stack offers a helper callback function, *AdpUpdNonVolatileDataIndication*, which is called every time any of these IBs changes, so the upper layer can always have the latest values just in case they need to be recovered. This callback function only informs that the values have changed, but it is the upper layer that has to retrieve the IB values and keep the record.

Proposed Approach

The methods used to achieve this data preservation are up to the final user, but Microchip provides an example of implementation in the *Storage* module, which, in turn, makes use of the *HAL* module to access low-level hardware. The approach is a combination of data records on both GPBRs and Flash memory, where GPBRs are used in case of a microcontroller reset, and Flash memory is used in case of Power-Down.

Power-Down detection is done by the *HAL* module, see 6.6. Power-Down Detection for details. The *HAL* module allows a callback function to be defined, which will be called when a Power-Down is detected.

Along with the IBs mentioned above, some extra data is stored:

- A counter incremented on each start-up, used to increase randomness in number generation on successive system start-ups, in microcontrollers that do not include a built-in TRNG block
- A version number and a CRC used to ensure the integrity of the stored data

Data Storage/Recovery Algorithm

Below are the steps followed to ensure data preservation:

1. During execution, and upon *AdpUpdNonVolatileDataIndication* callback, the Storage module retrieves the IBs from the G3 stack and stores them on GPBRs. This way we ensure that in case of a reset (other than Power-Down), we can retrieve information from these GPBRs.
2. When the Power-Down Callback function is called, the data is saved to Flash memory for further recovery.
3. Upon start-up, the reset source is checked and one of the following actions is taken:
   a. In case of a Power-Down reset, Flash memory has the latest info written on Power-Down detection and GPBRs are empty. Then, data is recovered from Flash memory and, after validation, it is written to GPBRs and set to G3 IBs.
   b. In case of any other reset cause, GPBRs contain the latest information and Flash does not. Then, data is recovered from GPBRs and set to G3 IBs.
4. Back to step 1.

Flash Storage Options

The provided storage implementation example has different options to choose the target Flash memory where the G3 data will be stored.

This option is set on the provided examples on the *conf_hal.h* file, where one of the following options has to be defined:

- *CONF_STORAGE_INTERNAL_FLASH* – Uses microcontroller internal Flash to store data
- *CONF_STORAGE_USER_SIGNATURE* – Uses a microcontroller special Flash area called User Signature to store data
- *CONF_STORAGE_AT45DBX_FLASH* – Uses an external Flash chip mounted on some Microchip EKs to store data

The Flash memory area is read and erased at start-up. The erase operation is done at start-up in order to save time on Power-Down, when time is critical and, thus, only the write operation is performed there.

The time for the writing operation in the different Flash options is shown in the table below.

Config Option	Writing Time (µs)
CONF_STORAGE_INTERNAL_FLASH	150
CONF_STORAGE_USER_SIGNATURE	270
CONF_STORAGE_AT45DBX_FLASH	50

3.5 Firmware Update Process

G3-PLC specification does not define a standard firmware update process. The stack is upgradeable using application-based mechanisms.

The user application is responsible for performing the memory management and steps for the firmware update process.

4. Operating System Support

4.1 Overview

User applications can run over an operating system, which means that the G3-PLC stack implementation needs to support that.

The OSS intends to transform the Microcontroller mode operation into a task-mode operation typical of operating systems. In order to do that, it creates and manages a single task where all active layers and interfaces are included. The user does not need to take care of controlling how the G3-PLC stack is running and can create their applications normally. The current implementation of the OSS is based on FreeRTOS but the user can modify it appropriately to use any other RTOS.

4.2 G3-PLC Process. Requirements, Handling and Usage of the G3-PLC FW Stack

When RTOS is used, the user has to take into account the requirements of different tasks running, to configure parameters such as the stack for each one of the tasks.

The given implementation takes into account the needed resources for the provided example, but any modification may have a direct impact on these needs. Therefore, the FreeRTOS configuration file may have to be modified.

⚠ **CAUTION** Take into account that the provided RTOS support is just an example of integration of the stack in an RTOS. It is functional, but Microchip does not ensure the same reliability and performance as the Microcontroller mode implementation, as the latter is the reference implementation.

5. Example Applications

⚠ CAUTION Note that all provided application examples were configured to work on our evaluation boards. Every firmware project must define a Board Support Package (BSP) according to the hardware to be used.

Along with the G3-PLC FW stack, some application examples are provided to show how to use the stack.

Examples can be divided into two groups:

- ADP examples, using the whole stack (ADP + MAC + PHY), to evaluate Microchip's G3 solution as a whole
- PLC PHY examples, using only the PLC PHY layer, to evaluate the physical layer features, or to be used connected to a PC tool for demonstration purposes. For PL360, MAC RT examples are also provided for the same purpose.

Depending on the target platform, the example applications are provided for CENELEC-A, CENELEC-B, FCC and ARIB working bands on PLC. A suffix in the project name indicates the band it is for. PL360 and ATPL250 platforms support bands in a different way:

- PL360 supports CENELEC-A, CENELEC-B and FCC bands by using different binary files, one for each band.
  - Suffix "_cen_a" indicates that the CENELEC-A binary is linked in the application and loaded to PL360 at start-up.
  - Suffix "_cen_b" indicates that the CENELEC-B binary is linked in the application and loaded to PL360 at start-up.
  - Suffix "_fcc" indicates that the FCC binary is linked in the application and loaded to PL360 at start-up.
  - Projects for the PL360G55CB board only support CENELEC-B and the behavior is the same as in "_cen_b" ones.
  - A special case is the PL360G55CF board, which supports both CENELEC-A and FCC bands. In these projects, both binaries are linked in the application and the one loaded to PL360 is decided at run time, depending on the Host Controller. The one loaded by default is determined by the band definition in the *conf_project.h* file.
- ATPL250 supports CENELEC-A, FCC and ARIB bands by using different PHY layer implementations.
  - Suffix "_cen_a" indicates that a specific CENELEC-A PHY layer is used. This layer is optimized in both Code and Data size for this band.
  - Suffix "_multiband" indicates that a generic multiband PHY layer is used. This layer can be used in any of the three supported bands, and it is initialized at run time through its initialization function. The band initialized by default is determined by the band definition on *conf_project.h* file.
- Projects supporting the G3 Hybrid Profile add the "_hybrid" suffix at the end of their name.

The band configuration on band-specific projects is already pre-defined in the provided conf_project.h file, and changing it will cause malfunctions. On the other hand, on multiband projects, a default band is pre-defined in the provided conf_project.h file, but it is possible to change it to the other band if desired.

Attention: As stated above, provided examples are configured to take full advantage of Evaluation Board capabilities, ignoring aspects such as power consumption that a final implementation will take care of. For an example of an alternative clock configuration for low requirements, see 5.4. Clock Configuration Example for Low Requirements Module.

5.1 ADP Examples

5.1.1 DLMS App

The DLMS App is an example that shows how the G3 API along with an IPv6 stack can be used in a typical DLMS application. This application is provided for device and coordinator, implementing a DLMS server and client respectively. The coordinator performs a continuous data collection that generates data traffic in the G3 network. Besides DLMS traffic, the application can be configured to generate ICMPv6 traffic though PINGs to other nodes.

Coordinator
Application header file *app_dlms_coord.h* contains some configuration constants for the coordinator which depend on the number of devices connected and the number of hops required to reach the coordinator if complex setups are used. These setup-dependent constants, which can be changed by the user, are:

- *DLMS_APP_ENABLE_PATH_REQ*. Enables the path request to all devices from the coordinator. If enabled, a path request round will be executed before each cycle. This is not defined by default.
  Note: As the structure to hold the path request is quite memory consuming, when this definition is enabled, the number of supported devices may have to be lowered in order to fit in RAM memory.
- *DLMS_TIME_WAIT_RESPONSE*. Time in milliseconds to consider a data timeout when no response is received from a device and continue with the next data request. Depending on the setup, and especially when hops and routes are used, the round trip time of a data request-response may be high. The default value is 50 seconds. The Route Repair process can be triggered if communication fails at some point, and the Timeout for such process is defined by the G3 Spec to 40 seconds by default, so the application timeout is chosen to take this into account.
- *DLMS_APP_WAIT_REG_NODES*. When defined, whenever a bootstrap or routing activity is detected in the network, the timer that waits for TIMER_WAITING_START is reloaded, so cycles would only start after the TIMER_WAITING_START time without bootstrap or routing traffic. This avoids needing to know beforehand the time it takes to build the network so as to wait before cycles start. Defined by default.
- *DLMS_TIME_WAITING_IDLE*. Time in milliseconds to wait until the application starts sending data requests to nodes present in the network. Depending on DLMS_APP_WAIT_REG_NODES constant (explained above), this time can be an absolute time starting when the coordinator is powered-up, or can be dynamically reloaded if network activity is detected. When this timer expires, DLMS traffic starts, taking into account only the devices present in the network at that moment. Therefore, if the DLMS_APP_WAIT_REG_NODES is not defined, the DLMS_TIME_WAITING_IDLE time has to be long enough to let all devices perform their bootstrap process correctly. Bootstrap process time per device is deeply dependent on the number of devices joining the network (the more devices, the more collisions and thus the more delay until next try), and also on the number of hops needed to perform the join process (hops increase the time to reach coordinator and get a response). A time of one minute per device may fit almost any setup (it is more than enough in simple setups, but just to give a general rule). Default value is 120 seconds.

The file *app_ping_coord.h* contains the constant *DLMS_APP_ENABLE_PING_CYCLE*, which enables sending ping frames to all devices before each cycle. Not defined by default.

Device
The file *app_ping_dev.h* contains the constant *DLMS_APP_ENABLE_PING_ALARM*, which enables sending ping frames periodically from the devices, using the interval definitions contained in this same file. Not defined by default.

Common
Both Device and Coordinator applications can configure the G3 stack and IPv6 parameters in order to be able to run G3 Conformance and Performance tests. On the provided projects, there is a compilation constant *APP_CONFORMANCE_TEST* available on the *g3_app_config.h* file which enables this configuration at start-up, but the final user can configure this behavour at run time if preferred, modifying the application code by calling configuration functions upon a particular trigger.

5.1.2 Meter/Modem Apps

The Meter and Modem concept and background

The objective of the Meter and Modem applications is to demonstrate how to implement, in 2 separated devices communicating through a Serial Port, the DLMS and Metrology objects on one side, and the PLC Communications on the other. This is typically the distribution when there is an already functional Meter, to which a communications module is connected, whatever the media used for such communications.

Serial Communication Protocol between the Meter and Modem

The Meter and Modem applications are provided with 2 different communication protocols between them:

- Simple Management, also called “Transparent Mode”
- DLMS Management, which implements an HDLC protocol over the Serial Line

Any of these 2 options is enabled by defining one of the following Compilation Constants in the *g3_app_config.h* file:

```
#define SIMPLE_MGMT
#define DLMS_MGMT
```

Only one of them must be defined. DLMS management is the option enabled by default in the given example projects.

The Meter and Modem Roles and Exchanged Information

Regardless of the protocol used between the Meter and Modem, the information exchanged between devices is the same.

The Meter application is Reactive and does nothing until a request is received from the Modem.

The Modem is the Active part, and it is responsible for sending requests to the Meter (either requests from the Modem itself or coming from a remote node through an available communication medium) and sending the Response back through the medium where the request was received.

The G3 stack needs an EUI64 to be used as an Extended Address before Joining a Network, and this is the first task of the Modem, to request the Meter for a 64-bit string called “MeterID”, which will be used for that purpose.

After joining the Network using the MeterID, the Modem will wait for requests from a remote node, transfer those requests to the Meter using the selected protocol through the Serial Line, and send back the response obtained from the Meter to the remote node that originated the request.

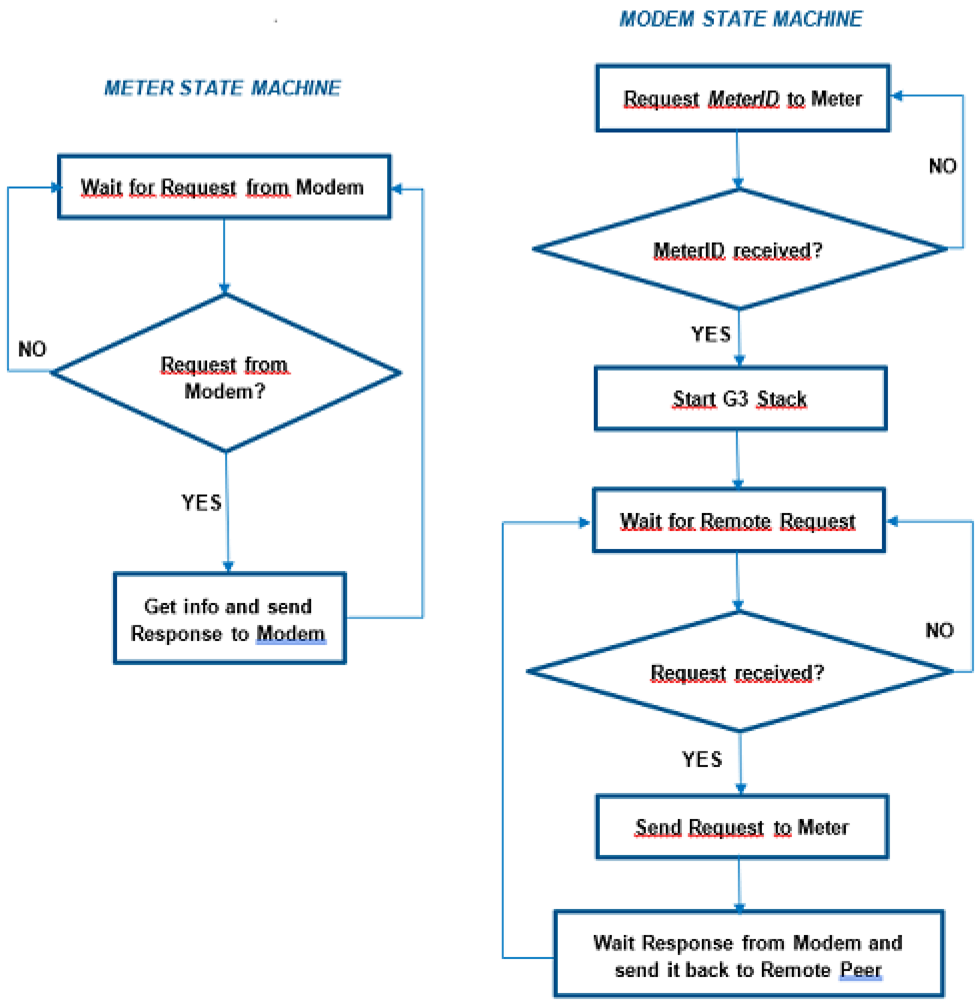
The given example uses DLMS as Application Layer on the Meter, so DLMS requests will be sent from the Modem to the Meter, and DLMS responses are what the Meter will send to the Modem.

For more information on how data is exchanged between devices, step through the provided example applications.

State Machines on the Meter and Modem

The State Machines running on the Meter and Modem devices can be depicted as follows:

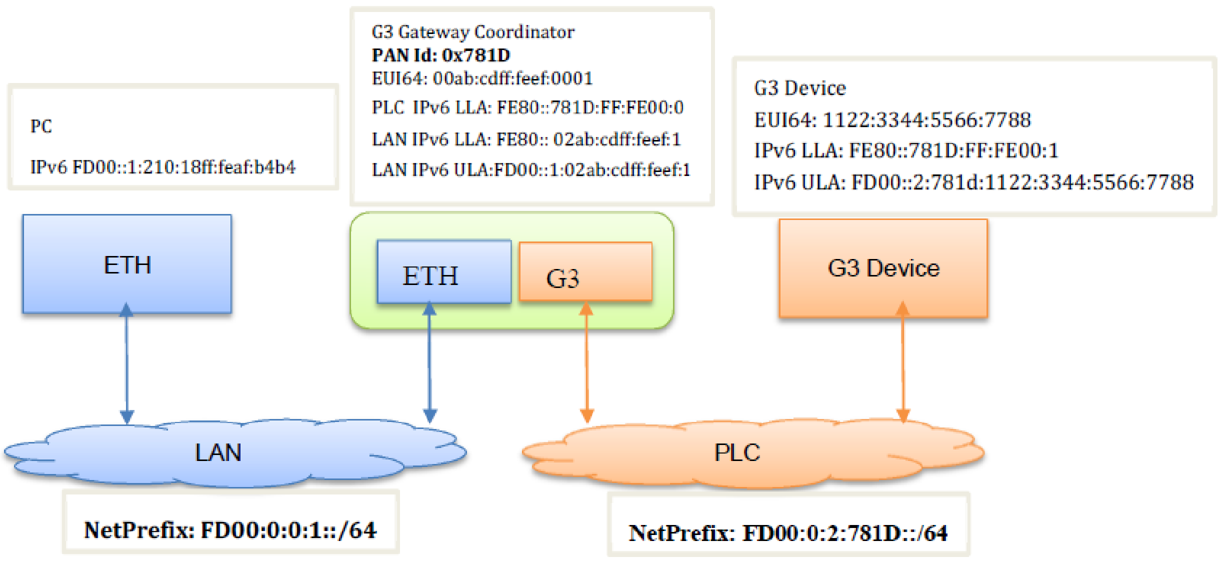
Figure 5-1. Meter and Modem State Machines



5.1.3 Gateway App

This application example implements a G3 coordinator with IPv6 routing capabilities. The gateway can be accessed via IPv6 and it acts as a bridge with the G3 network. For example, it is possible to perform ICMPv6 pings to G3 nodes from an external device, where such an external device speaks only IPv6 and does not know there is a G3 network below.

The following figure shows the concept of a G3 Gateway. The addressing scheme was chosen arbitrarily, and it is only for illustrative purposes.


G3 Gateway Coordinator
PAN Id: 0x781D
EUI64: 00ab:cdff:feef:0001
PLC IPv6 LLA: FE80::781D:FF:FE00:0
LAN IPv6 LLA: FE80:: 02ab:cdff:feef:1
LAN IPv6 ULA:FD00::1:02ab:cdff:feef:1
PC
IPv6 FD00::1:210:18ff:feaf:b4b4
G3 Device
EUI64: 1122:3344:5566:7788
IPv6 LLA: FE80::781D:FF:FE00:1
IPv6 ULA: FD00::2:781d:1122:3344:5566:7788
ETH
ETH
G3
G3 Device
LAN
PLC
NetPrefix: FD00:0:0:1::/64
NetPrefix: FD00:0:2:781D::/64

Detailed information about Gateway and PC configuration is provided in 16. Gateway Configuration.

5.1.4 ADP & MAC Serialization App

The ADP and MAC serialization is an application example that brings access to the ADP, MAC and Coordinator API through a serial connection. This application is useful for users who want to make intensive test of the stack or want to run the upper layers in another CPU.

The application allows control of the G3 PLC Stack at different levels. The user can make use of the ADP API (standard access) or access directly MAC Wrapper API as a shortcut for some tests. Serialization is also available to the provided Coordinator module in case the user wants to control the Bootstrap phase on the Coordinator side.

Serialization is done by means of 12. Serialization with Embedded USI.

5.2 PLC PHY Examples

5.2.1 PHY Tester

The PHY Tester tool is an application example that allows checking of the complete performance of the Microchip G3 PHY Layer on PLC boards. This example requires a board and a PC tool (PLC PHY Tester tool), which has to be installed in the user's host PC to interface with the boards.

The PLC PHY Tester tool controls the devices by means of a USI serialization. Users can perform PHY communication tests to check the capabilities of the PLC transceiver.

The example provided offers the serial interface configured through a certain UART or USB (depending on platform) at 230400 bps by default; the user can change these settings by means of 3.3.17. Universal Serial Interface Configuration – conf_usi.h.

5.2.2 PHY Sniffer

The PHY Sniffer is an application example that uses the PHY layer to monitor PLC frames in the G3 network and send them via USI serialization. The PLC Sniffer PC tool is also provided to interface with the board and interpret the G3 frames. This example requires only one board, and a G3 network to be monitored.

The example provided offers the serial interface configured through a certain UART or USB (depending on platform) at 230400 bps by default; the user can change these settings by means of 3.3.17. Universal Serial Interface Configuration – conf_usi.h.

5.2.3 TX Console

Because of the timing restrictions in the connection with the PC, the PLC PHY Tester tool may present limitations in applications or tests that require a very short time interval between consecutive frame transmissions.

The PHY Tx Test Console is an application example that demonstrates the complete performance of the Microchip G3 PHY Layer avoiding timing limitations in the PC host. That way, users can perform more specific PHY tests (e.g., short time interval between consecutive frames).

The PHY Tx Test Console application offers an interface to the user by means of a command console. In this console, users can configure several transmission parameters, such as modulation, frame data length and time interval between frames.

The example provided uses the console UART as a serial interface, configured through a certain UART or USB (depending on platform) at 921600 bps by default; the user can change these settings by means of 3.3.3. Board Configuration – conf_board.h and 3.3.15. Serial USART Service Configuration – conf_uart_serial.h.

5.2.4 PHY PLC & Go

This example is intended to show a simple application running on top of the G3 PHY layer.

It presents a point-to-multipoint chat application on a Serial Console. When a message is written and sent from a device, all other devices receive it and print it on the Console.

The PLC & Go Application Note is available for download on the Microchip website and describes this example in more detail.

5.2.5 MAC RT PLC & Go Plus

This example is intended to show a simple application running on top of the MAC RT layer when it is running on PL360.

It presents a point-to-multipoint chat application on a Serial Console. When a message is written and sent from a device, all other devices receive it and print it on the Console.

This example is available only for PL360 platform.

The PLC & Go Plus Application Note is available for download on the Microchip website and describes this example in more detail.

CAUTION **Notice:** This is not exactly a PHY application, and thus is not located under the '/thirdparty/g3/phy/atpl360/apps', but under '/thirdparty/g3/apps' instead.

5.3 RF PHY Examples

5.3.1 Hybrid PLC & RF PHY Tester

The Hybrid PLC & RF PHY Tester tool is an application example that allows checking of the complete performance of both Microchip G3 PLC & RF PHY Layers. This example requires a board and a host PC to interface with, where the user will run test scripts on top of a serialization library provided by Microchip.

Test scripts control the devices by means of a USI serialization. Users can perform PHY communication tests to check the capabilities of the selected transceiver, PLC or RF.

The example provided offers the serial interface configured through a certain UART or USB (depending on platform) at 230400 bps by default; the user can change these settings by means of 3.3.17. Universal Serial Interface Configuration – conf_usi.h.

5.3.2 Hybrid PLC & RF Sniffer

The Hybrid PLC & RF Sniffer is an application example that uses both RF and PLC PHY layers to monitor frames in a G3 Hybrid network and send them via USI serialization. The Microchip PLC Sniffer Tool, from version 2.0.3, is able to decode both PLC and RF frames, and is also provided with the example. This example requires only one board, and a G3 network to be monitored.

In case the board used has only one of the interfaces available, the example will run as a single medium sniffer, showing only PLC or RF frames. The same applies if it is used on a legacy PLC Network instead of a Hybrid Network, where only PLC frames will be shown.

The example provided offers the serial interface configured through a certain UART or USB (depending on platform) at 230400 bps by default; the user can change these settings by means of 3.3.17. Universal Serial Interface Configuration – conf_usi.h.

5.4 Clock Configuration Example for Low Requirements Module

As stated in chapters above, provided examples are configured to take full advantage of Evaluation Boards capabilities, ignoring aspects such as power consumption.

Microchip Kits mount external oscillators to provide CLK signal to their microcontrollers. By means of PLLs, the work frequency of the microcontroller is set to its maximum (or close to it) in provided examples, to get the highest performance as possible. But this is not a requirement of the G3 stack itself.

This chapter provides the changes in configuration files needed to be able to work at a lower frequency, using one of the internal RCs available. This leads to a decrease in Power Consumption and the possibility to remove the external crystal oscillator, both desired aspects in case a simpler solution is the target. For illustration purposes, a SAMG55-based platform is chosen to present the changes in configuration files.

Changes in conf_clock.h File

By default, projects are configured to use an external oscillator and a PLL to raise working frequency. In addition, a second PLL is used to generate the USB clock:

```
#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_PLLACK
#define CONFIG_SYSCLK_PRES          SYSCLK_PRES_1

/* Configure PLL0 to get MCK at 96 MHz */
#define CONFIG_PLL0_SOURCE          PLL_SRC_SLCK_XTAL
#define CONFIG_PLL0_MUL             2930
#define CONFIG_PLL0_DIV             1

/* Configure PLL1 to get USB CLK at 48 MHz */
#define CONFIG_PLL1_SOURCE          PLL_SRC_SLCK_XTAL
#define CONFIG_PLL1_MUL             1465
#define CONFIG_PLL1_DIV             1

#define CONFIG_USBCLK_SOURCE        USBCLK_SRC_PLL1
#define CONFIG_USBCLK_DIV           1
```

The alternative configuration to use the internal 24 MHz RC is the following:

```
#define CONFIG_SYSCLK_SOURCE        SYSCLK_SRC_MAINCK_24M_RC
#define CONFIG_SYSCLK_PRES          SYSCLK_PRES_1
```

No PLLs are used, as 24 MHz internal RC does not support connection to PLL, it is used directly as MCK; and USB CLK is not configured, as it requires 48 MHz to work properly.

Changes in conf_pplc_if.h file

Reducing MCK to 24 MHz has an impact on default SPI speed configuration. SPI uses DMA in the provided examples, and this limits the speed at which the SPI clock can be configured, related to MCK.

Default SPI speed is configured at 8 MHz in provided examples:

```
#define PPLC_CLOCK          8000000
```

But this has to be reduced when MCK is set to 24 MHz, so this definition has to be changed to:

```
#define PPLC_CLOCK          4000000
```

Working at 4 MHZ on SPI will make transactions take more time, but correct communication is ensured at this rate.

With this Clock and SPI Configuration, we have a fully functional G3 Stack running at 24 MHz, SPI transactions at 4 MHz, and without USB support, which is a perfectly valid approach for a PLC module with no extra requirements.

6. Microchip Implementation Particulars

This section describes some particularities of the Microchip G3 implementation, mainly referring to Physical Layer capabilities and features which make the difference between this product and others.

6.1 Different G3 Specification Availability

In April 2017, a new G3 Specification was released. It is interoperable functionally speaking with the previous one (from March 2015); however, the new specification introduces some differences in IBs at ADP and MAC layers, which require a slightly different integration with applications using the G3 stack.

PHY layer is common for both specifications, so applications running directly over PHY layer do not have to care about this.

Microchip G3 stack libraries and modules have to be initialized to be compliant with one of specifications. By default, the provided examples are configured to use Spec'17. To use the Spec'15 version, please do the following:

Open the file *conf_global.h* (3.3.19. Global Configuration Affecting all Projects – conf_global.h) and replace the default configuration:

```
#define SPEC_COMPLIANCE  17
/* #define SPEC_COMPLIANCE  15 */
```

with

```
/* #define SPEC_COMPLIANCE  17 */
#define SPEC_COMPLIANCE  15
```

The SPEC_COMPLIANCE compilation constant is used by libraries and modules to configure themselves at start-up and then dynamically behave as expected.

6.2 Table Sizes and Buffer Count Configuration

To improve G3 stack configuration capabilities and for user convenience, some G3 stack tables and buffer counts can be modified and set at compilation time. To do so, they are declared in auxiliary source files.

The configuration of these parameters depends on both user needs and on memory availability in the target platform.

These values can be set in the *conf_tables.h* file, present in all provided example applications.

The table sizes and buffer count configured in this file are the following:

Number of ADP Transmission Buffers

ADP defines a structure of buffers for transmission of three different sizes: 1280 bytes, 400 bytes and 100 bytes.

The reason for these buffer sizes are the following:

- 1280 bytes is the maximum size of an IPv6 packet coming from the application layer.
- 400 bytes is the maximum size that G3 MAC layer accepts. Typically used for short application IPv6 frames.
- 100 bytes is enough for most frames coming from inside the G3 stack, such as LOADng or LBP frames.

These buffer sizes are fixed; however, the number of buffers for each size is configurable through definitions. Default configuration is the following, depending on the Node type:

Coordinator

```
#define CONF_COUNT_ADP_BUFFERS_1280             1
#define CONF_COUNT_ADP_BUFFERS_400              8
#define CONF_COUNT_ADP_BUFFERS_100              8
```

Device

```
#define CONF_COUNT_ADP_BUFFERS_1280             1
#define CONF_COUNT_ADP_BUFFERS_400              3
#define CONF_COUNT_ADP_BUFFERS_100              3
```

Number of Simultaneous Fragmented Receptions
When receiving a fragmented frame, the ADP layer tracks fragments in a table for further reassembly. This table size, thus, defines the number of simultaneous fragmented frames that can be processed. Default configuration is the following, depending on the Node type:

Coordinator

```
#define CONF_ADP_FRAGMENTED_TRANSFER_TABLE_SIZE    3
```

Device

```
#define CONF_ADP_FRAGMENTED_TRANSFER_TABLE_SIZE    1
```

LOADng Routing Tables
The Routing Table and Routing Set functionality is well-defined in the G3 specification (and beyond the scope of this guide), but not the size that these tables have to have. In addition, the number of simultaneous Route Requests where a Route Reply has to be issued can also be configured. Default configuration is the following, depending on the Node type:

Coordinator

```
#define CONF_ADP_ROUTING_TABLE_SIZE                2000
#define CONF_ADP_ROUTING_SET_SIZE                  400
#define CONF_LOADNG_RREP_GENERATION_TABLE_SIZE     10
```

Device

```
#define CONF_ADP_ROUTING_TABLE_SIZE                150
#define CONF_ADP_ROUTING_SET_SIZE                  30
#define CONF_LOADNG_RREP_GENERATION_TABLE_SIZE     3
```

MAC Layer Tables
Neighbour Table and POS Table functionality is well-defined in the G3 specification (and beyond scope of this guide), but not the size that these tables have to have. In addition, the Data Sequence Number Table, used to avoid processing duplicate frames, can also be configured in size. Default configuration is the following, depending on the Node type and Spec Compliance:

Coordinator

```
#if (SPEC_COMPLIANCE == 15)
  #define CONF_MAC_NEIGHBOUR_TABLE_ENTRIES       (2000)
  #define CONF_MAC_POS_TABLE_ENTRIES             (1)
#else  /* SPEC_COMPLIANCE >= 17 */
  #define CONF_MAC_NEIGHBOUR_TABLE_ENTRIES       (500)
  #define CONF_MAC_POS_TABLE_ENTRIES             (2000)
#endif
#define CONF_MAC_DSN_SHORT_TABLE_ENTRIES         (256)
#define CONF_MAC_DSN_EXTENDED_TABLE_ENTRIES      (16)
```

```
#define CONF_MAC_POS_TABLE_ENTRIES_RF            (500)
#define CONF_MAC_DSN_TABLE_ENTRIES_RF            (16)
```

Device

```
#if (SPEC_COMPLIANCE == 15)
  #define CONF_MAC_NEIGHBOUR_TABLE_ENTRIES       (75)
  #define CONF_MAC_POS_TABLE_ENTRIES             (1)
#else  /* SPEC_COMPLIANCE >= 17 */
  #define CONF_MAC_NEIGHBOUR_TABLE_ENTRIES       (35)
  #define CONF_MAC_POS_TABLE_ENTRIES              (100)
#endif
#define CONF_MAC_DSN_SHORT_TABLE_ENTRIES          (128)
#define CONF_MAC_DSN_EXTENDED_TABLE_ENTRIES       (16)
```

```


#define CONF_MAC_POS_TABLE_ENTRIES_RF             (100)
#define CONF_MAC_DSN_TABLE_ENTRIES_RF             (8)
```

IPv6 Stack Buffer Count
Although outside of the G3 stack scope, the number of buffers used in the IPv6 stack can also be configured here for unification purposes. Default configuration is the following, depending on the Node type:

Coordinator

```
#define CONF_NET_MEM_POOL_BUFFER_COUNT           16
```

Device

```
#define CONF_NET_MEM_POOL_BUFFER_COUNT           4
```

Nodes Supported at DLMS Level
Also outside of the G3 stack scope, the number of devices supported by a DLMS client can also be configured here. On Coordinator and Gateway applications, such a DLMS Client is provided, so DLMS queries can be sent to Devices to simulate a real Data traffic in a typical network. The number of DLMS Servers that can be supported by the provided client is configurable:

```
#define CONF_DLMS_MAX_DEV_NUM                    250
```

Routing and Neighbour Table Snapshots at DLMS Level
Also outside of the G3 stack scope, some tables have a mirrored snapshot on the provided DLMS server, which can also be configured here. Devices and Gateways have a snapshot of their Routing and Neighbour Tables to provide information upon certain DLMS queries. Default configuration is the following, depending on the Node type:

Gateway

```
#define CONF_MAX_NEIGHBOUR_DLMS_SNAPSHOT_SIZE    500
#define CONF_MAX_ROUTING_DLMS_SNAPSHOT_SIZE      2000
```

Device

```
#define CONF_MAX_NEIGHBOUR_DLMS_SNAPSHOT_SIZE    75
#define CONF_MAX_ROUTING_DLMS_SNAPSHOT_SIZE      150
```

6.3 Impedance Detection, Transmission States and Equalization

6.3.1 What are Transmission States?

Transmission States are configurations applied to the PHY layer in order to obtain the desired signal level depending on the user needs.

Depending on impedance, transmission parameters change in order to obtain the better performance/efficiency from the output driver.

Three different states are defined for this purpose:

- 0 - HI_STATE. Transmission is configured to comply with normative with the best performance/efficiency assuming there is a high-impedance (imp > 20 Ohm), either auto-detected or fixed by configuration.
- 1- LO_STATE. This is an intermediate state suitable for all impedance ranges when the required transmission level is low (controlled by upper layers).
- 2 - VLO_STATE. Transmission is configured to comply with normative with the best performance/efficiency assuming there is a very low-impedance (imp < 10 Ohm), either auto-detected or fixed by configuration.

The user may force one of these states on the PHY layer or use the recommended operation mode, which is Automatic Impedance Detection and Adaptation.

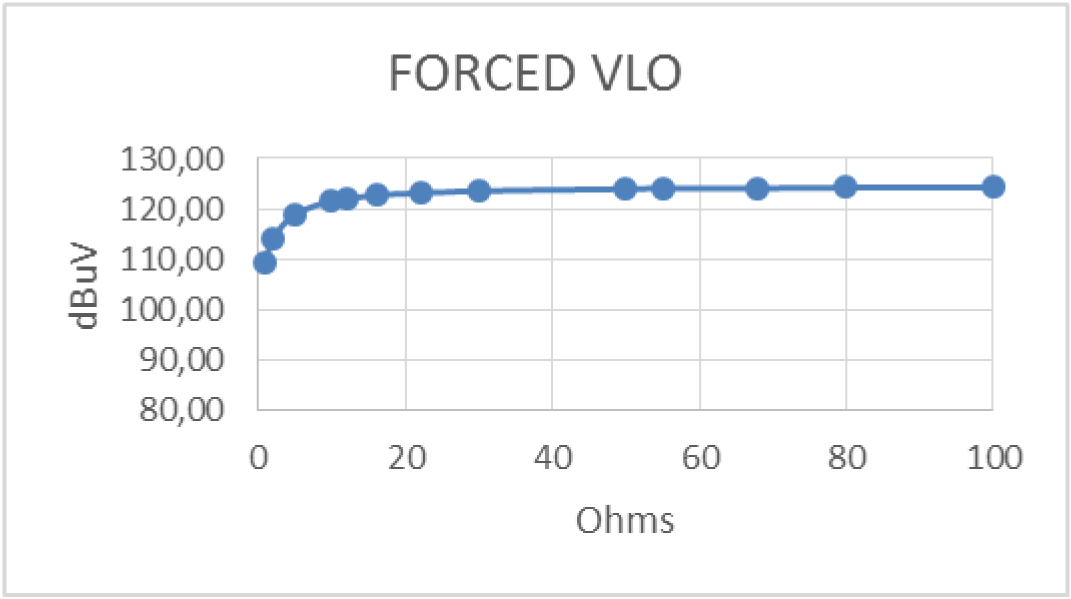
6.3.2 Forced Transmission Modes

As stated before, transmission modes are optimized for an impedance range, and out of such range, they do not work properly in terms of performance/efficiency.

Let us suppose we have set a signal injection objective of 110 dBuV for both HI and VLO modes.

The following figures show the response of forced modes in an impedance range from 0 to 50 ohms.

Figure 6-1. Forced Transmission Modes vs. Impedance



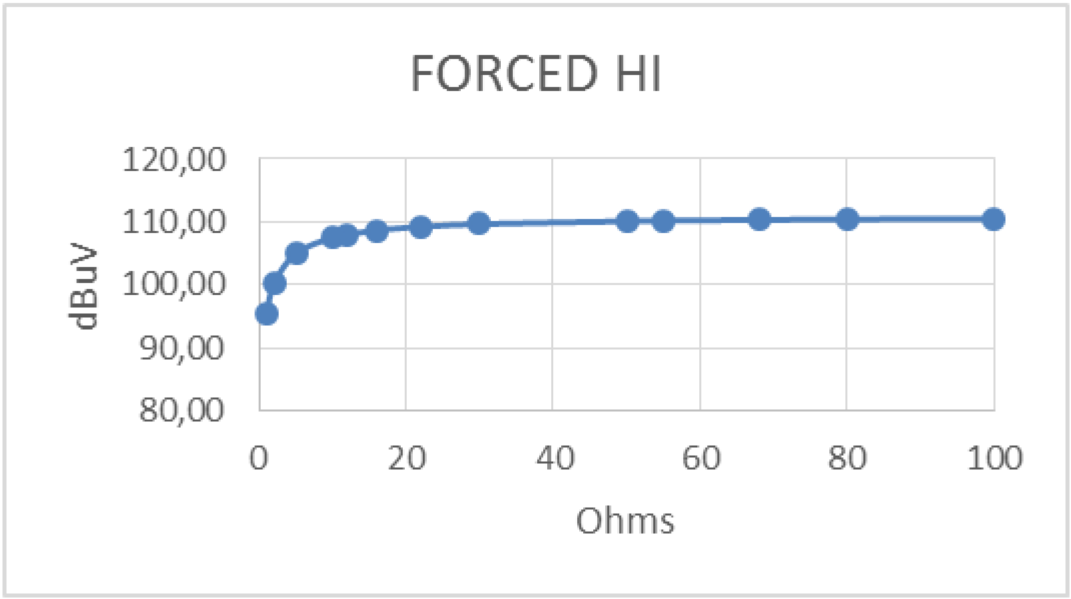


Figure shows that each mode is injecting the desired signal in its intended range, but out of such range, each mode is not achieving the injection objectives. Also can be seen that the response has a much more stable trend for impedance above 20 ohms, and below that, the transmission driver is more dependent on impedance detected.

Figures show the response when both State and Gain are Fixed. There is also a configuration where State is fixed, but slight changes in Gain are allowed (see 7.3.26. PHY_ID_CFG_AUTODETECT_IMPEDANCE (0x0161) for details), but such slight changes will not be enough to meet injection objective.

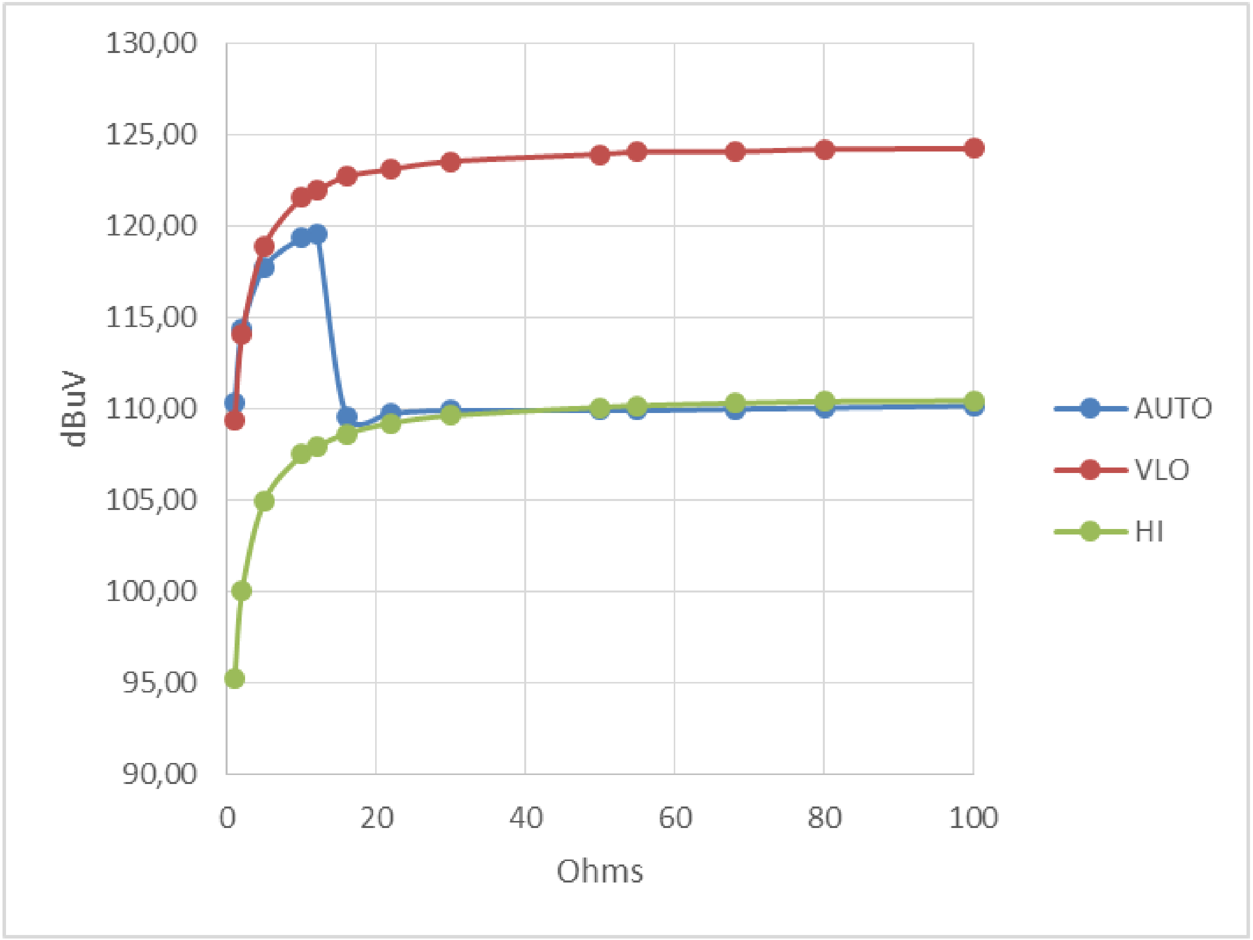
To meet objective in all impedance range (with some limitations in low impedance as explained above), we have to rely on Impedance Detection and Adaptation algorithm implemented in the Physical layer.

6.3.3 Impedance Detection and Adaptation Algorithm

The Physical layer implements an algorithm that monitors the output signal level and infers the impedance value against what it is transmitting to. Depending on the inferred impedance, the Physical layer sets the proper transmission state to obtain the desired output level. Enabled or not, this detection and adaptation is controlled through the PIB *PHY_PARAM_CFG_AUTODETECT_BRANCH* (see PIB for details about auto-detect configuration).

The following figure shows the response in AUTO_STATE_VAR_GAIN (1) mode, compared to the forced modes described before:

Figure 6-2. Impedance Adaptation vs. Forced Modes



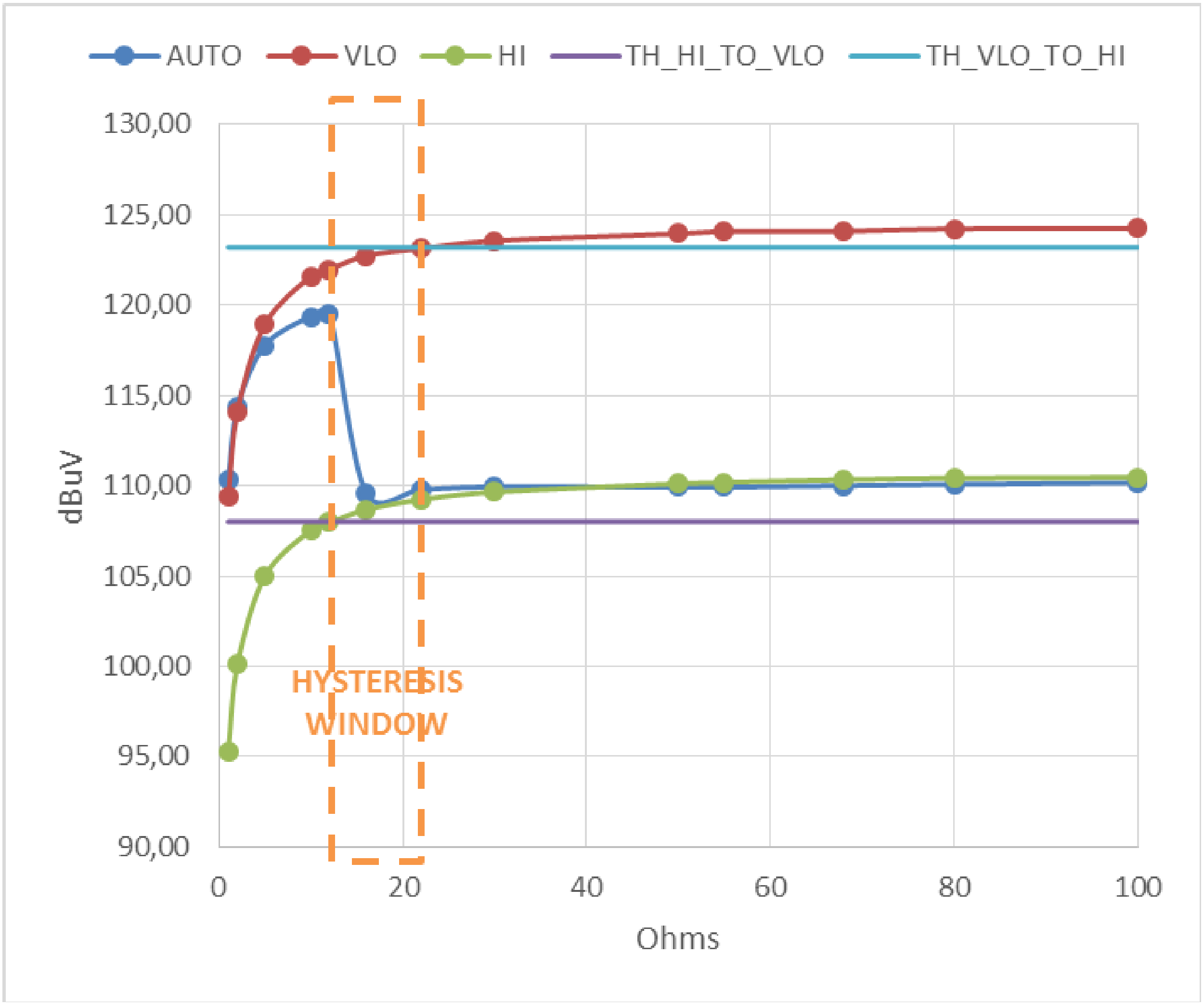
This adaptation is achieved by defining thresholds to switch from HI to VLO mode, and vice versa, which determine the points where transmission state is changed.

These thresholds are defined such that the switch from HI to VLO is done in a lower impedance than the switch from VLO to HI, obtaining a hysteresis window to avoid continuous switching between states which will lead to a continuous change in signal injection, which is an undesirable scenario.

The usage of LO mode is reserved to transmission with a forced attenuation, as explained later in this section.

The following figure illustrates such thresholds and the hysteresis window they achieve:

Figure 6-3. Thresholds and Hysteresis Window



As shown in the figure, the Auto response was obtained by sweeping from high to low impedance; this is why the switch is done near the lower value of the hysteresis window (some point between 10 and 15 ohms). If the sweep is done in the opposite way, from low to high impedance, the transition will be done near the higher value of the hysteresis window. Therefore, the curve will be slightly different inside such window. It will follow the VLO curve inside the window, and, then, switch to the HI curve just above 20 ohms.

It also shows that the Auto curve does not follow the exact trend of the forced ones. This is because the algorithm performs a fine tuning in each transmission according to the detected signal value, trying to adjust more closely to the signal injection objectives.

In this algorithm, the LO state is reserved to be used when the upper layers set an attenuation value (which in G3 is determined by the Neighbour Table 9.7.25. MAC_WRP_PIB_NEIGHBOUR_TABLE (0x0000010A) information in TxGain field) greater than a certain value (determined by the transceiver used), where the algorithm does not have enough resolution to properly determine the switching between states. Thus, it uses the LO state, which is a static state, whose configuration is aimed to this high attenuation circumstance, and considered valid in all the impedance range. As soon as the upper layer requests a transmission with less attenuation, the algorithm will be back to the mode switching between HI and VLO depending on the inferred impedance.

6.3.4 Equalization

Usually the output transmission driver has a non-flat response inside the working band (ripple). Depending on the difference between carriers, we may fall in a response out of the specification which has to be corrected to meet constraints.

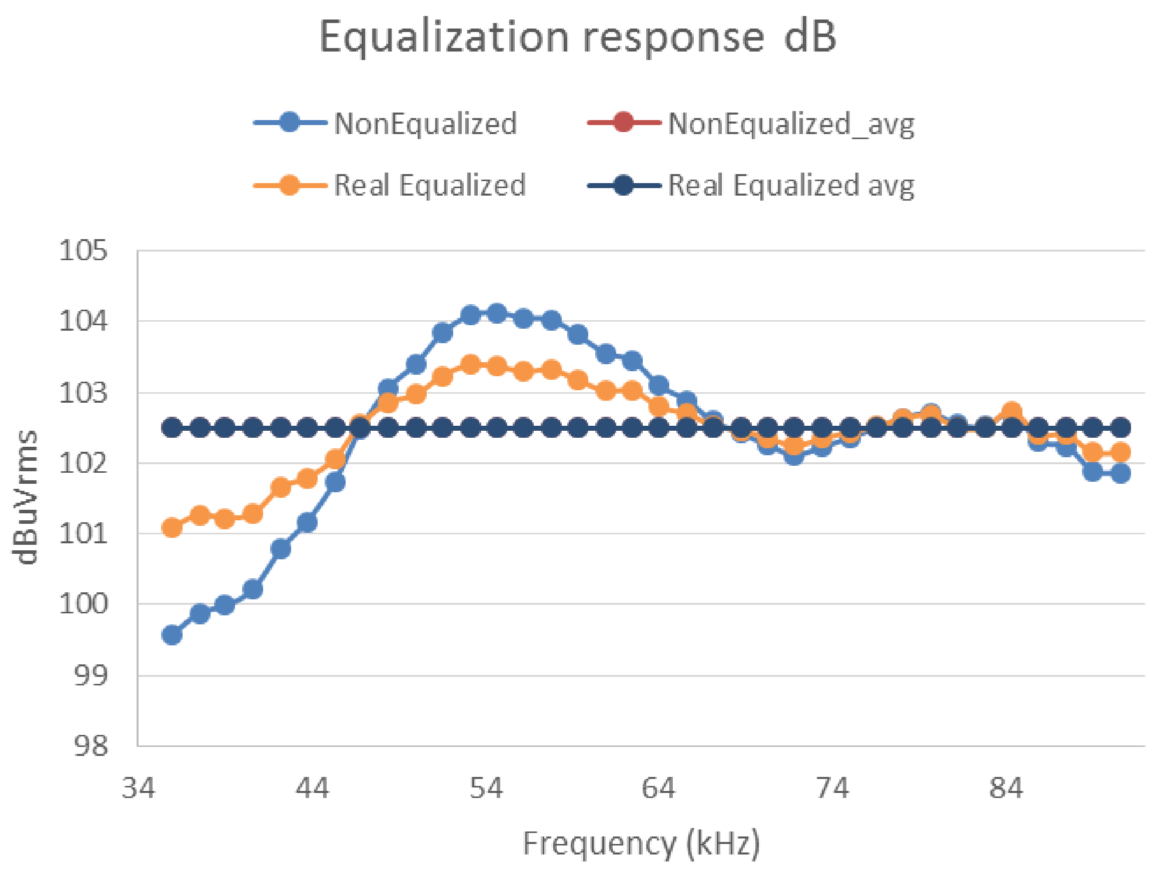
The Physical layer is able to perform this equalization by means of signal pre-distortion, which compensates the effect of the external driver, reducing the ripple, to give a final response closer to a flat one.

Equalization has a tight relation with transmission states seen in previous sections; in fact, each transmission state defines a particular equalization. As configurations are different for each state, the response is also different, which requires different equalization.

An important aspect to take into account is that the objective of the equalization is to reduce ripple, giving a flatter response, but keeping the average signal injection. Therefore, care must be taken in order to reduce signal in some carriers and increase it in others, resulting in the same response after integrating the signal level inside the operation band.

The following figure shows an example of equalization in the G3 Cenelec-A band:

Figure 6-4. Equalization Example



The figure shows how the ripple is reduced after equalization, keeping the same signal average in band (overlapped curves).

6.3.5 Calibration of Signal Level and Equalization

In the provided Physical layer, the transmission level and equalization parameters are configured to work properly with the provided HW (EKs).

The final user may need to calibrate all these parameters when working on their final HW. This is not a trivial step, so Microchip provides a tool which will perform the necessary tests and obtain the values that the user will overwrite in the provided Physical layer in order to meet expectations, after requesting that the user introduce the desired targets in terms of signal level and ripple.

The tool does currently support the PL360 platform. Calibration on the ATPL250 platform is still possible, but will require manual operation and monitoring.

The process to follow and the tool usage not within the scope of this guide, and will be provided in a separate document.

6.4 ZC Offset Configuration

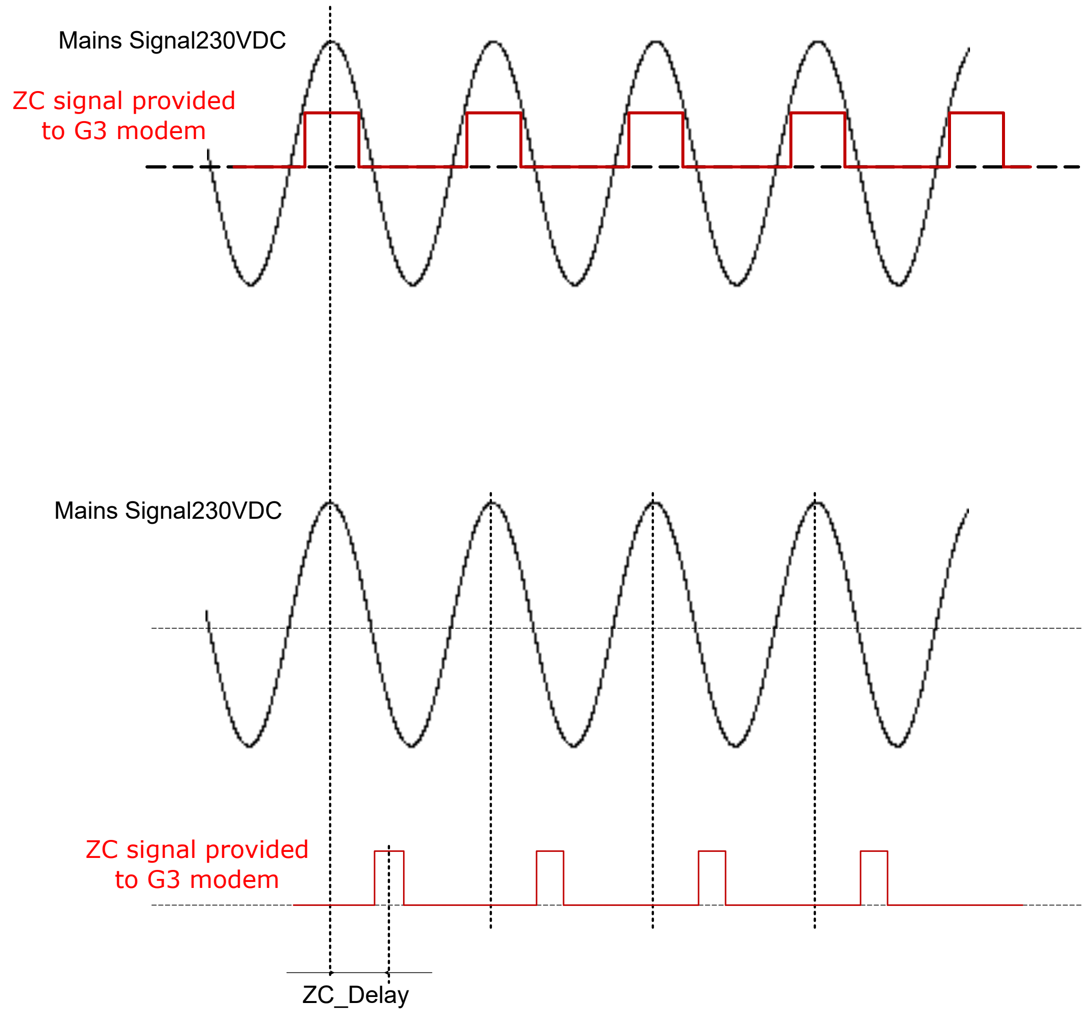
The PHY layer calculates the electrical phase difference between transmitter and receiver for a given frame. To do so, both transmitter and receiver have to be able to read the Zero-Cross time of the mains.

Both ATPL250 and PL360 have a dedicated input pin for connection to a signal which provides such mains Zero-Cross. Due to design reasons, the signal on the input pin may not be fully synchronized with the real mains Zero-Cross.

The following figure shows two different cases of the ZC signal provided to the modem.

- The first one shows a typical response of a Zener-based circuit to detect Zero-Cross, where the highest point of mains signal and the middle point of the pulse are aligned. In this case no configuration is needed, as there is no delay between these points.
- Below is an example where the provided ZC signal comes from an external device, which may have a delay in generating the pulse, and whose pulse width is not related to mains frequency. The figure shows the time that has to be measured to be set as ZC delay in the modem parameters, so further calculations refer to the real mains ZC.

Figure 6-5. ZC Delay Calculation


Mains Signal230VDC
ZC signal provided to G3 modem
Mains Signal230VDC
ZC signal provided to G3 modem
ZC_Delay

The way to configure this ZC delay is different for ATPL250 and PL360 modems.

For ATPL250, a compilation constant is provided in file *conf_fw.h*:

```
#define CONF_ZC_OFFSET_CORRECTION    (uint32_t)0
```

For PL360, this is configured via PIB (ATPL360_REG_ZC_CONF_DELAY, ATPL360_REG_ZC_CONF_INV). Please refer to the PL360 Host Controller documentation for more details.

The default value of this parameter for both platforms is configured according to Microchip EK boards requirements. The user can change it in its implementation if necessary.

6.5 Noise Detection and Compensation

On both the ATPL250 and PL360 platforms, Microchip G3 stack is able to detect and compensate narrow band and impulsive noises. As both detection and compensation are different depending on noise type, they will be explained in different sections.

6.5.1 Narrow Band Noise

Narrow band noise is continuously monitored in the line, at a given time rate with some exceptions affecting this period.

This functionality can be enabled/disabled through IB PHY_PARAM_ENABLE_AUTO_NOISE_CAPTURE. It is enabled by default.

The periodicity is set through PHY_PARAM_TIME_BETWEEN_NOISE_CAPTURES IB. The default value is different in PL360 (1 second) and ATPL250 (60 seconds) platforms. This is due the following: While capturing narrow band noise in ATPL250 we are not able to receive a frame for 10 milliseconds, so capture is spaced several seconds (60 by default) to avoid frame reception loss. In addition, as typical traffic in channel uses to imply several frames in a short time, there is a functionality, controlled by PHY_PARAM_DELAY_NOISE_CAPTURE_AFTER_RX IB, which delays the next noise capture after a frame is received, this way, we ensure no capture is triggered during a burst of frames in the line. PL360 does not have these limitations in the capture; if a frame is detected, the capture is automatically aborted and the frame is received. For this reason, periodicity and capture delays are treated in a different way. Please refer to PL360 host controlled documentation for details.

The compensation of narrow band noise, if detected, is the same in both platforms. A notch filter is configured in the same frequency as narrow band noise is detected, to reduce its amplitude in reception stage.

The information regarding whether a notch filter is configured and in which carrier can be retrieved by upper layers by means of PHY_PARAM_RRC_NOTCH_ACTIVE and PHY_PARAM_RRC_NOTCH_INDEX IBs.

The effect of narrow band noise in the MAC layer may be a sub-band cancellation in the next Tone Map Response, but this decision is not fixed. It will be evaluated by the Tone Map decision algorithm depending on SNR in the carriers near the narrow band noise. For more information about Tone Map Response functionality, please refer to G3 Specification.

6.5.2 Impulsive Noise

Impulsive noise detection and compensation is done in the same way in both the PL360 and ATPL250 platforms.

It is monitored during frame reception, where SNR of different symbols are compared to decide whether an impulsive noise is present. This functionality is always present and cannot be configured, as it has no drawbacks in frame reception itself.

The compensation of impulsive noise will be done by MAC layer in the following Tone Map Responses.

On the ATPL250 platform, the algorithm will force the ROBO modulation choice, as it is the only one that behaves correctly against this type of noise.

On the other hand, PL360 applies different techniques to improve robustness against impulsive noise that may not require the use of ROBO modulation, but a combination of modulation/tonemapping.

For more information about Tone Map Response functionality, refer to G3 Specification.

6.6 Power-Down Detection

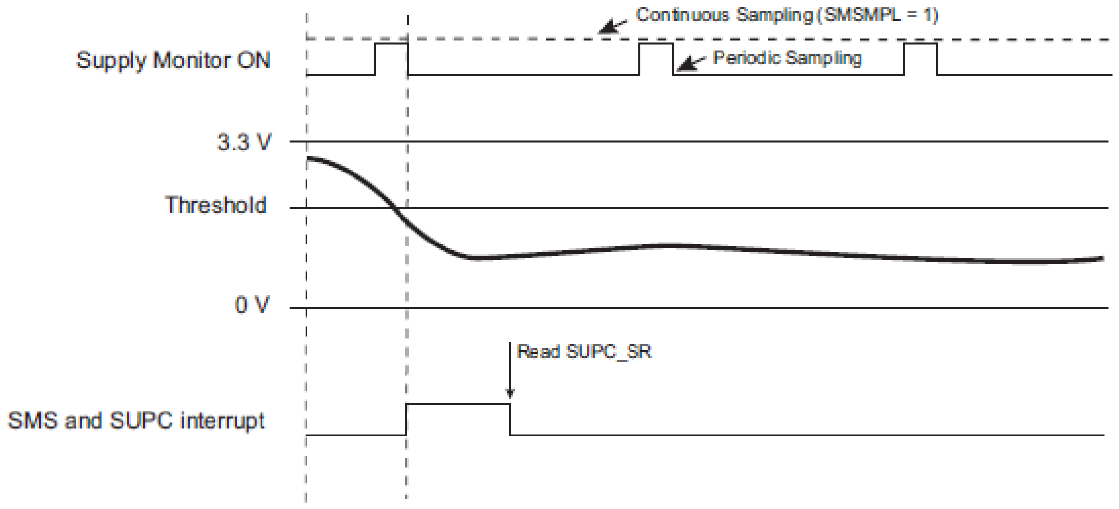
Requirements
Applications using the G3 stack need to be aware of when a device is powered down, in order to store some information before power is completely lost.

This section explains the Power-Down detection approach used in the G3 examples, although other alternative methods are presented as well. Regarding the data storage, please refer to 3.4. Persistent Data Storage.

Approach 1. Supply Monitor.

The default configuration of the provided G3 examples makes use of the Supply Controller, a HW block present in all SAM devices, which can be configured to trigger an interrupt when a certain (configurable) threshold is reached.

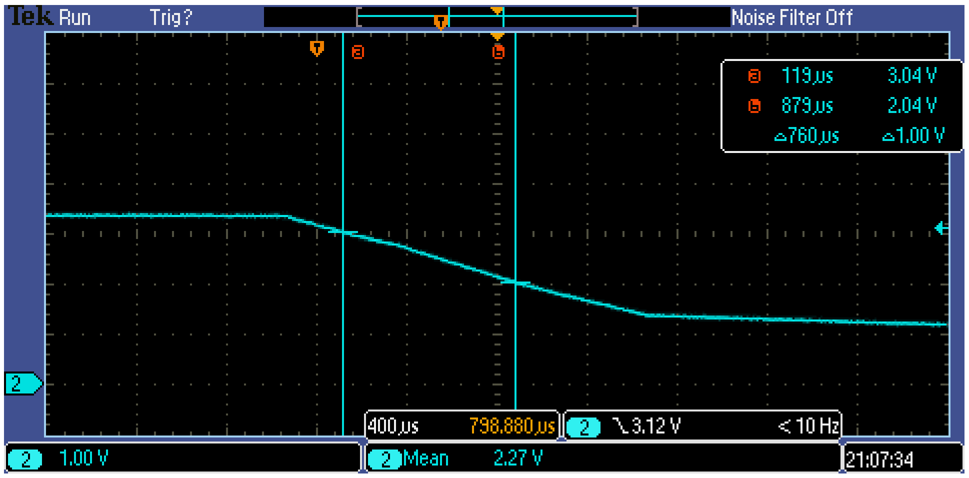
Figure 6-6. Supply Monitor



The default configuration in the examples is Continuous Sampling and Threshold = 3V. The response time of this approach (Continuous Sampling) is one cycle of SLCK (32kHz), which gives 31.25 µs.

The supply monitor is the fastest solution in terms of response time, and the less CPU consuming option, as everything is managed by HW until interrupt is triggered. The drawback is that the time to perform the storage is short. The following capture shows the 3V3 falling slope measured in a PL360G55CB_EK supplied by USB, which is considered the worst case.

Figure 6-7. PL360G55CB EK 3V3 Fall



The capture shows that, from Power-Down detection on 3V, there are about 900 µs until voltage falls to 2V. The time required to store data in User Signature plus the worst case in detection time sum up to 300 µs.

⚠ CAUTION This time measurements and constraints are based on Michochip EKs and the selected storage implementation. Any change in software or hardware has an impact and the validation of the approach is required.

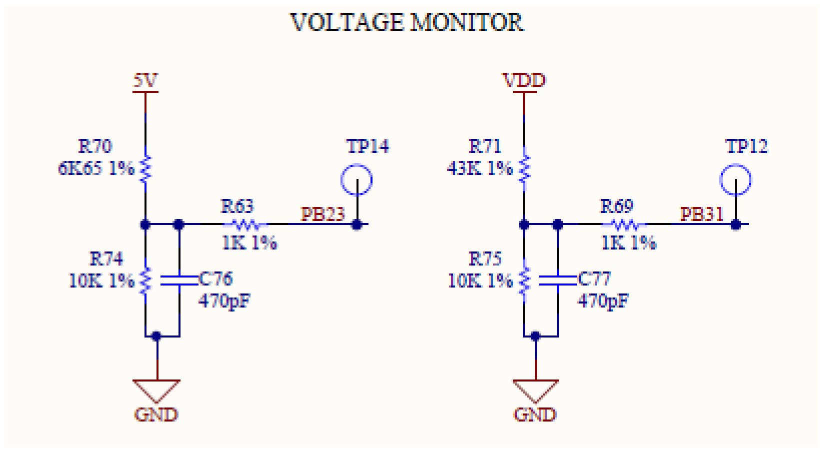
The conclusion is that this approach is OK for the provided examples in the available EKs, but all this information regarding times and voltages has to be taken into account when working on a new design, both HW and SW.

Approach 2. External Voltage Divider.
An alternative method, available in some Microchip PLC EKs, is to have a voltage divider connecting one of the power supplies of the board to a microcontroller GPIO with ADC capabilities.

Knowing the ADC value in the range of valid power supply voltages, a threshold is defined where it is estimated that the voltage has reached a value low enough to consider that the supply is about to fail (Power-Down).

In the provided implementation, the ADC is connected to a Timer Counter peripheral which periodically triggers ADC Conversions, which output value is compared to the threshold to detect whether power is falling down.

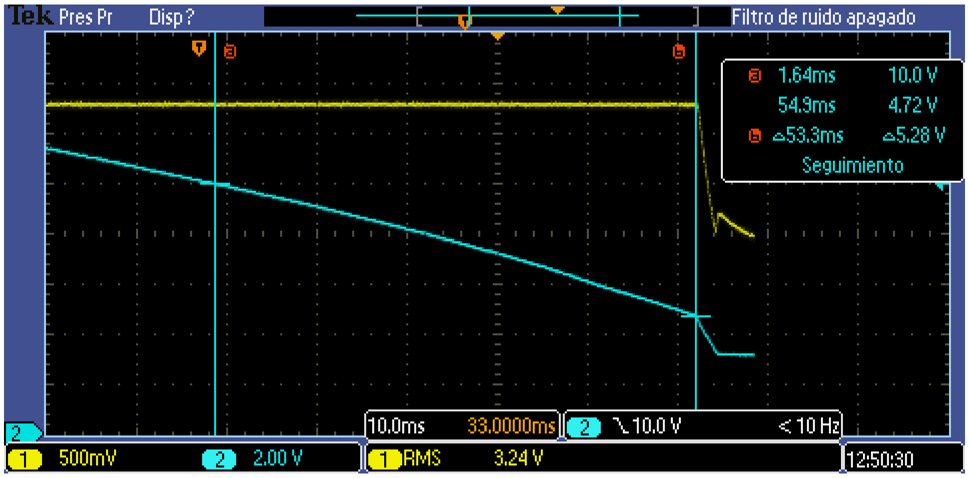
Figure 6-8. Voltage Divider



As seen in the picture, taken from ATPL250-EK documentation, 2 options are provided, connected to 5V and 12V (VDD) respectively. The provided implementation uses the VDD option to detect power fail.

The response time in this case depends on the rate at which the timer is configured to trigger conversions. The default rate is configured to a conversion every 10 ms, which is thus the response time. Although this may look like a poor response time, it may be fast enough in designs where VDD and 3V3 supplies are mounted in cascade (such as in ATPL250_EK) and the capacitors in VDD provide an extra time before the 3V3 supply is affected by the Power-Down.

Figure 6-9. ATPL250AMB 12V to 3V3 Fall



The picture shows the instant when the ADC output value is below the threshold (corresponding to 10V in VDD in the blue line), and the time when the 3V3 supply is affected (yellow line falling). Time between these events is about 55 ms, this is why a 10 ms response time is enough.

This option, having a worse response time, and being more CPU consuming (each ADC conversion triggers an interrupt where threshold is compared by SW) than approach 1, could still be the better option in cases where tasks to perform on Power-Down take more time, as we are in the range of milliseconds (instead of microseconds as in approach 1).

Alternative Approaches.
The provided examples offer the 2 approaches explained above, but the decision on how to detect the Power-Down is up to the final user, and different alternatives can be used.

It is possible that the design where G3 will be integrated has also a means of detecting Power-Down due to external needs. Then, the circuitry detecting the Power-Down can be connected to the microcontroller running G3 and configure an interrupt in such PIO which will then be in charge of the data storage.

Or even the SW solution of which G3 is a part has already a routine to perform some tasks at Power-Down, whatever the means of detection. Then the G3 Data Storage can be done along with such tasks, and no dedicated Power-Down detection will be necessary for G3.

Implementation and Configuration
The implementation of the presented approaches can be found in the HAL (Hardware Abstraction Layer) module in the provided G3 examples, as it is platform-dependent and implementations may differ depending on the microcontroller used, or the capabilities of the external circuitry.

The configuration of the method used and its corresponding parameters is done in the *conf_hal.h* file of the examples.

Any change that final user may test that is related to available approaches, or implementation of new ones, will be done in those files.

6.7 Voltage and Thermal Monitor for PLC Driver

On the PL460, being a device with an integrated PLC Tx Driver, it is important to ensure that the supply voltage level of the amplification stage is within the expected range.

To ensure this, the provided examples implement a voltage monitor using an analog input in the host micro-controller, which checks if the voltage is out of range, and, in case it is detected, uses an output pin connected to PL460 to indicate that transmission is not allowed. This is done by sampling the voltage and comparing it to determined thresholds (which depend on the circuitry used for the Monitor). When transmission is disabled by the pin, the PL460 aborts any transmission in progress or programmed, and new transmission requests are denied. This requires 2 PIOs in the Host microcontroller, one Analog Input to sample the voltage, and one Digital Output to indicate to the PL460 that transmission is not allowed.

The PL460 also implements a thermal monitor, which indicates through a Pin if the Die Temperature is above 110ºC. The provided examples configure a Digital Input in the host microcontroller to detect this temperature limit and, thus, not request more transmissions to the PL460 until the temperature is back to safe levels. As stated, this requires an additional PIO in the microcontroller configured as a digital input to get the thermal warning from the PL460. If this pin is not used by the host microcontroller, the PL460 has another internal pin that indicates if the die temperature is above 120ºC to disable transmissions in extreme temperature conditions.

For more details on the required PIOs and their configuration, refer to the 3.3.10. PPLC Interface Configuration – conf_ pplc_if.h file on any of the provided examples for the PL460 platform.

7. API of PHY Layer (ATPL250 Platform)

The PHY layer described here applies only to ATPL250 platform.

PL360 does not require a PHY layer in the Host microcontroller, it is self-contained in PL360 device, and access to it is managed by the Host Controller component. For further information regarding this component, please refer to the PL360 Host Controller documentation.

7.1 PHY SAP

7.1.1 PHY Initialization Function

G3-PLC PHY Layer must be always initialized when the device starts the execution. The following function initializes the physical parameters and configures the transceiver device:

```
void phy_init(uint8_t uc_serial_enable, uint8_t uc_band);
```

Parameters:

uc_serial_enable	Enables PHY layer serialization when set to true.
uc_band	Sets the band to be used by PHY layer. This is only used on ATPL250 multiband projects. On fixed Cenelec-A projects, PHY layer is not configurable and thus this parameter is ignored.

This function performs the following actions:
- Sets the handlers for the PHY interruption
- Initializes the PPLC driver
- And, if necessary, initializes the USI serial interface for the PHY layer by means of the input parameter

7.1.2 PHY Event Handler Function

The following function is called every program cycle by the PalEventHandler() function (see 8.1.2. PAL Event Handler Function):

```
uint8_t phy_process(void);
```

First, this function checks all PHY events:
- PHY parameters reset and reconfiguration
- Perform a Noise Capture
- End of transmission of PLC message
- End of reception of PLC message
- Reed-Solomon event

And then, triggers the corresponding PHY callbacks.

The return value of this function indicates whether it should be called again immediately due to an urgent event about to occur, or otherwise it is secure to wait for the next program cycle to call it.

7.1.3 Setting PHY Callbacks

The separation between layers involves relying on callback functions for event notifications. That means that, during stack initialization, upper layers must indicate to their immediate lower layer which function should be invoked when an event takes place.

In case of this implementation of the G3-PLC PHY layer, this is done by means of the *phy_set_callbacks* function:

```
void phy_set_callbacks(struct TPhyCallbacks *p_callbacks);
```

The structure used as input of the *phy_set_callbacks* function contains two fields which are pointers to the functions to be executed after a PLC transmission or reception:

```
static struct TPhyCallbacks g_pal_phy_callbacks = {
    pal_cb_phy_data_confirm,
    pal_cb_phy_data_indication
};
```

The user will need to redefine these callback pointers if using a different FW than the PAL layer provided by Microchip on top of the PHY layer.

7.2 PHY Primitives

7.2.1 PHY Data Primitives

7.2.1.1 PHY-DATA.request

This function sends a frame using the PHY layer:

```
uint8_t phy_tx_frame(xPhyMsgTx_t *px_msg)
```

The function's parameter structure is the following:

```
typedef struct _xPhyMsgTx_t {
    uint8_t m_uc_buff_id;
    uint8_t m_uc_tx_mode;
    uint8_t m_uc_tx_power;
    enum mod_types e_mod_type;
    enum mod_schemes e_mod_scheme;
    uint8_t m_uc_pdc;
    uint8_t m_auc_tone_map[TONE_MAP_SIZE];
    uint8_t m_uc_2_rs_blocks;
    uint8_t m_auc_preemphasis[NUM_SUBBANDS];
    enum delimiter_types e_delimiter_type;
    uint8_t *m_puc_data_buf;
    uint16_t m_us_data_len;
    uint32_t m_ul_tx_time;
} xPhyMsgTx_t;
```

Fields of the structure:

m_uc_buff_id	Buffer identifier. Ignored in the current version and set internally to 0
m_uc_tx_mode	Transmission Mode (forced, delayed, ...) (Related Constants explained below)
m_uc_tx_power	Power to transmit [0..15] [0 = Full gain, 1 = (Full gain - 3dB), 2 = (Full gain - 6dB) and so on]
e_mod_type	Modulation type (Related Constants explained below)
e_mod_scheme	Modulation scheme (Related Constants explained below)

m_uc_pdc	Phase Detector Counter. Calculated and filled internally by PHY layer
m_auc_tone_map	Tone Map to use on transmission (Related Constants explained below)
uc_2_rs_blocks	Flag to indicate whether 2 RS blocks have to be used (only used in FCC bandplan)
m_auc_preemphasis	Preemphasis for transmission. Same as m_uc_tx_power but for each subband (Related Constants explained below)
e_delimiter_type	DT field to be used in header (Related Constants explained below)
m_puc_data_buf	Pointer to data buffer
m_us_data_len	Length of the data buffer
m_ul_tx_time	Instant when transmission has to start referred to 1us PHY counter (Absolute value, not relative). Only used when m_uc_tx_mode is delayed transmission

Related Constants affecting above parameters:

```
/* TX Mode flags */
/* TX Mode: Not Forced Not Delayed transmission, Carrier Sense before transmission,
transmission stars immediately after request */
#define TX_MODE_NOT_FORCED_NOT_DELAYED_TX    0x00
/* TX Mode: Forced transmission, Noo Carrier Sense before transmission, transmission
stars immediately after request */
#define TX_MODE_FORCED_TX                    0x01
/* TX Mode: Delayed transmission, Carrier Sense before transmission, transmission in a
specified instant in time */
#define TX_MODE_DELAYED_TX                   0x02
/* TX Mode: Forced and Delayed transmission, No Carrier Sense, transmission in a
specified instant in time */
#define TX_MODE_FORCED_AND_DELAYED_TX        0x03

/* Modulation types */
enum mod_types {
    MOD_TYPE_BPSK = 0,
    MOD_TYPE_QPSK = 1,
    MOD_TYPE_8PSK = 2,
    MOD_TYPE_QAM = 3, /* NOTE: Not supported at MAC level */
    MOD_TYPE_BPSK_ROBO = 4
};

/* Modulation schemes */
enum mod_schemes {
    MOD_SCHEME_DIFFERENTIAL = 0,
    MOD_SCHEME_COHERENT = 1
};

/* Frame Delimiter Types */
enum delimiter_types {
    DT_SOF_NO_RESP = 0, /* Data frame requiring ACK */
    DT_SOF_RESP = 1, /* Data frame Not requiring ACK */
    DT_ACK = 2, /* Positive ACK */
    DT_NACK = 3 /* Negative ACK */
};

/* Tone Map size for Cenelec-A bandplan */
#define TONE_MAP_SIZE_CENELEC     1
/* Tone Map size for FCC and ARIB bandplans */
#define TONE_MAP_SIZE_FCC_ARIB    3

#if defined(CONF_CENELEC_A)
    #define TONE_MAP_SIZE TONE_MAP_SIZE_CENELEC
#else
    #define TONE_MAP_SIZE TONE_MAP_SIZE_FCC_ARIB
#endif

/* Number of subbands for each bandplan */
#define NUM_SUBBANDS_CENELEC_A    6
#define NUM_SUBBANDS_FCC          24
#define NUM_SUBBANDS_ARIB         18

#if defined(CONF_CENELEC_A)
```

```
    #define NUM_SUBBANDS NUM_SUBBANDS_CENELEC_A
    #elif defined(CONF_FCC)
    #define NUM_SUBBANDS NUM_SUBBANDS_FCC
    #elif defined(CONF_ARIB)
    #define NUM_SUBBANDS NUM_SUBBANDS_ARIB
    #endif
```

The function returns a transmission result value.

```
    /* TX Result values */
    enum tx_result_values {
        PHY_TX_RESULT_PROCESS = 0,      /* already in process */
        PHY_TX_RESULT_SUCCESS = 1,      /* end successfully */
        PHY_TX_RESULT_INV_LENGTH = 2,   /* invalid length error */
        PHY_TX_RESULT_BUSY_CH = 3,      /* busy channel error */
        PHY_TX_RESULT_BUSY_TX = 4,      /* busy in transmission error */
        PHY_TX_RESULT_BUSY_RX = 5,      /* busy in reception error */
        PHY_TX_RESULT_INV_SCHEME = 6,   /* invalid modulation scheme */
        PHY_TX_RESULT_TIMEOUT = 7,      /* timeout error */
        PHY_TX_RESULT_INV_TONEMAP = 8,  /* invalid tone map error */
        PHY_TX_RESULT_NO_TX = 255       /* No transmission ongoing */
    };
```

7.2.1.2 PHY-DATA.confirm

This data confirm callback executes the function set by the upper layer at the initialization of the PAL layer (See 8.1.1. PAL Initialization Function). The pointer to the function is set in:

```
typedef void (*phy_pd_data_confirm)(xPhyMsgTxResult_t *px_tx_result);
```

The result is reported into the following structure:

```
typedef struct _xPhyMsgTxResult_t {

    uint8_t m_uc_id_buffer;

    enum tx_result_values e_tx_result;

    uint32_t m_ul_rms_calc;

    uint32_t m_ul_end_tx_time;

} xPhyMsgTxResult_t;
```

Fields of the structure:

m_uc_id_buffer	Buffer to which result refers to. Fixed to 0, as value in Data Request is ignored
e_tx_result	TX Result (Related Constants defined in PHY-DATA.request)
m_ul_rms_calc	RMS_CALC value after transmission. Allows to estimate the amplitude of the transmitted signal
m_ul_end_tx_time	Instant when frame transmission ended referred to 1 us PHY counter

7.2.1.3 PHY-DATA.indication

This primitive is used to transfer received data from the PHY sublayer to the upper layer.

```
typedef void (*phy_pd_data_indication)(xPhyMsgRx_t *px_msg);
```

The information is reported into the following structure:

```
typedef struct _xPhyMsgRx_t {

    uint8_t m_uc_buff_id;

    enum mod_types e_mod_type;

    enum mod_schemes e_mod_scheme;
    uint8_t m_auc_tone_map[TONE_MAP_SIZE];
    uint16_t m_us_evm_header;
    uint16_t m_us_evm_payload;
    uint16_t m_us_rssi;
    uint16_t m_us_agc_factor;
    uint8_t m_uc_zct_diff;
    enum delimiter_types e_delimiter_type;
    uint8_t m_uc_rs_corrected_errors;
    uint8_t *m_puc_data_buf;
    uint16_t m_us_data_len;
    uint32_t m_ul_rx_time;
    uint32_t m_ul_frame_duration;
} xPhyMsgRx_t;
```

Fields of the structure:

m_uc_buff_id	Buffer identifier
e_mod_type	Modulation type of the last received frame. Related Constants defined in PHY-DATA.request
e_mod_scheme	Modulation scheme of the last received frame. Related Constants defined in PHY-DATA.request
m_auc_tone_map	Tone Map in received frame. Related Constants defined in PHY-DATA.request
m_us_evm_header	EVM on header reception
m_us_evm_payload	EVM on payload reception
m_us_rssi	Received Signal Strength Indicator
m_us_agc_factor	AGC factor
m_uc_zct_diff	Phase difference with transmitting node
e_delimiter_type	DT field coming in header. Related Constants defined in PHY-DATA.request
m_uc_rs_corrected_errors	Errors corrected by RS (errors == 0xff means could not correct)
m_puc_data_buf	Pointer to data buffer containing received frame
m_us_data_len	Length of received frame
m_ul_rx_time	Instant when frame was received (end of frame) referred to 1us PHY counter
m_ul_frame_duration	Frame duration in us (Preamble + FCH + Payload)

7.2.2 PHY Management Primitives

7.2.2.1 PHY_SET.request

```
uint8_t phy_set_cfg_param(uint16_t us_id, void *p_val, uint16_t us_len);
```

The input parameters are the following:

us_id	PIB ID (see atpl250.h file)
**p_val*	Pointer to parameter value to set

us_len	Length of parameter

Possible return values are:
- 0 - PHY_CFG_SUCCESS
- 1 - PHY_CFG_INVALID_INPUT (invalid input parameter or read only)

7.2.2.2 PHY_GET.request

```
uint8_t phy_get_cfg_param(uint16_t us_id, void *p_val, uint16_t us_len);
```

The input parameters are the following:

us_id	PIB ID (see atpl250.h file)
**p_val*	Pointer where read value will be stored
us_len	Length of parameter

Possible return values are:
- 0 - PHY_CFG_SUCCESS
- 1 - PHY_CFG_INVALID_INPUT (invalid input parameter)

7.2.2.3 PHY_RESET.request

It is possible to force a reset of the PHY layer during program execution, calling the function:

```
void phy_reset_request (void)
```

This function performs the following actions:
- Resets all PHY layer variables and state machines
- Initializes the PPLC driver

7.3 PIB Objects Specification and Access

Following is a list of available PIBs with a brief description and possible values.

7.3.1 PHY_ID_INFO_PRODUCT (0x0100)

Product identifier. Depending on Device.

Access: Read only.

Value Range: 10-character string.

Default Value: String identifying the Device for which code is compiled.

7.3.2 PHY_ID_INFO_MODEL (0x010A)

Model identifier.

Access: Read Only.

Value Range: 0-65535.

Default Value: Depending on used model.

7.3.3 PHY_ID_INFO_VERSION (0x010C)

PHY layer version identifier.

Access: Read Only.

Value Range: 0-4294967295.

Default Value: Depending on used version.

7.3.4 PHY_ID_TX_TOTAL (0x0110)

Correctly transmitted frames counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.5 PHY_ID_TX_TOTAL_BYTES (0x0114)

Transmitted bytes counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.6 PHY_ID_TX_TOTAL_ERRORS (0x0118)

Total transmission errors counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.7 PHY_ID_BAD_BUSY_TX (0x011C)

Transmission failure due to already in transmission counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.8 PHY_ID_TX_BAD_BUSY_CHANNEL (0x0120)

Transmission failure due to busy channel counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.9 PHY_ID_TX_BAD_LEN (0x0124)

Transmission failure due to incorrect length (too short - too long) counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.10 PHY_ID_TX_BAD_FORMAT (0x0128)

Transmission failure due to wrong format (Tone-map not valid) counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.11 PHY_ID_TX_TIMEOUT (0x012C)

Timeout error during transmission counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.12 PHY_ID_RX_TOTAL (0x0130)

Total received frames counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.13 PHY_ID_RX_TOTAL_BYTES (0x0134)

Total received bytes counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.14 PHY_ID_RX_RS_ERRORS (0x0138)

Frames discarded by Reed Solomon block counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.15 PHY_ID_RX_EXCEPTIONS (0x013C)

Frames discarded due to decoding errors during reception counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.16 PHY_ID_RX_BAD_LEN (0x0140)

Frames discarded due to incorrect length (too short - too long) counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.17 PHY_ID_RX_BAD_CRC_FCH (0x0144)

Frames discarded due to incorrect CRC in header counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.18 PHY_ID_RX_FALSE_POSITIVE (0x0148)

Frames discarded due to incomplete preamble synchronization counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.19 PHY_ID_RX_BAD_FORMAT (0x014C)

Frames discarded due to unexpected values in header counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.20 PHY_ID_TIME_FREELINE (0x0150)

Not implemented in this version.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.21 PHY_ID_TIME_BUSYLINE (0x0154)

Not implemented in this version.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.22 PHY_ID_TIME_BETWEEN_NOISE_CAPTURES (0x0158)

Time between noise captures (in ms) to detect tonal noise.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 60000.

7.3.23 PHY_ID_LAST_TX_MSG_PAYLOAD_SYMBOLS (0x015C)

Number of payload symbols of last transmitted frame.

Access: R/W.

Value Range: 0-65535.

Default Value: 65535.

7.3.24 PHY_ID_LAST_RX_MSG_PAYLOAD_SYMBOLS (0x015E)

Number of payload symbols of last received frame.

Access: R/W.

Value Range: 0-65535.

Default Value: 65535.

7.3.25 PHY_ID_FCH_SYMBOLS (0x0160)

Number of symbols in sent/received FCH frames. Number of bytes in FCH is constant, but symbols may vary depending on tone-masking.

Access: R/W.

Value Range: 0-255.

Default Value: Initialized depending on Tone-mask.

7.3.26 PHY_ID_CFG_AUTODETECT_IMPEDANCE (0x0161)

Autodetect impedance state, where possible values are:

- 0 - FIXED_STATE_FIXED_GAIN. Transmission state is fixed as impedance detection is not allowed. Transmission gain is kept constant.
- 1- AUTO_STATE_VAR_GAIN. Transmission state may change depending on impedance detected. This may lead to big changes in transmission gain in order to comply with normative.
- 2 - FIXED_STATE_VAR_GAIN. Transmission state is fixed but impedance detection is allowed to perform little transmission gain changes for better signal quality.

Access: R/W.

Value Range: 0-2.

Default Value: 1- AUTO_STATE_VAR_GAIN.

7.3.27 PHY_ID_CFG_IMPEDANCE (0x0162)

Transmission mode corresponding to impedance state, where possible values are:

- 0 - HI_STATE. Transmission is configured to comply with normative with the best performance/efficiency assuming there is a high-impedance (imp > 20 Ohm), either auto-detected or fixed by configuration
- 1- LO_STATE. This is an intermediate state suitable for all impedance range when the required transmission level is low (controlled by upper layers)
- 2 - VLO_STATE. Transmission is configured to comply with normative with the best performance/efficiency assuming there is a very low-impedance (imp < 10 Ohm), either auto-detected or fixed by configuration

Access: R/W.

Value Range: 0-2.

Default Value: 0 - HI_STATE.

7.3.28 PHY_ID_RRC_NOTCH_ACTIVE (0x0163)

Indicates whether a Notch filter is active, used to compensate effect of a tonal noise.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

7.3.29 PHY_ID_RRC_NOTCH_INDEX (0x0164)

Indicates the carrier index where the Notch filter is active (if any), 0 is set if no filter is active.

Access: R/W.

Value Range: 0, [23-58 Cen A], [33-104 FCC], [33-86 ARIB].

Default Value: 0.

7.3.30 PHY_ID_RRC_NOTCH_AUTODETECT (0x0165)

When set to 1, triggers a tonal noise detection so a Notch filter will be applied where noise is detected if needed.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

7.3.31 PHY_ID_ENABLE_AUTO_NOISE_CAPTURE (0x0166)

Enables a periodic tonal noise detection so a Notch filter will be applied where noise is detected if needed. Period is set in 7.3.22. PHY_ID_TIME_BETWEEN_NOISE_CAPTURES (0x0158) parameter.

Access: R/W.

Value Range: 0-1.

Default Value: 1.

7.3.32 PHY_ID_DELAY_NOISE_CAPTURE_AFTER_RX (0x0167)

If set, the next periodic tonal noise detection is canceled after a successful frame reception, and delayed a 7.3.22. PHY_ID_TIME_BETWEEN_NOISE_CAPTURES (0x0158) period. The purpose is not to interfere with frame receptions when there is traffic in the network.

Access: R/W.

Value Range: 0-1.

Default Value: 1.

7.3.33 PHY_ID_LEGACY_MODE (0x0168)

Allows Legacy Modes at PHY level. Must be set to the same value as 9.7.37. MAC_WRP_PIB_CENELEC_LEGACY_MODE (0x00000115) or 9.7.38. MAC_WRP_PIB_FCC_LEGACY_MODE (0x00000116). Possible values are:

- 0 - Legacy modes are used for backwards compatibility with old devices.
- 1 - No legacy compatibility is used.

Access: R/W.

Value Range: 0-1.

Default Value: 1.

7.3.34 PHY_ID_STATIC_NOTCHING (0x0169)

16-byte array where each bit represents one of the 128 possible carriers. Value '1' in one bit indicates that the carrier is affected by tone-mask, i.e. no energy is injected in such carrier during transmissions. Value of notched carriers must be in line with 9.7.32. MAC_WRP_PIB_TONE_MASK (0x00000110).

Access: R/W.

Value Range: 0-1 in bits where carriers are used by the band-plan [23-58 Cen A], [33-104 FCC], [33-86 ARIB].

Default Value: All bits set to '0'.

7.3.35 PHY_ID_PLC_DISABLE (0x016A)

Disables PLC Tx/Rx for device. Data indication primitives are not generated, and data confirms are issued after a data request without actually transmitting.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

7.3.36 PHY_ID_NOISE_PEAK_POWER (0x016B)

Returns detected noise peak power (in dBuV) on last noise capture, in case a peak was detected (PHY_PARAM_RRC_NOTCH_ACTIVE == 1). If no peak was detected, it returns the noise average in band measured on last noise capture.

Access: R/W.

Value Range: 0-255.

Default Value: 0.

7.3.37 PHY_ID_LAST_MSG_LQI (0x016C)

LQI value of the last received frame.

Access: R/W.

Value Range: 0-255.

Default Value: 0.

7.3.38 PHY_ID_TRIGGER_SIGNAL_DUMP (0x016F)

Triggers a signal capture at ADC in the reception stage. Please note this feature is intended to be used in PHY layer projects and can only be used if functionality is enabled at compilation, because the buffer needed to store ADC samples takes all the available RAM memory of the device. It is not available when running the whole G3 stack.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

7.3.39 PHY_ID_LAST_RMSCALC_CORRECTED (0x0170)

Stores a value related to the transmission level of the last frame. It is used for calibration of the impedance detection algorithm. For more details about this algorithm, refer to 6.3. Impedance Detection, Transmission States and Equalization.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

7.3.40 PHY_ID_TONE_MAP_RSP_ENABLED_MODS (0x0174)

When selecting the modulation to be notified in a Tone Map Response frame, PHY layer is asked to take the decision. With this PIB we can limit the decision range, enabling or disable different modulations through the 8 bits of the PIB. Bit order is 8PSK_D, 8PSK_C, QPSK_D, QPSK_C, BPSK_D, BPSK_C, ROBO_D, ROBO_C. 1 means enable and 0 means disable. For example, the value to allow only differential modulations would be 0xAA, and to allow only coherent ones, it would be 0x55.

Access: R/W.

Value Range: 0-255.

Default Value: 255 (All Modulations Enabled).

7.3.41 PHY_ID_FORCE_NO_OUTPUT_SIGNAL (0x0175)

Disables Output Signal Power when transmitting. This mode is used in some common impedance measurement tests and it is offered for the final user to be able to perform such tests using the real PHY layer and not a modified one only for those tests.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

7.3.42 PHY_ID_RESET_PHY_STATS (0x0176)

Resets all PHY Statistics PIBs at once.

Access: W.

Value Range: 8-bits. Any value triggers statistics reset.

Default Value: N/A.

8. API of PAL Layer (ATPL250 Platform)

This section details the association between the names of the standard primitives of the PAL layer API and the names of the corresponding functions implemented in this FW stack. This applies only to ATPL250 platform, as PL360 device is managed at MAC RT level when using G3 stack.

8.1 PAL SAP

8.1.1 PAL Initialization Function

This function is called by upper layers during stack initialization:

```
void PalInitialize(struct TPalNotifications *pNotifications, uint8_t u8Band);
```

This function performs the setting of the PHY callbacks, initializes the PHY layer, and configures the SPI channel for communication with ATPL250.

8.1.2 PAL Event Handler Function

This function is called every program cycle by the Event Handler of upper layer:

```
void PalEventHandler(void);
```

This function calls the *phy_process()* function (See 7.1.2. PHY Event Handler Function) to manage PHY layer processes.

8.2 PAL Primitives

Please note that the semantics of the PAL primitives cover the requirements of the G3-PLC PHY specs and provide more information. In addition, note that there are two management primitives not included into the spec:

- *PLME_RESET.request*, which forces a reset of PHY layer below.
- *GET_TIME*, which retrieves the PHY layer Sys Tick, used as time base in MAC low level time control duties such as CSMA or Inter Frame Spacing.

8.2.1 Data Primitives

DATA PRIMITIVE	**Microchip Function**
PD-DATA.request	void **PhyPdDataRequest**(struct TPdDataRequest *pParameters);
PD-DATA.confirm	typedef void(***PhyPdDataConfirm**)(struct TPdDataConfirm *pParameters);
PD-DATA.indication	typedef void(***PhyPdDataIndication**)(struct TPdDataIndication *pParameters);
PD-ACK.request	void **PhyPdAckRequest**(struct TPdAckRequest *pParameters);
PD-ACK.confirm	typedef void(***PhyPdAckConfirm**)(struct TPdAckConfirm *pParameters);
PD-ACK.indication	typedef void(***PhyPdAckIndication**)(struct TPdAckIndication *pParameters);

8.2.2 Management Primitives

MANAGEMENT PRIMITIVE	**Microchip Function**
PLME_SET.request	void **PhyPlmeSetRequest**(struct TPlmeSetRequest *pParameters);
PLME_SET.confirm	typedef void(***PhyPlmeSetConfirm**)(struct TPlmeSetConfirm *pParameters);
PLME_GET.request	void **PhyPlmeGetRequest**(struct TPlmeGetRequest *pParameters);

PLME_GET.confirm	typedef void(***PhyPlmeGetConfirm**)(struct TPlmeGetConfirm *pParameters);
PLME_SET_TRX_STATE.request	void **PhyPlmeSetTrxStateRequest**(struct TPlmeSetTrxStateRequest *pParameters);
PLME_SET_TRX_STATE.confirm	typedef void(***PhyPlmeSetTrxStateConfirm**)(struct TPlmeSetTrxStateConfirm *pParameters);
PLME_CS.request	void **PhyPlmeCsRequest**(struct TPlmeCsRequest *pParameters);
PLME_CS.confirm	typedef void(***PhyPlmeCsConfirm**)(struct TPlmeCsConfirm *pParameters);
PLME_RESET.request	void **PhyPlmeResetRequest**(void);
GET_TIME	uint32_t **PhyGetTime**(void);

9. API of DATA LINK Layer

This section details the API of the DATA LINK layer modules and their auxiliary files.

9.1 PLC MAC SAP

The functions detailed here (initialization and event handler) are not part of the G3 spec.

These are functions specific to the Microchip implementation of the G3 stack. The ADP layer is in charge of calling these functions through the MAC Wrapper module, so the user does not need to care about calling them.

9.1.1 PLC MAC Initialization Function

The *MacInitialize* function is used to initialize the PLC MAC and layers below.

```
void MacInitialize(struct TMacNotifications *pNotifications, uint8_t u8Band, struct
TMacTables *pTables, uint8_t u8SpecCompliance)
```

Parameters:

**pNotifications*	Structure with callbacks used to notify MAC specific events (if NULL the layer is de-initialized)
u8Band	Frequency band. This has to match the hardware. Use one of the constants: • BAND_CENELEC_A • BAND_CENELEC_B • BAND_FCC • BAND_ARIB
**pTables*	Pointer to PLC MAC Tables structure. Tables themselves are declared and initialized in Mac Wrapper module so user can change their size to fit their needs
u8SpecCompliance	Spec Compliance (G3 Spec'15 or Spec'17), initialized here so library can behave compliant with any spec depending on this parameter

9.1.2 PLC MAC Event Handler Function

The *MacEventHandler* function is called periodically, to execute the PLC MAC layer State Machine and, thus, process pending Data Requests, Confirms and Indications. This function will also call the event handler of the layers below.

```
void MacEventHandler(void)
```

9.2 PLC MAC Primitives

All the functions and the structures presented in this section are defined in files *MacApi.h*, *MacDefs.h* and *MacMib.h*.

Refer to these files for details of structure contents and types.

9.2.1 PLC Mac Data Primitives

DATA PRIMITIVE	Microchip Function
MCPS-DATA.request	void **MacMcpsDataRequest**(struct TMcpsDataRequest *pParameters)
MCPS-DATA.confirm	void (***MacMcpsDataConfirm**)(struct TMcpsDataConfirm *Parameters)
MCPS-DATA.indication	void (***MacMcpsDataIndication**)(struct TMcpsDataIndication *pDataIndication)
Beyond Spec	void (***MacMcpsMacSnifferIndication**)(struct TMcpsMacSnifferIndication *pParameters)

9.2.2 PLC Mac Management Primitives

MANAGEMENT PRIMITIVE	Microchip Function
MLME-GET.request	void **MacMlmeGetRequest**(struct TMlmeGetRequest *pParameters)
MLME-GET.confirm	void (***MacMlmeGetConfirm**)(struct TMlmeGetConfirm *pParameters)
MLME-SET.request	void **MacMlmeSetRequest**(struct TMlmeSetRequest *pParameters)
MLME-SET.confirm	void (***MacMlmeSetConfirm**)(struct TMlmeSetConfirm *pParameters)
MLME-RESET.request	void **MacMlmeResetRequest**(struct TMlmeResetRequest *pParameters)
MLME-RESET.confirm	void (***MacMlmeResetConfirm**)(struct TMlmeResetConfirm *pParameters)
MLME-SCAN.request	void **MacMlmeScanRequest**(struct TMlmeScanRequest *pParameters)
MLME-BEACON-NOTIFY.indication	void (***MacMlmeBeaconNotify**)(struct TMlmeBeaconNotifyIndication *pParameters)
MLME-SCAN.confirm	typedef void (***MacMlmeScanConfirm**)(struct TMlmeScanConfirm *pParameters)
MLME-START.request	void **MacMlmeStartRequest**(struct TMlmeStartRequest *pParameters)
MLME-START.confirm	void (***MacMlmeStartConfirm**)(struct TMlmeStartConfirm *pParameters)
MLME-COMM-STATUS.indication	typedef void (***MacMlmeCommStatusIndication**)(struct TMlmeCommStatusIndication *pParameters)
Beyond Spec	enum EMacStatus **MacMlmeGetRequestSync**(enum EMacPibAttribute eAttribute, uint16_t u16Index, struct TMacPibValue *pValue)
Beyond Spec	enum EMacStatus **MacMlmeSetRequestSync**(enum EMacPibAttribute eAttribute, uint16_t u16Index, const struct TMacPibValue *pValue)

9.3 PLC MAC Real Time

The lowest level of the PLC MAC layer, where strict response time duties reside, is separated from the MAC library and provided as a separated module.

This is a strong advantage when working with the PL360 device, as this module is integrated in the code running inside the transceiver with the following advantages:

- Response time requirements of the G3 stack dramatically decrease. The G3 periodic call (*AdpEventHandler* function), can be called every 20 ms, instead of the required 1-ms rate when MAC RT runs in the host microcontroller, which is the case for the ATPL250 device
- Reduction in both code and data memory utilization in the host microcontroller

On the ATPL250, there is no clear advantage of using it as a separate module, but it is provided as source code out of the MAC library to keep the same MAC library and architecture in both devices.

MAC Real Time duties are:

- CRC check
- ACK generation if requested
- CSMA
- ARQ
- Segmentation and reassembly

9.4 RF MAC SAP

Functions detailed here (initialization and event handler) are not part of the G3 Hybrid spec.

These are functions specific to the Microchip implementation of the G3 stack. The ADP layer is in charge of calling these functions through the MAC Wrapper module, so the user does not need to care about calling them.

9.4.1 RF MAC Initialization Function

The *MacInitializeRF* function is used to initialize the RF MAC and layers below.

```
void MacInitializeRF(struct TMacNotificationsRF *pNotifications, struct TMacTablesRF *pTables)
```

Parameters:

**pNotifications*	Structure with callbacks used to notify RF MAC specific events (if NULL the layer is de-initialized)
**pTables*	Pointer to RF MAC Tables structure. Tables themselves are declared and initialized in the Mac Wrapper module so the user can change their size to fit their needs

9.4.2 RF MAC Event Handler Function

The *MacEventHandlerRF* function is called periodically, to execute the RF MAC layer State Machine, and, thus, process pending Data Requests, Confirms and Indications. This function will also call the event handler of the layers below.

```
void MacEventHandlerRF(void)
```

9.5 RF MAC Primitives

All the functions and the structures presented in this section are defined in files *MacRfApi.h*, *MacRfDefs.h* and *MacRfMib.h*.

Refer to these files for details of structure contents and types.

9.5.1 RF MAC Data Primitives

DATA PRIMITIVE	**Microchip Function**
MCPS-DATA.request	void **MacMcpsDataRequestRF**(struct TMcpsDataRequest *pParameters)
MCPS-DATA.confirm	void (***MacMcpsDataConfirm**)(struct TMcpsDataConfirm *Parameters)
MCPS-DATA.indication	void (***MacMcpsDataIndication**)(struct TMcpsDataIndication *pDataIndication)
Beyond Spec	void (***MacMcpsMacSnifferIndication**)(struct TMcpsMacSnifferIndication *pParameters)

9.5.2 RF MAC Management Primitives

MANAGEMENT PRIMITIVE	**Microchip Function**
MLME-GET.request	void **MacMlmeGetRequestRF**(struct TMlmeGetRequestRF *pParameters)
MLME-GET.confirm	void (***MacMlmeGetConfirmRF**)(struct TMlmeGetConfirmRF *pParameters)
MLME-SET.request	void **MacMlmeSetRequestRF**(struct TMlmeSetRequestRF *pParameters)
MLME-SET.confirm	void (***MacMlmeSetConfirmRF**)(struct TMlmeSetConfirmRF *pParameters)
MLME-RESET.request	void **MacMlmeResetRequestRF**(struct TMlmeResetRequest *pParameters)
MLME-RESET.confirm	void (***MacMlmeResetConfirm**)(struct TMlmeResetConfirm *pParameters)
MLME-SCAN.request	void **MacMlmeScanRequestRF**(struct TMlmeScanRequest *pParameters)

MLME-BEACON-NOTIFY.indication	void (***MacMlmeBeaconNotify**)(struct TMlmeBeaconNotifyIndication *pParameters)
MLME-SCAN.confirm	typedef void (***MacMlmeScanConfirm**)(struct TMlmeScanConfirm *pParameters)
MLME-START.request	void **MacMlmeStartRequestRF**(struct TMlmeStartRequest *pParameters)
MLME-START.confirm	void (***MacMlmeStartConfirm**)(struct TMlmeStartConfirm *pParameters)
MLME-COMM-STATUS.indication	typedef void (***MacMlmeCommStatusIndication**)(struct TMlmeCommStatusIndication *pParameters)
Beyond Spec	enum EMacStatus **MacMlmeGetRequestSyncRF**(enum EMacPibAttributeRF eAttribute, uint16_t u16Index, struct TMacPibValue *pValue)
Beyond Spec	enum EMacStatus **MacMlmeSetRequestSyncRF**(enum EMacPibAttributeRF eAttribute, uint16_t u16Index, const struct TMacPibValue *pValue)

Note that the types specific to the RF MAC layer have the “RF” suffix on their name. Some of the RF MAC API functions share the types with PLC MAC Type definitions, and, thus, no suffix is present on their names.

9.6 MAC Wrapper

This layer is created with two main goals:

- To serve as a clean architectural boundary between the MAC and ADP layers. All requests from the ADP layer to MAC, likewise all callbacks from MAC layer to ADP, are routed through this layer.
- To define MAC layer tables and their size. This way, the MAC library is independent from the table sizes, and, thus, can be used for different purposes (such as Device or Coordinator), or sizes can be accommodate to fit custom needs of final users. So it acts as a *MAC library configuration* file.

Although not considered main goals, having this wrapper as a separate layer accessible outside of the G3 libraries also permits:

- Being used for debug purposes
- Monitoring the traffic between layers
- Implementing beyond-spec features when events occur between the ADP and MAC layers

⚠ CAUTION The correct operation between the ADP and MAC layers is ensured if the MAC wrapper is kept as-is. The user may add code in this layer, but always at their own risk and under their responsibility. If the user adds any code, it **must always** be added after the code already present in the functions to ensure primary operations are done first, and added code is executed only after such primary operation is finished.

The following table presents the relation between the MAC wrapper functions and PLC MAC primitives.

MAC Wrapper Function	Corresponding PLC MAC Primitive
MacWrapperInitialize	MacInitialize
MacWrapperEventHandler	MacEventHandler
MacWrapperMcpsDataRequest	McpsDataRequest
MacWrapperMlmeGetRequest	MlmeGetRequest
MacWrapperMlmeGetRequestSync	MlmeGetRequestSync
MacWrapperMlmeSetRequest	MlmeSetRequest
MacWrapperMlmeSetRequestSync	MlmeSetRequestSync
MacWrapperMlmeResetRequest	MlmeResetRequest

MacWrapperMlmeScanRequest	MlmeScanRequest
MacWrapperMlmeStartRequest	MlmeStartRequest
_Callback_MacWrapperMcpsDataConfirm	McpsDataConfirm
_Callback_MacWrapperMcpsDataIndication	McpsDataIndication
_Callback_MacWrapperMlmeGetConfirm	MlmeGetConfirm
_Callback_MacWrapperMlmeSetConfirm	MlmeSetConfirm
_Callback_MacWrapperMlmeResetConfirm	MlmeResetConfirm
_Callback_MacWrapperMlmeBeaconNotify	MlmeBeaconNotifyIndication
_Callback_MacWrapperMlmeScanConfirm	MlmeScanConfirm
_Callback_MacWrapperMlmeStartConfirm	MlmeStartConfirm
_Callback_MacWrapperMlmeCommStatusIndication	MlmeCommStatusIndication
_Callback_MacWrapperMcpsMacSnifferIndication	McpsMacSnifferIndication

The following table presents the relation between the MAC wrapper functions and RF MAC primitives.

MAC Wrapper Function	**Corresponding RF MAC Primitive**
MacWrapperInitializeRF	MacInitializeRF
MacWrapperEventHandlerRF	MacEventHandlerRF
MacWrapperMcpsDataRequestRF	McpsDataRequestRF
MacWrapperMlmeGetRequestRF	MlmeGetRequestRF
MacWrapperMlmeGetRequestSyncRF	MlmeGetRequestSyncRF
MacWrapperMlmeSetRequestRF	MlmeSetRequestRF
MacWrapperMlmeSetRequestSyncRF	MlmeSetRequestSyncRF
MacWrapperMlmeResetRequestRF	MlmeResetRequestRF
MacWrapperMlmeScanRequestRF	MlmeScanRequestRF
MacWrapperMlmeStartRequestRF	MlmeStartRequestRF
_Callback_MacWrapperMcpsDataConfirmRF	McpsDataConfirmRF
_Callback_MacWrapperMcpsDataIndicationRF	McpsDataIndicationRF
_Callback_MacWrapperMlmeGetConfirmRF	MlmeGetConfirmRF
_Callback_MacWrapperMlmeSetConfirmRF	MlmeSetConfirmRF
_Callback_MacWrapperMlmeResetConfirmRF	MlmeResetConfirmRF
_Callback_MacWrapperMlmeBeaconNotifyRF	MlmeBeaconNotifyIndicationRF
_Callback_MacWrapperMlmeScanConfirmRF	MlmeScanConfirmRF
_Callback_MacWrapperMlmeStartConfirmRF	MlmeStartConfirmRF
_Callback_MacWrapperMlmeCommStatusIndicationRF	MlmeCommStatusIndicationRF
_Callback_MacWrapperMcpsMacSnifferIndicationRF	McpsMacSnifferIndicationRF

9.7 MIB Objects Specification and Access

As the access to any of the MAC layers has to always be performed through the MAC Wrapper, this chapter enumerates objects using the name as defined in the *mac_wrapper_defs.h* file.

The default endianness of all MIB is little-endian. In other cases, it will be explicitly specified.

9.7.1 MAC_WRP_PIB_ACK_WAIT_DURATION (0x00000040)

Duration of acknowledgement period in microseconds.

Access: Read-only.

Value Range: 0-65535.

Default Value: aSymbolTime x (aRIFS + aCIFS) + aAckTime.

9.7.2 MAC_WRP_PIB_MAX_BE (0x00000047)

Maximum value of back-off exponent. It should always be greater than macMinBE.

Access: R/W.

Value Range: 0–20.

Default Value: 8.

9.7.3 MAC_WRP_PIB_BSN (0x00000049)

Beacon frame sequence number.

Access: R/W.

Value Range: 0-255.

Default Value: random.

9.7.4 MAC_WRP_PIB_DSN (0x0000004C)

Data frame sequence number.

Access: R/W.

Value Range: 0-255.

Default Value: random.

9.7.5 MAC_WRP_PIB_MAX_CSMA_BACKOFFS (0x0000004E)

Maximum number of back-off attempts.

Access: R/W.

Value Range: 0-255.

Default Value: 50.

9.7.6 MAC_WRP_PIB_MIN_BE (0x0000004F)

Minimum value of back-off exponent.

Access: R/W.

Value Range: 0–20.

Default Value: 3.

9.7.7 MAC_WRP_PIB_PAN_ID (0x00000050)

PAN ID of the network.

Access: R/W.

Value Range: 0-65535.

Default Value: 65535.

9.7.8 MAC_WRP_PIB_PROMISCUOUS_MODE (0x00000051)

Promiscuous mode enabled as defined in IEEE 802.15.4 specification.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

9.7.9 MAC_WRP_PIB_SHORT_ADDRESS (0x00000053)

Device short address.

Access: R/W.

Value Range: 0-65535.

Default Value: 65535.

9.7.10 MAC_WRP_PIB_MAX_FRAME_RETRIES (0x00000059)

Maximum number of retransmission.

Access: R/W.

Value Range: 0-10.

Default Value: 5.

9.7.11 MAC_WRP_PIB_TIMESTAMP_SUPPORTED (0x0000005C)

MAC frame time stamp support enabled.

Access: Read-only.

Value Range: 0-1.

Default Value: 1

9.7.12 MAC_WRP_PIB_SECURITY_ENABLED (0x0000005D)

Security enabled.

Access: Read-only.

Value Range: 0-1.

Default Value: 1.

9.7.13 MAC_WRP_PIB_KEY_TABLE (0x00000071)

This attribute holds GMK keys required for MAC layer ciphering. The attribute can hold two 16-byte keys. The row index corresponds to the key identifier value. For security reasons, the key entries cannot be read, only written or deleted.

Access: Write only.

Number of entries: 2.

Entry format:

```
struct TMacSecurityKey {
    bool m_bValid;
    uint8_t m_au8Key[MAC_SECURITY_KEY_LENGTH]; // 16 bytes
};
```

Default Value: empty table.

Add Entry: Call Set function with index [0-1], length = 16, value = 16-byte key value. Valid flag automatically set.

Delete Entry: Call Set function with index [0-1], length = 0: Key value and Valid flag automatically cleared.

9.7.14 MAC_WRP_PIB_FRAME_COUNTER (0x00000077)

The outgoing frame counter for this device. This value has to be restored across power fail.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.15 MAC_WRP_PIB_HIGH_PRIORITY_WINDOW_SIZE (0x00000100)

The high priority contention window size, in number of slots.

Access: R/W.

Value Range: 1-7:

Default Value: 7.

9.7.16 MAC_WRP_PIB_TX_DATA_PACKET_COUNT (0x00000101)

Statistic counter of successfully transmitted unicast MSDUs.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.17 MAC_WRP_PIB_RX_DATA_PACKET_COUNT (0x00000102)

Statistic counter of successfully received unicast MSDUs.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.18 MAC_WRP_PIB_TX_CMD_PACKET_COUNT (0x00000103)

Statistic counter of successfully transmitted command packets.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.19 MAC_WRP_PIB_RX_CMD_PACKET_COUNT (0x00000104)

Statistic counter of successfully received command packets.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.20 MAC_WRP_PIB_CSMA_FAIL_COUNT (0x00000105)

Counts the number of times the CSMA back-offs reach macMaxCSMABackoffs.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.21 MAC_WRP_PIB_CSMA_NO_ACK_COUNT (0x00000106)

Counts the number of times an ACK is not received while transmitting a unicast data frame (The loss of ACK is attributed to collisions).

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.22 MAC_WRP_PIB_RX_DATA_BROADCAST_COUNT (0x00000107)

Statistic counter of successfully received broadcast frames.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.23 MAC_WRP_PIB_TX_DATA_BROADCAST_COUNT (0x00000108)

Statistic counter of the number of broadcast frames sent.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.24 MAC_WRP_PIB_BAD_CRC_COUNT (0x00000109)

Statistic counter of the number of frames received with bad CRC.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.25 MAC_WRP_PIB_NEIGHBOUR_TABLE (0x0000010A)

The neighbour table.

Access: R/W.

Number of entries:

	Spec'15	**Spec'17**
Device	75	35
Coordinator Lite	500	100
Coordinator	2000	500

The table size is configurable for each project by means of the *conf_tables.h* file.

Entry format:

```
struct TNeighbourEntry {
    TShortAddress m_nShortAddress; // 2 bytes
    struct TPhyToneMap m_ToneMap; // 3 bytes (only first used in Cenelec band-plan)
    uint8_t m_nModulationType : 3;
    uint8_t m_nTxGain : 4;
    uint8_t m_nTxRes : 1;
    struct TMacTxCoef m_TxCoef; // 6 bytes
    uint8_t m_nModulationScheme : 1;
    uint8_t m_nPhaseDifferential : 3;
    uint8_t m_u8Lqi;
    uint16_t m_u16TmrValidTime; // TMResponse parameters validity time (in seconds)
    uint16_t m_u16NeighbourValidTime; // Entry validity time (in seconds)
};
```

Short Address	Tone Map	TxRes	TxGain	Modulation Type	TxCoef	Not Used	Phase Differential	Modulation Scheme	Lqi	Tmr ValidTime	Neighbour ValidTime
2 bytes	3 bytes	1 byte			6 bytes	1 byte			1 byte	2 bytes	2 bytes
		1 bit	4 bits	3 bits		4 bits	3 bits	1 bit			

The NeighbourValidTime parameter was removed in the April ‘17 G3 Specification, so the entry size depends on which G3 spec compliance is running (compliant with Spec'17 or with Spec'15). Internally, the entry contains all the fields, but when running Spec’17, NeighbourValidTime is neither used nor returned when getting the IB.

This table is ordered according to m_u16NeighbourValidTime (Spec'15) or m_u16TmrValidTime (Spec'17). The newest entries are placed at top of the table.

Default Value: empty table

Get Entry: Call Get function with index [0-74] or [0-34], returned value will have the format described above.

Add Entry: Call Set function with index [0-74] or [0-34], index will be ignored and new entry will be placed according to m_u16NeighbourValidTime (Spec'15) or m_u16TmrValidTime (Spec'17) as described above.

Delete Entry: Set entry with short address to be deleted and valid time (m_u16NeighbourValidTime (Spec'15) or m_u16TmrValidTime (Spec'17)) equal to 0. Index is ignored, entry will be searched by short address and deleted.

To get all the active entries, the first step is to get the number of such active entries by means of 9.7.60. MAC_WRP_PIB_MANUF_NEIGHBOUR_TABLE_COUNT (0x08000012). Then, because the table is ordered, all the entries can be obtained by iterating from 0 to the number of active entries.

9.7.26 MAC_WRP_PIB_FREQ_NOTCHING (0x0000010B)

This attribute is not available from the April ‘17 G3 Specification, so it cannot be referenced when using the compliant with Spec'17 mode.

The attribute was intended ti set the S-FSK 63 to 74 kHz frequency notching. **Even if running in Spec'15 mode, this attribute is ignored. Use MAC_WRP_PIB_TONE_MASK instead.**

Access: R/W.

Value Range: 0-1.

Default Value: 0.

9.7.27 MAC_WRP_PIB_CSMA_FAIRNESS_LIMIT (0x0000010C)

Channel access fairness limit. Specifies after how many failed back-off attempts, back-off exponent is set to minBE. Should be at least (2×(macMaxBE-macMinBE)).

Access: R/W.

Value Range: 0-255.

Default Value: 15.

9.7.28 MAC_WRP_PIB_TMR_TTL (0x0000010D)

Maximum time to live of tone map parameters entry in the neighbor table, in minutes.

Access: R/W.

Value Range: 0-255.

Default Value: 2 in Spec'15, 10 in Spec'17.

9.7.29 MAC_WRP_PIB_NEIGHBOUR_TABLE_ENTRY_TTL (0x0000010E)

This attribute is not available from the April ‘17 G3 Specification, so it cannot be referenced when running mode compliant with Spec'17. In such case, the concept of Neighbour entries validity does not exist, and only the contents validity is taken into account, which is controlled by 9.7.28. MAC_WRP_PIB_TMR_TTL (0x0000010D).

Maximum time to live for an entry in the neighbor table, in minutes.

Access: R/W.

Value Range: 0-255.

Default Value: 255.

9.7.30 MAC_WRP_PIB_POS_TABLE_ENTRY_TTL (0x0000010E)

This attribute is new with the April ‘17 G3 Specification, so it cannot be referenced when running mode compliant with Spec'15. Note that the numeric ID has been reused from MAC_WRP_PIB_NEIGHBOUR_TABLE_ENTRY_TTL of previous Specs. Be careful as it may lead to confusion.

Maximum time to live for an entry in the POS table, in minutes.

Access: R/W.

Value Range: 0-255.

Default Value: 255.

9.7.31 MAC_WRP_PIB_RC_COORD (0x0000010F)

Route cost to coordinator to be used in the beacon payload as RC_COORD.

Access: R/W.

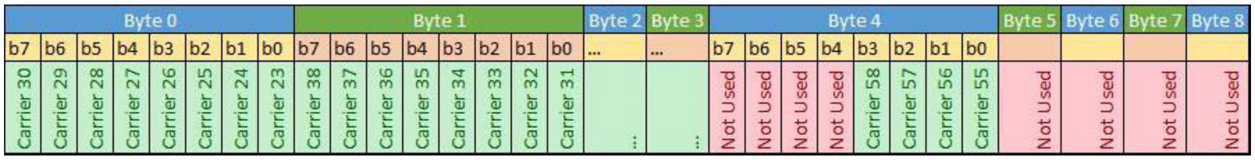
Value Range: 0-65535.

Default Value: 65535.

9.7.32 MAC_WRP_PIB_TONE_MASK (0x00000110)

Defines the tone mask to use during symbol formation. It is represented using one bit per carrier. Holds up to 72 carriers (for FCC) so its size is 9 bytes. Bits set indicate carrier used; bits cleared indicate carrier masked. Depending on the band used, a different number of bits are taken into account.

This PIB is a 9-byte array coded little endian (LSB first). In the Cenelec-A band, the carriers are placed like this:



Byte 0								Byte 1								Byte 2	Byte 3	Byte 4								Byte 5	Byte 6	Byte 7	Byte 8
b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0	...	...	b7	b6	b5	b4	b3	b2	b1	b0				
Carrier 30	Carrier 29	Carrier 28	Carrier 27	Carrier 26	Carrier 25	Carrier 24	Carrier 23	Carrier 38	Carrier 37	Carrier 36	Carrier 35	Carrier 34	Carrier 33	Carrier 32	Carrier 31	⋮	⋮	Not Used	Not Used	Not Used	Not Used	Carrier 58	Carrier 57	Carrier 56	Carrier 55	Not Used	Not Used	Not Used	Not Used

Access: R/W.

Value Range: 0x00-0xFF for each of 9 bytes.

Default Value: All bytes set to 0xFF (All carriers are enabled).

9.7.33 MAC_WRP_PIB_BEACON_RANDOMIZATION_WINDOW_LENGTH (0x00000111)

Duration time in seconds for beacon randomization.

Access: R/W.

Value Range: 1-254.

Default Value: 12.

9.7.34 MAC_WRP_PIB_A (0x00000112)

This parameter controls the adaptive CW linear decrease.

Access: R/W.

Value Range: 3-20.

Default Value: 8.

9.7.35 MAC_WRP_PIB_K (0x00000113)

Rate adaptation factor for channel access fairness limit.

Access: R/W.

Value Range: 1-macCSMAFairnessLimit.

Default Value: 5.

9.7.36 MAC_WRP_PIB_MIN_CW_ATTEMPTS (0x00000114)

Number of consecutive attempts while using minimum CW.

Access: R/W.

Value Range: 0-255

Default Value: 10.

9.7.37 MAC_WRP_PIB_CENELEC_LEGACY_MODE (0x00000115)

This read only attribute indicates the capability of the node. This means that, when set to 1, the following configuration is used:

- Full block interleaving
- Interleaver parameters ni and nj are swapped when I(i,j) = 0

Access: Read-only.

Value Range: 0-1.

Default Value: 1.

9.7.38 MAC_WRP_PIB_FCC_LEGACY_MODE (0x00000116)

This read only attribute indicates the capability of the device. This means that, when set to 1, the following configuration is used:

- Coherent FCH modulation
- Full block interleaving
- Interleaver parameters ni and nj are swapped when I(i,j) = 0
- Two RS blocks capability

Access: Read-only.

Value Range: 0-1.

Default Value: 1.

9.7.39 MAC_WRP_PIB_BROADCAST_MAX_CW_ENABLE (0x0000011E)

If enabled (1), MAC uses maximum contention window for Broadcast frames.

This attribute is new on G3 Specification April '17, so it cannot be referenced when running mode compliant with Spec'15.

Access: R/W.

Value Range: 0-1

Default Value: 0.

9.7.40 MAC_WRP_PIB_TRANSMIT_ATTEN (0x0000011F)

Attenuation of the output level in dB.

This attribute is new with the April ‘17 G3 Specification, so it cannot be referenced when running mode compliant with Spec'15.

Access: R/W.

Value Range: 0-25

Default Value: 0.

9.7.41 MAC_WRP_PIB_POS_TABLE (0x00000120)

The POS table.

This attribute is new for the April ‘17 G3 Specification, so it cannot be referenced when running mode compliant with Spec'15.

Access: R/W.

Number of entries: Default configuration is set to 100 for Device, 500 for Coordinator Lite, 2000 for Coordinator. Anyway the Table size is configurable for each project by means of the *conf_tables.h* file.

Entry format:

```
struct TPOSEntry {
    TShortAddress m_nShortAddress; // 2 bytes
    uint8_t m_u8Lqi;
    uint16_t m_u16POSValidTime; // Entry validity time (in seconds)
};
```

This table is ordered according to m_u16POSValidTime. Newest entries are placed at top of the table.

Default Value: empty table.

Get Entry: Call Get function with index [0-99], returned value will have the format described above.

Add Entry: Call Set function with index [0-99], index will be ignored and new entry will be placed according to m_u16POSValidTime as described above.

Delete Entry: Set entry with the short address to be deleted and valid time equal to 0. Index is ignored and the entry will be searched by short address and deleted.

To get all the active entries, first step to get the number of such active entries by means of 9.7.72. MAC_WRP_PIB_MANUF_POS_TABLE_COUNT (0x0800001E). Then, since the table is ordered, all entries can be obtained by iterating from 0 to the number of active entries.

9.7.42 MAC_WRP_PIB_MANUF_DEVICE_TABLE (0x08000000)

The device table as specified in G3 Spec. Stores Frame Counter of neighbour nodes from which frames are received.

Access: Read only.

Number of entries: 128 by default. Can be modified through MAC_MAX_DEVICE_TABLE_ENTRIES definition in Mac Wrapper.

Entry format:

```
struct TDeviceTableEntry {
  TPanId m_nPanId; // 2 bytes
  TShortAddress m_nShortAddress; // 2 bytes
  uint32_t m_u32FrameCounter; // 4 bytes
};
```

Default Value: empty table.

Get Entry: Call Get function with index [0-127], returned value will have the format described above.

9.7.43 MAC_WRP_PIB_MANUF_EXTENDED_ADDRESS (0x08000001)

Extended address of the node. Format EUI64. (little endian encoding)

Access: R/W.

Value Range: 0-255 for each of 8 bytes.

Default value: All bytes set to 0.

9.7.44 MAC_WRP_PIB_MANUF_NEIGHBOUR_TABLE_ELEMENT (0x08000002)

This object has the same content as the neighbour table, but it is retrieved using the short address of the node, instead of the index inside the table. To do so, set the short address in the index field in the Get IB function.

Access: Read-only.

9.7.45 MAC_WRP_PIB_MANUF_BAND_INFORMATION (0x08000003)

This read only object returns information about the band.

The intention of this IB is more internal than for G3 user. It is used by G3 stack to perform calculations based on these parameters.

Entry format:

```
struct TPhyBandInformation {
  uint8_t m_u8Band; // 1 byte
  uint8_t m_u8Tones; // 1 byte
  uint8_t m_u8Carriers; // 1 byte
  uint8_t m_u8TonesInCarrier; // 1 byte
  uint8_t m_u8FlBand; // 1 byte
  uint16_t m_u16FlMax; // 2 bytes
  uint8_t m_u8MaxRsBlocks; // 1 byte
  uint8_t m_u8TxCoefBits; // 1 byte
  uint8_t m_u8PilotsFreqSpa; // 1 byte
};
```

- 1 byte – band (CENELEC_A = 0, CENELEC_B = 1, FCC = 2, ARIB = 3)
- 1 byte – number of tones in band
- 1 byte – number of carrier groups
- 1 byte – tones in a carrier group
- 1 byte – FLBand. Symbol grouping in FCH coding (4 for CENELEC A & B, 1 for FCC & ARIB)
- 1 byte – reserved
- 2 byte – FLmax. Maximum symbols as coded in FCH (63 for CENELEC A & B, 511 for FCC & ARIB)
- 1 byte – maximum number of RS blocks (2 for FCC, 1 for other bands)
- 1 byte – number of bits in one entry of the TXCOEF table (4 for CENELEC A & B, 2 for FCC & ARIB)
- 1 byte – Pilot tones spacing in number of carriers (8 for CENELEC_B, 12 for other bands)

Access: Read-only.

9.7.46 MAC_WRP_PIB_MANUF_COORD_SHORT_ADDRESS (0x08000004)

Short address of the coordinator. Set after bootstrap process.

Access: Read-only.

Value Range: 0-65535.

Default Value: 0.

9.7.47 MAC_WRP_PIB_MANUF_MAX_MAC_PAYLOAD_SIZE (0x08000005)

Maximal length of an MSDU.

Access: Read-only.

Value Range: 0-65535.

Default value: 400.

9.7.48 MAC_WRP_PIB_MANUF_SECURITY_RESET (0x08000006)

Resets the device table. Intended for testing purposes, not to be used in final products.

Access: write only.

9.7.49 MAC_WRP_PIB_MANUF_FORCED_MOD_SCHEME (0x08000007)

Forces payload modulation scheme to Differential or Coherent mode for every transmitted frame, instead of using neighbour table info. For testing purposes.

Access: R/W.

Value Range: 0-2.

0 – Use Neighbour table info.

1 – Force Differential scheme.

2 – Force Coherent scheme.

Default value: 0.

9.7.50 MAC_WRP_PIB_MANUF_FORCED_MOD_TYPE (0x08000008)

Forces payload modulation type to any of the available modes for every transmitted frame, instead of using neighbour table info. For testing purposes.

Access: R/W.

Value Range: 0-4.

0 – Use Neighbour table info.

1 – Force ROBO.

2 – Force BPSK.

3 – Force QPSK.

4 – Force 8PSK.

Default value: 0.

9.7.51 MAC_WRP_PIB_MANUF_FORCED_TONEMAP (0x08000009)

Forces the payload tone map to any value for every transmitted frame, instead of using neighbour table information. For testing purposes.

Access: R/W.

Entry format:

```
struct TPhyToneMap {
    uint8_t m_au8Tm[3];
};
```

A value of “all zeros” means the tone map is not forced and the neighbour table information will be used.

Default value: [0x00, 0x00, 0x00].

Note: For the Cenelec A & B Bands, only first element of the array is used, as tone map length is 6 or 4 bits respectively.

9.7.52 MAC_WRP_PIB_MANUF_FORCED_MOD_SCHEME_ON_TMRESPONSE (0x0800000A)

Forces Modulation scheme bit to Differential or Coherent mode on Tone Map Response message, instead of using internal G3 stack info.

Access: R/W.

Value Range: 0-2.

0 – Use internal G3 stack info.

1 – Force Differential scheme.

2 – Force Coherent scheme.

Default value: 0.

9.7.53 MAC_WRP_PIB_MANUF_FORCED_MOD_TYPE_ON_TMRESPONSE (0x0800000B)

Forces modulation type bits to any of the available modes on the tone map response message, instead of using internal G3 stack info.

Access: R/W.

Value Range: 0-4.

0 – Use internal G3 stack info.

1 – Force ROBO.

2 – Force BPSK.

3 – Force QPSK.

4 – Force 8PSK.

Default value: 0.

9.7.54 MAC_WRP_PIB_MANUF_FORCED_TONEMAP_ON_TMRESPONSE (0x0800000C)

Forces the tone map field to any value on the tone map response message, instead of using internal G3 stack info.

Access: R/W.

Entry format:

```
struct TPhyToneMap {
    uint8_t m_au8Tm[3];
};
```

A value of “all zeros” means Tone Map field is not forced and internal G3 stack info will be used.

Default value: [0x00, 0x00, 0x00].

Note: For the Cenelec A & B Bands, only first element of the array is used, as tone map length is 6 or 4 bits respectively.

9.7.55 MAC_WRP_PIB_MANUF_LAST_RX_MOD_SCHEME (0x0800000D)

Gets Modulation scheme of the last received data frame.

Access: Read only.

Value Range: 0-1.

0 – Differential scheme.

1 – Coherent scheme.

Default value: 0.

9.7.56 MAC_WRP_PIB_MANUF_LAST_RX_MOD_TYPE (0x0800000E)

Gets modulation type of the last received data frame.

Access: Read only.

Value Range: 0-3.

0 – ROBO.

1 – BPSK.

2 – QPSK.

3 – 8PSK.

Default value: 0.

9.7.57 MAC_WRP_PIB_MANUF_LBP_FRAME_RECEIVED (0x0800000F)

Flag indicating whether LBP frames are being received since last time MIB was set to 0. This IB is intended to be used for collision avoidance during bootstrap phase.

Access: R/W.

Value Range: 0-1.

Default value: 0.

9.7.58 MAC_WRP_PIB_MANUF_LNG_FRAME_RECEIVED (0x08000010)

Flag indicating whether LOADng frames are being received since last time MIB was set to 0. This IB is intended to be used for collision avoidance during bootstrap phase.

Access: R/W.

Value Range: 0-1.

Default value: 0.

9.7.59 MAC_WRP_PIB_MANUF_BCN_FRAME_RECEIVED (0x08000011)

Flag indicating whether beacon frames are being received since last time MIB was set to 0. This IB is intended to be used for collision avoidance during bootstrap phase.

Access: R/W.

Value Range: 0-1.

Default value: 0.

9.7.60 MAC_WRP_PIB_MANUF_NEIGHBOUR_TABLE_COUNT (0x08000012)

Gets the number of valid elements in the neighbour table.

Access: Read only.

Value Range: 0-65535.

Default value: 0.

9.7.61 MAC_WRP_PIB_MANUF_RX_OTHER_DESTINATION_COUNT (0x08000013)

Statistic counter of discarded packets due to other destination.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.62 MAC_WRP_PIB_MANUF_RX_INVALID_FRAME_LENGTH_COUNT (0x08000014)

Statistic counter of discarded packets due to invalid frame length.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.63 MAC_WRP_PIB_MANUF_RX_MAC_REPETITION_COUNT (0x08000015)

Statistic counter of discarded packets due to MAC repetition.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.64 MAC_WRP_PIB_MANUF_RX_WRONG_ADDR_MODE_COUNT (0x08000016)

Statistic counter of discarded packets due to Wrong Addressing Mode.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.65 MAC_WRP_PIB_MANUF_RX_UNSUPPORTED_SECURITY_COUNT (0x08000017)

Statistic counter of discarded packets due to unsupported security.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.66 MAC_WRP_PIB_MANUF_RX_WRONG_KEY_ID_COUNT (0x08000018)

Statistic counter of discarded packets due to wrong key ID.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.67 MAC_WRP_PIB_MANUF_RX_INVALID_KEY_COUNT (0x08000019)

Statistic counter of discarded packets due to invalid key.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.68 MAC_WRP_PIB_MANUF_RX_WRONG_FC_COUNT (0x0800001A)

Statistic counter of discarded packets due to wrong frame counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.69 MAC_WRP_PIB_MANUF_RX_DECRYPTION_ERROR_COUNT (0x0800001B)

Statistic counter of discarded packets due to decryption error.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.70 MAC_WRP_PIB_MANUF_RX_SEGMENT_DECODE_ERROR_COUNT (0x0800001C)

Statistic counter of discarded packets due to segment decode error.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.71 MAC_WRP_PIB_MANUF_ENABLE_MAC_SNIFFER (0x0800001D)

MAC Sniffer mode enabled. This mode processes all frames at MAC level, regardless of addressing, and outputs all correct frames by means of MacMcpsMacSnifferIndication callback. The difference with Promiscuous mode is that, in Sniffer mode, the device will still work as a regular node in the network, treating frames addressed to it like a normal device would do. Summarizing, Sniffer mode is the combination of Normal mode plus Promiscuous mode.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

9.7.72 MAC_WRP_PIB_MANUF_POS_TABLE_COUNT (0x0800001E)

Gets number of valid elements in the POS Table.

POS Table is new for the April ‘17 G3 Specification, so this attribute cannot be referenced when running mode compliant with Spec'15.

Access: Read only.

Value Range: 0-65535.

Default value: 0.

9.7.73 MAC_WRP_PIB_MANUF_RETRIES_LEFT_TO_FORCE_ROBO (0x0800001F)

This IB is used to force ROBO mode when a certain number of MAC retries are left before discarding the frame due to no ACK reception. It can be used to increase the robustness of communications after retries. Of course robustness increase causes throughput decrease.

Access: R/W.

Value Range: 0-macMaxFrameRetries.

Default value: 0.

9.7.74 MAC_WRP_PIB_MANUF_MAC_INTERNAL_VERSION (0x08000021)

Gets current MAC internal version number.

Entry format:

```
struct TMacSoftVersion {
  uint8_t m_u8Major;
  uint8_t m_u8Minor;
  uint8_t m_u8Revision;
  uint8_t m_u8Year; // year since 2000
  uint8_t m_u8Month;
  uint8_t m_u8Day;
};
```

Access: Read only.

Value Range: N/A.

Default value: Current MAC version.

9.7.75 MAC_WRP_PIB_MANUF_MAC_RT_INTERNAL_VERSION (0x08000022)

Gets current MAC Real Time internal version number.

Entry format:

```
struct TMacRtSoftVersion {
  uint8_t m_u8Major;
  uint8_t m_u8Minor;
  uint8_t m_u8Revision;
  uint8_t m_u8Year; // year since 2000
  uint8_t m_u8Month;
  uint8_t m_u8Day;
};
```

Access: Read only.

Value Range: N/A.

Default value: Current MAC Real Time version.

9.7.76 MAC_WRP_PIB_MANUF_RESET_MAC_STATS (0x08000023)

Resets all IBs related to MAC statistics at once.

Access: W.

Value Range: 8-bits. Any value triggers statistics reset.

Default Value: N/A.

9.7.77 MAC_WRP_PIB_MANUF_SLEEP_MODE (0x08000024)

This IB is used to set/clear the Sleep mode of the PL360, which is available under certain conditions. This mode is detailed in the PL360 Host Controller documentation.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

9.7.78 MAC_WRP_PIB_MANUF_DEBUG_SET (0x08000025)

This IB is used for the special Debug mode of the PL360. This mode is detailed in the PL360 Host Controller documentation.

9.7.79 MAC_WRP_PIB_MANUF_DEBUG_READ (0x08000026)

This IB is used for the special Debug mode of the PL360. This mode is detailed in the PL360 Host Controller documentation.

9.7.80 MAC_WRP_PIB_MANUF_PHY_PARAM (0x08000020)

Gives access to some PHY layer parameters and statistics. It is treated as a table, where the index determines which parameter to access. Available parameters and their corresponding indexes are the following:

A link to the corresponding PHY IB is provided for more detailed information.

MAC_WRP_PHY_PARAM_VERSION, index 0x010C. (7.3.3. PHY_ID_INFO_VERSION (0x010C))

Description: Retrieves PHY layer version number. 32 bits.

Access: Read only.

Default Value: N/A.

MAC_WRP_PHY_PARAM_TX_TOTAL, index 0x0110. (7.3.4. PHY_ID_TX_TOTAL (0x0110))

Description: Correctly transmitted frame count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_TX_TOTAL_BYTES, index 0x0114. (7.3.5. PHY_ID_TX_TOTAL_BYTES (0x0114))

Description: Transmitted bytes count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_TX_TOTAL_ERRORS, index 0x0118. (7.3.6. PHY_ID_TX_TOTAL_ERRORS (0x0118))

Description: Transmission errors count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_BAD_BUSY_TX, index 0x011C. (7.3.7. PHY_ID_BAD_BUSY_TX (0x011C))

Description: Transmission failure due to already in transmission. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_TX_BAD_BUSY_CHANNEL, index 0x0120.
(7.3.8. PHY_ID_TX_BAD_BUSY_CHANNEL (0x0120))

Description: Transmission failure due to busy channel. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_TX_BAD_LEN, index 0x0124. (7.3.9. PHY_ID_TX_BAD_LEN (0x0124))

Description: Bad length in message (too short - too long). 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_TX_BAD_FORMAT, index 0x0128. (7.3.10. PHY_ID_TX_BAD_FORMAT (0x0128))

Description: Message to transmit in bad format. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_TX_TIMEOUT, index 0x012C. (7.3.11. PHY_ID_TX_TIMEOUT (0x012C))

Description: Timeout error in transmission. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_TOTAL, index 0x0130. (7.3.12. PHY_ID_RX_TOTAL (0x0130))

Description: Correctly received messages count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_TOTAL_BYTES, index 0x0134. (7.3.13. PHY_ID_RX_TOTAL_BYTES (0x0134))

Description: Received bytes count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_RS_ERRORS, index 0x0138. (7.3.14. PHY_ID_RX_RS_ERRORS (0x0138))

Description: Reception RS errors count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_EXCEPTIONS, index 0x013C. (7.3.15. PHY_ID_RX_EXCEPTIONS (0x013C))

Description: Reception Exceptions count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_BAD_LEN, index 0x0140. (7.3.16. PHY_ID_RX_BAD_LEN (0x0140))

Description: Bad length in message (too short - too long). 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_BAD_CRC_FCH, index 0x0144. (7.3.17. PHY_ID_RX_BAD_CRC_FCH (0x0144))

Description: Bad CRC in received FCH. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_FALSE_POSITIVE, index 0x0148. (7.3.18. PHY_ID_RX_FALSE_POSITIVE (0x0148))

Description: Incomplete synchronization. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_RX_BAD_FORMAT, index 0x014C. (7.3.19. PHY_ID_RX_BAD_FORMAT (0x014C))

Description: Received message in bad format. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_PHY_PARAM_ENABLE_AUTO_NOISE_CAPTURE, index 0x0166. (7.3.31. PHY_ID_ENABLE_AUTO_NOISE_CAPTURE (0x0166))

Description: Noise autodetection and adaptation. 8 bits.

Access: R/W.

Default Value: 1.

MAC_WRP_PHY_PARAM_TIME_BETWEEN_NOISE_CAPTURES, index 0x0158. (7.3.22. PHY_ID_TIME_BETWEEN_NOISE_CAPTURES (0x0158))

Description: Time between noise captures (in ms). 32 bits.

Access: R/W.

Default Value: 60000.

MAC_WRP_PHY_PARAM_CFG_AUTODETECT_BRANCH, index 0x0161. (7.3.26. PHY_ID_CFG_AUTODETECT_IMPEDANCE (0x0161))

Description: Automatic Impedance detection by PHY layer. 8 bits treated as Boolean.

Access: R/W.

Default Value: 2.

MAC_WRP_PHY_PARAM_CFG_IMPEDANCE, index 0x0162. (7.3.27. PHY_ID_CFG_IMPEDANCE (0x0162))

Description: Impedance value to set in case Autodetect is disabled. 8 bits.

Access: R/W.

Default Value: 2 (VLO impedance).

MAC_WRP_PHY_PARAM_RRC_NOTCH_ACTIVE, index 0x0163. (7.3.28. PHY_ID_RRC_NOTCH_ACTIVE (0x0163))

Description: Indicate if notch filter is active or not. 8 bits.

Access: R/W.

Default Value: 0.

MAC_WRP_PHY_PARAM_RRC_NOTCH_INDEX, index 0x0164. (7.3.29. PHY_ID_RRC_NOTCH_INDEX (0x0164))

Description: Index of the notch filter. 8 bits.

Access: R/W.

Default Value: 0.

MAC_WRP_PHY_PARAM_DELAY_NOISE_CAPTURE_AFTER_RX, index 0x0167. (7.3.32. PHY_ID_DELAY_NOISE_CAPTURE_AFTER_RX (0x0167))

Description: Next noise capture is delayed if a frame is correctly received. 8 bits.

Access: R/W.

Default Value: 1.

MAC_WRP_PHY_PARAM_PLC_DISABLE, index 0x016A. (7.3.35. PHY_ID_PLC_DISABLE (0x016A))

Description: Disable PLC Tx/Rx. 8 bits.

Access: R/W.

Default Value: 0.

MAC_WRP_PHY_PARAM_NOISE_PEAK_POWER, index 0x016B. (7.3.36. PHY_ID_NOISE_PEAK_POWER (0x016B))

Description: Returns Detected Noise Peak Power on last Noise Capture, in case a Peak was detected (MAC_WRP_PHY_PARAM_RRC_NOTCH_ACTIVE == 1). If no Peak was detected, it returns the Noise average in band measured on last Noise Capture. 8 bits.

Access: R/W.

Default Value: 0.

MAC_WRP_PHY_PARAM_LAST_MSG_LQI, index 0x016C. (7.3.37. PHY_ID_LAST_MSG_LQI (0x016C))

Description: Returns LQI value calculated for last received frame. 8 bits.

Access: R/W.

Default Value: 0.

MAC_WRP_PHY_PARAM_LAST_MSG_RSSI, index 0x016D.

Description: Returns RSSI value calculated for last received frame. Only available in PL360 platform including MAC RT. 16 bits.

Access: R/W.

Default Value: 0.

MAC_WRP_PHY_PARAM_ACK_TX_CFM, index 0x016E.

Description: Counts the number of successful ACK transmissions. Only available in PL360 platform including MAC RT. 16 bits.

Access: R/W.

Default Value: 0.

MAC_WRP_PHY_PARAM_TONE_MAP_RSP_ENABLED_MODS, index 0x0174. (7.3.40. PHY_ID_TONE_MAP_RSP_ENABLED_MODS (0x0174))

Description: Enables / Disables modulations to be chosen by the PHY layer algorithm when a Tone Map Response has to be sent. 8 bits.

Access: R/W.

Default Value: 0xFF.

9.7.81 MAC_WRP_PIB_DSN_RF (0x00000200)

Data frame sequence number.

Access: R/W.

Value Range: 0-255.

Default Value: random.

9.7.82 MAC_WRP_PIB_MAX_BE_RF (0x00000201)

Maximum value of back-off exponent. It should always be greater than macMinBERF.

Access: R/W.

Value Range: 3–8.

Default Value: 5.

9.7.83 MAC_WRP_PIB_MAX_CSMA_BACKOFFS_RF (0x00000202)

Maximum number of back-off attempts.

Access: R/W.

Value Range: 0-5.

Default Value: 4.

9.7.84 MAC_WRP_PIB_MAX_FRAME_RETRIES_RF (0x00000203)

Maximum number of retransmission.

Access: R/W.

Value Range: 0-7.

Default Value: 3.

9.7.85 MAC_WRP_PIB_MIN_BE_RF (0x00000204)

Minimum value of back-off exponent.

Access: R/W.

Value Range: 0–macMaxBeRf.

Default Value: 3.

9.7.86 MAC_WRP_PIB_TIMESTAMP_SUPPORTED_RF (0x00000205)

Support of the optional timestamping feature for incoming and outgoing Data frames.

Access: Read-only.

Value Range: Always set to False (Timestamp feature not supported).

Default Value: False.

9.7.87 MAC_WRP_PIB_DEVICE_TABLE_RF (0x00000206)

The device table as specified in the G3 Spec. It stores the Frame Counter of the neighbour nodes from which the frames are received on the RF medium.

Access: Read only.

Number of entries: 128 by default. Can be modified through MAC_MAX_DEVICE_TABLE_ENTRIES_RF definition in Mac Wrapper.

Entry format:

```
struct TDeviceTableEntry {
  TPanId m_nPanId; // 2 bytes
  TShortAddress m_nShortAddress; // 2 bytes
  uint32_t m_u32FrameCounter; // 4 bytes
};
```

Default Value: empty table.

Get Entry: Call PIB Get function with index [0-127], the returned value will have the format described above.

9.7.88 MAC_WRP_PIB_FRAME_COUNTER_RF (0x00000207)

The outgoing frame counter for this device. RF MAC layer maintains its own Frame Counter (independent from PLC one). This value has to be restored across power fail.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.89 MAC_WRP_PIB_DUPLICATE_DETECTION_TTL_RF (0x00000208)

Maximum time a received pair (source address + sequence-number) is retained for duplicate frame detection, in seconds.

Access: R/W.

Value Range: 0-255.

Default Value: 3.

9.7.90 MAC_WRP_PIB_COUNTER_OCTETS_RF (0x00000209)

The size of the MAC metrics counters in octets.

Access: Read-only.

Value Range: Always set to 4.

Default Value: 4.

9.7.91 MAC_WRP_PIB_RETRY_COUNT_RF (0x0000020A)

The number of transmitted frames that required exactly one retry before acknowledgment.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.92 MAC_WRP_PIB_MULTIPLE_RETRY_COUNT_RF (0x0000020B)

The number of transmitted frames that required more than one retry before acknowledgment.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.93 MAC_WRP_PIB_TX_FAIL_COUNT_RF (0x0000020C)

The number of transmitted frames that did not result in an acknowledgment after macMaxFrameRetries.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.94 MAC_WRP_PIB_TX_SUCCESS_COUNT_RF (0x0000020D)

The number of transmitted frames that were acknowledged successfully after the initial Data frame transmission.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.95 MAC_WRP_PIB_FCS_ERROR_COUNT_RF (0x0000020E)

The number of frames that were discarded due to an incorrect FCS.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.96 MAC_WRP_PIB_SECURITY_FAILURE_COUNT_RF (0x0000020F)

The number of frames that were returned from the security procedure with any Status other than SUCCESS.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.97 MAC_WRP_PIB_DUPLICATE_FRAME_COUNT_RF (0x00000210)

The number of frames that contained the same sequence number as a frame previously received.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.98 MAC_WRP_PIB_RX_SUCCESS_COUNT_RF (0x00000211)

The number of frames that were received correctly.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.99 MAC_WRP_PIB_NACK_COUNT_RF (0x00000212)

The number of transmitted frames that were not acknowledged.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.100 MAC_WRP_PIB_USE_ENHANCED_BEACON_RF (0x00000213)

Use of Enhanced Beacon by this device.

Access: Read-only.

Value Range: Always set to True.

Default Value: True.

9.7.101 MAC_WRP_PIB_EB_HEADER_IE_LIST_RF (0x00000214)

List of IDs of Header IEs to include in the Enhanced Beacon. Only vendor-specific Header IEs are used in this implementation.

Access: Read-only.

Value Range: Always one element containing ID 0x00 (Vendor Specific IE).

Default Value: 0x00.

9.7.102 MAC_WRP_PIB_EB_PAYLOAD_IE_LIST_RF (0x00000215)

List of IDs of Payload IEs to include in the Enhanced Beacon. No Payload IEs are used in this implementation.

Access: Read-only.

Value Range: Always empty list (length 0).

Default Value: N/A.

9.7.103 MAC_WRP_PIB_EB_FILTERING_ENABLED_RF (0x00000216)

Indicates if devices should perform filtering in response to Enhanced Beacon Request command.

Access: Read-only.

Value Range: Always set to False.

Default Value: False.

9.7.104 MAC_WRP_PIB_EBSN_RF (0x00000217)

BSN used for Enhanced Beacon frames.

Access: R/W.

Value Range: 0-255.

Default Value: random.

9.7.105 MAC_WRP_PIB_EB_AUTO_SA_RF (0x00000218)

Indicates if Enhanced Beacon frames generated by the MAC in response to Enhanced Beacon frame include Source Address field.

Access: Read-only.

Value Range: Always set to Short Address identifier (2).

Default Value: 2.

9.7.106 MAC_WRP_PIB_SEC_SECURITY_LEVEL_LIST_RF (0x0000021A)

Security Level Descriptor of Frame Type indicated by index.

Access: Read-only.

Entry format:

```
struct TSecurityLevelDescriptorRF {
    uint8_t m_u8FrameType;
    uint8_t m_u8CommandId;
    uint8_t m_u8SecurityMinimum; // Fixed to 0
    bool m_bOverrideSecurityMinimum; // Fixed to False
};
```

Default Value: N/A.

9.7.107 MAC_WRP_PIB_POS_TABLE_RF (0x0000021C)

The POS table.

Access: R/W.

Number of entries: Default configuration is set to 100 for Device, 500 for Coordinator Lite, 2000 for Coordinator. The table size is configurable for each project by means of the *conf_tables.h* file.

Entry format:

```
struct TPOSEntry {
    TShortAddress m_nShortAddress; // 2 bytes
    uint8_t m_u8ForwardLqi;
    uint8_t m_u8ReverseLqi;
    uint8_t m_u8DutyCycle;
    uint8_t m_u8ForwardTxPowerOffset;
    uint8_t m_u8ReverseTxPowerOffset;
    uint16_t m_u16POSValidTime; // Entry validity time (in seconds)
};
```

This table is ordered according to m_u16POSValidTime. Newest entries are placed at top of the table.

Default Value: empty table.

Get Entry: Call Get function with index [0-99], returned value will have the format described above.

Add Entry: Call Set function with index [0-99], index will be ignored and new entry will be placed according to m_u16POSValidTime as described above.

Delete Entry: Set entry with the short address to be deleted and valid time equal to 0. Index is ignored and the entry will be searched by short address and deleted.

To get all the active entries, first step to get the number of such active entries by means of 9.7.136. MAC_WRP_PIB_MANUF_POS_TABLE_COUNT_RF (0x08000218). Then, because the table is ordered, all entries can be obtained by iterating from 0 to the number of active entries.

9.7.108 MAC_WRP_PIB_OPERATING_MODE_RF (0x0000021D)

The RF operating mode as defined in clause 19.3 of [IEEE 802.15.4-2020]. Tables 19.7 and 19.8.

Operating Mode defines Data Rate, Modulation, Modulation Index and Channel Spacing for a given working band.

Access: R/W.

Value Range: 1-4.

Default Value: 1.

9.7.109 MAC_WRP_PIB_CHANNEL_NUMBER_RF (0x0000021E)

The channel number as defined in clause 10.1.3.9 of [IEEE 802.15.4-2020] to be used for channel scanning and data transmission.

Available channels depend on working band, as well as channel spacing.

Access: R/W.

Value Range: 0-7279.

Default Value: 0.

9.7.110 MAC_WRP_PIB_DUTY_CYCLE_USAGE_RF (0x0000021F)

Current usage of maximum allowed duty cycle in percent over the current sliding-window measurement period, i.e., (t_on / macDutyCycleLimit_RF) * 100.

Access: R/W.

Value Range: 0-100.

Default Value: 0.

9.7.111 MAC_WRP_PIB_DUTY_CYCLE_PERIOD_RF (0x00000220)

Duration of the measurement period in seconds, e.g., 3600 seconds as default for [ETSI EN 303 204].

Access: R/W.

Value Range: 1-65535.

Default Value: 3600.

9.7.112 MAC_WRP_PIB_DUTY_CYCLE_LIMIT_RF (0x00000221)

Duration of the allowed transmission time in seconds, e.g., 2.5%/10% of macDutyCyclePeriod_RF (3600 seconds as default for [ETSI EN 303 204]).

Tip: If this attribute is set equal to macDutyCyclePeriod_RF the value of macDutyCycleUsage_RF will be fixed to zero, and, thus, transmission will never be canceled due to Duty Cycle limits. This is the recommended setting for test environments, where no regulation is required and maximum throughput is desired.

Access: R/W.

Value Range: 1-65535.

Default Value: 90 (PAN Device), 360 (PAN Coordinator).

9.7.113 MAC_WRP_PIB_DUTY_CYCLE_THRESHOLD_RF (0x00000222)

Duty cycle threshold for stopping RF transmissions.

Access: R/W.

Value Range: 0-100.

Default Value: 90.

9.7.114 MAC_WRP_PIB_DISABLE_PHY_RF (0x00000223)

Disable RF PHY Tx and Rx.

Access: R/W.

Value Range: True / False.

Default Value: False.

9.7.115 MAC_WRP_PIB_MANUF_SECURITY_RESET_RF (0x08000203)

Resets the device table. Intended for testing purposes, not to be used in final products.

Access: Write only.

9.7.116 MAC_WRP_PIB_MANUF_LBP_FRAME_RECEIVED_RF (0x08000204)

Flag indicating whether LBP frames are being received since last time MIB was set to 0. This IB is intended to be used for collision avoidance during bootstrap phase.

Access: R/W.

Value Range: 0-1.

Default value: 0.

9.7.117 MAC_WRP_PIB_MANUF_LNG_FRAME_RECEIVED_RF (0x08000205)

Flag indicating whether LOADng frames are being received since last time MIB was set to 0. This IB is intended to be used for collision avoidance during bootstrap phase.

Access: R/W.

Value Range: 0-1.

Default value: 0.

9.7.118 MAC_WRP_PIB_MANUF_BCN_FRAME_RECEIVED_RF (0x08000206)

Flag indicating whether Beacon frames are being received since last time MIB was set to 0. This IB is intended to be used for collision avoidance during bootstrap phase.

Access: R/W.

Value Range: 0-1.

Default value: 0.

9.7.119 MAC_WRP_PIB_MANUF_RX_OTHER_DESTINATION_COUNT_RF (0x08000207)

Statistic counter of discarded packets due to other destination.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.120 MAC_WRP_PIB_MANUF_RX_INVALID_FRAME_LENGTH_COUNT_RF (0x08000208)

Statistic counter of discarded packets due to invalid frame length.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.121 MAC_WRP_PIB_MANUF_RX_WRONG_ADDR_MODE_COUNT_RF (0x08000209)

Statistic counter of discarded packets due to Wrong Addressing Mode.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.122 MAC_WRP_PIB_MANUF_RX_UNSUPPORTED_SECURITY_COUNT_RF (0x0800020A)

Statistic counter of discarded packets due to unsupported security.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.123 MAC_WRP_PIB_MANUF_RX_WRONG_KEY_ID_COUNT_RF (0x0800020B)

Statistic counter of discarded packets due to wrong key ID.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.124 MAC_WRP_PIB_MANUF_RX_INVALID_KEY_COUNT_RF (0x0800020C)

Statistic counter of discarded packets due to invalid key.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.125 MAC_WRP_PIB_MANUF_RX_WRONG_FC_COUNT_RF (0x0800020D)

Statistic counter of discarded packets due to wrong frame counter.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.126 MAC_WRP_PIB_MANUF_RX_DECRYPTION_ERROR_COUNT_RF (0x0800020E)

Statistic counter of discarded packets due to decryption error.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.127 MAC_WRP_PIB_MANUF_TX_DATA_PACKET_COUNT_RF (0x0800020F)

Statistic counter of successfully transmitted unicast MSDUs.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.128 MAC_WRP_PIB_MANUF_RX_DATA_PACKET_COUNT_RF (0x08000210)

Statistic counter of successfully received unicast MSDUs.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.129 MAC_WRP_PIB_MANUF_TX_CMD_PACKET_COUNT_RF (0x08000211)

Statistic counter of successfully transmitted command packets.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.130 MAC_WRP_PIB_MANUF_RX_CMD_PACKET_COUNT_RF (0x08000212)

Statistic counter of successfully received command packets.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.131 MAC_WRP_PIB_MANUF_CSMA_FAIL_COUNT_RF (0x08000213)

Counts the number of times the CSMA back-offs reach macMaxCSMABackoffsRF.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.132 MAC_WRP_PIB_MANUF_RX_DATA_BROADCAST_COUNT_RF (0x08000214)

Statistic counter of successfully received broadcast frames.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.133 MAC_WRP_PIB_MANUF_TX_DATA_BROADCAST_COUNT_RF (0x08000215)

Statistic counter of the number of broadcast frames sent.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.134 MAC_WRP_PIB_MANUF_BAD_CRC_COUNT_RF (0x08000216)

Statistic counter of the number of frames received with bad CRC.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 0.

9.7.135 MAC_WRP_PIB_MANUF_ENABLE_MAC_SNIFFER_RF (0x08000217)

RF MAC Sniffer mode enabled. This mode processes all frames at MAC level, regardless of addressing, and outputs all correct frames by means of MacMcpsMacSnifferIndicationRF callback. The difference with Promiscuous mode is that, in Sniffer mode, the device will still work as a regular node in the network, treating frames addressed to it like a normal device would do. Summarizing, Sniffer mode is the combination of Normal mode plus Promiscuous mode.

Access: R/W.

Value Range: 0-1.

Default Value: 0.

9.7.136 MAC_WRP_PIB_MANUF_POS_TABLE_COUNT_RF (0x08000218)

Gets number of valid elements in the RF POS Table.

Access: Read only.

Value Range: 0-65535.

Default value: 0.

9.7.137 MAC_WRP_PIB_MANUF_MAC_INTERNAL_VERSION_RF (0x08000219)

Gets current RF MAC internal version number.

Entry format:

```
struct TMacSoftVersion {
  uint8_t m_u8Major;
  uint8_t m_u8Minor;
  uint8_t m_u8Revision;
  uint8_t m_u8Year; // year since 2000
  uint8_t m_u8Month;
  uint8_t m_u8Day;
};
```

Access: Read only.

Value Range: N/A.

Default value: Current RF MAC version.

9.7.138 MAC_WRP_PIB_MANUF_RESET_MAC_STATS_RF (0x0800021A)

Resets all IBs related to RF MAC statistics at once.

Access: W.

Value Range: 8-bits. Any value triggers statistics reset.

Default Value: N/A.

9.7.139 MAC_WRP_PIB_MANUF_POS_TABLE_ELEMENT_RF (0x0800021B)

This object has the same content as the RF POS table, but it is retrieved using the short address of the node, instead of the index inside the table. To do so, set the short address in the index field in the Get IB function.

Entry format:

```
struct TPOSEntry {
    TShortAddress m_nShortAddress; // 2 bytes
    uint8_t m_u8ForwardLqi;
    uint8_t m_u8ReverseLqi;
    uint8_t m_u8DutyCycle;
    uint8_t m_u8ForwardTxPowerOffset;
    uint8_t m_u8ReverseTxPowerOffset;
    uint16_t m_u16POSValidTime; // Entry validity time (in seconds)
};
```

Access: Read-only.

9.7.140 MAC_WRP_PIB_MANUF_ACK_TX_DELAY_RF (0x0800021C)

Time (in µs) between end of reception and start of ACK transmission.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 2000.

9.7.141 MAC_WRP_PIB_MANUF_ACK_RX_WAIT_TIME_RF (0x0800021D)

Time (in µs) to wait for an ACK reception after end of frame transmission before time-out.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 30000.

9.7.142 MAC_WRP_PIB_MANUF_ACK_CONFIRM_WAIT_TIME_RF (0x0800021E)

Time (in µs) to wait for an ACK transmission confirm after TX request, before time-out.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 30000.

9.7.143 MAC_WRP_PIB_MANUF_DATA_CONFIRM_WAIT_TIME_RF (0x0800021F)

Time (in µs) to wait for an Data Frame transmission confirm after TX request, before time-out.

Access: R/W.

Value Range: 0-4294967295.

Default Value: 90000.

9.7.144 MAC_WRP_PIB_MANUF_PHY_PARAM_RF (0x08000220)

Gives access to some RF PHY layer parameters and statistics. It is treated as a table, where the index determines which parameter to access. Available parameters and their corresponding indexes are the following:

MAC_WRP_RF_PHY_PARAM_DEVICE_ID, index 0x0000.

Description: RF device identifier. 16 bits.

Access: Read only.

Default Value: N/A.

MAC_WRP_RF_PHY_PARAM_FW_VERSION, index 0x0001.

Description: RF PHY layer firmware version number. 6 bytes.
{major, minor, revision, year, month, day}

Access: Read only.

Default Value: N/A.

MAC_WRP_RF_PHY_PARAM_DEVICE_RESET, index 0x0002.

Description: This IB performs an RF device reset by means of its Reset Pin. 8 bits.

Access: Write only.

Default Value: N/A.

MAC_WRP_RF_PHY_PARAM_TRX_RESET, index 0x0080.

Description: This IB performs an RF transceiver (RF09 or RF24) reset through an SPI command. It allows resetting only one of the two available transceivers. 8 bits.

Access: Write only.

Default Value: N/A.

MAC_WRP_RF_PHY_PARAM_TRX_SLEEP, index 0x0081.

Description: RF transceiver (RF09 or RF24) sleep. 8 bits.

Access: Write only.

Default Value: N/A.

MAC_WRP_RF_PHY_PARAM_PHY_CONFIG, index 0x0100.

Description: RF PHY configuration structure (see “at86rf_phy_cfg_t” in code).

Access: R/W.

Default Value: N/A.

MAC_WRP_RF_PHY_PARAM_PHY_BAND_OPERATING_MODE, index 0x0101.

Description: RF PHY band and operating mode. 16 bits.

Access: R/W.

Default Value: AT86RF_SUN_FSK_BAND_863_OPM1.

MAC_WRP_RF_PHY_PARAM_PHY_CHANNEL_NUM, index 0x0120.

Description: RF channel number used for transmission and reception. 16 bits.

Access: R/W.

Default Value: 29 (866 MHz in default Band and Operating Mode).

MAC_WRP_RF_PHY_PARAM_PHY_CHANNEL_FREQ_HZ, index 0x0121.

Description: RF frequency in Hz used for transmission and reception. 32 bits.

Access: Read only.

Default Value: 866000000.

MAC_WRP_RF_PHY_PARAM_PHY_CCA_ED_CONFIG, index 0x0140.

Description: Configuration of Energy Detection for CCA. 3 bytes (see “at86rf_cca_ed_cfg_t” in code). {duration, threshold}

Access: R/W.

Default Value: {160, -81}.

MAC_WRP_RF_PHY_PARAM_PHY_CCA_ED_DURATION, index 0x0141.

Description: Duration in us of Energy Detection for CCA. 16 bits.

Access: R/W.

Default Value: 160.

MAC_WRP_RF_PHY_PARAM_PHY_CCA_ED_THRESHOLD, index 0x0142.

Description: Threshold in dBm of for CCA with Energy Detection. 8 bits.

Access: R/W.

Default Value: -81.

MAC_WRP_RF_PHY_PARAM_PHY_TURNAROUND_TIME, index 0x0160.

Description: Turnaround time in us. 16 bits.

Access: Read only.

Default Value: 1000.

MAC_WRP_RF_PHY_PARAM_PHY_TX_PAY_SYMBOLS, index 0x0180.

Description: Number of payload symbols in last transmitted message. 16 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_PAY_SYMBOLS, index 0x0181.

Description: Number of payload symbols in last received message. 16 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_TOTAL, index 0x01A0.

Description: Successfully transmitted messages count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_TOTAL_BYTES, index 0x01A1.

Description: Successfully transmitted bytes count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_TOTAL, index 0x01A2.

Description: Total Transmission errors count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_BUSY_TX, index 0x01A3.

Description: Transmission errors count due to already in transmission. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_BUSY_RX, index 0x01A4.

Description: Transmission errors count due to already in reception. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_BUSY_CHN, index 0x01A5.

Description: Transmission errors count due to busy channel. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_BAD_LEN, index 0x01A6.

Description: Transmission errors count due to bad message length. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_BAD_FORMAT, index 0x01A7.

Description: Transmission errors count due to bad format. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_TIMEOUT, index 0x01A8.

Description: Transmission errors count due to timeout. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_ERR_ABORTED, index 0x01A9.

Description: Transmission aborted count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_TX_CFM_NOT_HANDLED, index 0x01AA.

Description: Transmission confirms not handled by upper layer count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_TOTAL, index 0x01B0.

Description: Successfully received messages count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_TOTAL_BYTES, index 0x01B1.

Description: Successfully received bytes count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_ERR_TOTAL, index 0x01B2.

Description: Total Reception errors count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_ERR_FALSE_POSITIVE, index 0x01B3.

Description: Reception false positive count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_ERR_BAD_LEN, index 0x01B4.

Description: Reception errors count due to bad message length. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_ERR_BAD_FORMAT, index 0x01B5.

Description: Reception errors count due to bad format or bad FCS in header. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_ERR_BAD_FCS_PAY, index 0x01B6.

Description: Reception errors count due to bad FCS in payload. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_ERR_ABORTED, index 0x01B7.

Description: Reception aborted count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_OVERRIDE, index 0x01B8.

Description: Reception override (another message with higher signal level) count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_RX_IND_NOT_HANDLED, index 0x01B9.

Description: Reception indications not handled by upper layer count. 32 bits.

Access: Read only.

Default Value: 0.

MAC_WRP_RF_PHY_PARAM_PHY_STATS_RESET, index 0x01C0.

Description: Reset Phy Statistics. 8 bits.

Access: Write only.

Default Value: N/A.

MAC_WRP_RF_PHY_PARAM_MAC_UNIT_BACKOFF_PERIOD, index 0x0200.

Description: Backoff period unit in us (aUnitBackoffPeriod in IEEE 802.15.4) used for CSMA-CA . 16 bits.

Access: Read only.

Default Value: 1160.

MAC_WRP_RF_PHY_PARAM_TX_FSK_FEC, index 0x8000.

Description: SUN FSK FEC enabled or disabled for transmission (phyFskFecEnabled in IEEE 802.15.4). 8 bits.

Access: R/W.

Default Value: 0 (Disabled).

MAC_WRP_RF_PHY_PARAM_TX_OFDM_MCS, index 0x8001.

Description: SUN OFDM MCS (Modulation and coding scheme) used for transmission. 8 bits.

Access: R/W.

Default Value: 0 (BPSK, 1/2 convolutional encoder rate, 4x frequency repetition).

9.8 Hybrid Abstraction Layer

The Hybrid Abstraction Layer, or HyAL, has the role of abstracting the upper layers (ADP, LOADng, LBP) from having 2 different MAC layers in the Hybrid Profile.

The intention when the Hybrid Profile was defined, was to keep those upper layers as agnostic as possible from the fact that there may be only a PLC MAC layer or two (PLC & RF) MAC layers below. Thus, a unique API was needed regardless of the MAC layers present.

To do so, HyAL has the API of an IEEE802.15.4 MAC layer, but adding a new parameter to some of the primitives, the **MediaType**, is explained in the following sections.

Refer to 1. General Architecture for a complete view of the layers on the G3 stack.

9.8.1 HyAL SAP

9.8.1.1 HyALInitialize

```
void HyALInitialize(struct THyALNotifications *pNotifications, uint8_t u8Band);
```

This function initializes the PLC and RF MAC layers and the callbacks for upper layers.

Parameters:

pNotifications	Structure with callbacks used to notify ADP with events coming from both PLC and RF MAC layers
u8Band	Working PLC band (should be inline with the hardware)

```
struct THyALNotifications {
    HyALDataConfirm m_HyALDataConfirm;
    HyALDataIndication m_HyALDataIndication;
    HyALGetConfirm m_HyALGetConfirm;
    HyALSetConfirm m_HyALSetConfirm;
    HyALResetConfirm m_HyALResetConfirm;
    HyALBeaconNotify m_HyALBeaconNotifyIndication;
    HyALScanConfirm m_HyALScanConfirm;
    HyALStartConfirm m_HyALStartConfirm;
    HyALCommStatusIndication m_HyALCommStatusIndication;
    HyALSnifferIndication m_HyALSnifferIndication;
};

uint8_t u8Band;
    ADP_BAND_CENELEC_A = 0
    ADP_BAND_CENELEC_B = 1
    ADP_BAND_FCC = 2
    ADP_BAND_ARIB = 3
```

9.8.1.2 HyALEventHandler

```
void HyALEventHandler(void);
```

Performs periodic tasks of Hybrid Abstraction Layer and calls RF and PLC MAC Event Handlers.

9.8.2 HyAL Primitives

9.8.2.1 HyAL-DATA.request

```
void HyALDataRequest(struct THyALDataRequest *pParameters);
```

The Data Request primitive is the equivalent of MCPS-DATA.request, but with the *MediaType* field added to the parameters structure. MediaType usage is detailed in 9.8.3. Media Type Definition.

By receiving this command, the hybrid abstraction layer forwards the frame to the appropriate MAC sublayer, based on the MediaType and DstAddr parameters. For unicast transmissions (DstAddr different from 0xFFFF):

- If MediaType is 0x00, the hybrid abstraction layer sends a MCPS-DATA.request to the PLC MAC. If the transmission is successful, a HyAL-DATA.confirm is generated with status SUCCESS and MediaType = 0x00. If the transmission fails (MCPS-DATA.confirm from PLC MAC with status different from SUCCESS), a second attempt will be performed on the backup medium.
- If MediaType is 0x01, the hybrid abstraction layer sends a MCPS-DATA.request to the RF MAC. If the transmission is successful, a HyAL-DATA.confirm is generated with status SUCCESS and MediaType = 0x01. If the transmission fails (MCPS-DATA.confirm from RF MAC with status different from SUCCESS), a second attempt will be performed on the backup medium.
- If MediaType is 0x03, the hybrid abstraction layer sends a MCPS-DATA.request to the PLC MAC. On reception of the corresponding MCPS-DATA.confirm from the PLC MAC, a HyAL-DATA.confirm is generated, with status = MCPS-DATA.confirm status and MediaType = 0x00.
- If MediaType is 0x04, the hybrid abstraction layer sends a MCPS-DATA.request to the RF MAC. On reception of the corresponding MCPS-DATA.confirm from the RF MAC, a HyAL-DATA.confirm is generated, with status = MCPS-DATA.confirm status and MediaType = 0x01.

For broadcast transmissions (DstAddr = 0xFFFF) the hybrid abstraction layer sends a MCPS-DATA.request to the PLC MAC and to the RF MAC:

- If the transmission is successful on either of the two media, a HyAL-DATA.confirm is generated with status SUCCESS and MediaType = 0x02.
- If transmission is not possible on any media, a HyAL-DATA.confirm is generated with status set to the proper error and MediaType = 0x02.

9.8.2.2 HyAL-DATA.confirm

```
typedef void (*HyALDataConfirm)(struct THyALDataConfirm *pParameters);
```

The Data Confirm primitive is the equivalent of MCPS-DATA.confirm, but with the *MediaType* field added to the parameters structure. MediaType usage is detailed in 9.8.3. Media Type Definition.

The hybrid abstraction layer generates a HyAL-DATA.confirm following the receipt of (at least) a MCPS-DATA.confirm from the layer below.

In case of broadcast transmission (DstAddr = 0xFFFF), only one HyAL-DATA.confirm is generated after receiving confirmation both from PLC MAC and RF MAC.

9.8.2.3 HyAL-DATA.indication

```
typedef void (*HyALDataIndication)(struct THyALDataIndication *pParameters);
```

The Data Indication primitive is the equivalent of the MCPS-DATA.indication, but with the *MediaType* field added to the parameters structure. MediaType usage is detailed in 9.8.3. Media Type Definition.

The hybrid abstraction layer generates a HyAL-DATA.indication following the receipt of a MCPS-DATA.indication, either from MAC PLC or MAC RF.

A duplicate detection is implemented in the HyAL to drop indications for the same frame coming from a second medium, if already received on the other.

9.8.2.4 HyAL-SCAN.request

```
void HyALScanRequest(struct THyALScanRequest *pParameters);
```

By receiving this command, the hybrid abstraction layer initiates the scan procedure sending a MLME-SCAN.request both to the PLC MAC and to the RF MAC.

9.8.2.5 HyAL-SCAN.confirm

```
typedef void (*HyALScanConfirm)(struct THyALScanConfirm *pParameters);
```

The hybrid abstraction layer generates a HyAL-SCAN.confirm following the receipt of a MLME-SCAN.confirm from both the PLC MAC and the RF MAC.

The status returned in HyAL-SCAN.confirm is SUCCESS if the scan procedure was successful on at least one of the two interfaces.

9.8.2.6 HyAL-BEACON-NOTIFY.indication

```
typedef void (*HyALBeaconNotify)(struct THyALBeaconNotifyIndication *pParameters);
```

The Beacon Notify Indication primitive is the equivalent of MLME-BEACON-NOTIFY.indication, but with the *MediaType* field added to the PAN Descriptor structure. MediaType usage is detailed in 9.8.3. Media Type Definition.

When receiving a MLME-BEACON-NOTIFY.indication, either from RF MAC or from PLC MAC, the hybrid abstraction layer fills the HyAL PAN Descriptor structure setting the MediaType properly and sends the HyAL-BEACON-NOTIFY.indication command to the higher layer.

9.8.2.7 HyAL-COMM-STATUS.indication

```
typedef void (*HyALCommStatusIndication)(struct THyALCommStatusIndication *pParameters);
```

The Comm Status Indication primitive is the equivalent of MLME-COMM-STATUS.indication, but with the *MediaType* field added to the parameters structure. MediaType usage is detailed in 9.8.3. Media Type Definition.

When receiving a MLME-COMM-STATUS.indication, either from RF MAC or from PLC MAC, the hybrid abstraction layer forwards the event through the HyAL-COMM-STATUS.indication command, setting the MediaType properly.

9.8.2.8 HyAL-START.request

```
void HyALStartRequest(struct THyALStartRequest *pParameters);
```

When receiving a HyAL-START.request primitive, the hybrid abstraction layer invokes the start requests of both RF and PLC MAC layers.

9.8.2.9 HyAL-START.confirm

```
typedef void (*HyALStartConfirm)(struct THyALStartConfirm *pParameters);
```

The hybrid abstraction layer generates a HyAL-START.confirm following the receipt of a MLME-START.confirm from both RF and PLC MAC layers.

The status returned in HyAL-START.confirm is SUCCESS if the start procedure was successful on both interfaces. Alternately, if the procedure failed on either one of the two interfaces or both, the first received error code value is returned.

9.8.2.10 HyAL-RESET.request

```
void HyALResetRequest(struct THyALResetRequest *pParameters);
```

When receiving a HyAL-RESET.request primitive, the hybrid abstraction layer is reset and, then, the reset requests for both RF and PLC MAC layers are invoked.

9.8.2.11 HyAL-RESET.confirm

```
typedef void (*HyALResetConfirm)(struct THyALResetConfirm *pParameters);
```

The hybrid abstraction layer generates a HyAL-RESET.confirm following the receipt of a MLME-RESET.confirm from both RF and PLC MAC layers.

The status returned in HyAL-RESET.confirm is SUCCESS if the start procedure was successful on both interfaces. Alternately, if the procedure failed on either one of the two interfaces or both, the first received error code value is returned.

9.8.2.12 HyAL-GET.request

```
void HyALGetRequest(struct THyALGetRequest *pParameters);
```

By receiving this command, the hybrid abstraction layer inspects the parameters to check whether the target MAC IB belongs to the PLC MAC or to the RF MAC, and, then, forwards the request to the corresponding MAC layer.

As an alternative, a Synchronous version of this function is provided, on which the result and value are returned as part of the function call, instead of waiting for a Confirm.

```
enum EMacWrpStatus HyALGetRequestSync(enum EMacWrpPibAttribute eAttribute, uint16_t u16Index,
struct TMacWrpPibValue *pValue);
```

9.8.2.13 HyAL-GET.confirm

```
typedef void (*HyALGetConfirm)(struct THyALGetConfirm *pParameters);
```

The hybrid abstraction layer generates a HyAL-GET.confirm following the receipt of a MLME-GET.confirm from either the PLC MAC or the RF MAC, and forwards it to upper layer.

9.8.2.14 HyAL-SET.request

```
void HyALSetRequest(struct THyALSetRequest *pParameters);
```

By receiving this command, the hybrid abstraction layer inspects the parameters to check whether the target MAC IB belongs to the PLC MAC or to the RF MAC, and, then, forwards the request to the corresponding MAC layer.

As an alternative, a Synchronous version of this function is provided, on which the result is returned as part of the function call, instead of waiting for a Confirm.

```
enum EMacWrpStatus HyALSetRequestSync(enum EMacWrpPibAttribute eAttribute, uint16_t u16Index,
const struct TMacWrpPibValue *pValue);
```

9.8.2.15 HyAL-SET.confirm

```
typedef void (*HyALSetConfirm)(struct THyALSetConfirm *pParameters);
```

The hybrid abstraction layer generates a HyAL-SET.confirm following the receipt of a MLME-SET.confirm from either the PLC MAC or the RF MAC, and forwards it to the upper layer.

9.8.2.16 HyAL-SNIFFER.indication

```
typedef void (*HyALSnifferIndication)(struct THyALSnifferIndication *pParameters);
```

When receiving a MLME-SNIFFER.indication, either from RF MAC or from PLC MAC, the hybrid abstraction layer forwards the event through the HyAL-SNIFFER.indication command.

9.8.3 Media Type Definition

The MediaType parameter used in many of the HyAL primitives has the following meaning, depending on the primitive type:

Table 9-1. MediaType Definition

MediaType Value	Primitive Type		
	Request	Confirm	Indication
0x00	Power Line interface Backup Radio Frequency interface	Power Line interface used by default	Power Line interface
0x01	Radio Frequency interfaceBackup Power Line interface	Radio Frequency interface used by default	Radio Frequency interface
0x02	Both Power Line and Radio Frequency interfaces	Both Power Line and Radio Frequency interfaces used for transmission	Not used

...........continued

MediaType Value	Primitive Type		
	Request	Confirm	Indication
0x03	Power Line interface No backup interface	Power Line interface used as backup after failure on Radio Frequency interface	Not used
0x04	Radio Frequency interface No backup interface	Radio Frequency interface used as backup after failure on Power Line interface	Not used

9.8.4 Backup Medium Usage

The hybrid abstraction layer implements a functionality that, when allowed, performs a second transmission attempt using the backup medium (the one that was not primarily intended for transmission).

This procedure occurs when the hybrid abstraction layer sends a MCPS-DATA.request to the lower layer (PLC MAC or RF MAC) and it receives a MCPS-DATA.confirm with a status different from SUCCESS.

Two scenarios are, therefore, possible:

- HyAL-DATA.request with MediaType = 0x00 (PLC). If the transmission over PLC medium fails (MCPS-DATA.confirm from PLC MAC with a status different from SUCCESS) the hybrid abstraction layer will check the RF POS table:
  - IF the DstAddr is in extended address mode or IF a valid entry is found for the DstAddr specified in the HyAL-DATA.request, send a MCPS-DATA.request to the RF MAC, and wait for the corresponding confirm.
    - IF the MCPS-DATA.confirm status is SUCCESS, generate a HyAL-DATA.confirm with SUCCESS status and MediaType = 0x04 (transmission over PLC failed).
    - ELSE, generate a HyAL-DATA.confirm with the status set to the proper error and MediaType = 0x04 (subsequent transmissions over PLC and RF failed, the status provides information for the RF transmission).
  - ELSE, generate a HyAL-DATA.confirm with the status set to the proper error and MediaType = 0x00.
- HyAL-DATA.request with MediaType = 0x01 (RF). If the transmission over the RF medium fails (MCPS-DATA.confirm from RF MAC with Status different from SUCCESS), the hybrid abstraction layer will check the PLC POS table:
  - IF the DstAddr is in extended address mode or IF a valid entry is found for the DstAddr specified in the HyAL-DATA.request, send a MCPS-DATA.request to the PLC MAC and wait for the corresponding confirm.
    - IF the MCPS-DATA.confirm status is SUCCESS, generate a HyAL-DATA.confirm with status SUCCESS and MediaType = 0x03 (transmission over RF failed).
    - ELSE, generate a HyAL-DATA.confirm with the status set to the proper error and MediaType = 0x03 (subsequent transmissions over RF and PLC failed, the status provides information for the PLC transmission).
  - ELSE, generate a HyAL-DATA.confirm with the status set to the proper error and MediaType = 0x01.

9.9 ADP SAP

Functions detailed here (initialization and event handler) are not part of the G3 Spec.

These are functions specific to Microchip implementation of the G3 Stack. Layers above ADP are in charge of calling these functions properly.

9.9.1 ADP Initialization Function

Use this function to initialize the ADP layer. The ADP layer should be initialized before doing any other operation on the G3 Stack.

This function takes care of initialization of all lower layers.

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

Parameters:

pNotifications	Structure with callbacks used to notify ADP specific events (if NULL the layer is de-initialized)
band	Working band (should be inline with the hardware)

```
struct TAdpNotifications {
    AdpDataConfirm fnctAdpDataConfirm;
    AdpDataIndication fnctAdpDataIndication;
    AdpDiscoveryConfirm fnctAdpDiscoveryConfirm;
    AdpDiscoveryIndication fnctAdpDiscoveryIndication;
    AdpNetworkStartConfirm fnctAdpNetworkStartConfirm;
    AdpNetworkJoinConfirm fnctAdpNetworkJoinConfirm;
    AdpNetworkLeaveIndication fnctAdpNetworkLeaveIndication;
    AdpNetworkLeaveConfirm fnctAdpNetworkLeaveConfirm;
    AdpResetConfirm fnctAdpResetConfirm;
    AdpSetConfirm fnctAdpSetConfirm;
    AdpMacSetConfirm fnctAdpMacSetConfirm;
    AdpGetConfirm fnctAdpGetConfirm;
    AdpMacGetConfirm fnctAdpMacGetConfirm;
    AdpLbpConfirm fnctAdpLbpConfirm;
    AdpLbpIndication fnctAdpLbpIndication;
    AdpRouteDiscoveryConfirm fnctAdpRouteDiscoveryConfirm;
    AdpPathDiscoveryConfirm fnctAdpPathDiscoveryConfirm;
    AdpNetworkStatusIndication fnctAdpNetworkStatusIndication;
    AdpBufferIndication fnctAdpBufferIndication;
    AdpPREQIndication fnctAdpPREQIndication;
    AdpUpdNonVolatileDataIndication fnctAdpUpdNonVolatileDataIndication;
    AdpRouteNotFoundIndication fnctAdpRouteNotFoundIndication;
};
enum TAdpBand {
    ADP_BAND_CENELEC_A = 0, ADP_BAND_CENELEC_B = 1, ADP_BAND_FCC = 2, ADP_BAND_ARIB = 3
};
```

9.9.2 ADP Event Handler Function

Event handler of the ADP layer. It must be called periodically, at the default 1ms rate or the one defined by the user as explained in 3.2.3.1. Time Control Initialization. This function calls the MacWrapperEventHandler function and the Event Handler call propagates through all lower layers. Returns *false* in case ADP Notification callbacks are not initialized, and *true* otherwise.

```
bool AdpEventHandler(void);
```

9.10 ADP Data Primitives

The ADPD is used to transport the NSDU to other devices on the network. Microchip implementation support all standard data primitives specified in G3-PLC Spec. The following chapters enumerate the ADPD functions.

All the functions and structures presented here are defined in files *AdpApi.h* and *AdpApiTypes.h*, available to the user.

Refer to these files for details of structure contents and types.

9.10.1 ADPD-DATA.request

The ADPD-DATA.request primitive requests the transfer of an application PDU (i.e., IPv6 packet) to another device or multiple devices.

```
void AdpDataRequest(uint16_t u16NsduLength, const uint8_t *pNsdu, uint8_t u8NsduHandle, bool
bDiscoverRoute, uint8_t u8QualityOfService);
```

Parameters:

u16NsduLength	The size of the NSDU, in bytes. [Valid range: 0-1280].
pNsdu	The NSDU to send.
u8NsduHandle	The handle of the NSDU to transmit. This parameter is used to identify in the AdpDataConfirm primitive which request it is concerned with. It can be randomly chosen by the application layer.
bDiscoverRoute	If TRUE, a route discovery procedure will be performed prior to sending the frame if a route to the destination is not available in the routing table. If FALSE, no route discovery is performed.
u8QualityOfService	The requested quality of service (QoS) of the frame to send. Allowed values are: [0x00 = normal priority; 0x01 = high priority].

When the execution of the DataRequest is finished, the callback *AdpDataConfirm* is called to inform the user about the status of the operation.

9.10.2 ADPD-DATA.confirm

The ADPD-DATA.confirm primitive reports the result of a previous ADPD-DATA.request primitive.

```
typedef void (*AdpDataConfirm)(struct TAdpDataConfirm *pDataConfirm);
```

The result is reported into the following structure:

```
struct TAdpDataConfirm {

    uint8_t m_u8Status;

    uint8_t m_u8NsduHandle;

};
```

Fields of the structure:

u8Status	The status code of a previous ADPD-DATA.request identified by its NsduHandle.
u8NsduHandle	The handle of the NSDU confirmed by this primitive.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.10.3 ADPD-DATA.indication

The ADPD-DATA.indication primitive is used to transfer received data from the adaptation sublayer to the upper layer.

When user data is received by the modem, the callback *AdpDataIndication* is called:

```
typedef void (*AdpDataIndication)(struct TAdpDataIndication *pDataIndication);
```

The information is reported into the following structure:

```
struct TAdpDataIndication {

    uint16_t m_u16NsduLength;

    const uint8_t *m_pNsdu;

    uint8_t m_u8LinkQualityIndicator;

};
```

Fields of the structure:

m_u16NsduLength	The size of the NSDU, in bytes.
m_pNsdu	Pointer to received NSDU.
m_u8LinkQualityIndicator	The value of the link quality during the reception of the frame.

9.11 ADP Management Service Primitives

The ADPM allows the transport of command frames used for network maintenance. The following chapters enumerate the ADPM functions.

All the functions and structures presented here are defined in files *AdpApi.h* and *AdpApiTypes.h*, available to the user.

Refer to these files for details of structure contents and types.

9.11.1 ADPM-DISCOVERY.request

The ADPM-DISCOVERY.request primitive allows the upper layer to request the ADPM to scan for networks operating in its POS.

```
void AdpDiscoveryRequest(uint8_t u8Duration);
```

Parameter:

u8Duration	The number of seconds the scan shall last.

9.11.2 ADPM-DISCOVERY.indication

The ADPM-Discovery.indication primitive is generated by the ADP layer upon Beacon reception. This primitive will be generated as many times as number of Beacons are received, thus, it is an application task to decide which PAN to join, depending on the info received on each of the indications.

```
typedef void (*AdpDiscoveryIndication)(struct TAdpPanDescriptor *pPanDescriptor);
```

The information is reported into the following structure:

```
struct TAdpPanDescriptor {
    uint16_t m_u16PanId;
    uint8_t m_u8LinkQuality;
    uint16_t m_u16LbaAddress;
    uint16_t m_u16RcCoord;
    uint8_t m_u8MediaType;
};
```

Fields of the structure:

u16PanId	PANId where Beacon comes from.
u8LinkQuality	Quality of Beacon reception.
m_ u16LbaAddress	Short address of device sending the Beacon.
m_u16RcCoord	Route Cost to Coordinator reported by Device sending the Beacon.
m_u8MediaType	MediaType through which the Beacon was received. Will be further used when calling 9.11.6. ADPM-NETWORK-JOIN.request primitive.

9.11.3 ADPM-DISCOVERY.confirm

The ADPM-DISCOVERY.confirm primitive is generated by the ADP layer upon completion of a previous ADPM-DISCOVERY.request.

```
typedef void (*AdpDiscoveryConfirm)(uint8_t m_u8Status);
```

Parameter:

m_u8Status	Result of the associated DISCOVERY.request.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.4 ADPM-NETWORK-START.request

The ADPM-NETWORK-START.request primitive allows the upper layer to request the starting of a new network. It shall only be invoked by a device designated as the PAN coordinator during the factory process.

```
void AdpNetworkStartRequest(uint16_t u16PanId);
```

Parameter:

u16PanId	The PANId of the network to create; determined at the application level [Valid range: 0x0000 - 0xFFFF. Note: value must be anded with 0xFCFF according to G3 spec].

9.11.5 ADPM-NETWORK-START.confirm

The ADPM-NETWORK-START.confirm primitive reports the status of an ADPM-NETWORK-START.request.

```
typedef void (*AdpNetworkStartConfirm)(struct TAdpNetworkStartConfirm *pNetworkStartConfirm);
```

Structure:

```
struct TAdpNetworkStartConfirm {

    uint8_t m_u8Status;

};
```

Parameter:

m_u8Status	The result of Network Start request.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.6 ADPM-NETWORK-JOIN.request

The ADPM-NETWORK-JOIN.request primitive allows the upper layer to join an existing network.

```
void AdpNetworkJoinRequest(uint16_t u16PanId, uint16_t u16LbaAddress, uint8_t u8MediaType);
```

Parameters:

u16PanId	The 16-bit PAN identifier of the network to join.
u16LbaAddress	The 16-bit short address of the device acting as a LoWPAN bootstrap agent as defined in Annex E of G3 Spec.
u8MediaType	The Media through which the Joining procedure will be triggered.

9.11.7 ADPM-NETWORK-JOIN.confirm

The ADPM-NETWORK-JOIN.confirm primitive is generated by the ADP layer to indicate the completion status of a previous ADPM-NETWORK-JOIN.request.

```
typedef void (*AdpNetworkJoinConfirm)(struct TAdpNetworkJoinConfirm *pNetworkJoinConfirm);
```

Structure:

```
struct TAdpNetworkJoinConfirm {
    uint8_t m_u8Status;
    uint16_t m_u16NetworkAddress;
    uint16_t m_u16PanId;
};
```

Fields:

m_u8Status	The status of the Join request.
m_u16NetworkAddress	The 16-bit network address that was allocated to the device. If the allocation fails, this address is equal to 0xFFFF.
m_u16PanId	The 16-bit address of the PAN of which the device is now a member.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.8 ADPM-NETWORK-JOIN.indication

Not implemented.

9.11.9 ADPM-NETWORK-LEAVE.request

The ADPM-NETWORK-LEAVE.request primitive allows a non-coordinator device to remove itself from the network.

```
void AdpNetworkLeaveRequest(void);
```

Important: Regardless of the result of this request, the G3 stack will reset internally before calling ADPM-NETWORK-LEAVE.confirm. Some IBs of the stack have to be preserved (read before reset, and restored after it), refer to the 3.4. Persistent Data Storage for details. This restore operation will be done automatically if 9.12.52. ADP_IB_MANUF_KEEP_PARAMS_AFTER_KICK_LEAVE = 0x080000CD is set to True, otherwise the user must take care of restoring them.

9.11.10 ADPM-NETWORK-LEAVE.indication

ADPM-NETWORK-LEAVE.indication primitive is generated by the ADP layer of a non-coordinator device to inform the upper layer that it has been unregistered from the network by the coordinator.

```
typedef void (*AdpNetworkLeaveIndication)(void);
```

9.11.11 ADPM-NETWORK-LEAVE.confirm

The ADPM-NETWORK-LEAVE.confirm primitive allows the upper layer to be informed of the status of its previous ADPM-NETWORK-LEAVE.request.

```
typedef void (*AdpNetworkLeaveConfirm)(struct TAdpNetworkLeaveConfirm *pLeaveConfirm);
```

Structure:

```
struct TAdpNetworkLeaveConfirm {
    uint8_t m_u8Status;
};
```

Field:

m_u8Status	The status of the request.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.12 ADPM-RESET.request

The ADPM-RESET.request primitive performs a reset of the adaptation sublayer and allows the resetting of the MIB attributes.

```
void AdpResetRequest(void);
```

9.11.13 ADPM-RESET.confirm

The ADPM-RESET.confirm primitive allows the upper layer to be notified of the completion of an ADPM-RESET.request primitive.

```
typedef void (*AdpResetConfirm)(struct TAdpResetConfirm *pResetConfirm);
```

Structure:

```
struct TAdpResetConfirm {
    uint8_t m_u8Status;
};
```

Field:

m_u8Status	The status of the request.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.14 ADPM-GET.request

The ADPM-GET.request primitive allows the upper layer to get the value of an attribute from the ADP information base.

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

Parameters:

u32AttributeId	The identifier of the ADP IB attribute to read.
u16AttributeIndex	The index within the table of the specified IB attribute to read. This parameter is valid only for IB attributes that are tables.

A synchronous implementation of this function is also available, thus no confirm is generated, instead a pointer to confirm the structure has to be provided.

```
void AdpGetRequestSync (uint32_t u32AttributeId, uint16_t u16AttributeIndex, struct
TAdpGetConfirm *pGetConfirm);
```

Parameters:

u32AttributeId	The identifier of the ADP IB attribute to read.
u16AttributeIndex	The index within the table of the specified IB attribute to read. This parameter is valid only for IB attributes that are tables.
pGetConfirm	Pointer to confirm structure to get the result. See structure definition in confirm primitive definition below.

9.11.15 ADPM-GET.confirm

The ADPM-GET.confirm primitive allows the upper layer to be informed of the status of a previously issued ADPM-GET.request primitive.

```
typedef void (*AdpGetConfirm)(struct TAdpGetConfirm *pGetConfirm);
```

Structure:

```
struct TAdpGetConfirm {

    uint8_t m_u8Status;

    uint32_t m_u32AttributeId;

    uint16_t m_u16AttributeIndex;

    uint8_t m_u8AttributeLength;

    uint8_t m_au8AttributeValue[64];

};
```

Fields:

m_u8Status	The status of the request.
m_u32AttributeId	The identifier of the IB attribute read.
m_u16AttributeIndex	The index within the table of the specified IB attribute to read. This parameter is valid only for IB attributes that are tables.
m_u8AttributeLength	The length of the value of the attribute read from the IB.
m_au8AttributeValue	The value of the attribute read from the IB.

Possible values of *m_u8Status* parameter are:

G3_SUCCESS	Value 0x00
G3_INVALID_PARAMETER	Value 0xA8
G3_INVALID_INDEX	Value 0xA7
G3_UNSUPPORTED_ATTRIBUTE	Value 0xB1

9.11.16 ADPM-MAC.GET.request

The ADPM-MAC.GET.request primitive allows the upper layer to get the value of an attribute from the MAC information base. The upper layer cannot directly access the MAC layer while ADP is running.

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

Parameters:

u32AttributeId	The identifier of the MAC IB attribute to read.
u16AttributeIndex	The index within the table of the specified IB attribute to read. This parameter is valid only for IB attributes that are tables.

A synchronous implementation of this function is also available, thus no confirm is generated; instead, a pointer to confirm structure has to be provided.

```
void AdpMacGetRequestSync (uint32_t u32AttributeId, uint16_t u16AttributeIndex, struct
TAdpMacGetConfirm *pGetConfirm);
```

Parameters:

u32AttributeId	The identifier of the ADP IB attribute to read.
u16AttributeIndex	The index within the table of the specified IB attribute to read. This parameter is valid only for IB attributes that are tables.
pGetConfirm	Pointer to confirm structure to get the result. See structure definition in confirm primitive definition below.

9.11.17 ADPM-MAC.GET.confirm

The ADPM-MAC.GET.confirm primitive allows the upper layer to be informed of the status of a previously issued ADPM-MAC.GET.request primitive.

```
typedef void (*AdpMacGetConfirm)(struct TAdpMacGetConfirm *pGetConfirm);
```

Structure:

```
struct TAdpMacGetConfirm {

    uint8_t m_u8Status;

    uint32_t m_u32AttributeId;

    uint16_t m_u16AttributeIndex;

    uint8_t m_u8AttributeLength;

    uint8_t m_au8AttributeValue[MAC_PIB_MAX_VALUE_LENGTH];

};
```

Fields:

m_u8Status	The status of the request.
m_u32AttributeId	The identifier of the IB attribute read.
m_u16AttributeIndex	The index within the table of the specified IB attribute read.
m_u8AttributeLength	The length of the value of the attribute read from the IB.
m_au8AttributeValue	The value of the attribute read from the IB.

Possible values of *m_u8Status* parameter are:

MAC_STATUS_SUCCESS	Value 0x00
MAC_STATUS_UNSUPPORTED_ATTRIBUTE	Value 0xF4
MAC_STATUS_INVALID_INDEX	Value 0xF9
MAC_STATUS_UNAVAILABLE_KEY	Value 0xF3
MAC_STATUS_DENIED	Value 0xE2
MAC_STATUS_INVALID_PARAMETER	Value 0xE8

9.11.18 ADPM-SET.request

The ADPM-SET.request primitive allows the upper layer to set the value of an attribute in the ADP information base.

```
void AdpSetRequest(uint32_t u32AttributeId, uint16_t u16AttributeIndex, uint8_t
u8AttributeLength, const uint8_t *pu8AttributeValue);
```

Parameters:

u32AttributeId	The identifier of the ADP IB attribute set.
u16AttributeIndex	The index within the table of the specified IB attribute.
u8AttributeLength	The length of the value of the attribute to set.
pu8AttributeValue	Pointer to the value of the attribute to set.

A synchronous implementation of this function is also available, thus no confirm is generated, instead a pointer to confirm structure has to be provided.

```
void AdpSetRequestSync (uint32_t u32AttributeId, uint16_t u16AttributeIndex, uint8_t
u8AttributeLength, const uint8_t *pu8AttributeValue, struct TAdpSetConfirm *pSetConfirm);
```

Parameters:

u32AttributeId	The identifier of the ADP IB attribute set.
u16AttributeIndex	The index within the table of the specified IB attribute.
u8AttributeLength	The length of the value of the attribute to set.
pu8AttributeValue	Pointer to the value of the attribute to set.
pSetConfirm	Pointer to confirm structure to get the result. See structure definition in confirm primitive definition below.

9.11.19 ADPM-SET.confirm

The ADPM-SET.confirm primitive allows the upper layer to be informed about a previous ADPM-SET.request primitive.

```
typedef void (*AdpSetConfirm)(struct TAdpSetConfirm *pSetConfirm);
```

Structure:

```
struct TAdpSetConfirm {

    uint8_t m_u8Status;

    uint32_t m_u32AttributeId;

    uint16_t m_u16AttributeIndex;

};
```

Fields:

m_u8Status	The status of the scan request.
m_u32AttributeId	The identifier of the IB attribute set.
m_u16AttributeIndex	The index within the table of the specified IB attribute.

Possible values of *m_u8Status* parameter are:

G3_SUCCESS	Value 0x00
G3_INVALID_PARAMETER	Value 0xA8
G3_INVALID_INDEX	Value 0xA7
G3_UNSUPPORTED_ATTRIBUTE	Value 0xB1
G3_READ_ONLY	Value 0xB0
G3_FAILED	Value 0xA2

9.11.20 ADPM-MAC.SET.request

The ADPM-MAC.SET.request primitive allows the upper layer to set the value of an attribute in the MAC information base. The upper layer cannot directly access the MAC layer while ADP is running.

```
void AdpMacSetRequest(uint32_t u32AttributeId, uint16_t u16AttributeIndex, uint8_t
u8AttributeLength, const uint8_t *pu8AttributeValue);
```

Parameters:

u32AttributeId	The identifier of the ADP IB attribute set.
u16AttributeIndex	The index within the table of the specified IB attribute.
u8AttributeLength	The length of the value of the attribute to set.
pu8AttributeValue	The value of the attribute to set.

A synchronous implementation of this function is also available, thus no confirm is generated, instead a pointer to confirm structure has to be provided.

```
void AdpMacSetRequestSync (uint32_t u32AttributeId, uint16_t u16AttributeIndex, uint8_t
u8AttributeLength, const uint8_t *pu8AttributeValue, struct TAdpMacSetConfirm*pSetConfirm);
```

Parameters:

u32AttributeId	The identifier of the ADP IB attribute set.
u16AttributeIndex	The index within the table of the specified IB attribute.
u8AttributeLength	The length of the value of the attribute to set.
pu8AttributeValue	Pointer to the value of the attribute to set.
pSetConfirm	Pointer to confirm structure to get the result. See structure definition in confirm primitive definition below.

9.11.21 ADPM-MAC.SET.confirm

The ADPM-MAC.SET.confirm primitive allows the upper layer to be informed about a previous ADPM-MAC.SET.request primitive.

```
typedef void (*AdpMacSetConfirm)(struct TAdpMacSetConfirm *pSetConfirm);
```

Structure:

```
struct TAdpMacSetConfirm {
    uint8_t m_u8Status;

    uint32_t m_u32AttributeId;

    uint16_t m_u16AttributeIndex;

};
```

Fields:

m_u8Status	The status of the request.
m_u32AttributeId	The identifier of the IB attribute set.
m_u16AttributeIndex	The index within the table of the specified IB attribute.

Possible values of *m_u8Status* parameter are:

MAC_STATUS_SUCCESS	Value 0x00
MAC_STATUS_UNSUPPORTED_ATTRIBUTE	Value 0xF4
MAC_STATUS_INVALID_INDEX	Value 0xF9
MAC_STATUS_READ_ONLY	Value 0xFB
MAC_STATUS_INVALID_PARAMETER	Value 0xE8

9.11.22 ADPM-NETWORK-STATUS.indication

The ADPM-NETWORK-STATUS.indication primitive allows the next higher layer of a PAN coordinator or a coordinator to be notified when a particular event occurs on the PAN.

```
typedef void (*AdpNetworkStatusIndication)(struct TAdpNetworkStatusIndication
*pNetworkStatusIndication);
```

Structure:

```
struct TAdpNetworkStatusIndication {

    uint16_t m_u16PanId;

    struct TAdpAddress m_SrcDeviceAddress;

    struct TAdpAddress m_DstDeviceAddress;

    uint8_t m_u8Status;

    uint8_t m_u8SecurityLevel;

    uint8_t m_u8KeyIndex;

    uint8_t m_u8MediaType;
};
```

9.11.23 ADPM-BUFFER.indication

The ADPM-BUFFER.indication primitive allows the upper layer to get info about the availability of buffers to hold a data transmission.

```
typedef void (*AdpBufferIndication)(struct TAdpBufferIndication *pBufferIndication);
```

Structure:

```
struct TAdpBufferIndication {
    bool m_bBufferReady;
};
```

The parameter m_bBufferReady indicates whether or not there are buffers available. This primitive is generated by the ADP layer when a data transmission takes the last available buffer, indicating that there are no more free buffers, and is also generated when, after no buffers are available, one of them becomes free for a new transmission.

9.11.24 ADPM-PREQ.indication

The AdpPREQIndication primitive allows the next higher layer to be notified when a PREQ frame is received in Unicast mode with an originator address equal to the coordinator address, and with a destination address equal to the device address.

```
typedef void (*AdpPREQIndication)(void);
```

This primitive is generated by ADP layer in order to indicate upper layer that the device has been addressed by means of a PREQ frame.

9.11.25 ADPM-UPD_NON_VOLATILE_DATA.indication

The AdpUpdNonVolatileDataIndication primitive allows the next higher layer to be notified when any of the parameters to be saved/restored before/after a reset change; this way, the upper layer can maintain a record of these parameters that is always updated.

```
typedef void (*AdpUpdNonVolatileDataIndication)(void);
```

It is up to the upper layer how to handle the change of these parameters, and their storage if necessary.

Refer to the 3.4. Persistent Data Storage for more details.

9.11.26 ADPM-ROUTE_NOT_FOUND.indication

The AdpRouteNotFoundIndication primitive is used to indicate the upper layer that a route is not available when a frame is to be sent. Absence of a route may be because there is no entry in the routing table when a data request comes from the upper layer, there is no entry in the routing table when a received frame has to be forwarded (in Unicast Addressing mode), or because a route has just broken due to no ACK reception for the frame being transmitted. This way, the upper layer knows that a route discover or repair process will be triggered to find a route to send such frame (always that IB configuration allows for triggering route discover/repair, and by default it does).

```
typedef void (*AdpRouteNotFoundIndication)(struct TAdpRouteNotFoundIndication
*pRouteNotFoundIndication);
```

Structure:

```
struct TAdpRouteNotFoundIndication{
    uint16_t m_u16SrcAddr;
    uint16_t m_u16DestAddr;
    uint16_t m_u16NextHopAddr;
    uint16_t m_u16PreviousHopAddr;
    uint16_t m_u16RouteCost;
    uint8_t m_u8HopCount;
    uint8_t m_u8WeakLinkCount;
    bool m_bRouteJustBroken;
    bool m_bCompressedHeader;
    uint16_t m_u16NsduLength;
    uint8_t *m_pNsdu;
};
```

Field:

m_u16SrcAddr	Originator address of the frame which found no route to be sent.
m_u16DestAddr	Final destination address of the frame which found no route to be sent.
m_u16NextHopAddr	If there was a route and it is just broken, next hop in broken route. In case there was no previous route, reported value is 0xFFFF.
m_u16PreviousHopAddr	Next hop in route to originator address. In case there is no route, reported value is 0xFFFF.

m_u16RouteCost	If there was a route and it is just broken, route cost of broken route. In case there was no previous route, reported value is 0xFFFF.
m_u8HopCount	If there was a route and it is just broken, hop count of broken route. In case there was no previous route, reported value is 0xFF.
m_u8WeakLinkCount	If there was a route and it is just broken, weak link count of broken route. In case there was no previous route, reported value is 0xFF.
m_bRouteJustBroken	Indicates whether there is no route because it is just broken (No ACK reception), or because there is no previous route.
m_bCompressedHeader	Indicates whether frame needing route, which is reported in this indication, has a compressed IPv6 header or a complete one.
m_u16NsduLength	Length of the frame needing route to be sent.
**m_pNsdu*	Pointer to frame needing route to be sent.

9.11.27 ADPM-ROUTE-DISCOVERY.request

The ADPM-ROUTE-DISCOVERY.request primitive allows the upper layer to initiate a route discovery.

```
void AdpRouteDiscoveryRequest(uint16_t u16DstAddr, uint8_t u8MaxHops);
```

Parameters:

u16DstAddr	The short unicast destination address of the route discovery.
u8MaxHops	This parameter indicates the maximum number of hops allowed for the route discovery [Valid range: 0x01 - 0x0E].

9.11.28 ADPM-ROUTE-DISCOVERY.confirm

The ADPM-ROUTE-DISCOVERY.confirm primitive allows the upper layer to be informed about a previous ADPM-ROUTE-DISCOVERY.request primitive.

```
typedef void (*AdpRouteDiscoveryConfirm)(struct TAdpRouteDiscoveryConfirm
*pRouteDiscoveryConfirm);
```

Structure:

```
struct TAdpRouteDiscoveryConfirm {
    uint8_t m_u8Status;
};
```

Field:

m_u8Status	The status of the route discovery.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.29 ADPM-PATH-DISCOVERY.request

The ADPM-PATH-DISCOVERY.request primitive allows the upper layer to initiate a path discovery.

```
void AdpPathDiscoveryRequest(uint16_t u16DstAddr, uint8_t u8MetricType);
```

Parameters:

u16DstAddr	The short unicast destination address of the path discovery.
u8MetricType	The metric type to be used for the path discovery. (Range: 0x00 - 0x0F).

9.11.30 ADPM-PATH-DISCOVERY.confirm

The ADPM-PATH-DISCOVERY.confirm primitive allows the upper layer to be informed of the completion of a path discovery (ADPM-PATH-DISCOVERY.request).

```
typedef void (*AdpPathDiscoveryConfirm)(struct TAdpPathDiscoveryConfirm
*pPathDiscoveryConfirm);
```

Structure:

```
struct TAdpPathDiscoveryConfirm {

    uint8_t m_u8Status;

    uint16_t m_u16DstAddr;

    uint16_t m_u16OrigAddr;

    uint8_t m_u8MetricType;

    uint8_t m_u8ForwardHopsCount;

    uint8_t m_u8ReverseHopsCount;

    struct THopDescriptor m_aForwardPath[16];

    struct THopDescriptor m_aReversePath[16];

};
```

Fields:

m_u8Status	The status of the path discovery (status can be INCOMPLETE and the other parameters contain the discovered path).
m_u16DstAddr	The short unicast destination address of the path discovery.
m_u16Originator	The originator of the path reply.
m_u8PathMetricType	Path metric type.
m_u8HopsCount	Number of path hops.
m_pau16HopAddressTable	Table with the address of each hop (according to m_u8HopsCount).
m_pau16HopAddressTable	Table with mns field of each hop (according to m_u8HopsCount).
m_pau8LinkCostTable	Table with cost information related to each hop (according to m_u8HopsCount).

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.31 ADPM-LBP.request

The ADPM-LBP.request primitive allows the upper layer of the client to send the LBP message to the server modem.

```
void AdpLbpRequest(const struct TAdpAddress *pDstAddr, uint16_t u16NsduLength, uint8_t
*pNsdu, uint8_t u8NsduHandle, uint8_t u8MaxHops, bool bDiscoveryRoute, uint8_t
u8QualityOfService, bool bSecurityEnable);
```

Parameters:

pDstAddr	16-bit address of LBA or LBD or 64-bit address (extended address of LBD).
u16NsduLength	The size of the NSDU, in bytes.
pNsdu	The NSDU to send.
u8NsduHandle	The handle of the NSDU to transmit. This parameter is used to identify in the AdpLbpConfirm primitive which request is concerned. It can be randomly chosen by the application layer.

u8MaxHops	The number of times the frame will be repeated by network routers.
bDiscoveryRoute	If TRUE, a route discovery procedure will be performed prior to sending the frame if a route to the destination is not available in the routing table. If FALSE, no route discovery is performed.
u8QualityOfService	The requested quality of service (QoS) of the frame to send. Allowed values are: • 0x00 = standard priority • 0x01 = high priority
bSecurityEnable	If TRUE, this parameter enables the MAC layer security for sending the frame.

9.11.32 ADPM-LBP.confirm

The ADPM-LBP.confirm primitive reports the result of a previous ADPM-LBP.request primitive.

```
typedef void (*AdpLbpConfirm)(struct TAdpLbpConfirm *pLbpConfirm);
```

Structure:

```
struct TAdpLbpConfirm {

    uint8_t m_u8Status;

    uint8_t m_u8NsduHandle;

};
```

Fields:

m_u8Status	The status code of a previous AdpLbpRequest identified by its NsduHandle.
m_u8NsduHandle	The handle of the NSDU confirmed by this primitive.

Status codes are detailed in 9.13. ADP Status Codes Description.

9.11.33 ADPM-LBP.indication

The ADPM-LBP.indication primitive is used to transfer a received LBP frame from the ADP layer to the upper layer.

```
typedef void (*AdpLbpIndication)(struct TAdpLbpIndication *pLbpIndication);
```

Structure:

```
struct TAdpLbpIndication {

    uint16_t m_u16SrcAddr;

    uint16_t m_u16NsduLength;

    uint8_t *m_pNsdu;

    uint8_t m_u8LinkQualityIndicator;

    bool m_bSecurityEnabled;

};
```

Fields:

m_SrcAddr	16 bits final destination address.
m_u16NsduLength	The size of the NSDU, in bytes; Range: 0 – 1280.
m_pNsdu	The NSDU received.
m_u8LinkQualityIndicator	The value of the link quality during reception of the frame.

m_bSecurityEnabled	TRUE if the frame was received with a security level greater or equal to adpSecurityLevel, FALSE otherwise.

9.12 ADPIB Objects Specification and Access

Microchip G3-PLC stacks supports all the information base (IB) attributes of the Adaptation Sublayer defined in the G3-PLC spec. The default endianness of all ADPIB is little endian; in other cases it will be explicitly specified.

In addition, there are some Microchip proprietary IB attributes of the Adaptation Sublayer with extra functionalities. All of them are described here.

9.12.1 ADP_IB_SECURITY_LEVEL = 0x00000000

This attribute indicates the minimum security level to be used for incoming and outgoing adaptation frames. Messages not respecting this level are dropped. The size of the attribute is 1 byte. The default value is 5 (ENC-MIC-32).

9.12.2 ADP_IB_PREFIX_TABLE = 0x00000001

This attribute contains the list of prefixes defined on this PAN. It is assumed that the link local IPv6 address exists independently and is not affected by the prefixes defined in the prefix table.

This attribute is a table of two elements.

Each element is encoded as following:

- Byte 0: indicates the prefix length in bits
- Byte 1: indicates the “on-link flag” (0 = not set, 1 = set)
- Byte 2: indicates the “autonomous address-configuration flag” (0 = not set, 1 = set)
- Bytes 3 - 6: indicate the valid lifetime in seconds
- Bytes 7 - 10: indicate the preferred lifetime in seconds
- Bytes 11 - 26 (variable length): indicate the prefix value

If the attribute length is set to 0, the prefix is deleted.

9.12.3 ADP_IB_BROADCAST_LOG_TABLE_ENTRY_TTL = 0x00000002

This attribute indicates the maximum time to live of an adpBroadcastLogTable entry (in minutes). The size of the attribute is 2 bytes. The value of the attribute is a 16 bits integer encoded little endian (LSB first). (e.g. 0x0200 = 2 minutes). Default value is 2 minutes.

9.12.4 ADP_IB_METRIC_TYPE = 0x00000003

This attribute defines the metric type to be used for routing purposes. The size of the attribute is 1 byte. The default value is 0x0F.

Possible values for this IB are:

- 0x00: TYPE_HOP_COUNT
- 0x0E: TYPE_METRIC_CAPACITY
- 0x0F: TYPE_METRIC_COMPOSITE

9.12.5 ADP_IB_LOW_LQI_VALUE = 0x00000004

The low LQI value defines the LQI value, used in metric computation, below which a link to a neighbour is considered as an unreliable link. The size of the attribute is 1 byte. The default value is 0x00.

9.12.6 ADP_IB_HIGH_LQI_VALUE = 0x00000005

The high LQI value defines the LQI value, used in metric computation, above which a link to a neighbour is considered as a reliable link. The size of the attribute is 1 byte. The default value is 0xFF.

9.12.7 ADP_IB_RREP_WAIT = 0x00000006

An RREP shall be generated after a delay of adpRREPWait seconds after either the arrival of the first RREQ or the transmission of the latest RREP. The size of the attribute is 1 byte. The default value is 4.

9.12.8 ADP_IB_CONTEXT_INFORMATION_TABLE = 0x00000007

This attribute contains the context information associated to each CID extension field. This attribute is a table of 16 elements. The CID field from the ICMPv6 / 6CO message indicates the index in the table.

Each element is encoded as following:

- Bytes 0 - 1: indicate the valid time in minutes. The value is a 16 bits integer encoded little endian (LSB first)
- Byte 2: indicates the validity of the CID for compression purposes (corresponds to C field from the ICMPv6 / 6CO message). A value of 0 means false, 1 means true
- Byte 3: indicates the context length in bits
- Bytes 4 - 19 (variable length): indicate the context value

If the attribute length is set to 0 the context is deleted.

This table is not ordered using any criteria and entries can be added at any index position, thus to get all the active entries, the whole table has to be read, or iterate until 9.12.71. ADP_IB_MANUF_CONTEXT_INFORMATION_TABLE_COUNT = 0x080000E1 entries have been found.

9.12.9 ADP_IB_COORD_SHORT_ADDRESS = 0x00000008

This attribute defines the short address of the coordinator. The size of the attribute is 2 bytes. The value is a 16 bits integer encoded little endian (LSB first). The default value is 0x0000.

9.12.10 ADP_IB_RLC_ENABLED = 0x00000009

This attribute controls the sending of RLCREQ frame by the device. The size of the attribute is 1. The value 0 indicates mechanism is disabled, value 1 mechanism enabled. The default value is 0.

9.12.11 ADP_IB_ADD_REV_LINK_COST = 0x0000000A

This attribute represents an additional cost to take into account a possible asymmetry in the link. The size of the attribute is 1 byte. The default value is 0.

9.12.12 ADP_IB_BROADCAST_LOG_TABLE = 0x0000000B

This attribute contains the broadcast log table. This attribute is read-only. The table contains 20 elements. Each element is 5 bytes in length:

- Bytes 0 - 1: indicate the source address of the broadcast packet (the address of the broadcast initiator). The address is encoded little endian (LSB first)
- Byte 2: indicates the sequence number
- Bytes 3 - 4: indicate the remaining time in minutes until this entry in the broadcast log table is considered valid (encoded little endian, LSB first)

This table is not ordered using any criteria, thus to get all the active entries the whole table has to be read, or iterate until 9.12.70. ADP_IB_MANUF_BROADCAST_LOG_TABLE_COUNT = 0x080000E0 entries have been found.

9.12.13 ADP_IB_ROUTING_TABLE = 0x0000000C

This attribute contains the routing table.

The routing table default sizes are 150 elements for Devices, 400 for Coordinator Lite and 2000 for Full Coordinator. Anyway the Table size is configurable for each project by means of the *conf_tables.h* file.

Each element has 9 bytes (all integers are encoded little endian, LSB first):

- Bytes 0 - 1: indicate the destination address
- Bytes 2 - 3: indicate the next hop address
- Bytes 4 - 5: indicate the route cost
- Byte 6, bits 0 - 3: indicate the weak links count
- Byte 6, bits 4 - 7: indicate the hop count

- Bytes 7 - 8: indicate the time this entry is valid in minutes

This attribute can also be written for testing purposes.

This table is ordered according to valid time. Newest entries are placed at the top of the table.

To get all the active entries, the first step is to get the number of such active entries by means of 9.12.47. ADP_IB_MANUF_ROUTING_TABLE_COUNT = 0x080000C8. Then, since the table is ordered, all entries can be obtained by iterating from 0 to the number of active entries.

9.12.14 ADP_IB_UNICAST_RREQ_GEN_ENABLE = 0x0000000D

This attribute controls the unicast RREQ generation. The length of the attribute is 1 byte. Value 0 indicates the mechanism is disabled, value 1 the mechanism is enabled. The default value is 1 (enabled).

9.12.15 ADP_IB_GROUP_TABLE = 0x0000000E

This attribute contains the group addresses to which the device belongs. The table size is 16. Note that the “All nodes address”, group 0x8001 is predefined and it doesn't need to be added to this attribute.

The length of each element is 2 bytes, defining the group address encoded little endian (lsb first). If the attribute element is set with the length 0, the group is deleted.

When reading the element, an extra byte is added indicating if the entry is valid or not: 0 means invalid, 1 means valid.

This table is not ordered using any criteria, thus to get all the active entries the whole table has to be read, or iterate until 9.12.72. ADP_IB_MANUF_GROUP_TABLE_COUNT = 0x080000E2 entries have been found.

9.12.16 ADP_IB_MAX_HOPS = 0x0000000F

This attribute defines the maximum number of hops to be used by the routing algorithm. The length of the attribute is 1 byte. The default value is 8.

9.12.17 ADP_IB_DEVICE_TYPE= 0x00000010

This attribute defines the type of device connected to the modem. The length of the attribute is 1 byte.

- 0: PAN device
- 1: PAN coordinator
- 2: Not defined (Default)

9.12.18 ADP_IB_NET_TRAVERSAL_TIME = 0x00000011

This attribute defines the maximum time in seconds that a packet is expected to take to reach any node from any node. The length of the attribute is 1 byte. The default value is 20.

9.12.19 ADP_IB_ROUTING_TABLE_ENTRY_TTL = 0x00000012

This attribute defines the maximum time-to-live of a routing table entry (in minutes). The length of the attribute is 2 bytes. The value is encoded little endian. Default value: 60 in Spec'15 / 360 in Spec'17.

9.12.20 ADP_IB_KR = 0x00000013

This attribute defines a weight factor for robust mode to calculate link cost. The length of the attribute is 1 byte. The default value is 0.

9.12.21 ADP_IB_KM = 0x00000014

This attribute defines a weight factor for modulation to calculate link cost. The length of the attribute is 1 byte. The default value is 0.

9.12.22 ADP_IB_KC = 0x00000015

This attribute defines a weight factor for number of active tones to calculate link cost. The length of the attribute is 1 byte. The default value is 0.

9.12.23 ADP_IB_KQ = 0x00000016

This attribute defines a weight factor for LQI to calculate route cost. The length of the attribute is 1 byte. Default value: 10 for CENELEC bandplans and 40 for FCC bandplan in Spec'15; in Spec'17 it is unified to 10 for all bandplans.

9.12.24 ADP_IB_KH = 0x00000017

This attribute defines a weight factor for hop to calculate link cost. The length of the attribute is 1 byte. Default value is 4 for CENELEC bandplans and 2 for FCC bandplan in Spec'15; in Spec'17 it is unified to 4 for all bandplans.

9.12.25 ADP_IB_RREQ_RETRIES = 0x00000018

This attribute defines the number of RREQ retransmissions in case of a RREP reception time out. The length of the attribute is 1 byte. The default value is 0.

9.12.26 ADP_IB_RREQ_RERR_WAIT / ADP_IB_RREQ_WAIT = 0x00000019

This attribute has different name depending on G3 Specification compliance. In G3 Specification April '17, it is stated that RERR frames are not delayed anymore, so its acronym has been removed from the attribute name. Both identifiers are valid (numeric ID does not change) and the reason to keep both is for backwards compatibility purposes.

This attribute defines the time in seconds to wait between two consecutive RREQ or RRER (only in Spec'15) generations. The length is 1 byte. The default value is 30.

9.12.27 ADP_IB_WEAK_LQI_VALUE = 0x0000001A

This attribute indicates the weak link value which defines the LQI value below which a link to a neighbour is considered as a weak link. The length is 1 byte. The default value is 52 (SNR of 3 dB).

9.12.28 ADP_IB_KRT = 0x0000001B

This attribute indicates a weight factor for the number of active routes in the routing table to calculate link cost. The length is 1 byte. The default value is 0.

9.12.29 ADP_IB_SOFT_VERSION = 0x0000001C

This attribute indicates the software version. This is a read-only attribute with a length of 6 bytes:

- Byte 0: major version number
- Byte 1: minor version number
- Byte 2: revision number
- Byte 3: version date: year (since 2000)
- Byte 4: version date: month
- Byte 5: version date: day

9.12.30 ADP_IB_SNIFFER_MODE = 0x0000001D

Note: This attribute is not supported in the current version of the stack.

Sniffer mode at ADP level activation/deactivation. The length of the attribute is 1 byte.

- 0: Disabled (Default)
- 1: Enabled

9.12.31 ADP_IB_BLACKLIST_TABLE = 0x0000001E

This attribute contains the list of the blacklisted neighbours.

The size of the table is 20 elements by default. This size can be modified if needed in the *RoutingWrapper.c* file.

Each element has 4 bytes:

- Bytes 0 - 1: the short address of the blacklisted neighbour
- Bytes 2 - 3: the remaining time in minutes while this entry is valid

If the attribute length is set to 0 the entry is deleted.

This table is not ordered using any criteria, thus to get all the active entries the whole table has to be read, or iterate until 9.12.69. ADP_IB_MANUF_BLACKLIST_TABLE_COUNT = 0x080000DF entries have been found.

9.12.32 ADP_IB_BLACKLIST_TABLE_ENTRY_TTL = 0x0000001F

This attribute indicates the maximum time-to-live of a blacklisted neighbour entry (in minutes). The length of the attribute is 2 bytes. The value is encoded little endian (lsb first). The default value is 10.

9.12.33 ADP_IB_MAX_JOIN_WAIT_TIME = 0x00000020

This attribute indicates the network join timeout in seconds for LBD. The size of the attribute is 2 bytes. The value is encoded little endian (lsb first). The default value is 20.

9.12.34 ADP_IB_PATH_DISCOVERY_TIME = 0x00000021

This attribute indicates the timeout for path discovery in seconds. The size of the attribute is 1 byte. The default value is 40.

9.12.35 ADP_IB_ACTIVE_KEY_INDEX = 0x00000022

This attribute indicates the index of the active GMK to be used for data transmission. The size of the attribute is 1 byte. The default value is 0.

9.12.36 ADP_IB_DESTINATION_ADDRESS_SET = 0x00000023

This attribute contains the Destination Address Set, which contains the list of the addresses of the device for which this LOADng router is providing connectivity. It is a new attribute from G3 Specification April '17 (please refer to that document for details about the attribute functionality), so it will only be available when running Spec'17 mode.

The size of the table is 1 element. Each element has 2 bytes:

- Bytes 0 - 1: the short address of the target destination

If the attribute length is set to 0 the entry is deleted.

9.12.37 ADP_IB_DEFAULT_COORD_ROUTE_ENABLED = 0x00000024

If TRUE, the adaptation layer adds a default route to the coordinator after successful completion of the bootstrapping procedure. If FALSE, no default route will be created. It is a new attribute from G3 Specification April '17 (please refer to that document for details about the attribute functionality), so it will only be available when running Spec'17 mode.

The size of the attribute is 1 byte. The default value is 0 (FALSE).

9.12.38 ADP_IB_DISABLE_DEFAULT_ROUTING = 0x000000F0

This attribute indicates if LOADng protocol should or should not be used. The size of the attribute is 1 byte. If 1 (true), the default routing (LOADng) is disabled. If 0 (false), the default routing (LOADng) is enabled. The default value is 0.

9.12.39 ADP_IB_MANUF_REASSEMBY_TIMER = 0x080000C0

This manufacturer attribute can be used to specify the reassembly timer (the maximum delay between the reception of two fragments from a message at ADP level). The size of the attribute is 2 bytes. The value is encoded little endian. Default value is 60 seconds.

9.12.40 ADP_IB_MANUF_IPV6_HEADER_COMPRESSION = 0x080000C1

This manufacturer attribute can be used to enable / disable the compression of the IPv6 headers. This applies only on the messages sent using the DataRequest service. The size of the attribute is 1 byte. The value 0 indicates the compression is disabled. Value 1 indicates the compression is enabled. The default value is 1.

9.12.41 ADP_IB_MANUF_EAP_PRESHARED_KEY = 0x080000C2

This manufacturer attribute specifies the network authentication key (PSK). For security reasons, this is a write-only attribute. The size of the attribute is 16 bytes.

9.12.42 ADP_IB_MANUF_EAP_NETWORK_ACCESS_IDENTIFIER = 0x080000C3

This manufacturer attribute specifies the network access identifier used during the EAP process. The size is a maximum of 36 bytes. The default value points to the extended address of the device.

9.12.43 ADP_IB_MANUF_BROADCAST_SEQUENCE_NUMBER = 0x080000C4

This manufacturer attribute can be used to get/set the broadcast sequence number. The size of the attribute is 1 byte.

9.12.44 ADP_IB_MANUF_REGISTER_DEVICE = 0x080000C5

This manufacturer attribute can be used to force the registration of a device into the network.

It is intended to be used for testing purposes only, because when used, all Encryption and Authentication process done during Bootstrap phase is bypassed.

This is a write-only attribute. The length of the attribute is 29 bytes:

- Bytes 0 - 7: indicate the extended address of the device (encoded big endian)
- Bytes 8 - 9: indicate the pan id (encoded little endian)
- Bytes 10 - 11: indicate the short address (encoded little endian)
- Byte 12: indicates the key index that will be used and where GMK will be stored
- Bytes 13 - 28: indicate the GMK key

9.12.45 ADP_IB_MANUF_DATAGRAM_TAG = 0x080000C6

This manufacturer attribute can be used to specify the datagram tag to be used for the fragmented transfers. The size of the attribute is 2 bytes. The value is encoded little endian. The default value is 0, which means the datagram will be randomly chosen.

9.12.46 ADP_IB_MANUF_RANDP = 0x080000C7

This manufacturer attribute can be used to specify the value of the RandP parameter used during the bootstrap. The size of the attribute is 16 bytes. The default value of all bytes is 0, which means the RandP will be chosen randomly.

9.12.47 ADP_IB_MANUF_ROUTING_TABLE_COUNT = 0x080000C8

This read-only manufacturer attribute can be used to get the number of active entries in the routing table. The size of the attribute is 4 bytes. The value represents a 32 bits integer encoded little endian (lsb first).

9.12.48 ADP_IB_MANUF_DISCOVER_SEQUENCE_NUMBER = 0x080000C9

This manufacturer attribute can be used to get/set the discover sequence number. The size of the attribute is 2 bytes. The value is encoded little endian.

9.12.49 ADP_IB_MANUF_FORCED_NO_ACK_REQUEST = 0x080000CA

This manufacturer attribute can be used to force all frames to be sent with No ACK request. The size of the attribute is 1 byte. The default value is 0, which means that the ACK request is decided by G3 mechanisms. Any other value forces all frames to be sent without ACK request.

9.12.50 ADP_IB_MANUF_LQI_TO_COORD = 0x080000CB

This manufacturer attribute retrieves LQI of the link which is the next hop in the route to coordinator. This link can be to coordinator itself if coordinator is a neighbour, or to another device if a route is needed to reach it.

The size of the attribute is 1 byte. The default value is 0, which means that coordinator is not a neighbour and there is no route to reach it (typically before starting to exchange any data with it).

9.12.51 ADP_IB_MANUF_BROADCAST_ROUTE_ALL = 0x080000CC

This manufacturer attribute can be used to activate specific mode where all the broadcast messages are propagated (in respect with broadcast sequence number and number of hops) even if the modem is not part of the destination group of the message.

The normal behaviour is to drop broadcast messages if device is not part of the destination group, thus default value is 0.

The size of the attribute is 1 byte, interpreted as Boolean.

9.12.52 ADP_IB_MANUF_KEEP_PARAMS_AFTER_KICK_LEAVE = 0x080000CD

This manufacturer attribute enables automatic preservation of some parameters after a reset caused by a Kick/Leave process. Default value for this parameter is false.

Parameters that are currently saved and restored when this IB is set to true are:

- ADP_IB_MANUF_BROADCAST_SEQUENCE_NUMBER
- ADP_IB_MANUF_DISCOVER_SEQUENCE_NUMBER
- MAC_PIB_FRAME_COUNTER

The size of the attribute is 1 byte. Default value: 0 for Spec'15, 1 for Spec'17.

9.12.53 ADP_IB_MANUF_ADP_INTERNAL_VERSION = 0x080000CE

This attribute indicates the internal ADP version. This is a read-only attribute and the length is 6 bytes:

- Byte 0: major version number
- Byte 1: minor version number
- Byte 2: revision number
- Byte 3: version date: year (since 2000)
- Byte 4: version date: month
- Byte 5: version date: day

9.12.54 ADP_IB_MANUF_CIRCULAR_ROUTES_DETECTED = 0x080000CF

This manufacturer attribute can be used to get the number of circular routes (routing loops) detected and corrected by device. The size of the attribute is 2 bytes.

9.12.55 ADP_IB_MANUF_LAST_CIRCULAR_ROUTE_ADDRESS = 0x080000D0

This manufacturer attribute can be used to get the next hop address involved in the last routing loop detected and corrected, it is provided for statistical purposes. The size of the attribute is 2 bytes.

9.12.56 ADP_IB_MANUF_IPV6_ULA_DEST_SHORT_ADDRESS = 0x080000D1

This manufacturer attribute is intended to be used in a G3 Gateway implementation. It allows upper layers to set the destination short address of the next data frame to be transmitted, which will be derived from an ULA address. This allows the ULA / short address translation to be managed by the gateway application, out of G3 stack scope. The size of the attribute is 2 bytes.

9.12.57 ADP_IB_MANUF_MAX_REPAIR_RESEND_ATTEMPTS = 0x080000D2

This manufacturer attribute sets the number of attempts of transmission for a frame when repair has been performed. The concept “attempt” refers to the whole cycle of transmission fail and successful repair. The aim of this attribute is to avoid an infinite loop of transmission fail + repair successful, limiting this cycle to a number of iterations. Default value is 5 and attribute size is 1 byte.

9.12.58 ADP_IB_MANUF_DISABLE_AUTO_RREQ = 0x080000D3

This manufacturer attribute allows controlling the stack behaviour when a route is not available for sending a frame. This situation happens if an existing route is broken due to No ACK reception after sending a frame, or if previous route does not exist or is no longer valid. The default behaviour is to automatically trigger a route discover or repair mechanism (thus default IB value is false), but the user can change this to discard the frame sending if there is no route available. When this IB is set, only route discovery processes requested by upper layer by means of AdpRouteDiscoveryRequest primitive will be executed. Please note that this behaviour is out of G3 spec, and may lead to interoperability issues. Attribute size is 1 byte. The default value is 0.

9.12.59 ADP_IB_MANUF_ALL_NEIGHBORS_BLACKLISTED_COUNT = 0x080000D5

As a protection, when a device detects that all its neighbours are blacklisted, the blacklist table is cleared so there is a chance to find routes again through any of them.

This manufacturer attribute counts how many times this protection has been triggered. Attribute size is 2 bytes. The default value is 0.

9.12.60 ADP_IB_MANUF_QUEUED_ENTRIES_REMOVED_TIMEOUT_COUNT = 0x080000D6

The ADP transmission queue implements a timeout so frames not released for transmission for a given time are discarded. Time-out is set to 2*ADP_IB_NET_TRAVERSAL_TIME.

This manufacturer attribute counts how many frames have been discarded due to time-out. Attribute size is 2 bytes. The default value is 0.

9.12.61 ADP_IB_MANUF_QUEUED_ENTRIES_REMOVED_ROUTE_ERROR_COUNT = 0x080000D7

When a route request/repair fails, any element in ADP transmission queue with the same destination as the one for which the route process has failed is purged. This prevents immediately starting a new route discovery process to the same destination that just failed, and thus avoids channel congestion due to the flooding of RREQs in the network.

This manufacturer attribute counts how many frames have been discarded due to the above explained reason. Attribute size is 2 bytes. The default value is 0.

9.12.62 ADP_IB_MANUF_PENDING_DATA_IND_SHORT_ADDRESS = 0x080000D8

This manufacturer attribute is used for inter-communication between ADP and Routing libraries. It is not intended to be used by the final user application.

Attribute size is 2 bytes. The default value is 0.

9.12.63 ADP_IB_MANUF_GET_BAND_CONTEXT_TONES = 0x080000D9

This manufacturer attribute is used for inter-communication between ADP and Routing libraries. It is not intended to be used by the final user application.

Attribute size is 1 byte. The default value is 0.

9.12.64 ADP_IB_MANUF_UPDATE_NON_VOLATILE_DATA = 0x080000DA

This manufacturer attribute is used for inter-communication between ADP and Routing libraries. It is not intended to be used by the final user application.

Attribute size is 1 byte. The default value is 0.

9.12.65 ADP_IB_MANUF_DISCOVER_ROUTE_GLOBAL_SEQ_NUM = 0x080000DB

This manufacturer attribute is used for inter-communication between ADP and Routing libraries. It is not intended to be used by the final user application.

Attribute size is 2 bytes. The default value is 0.

9.12.66 ADP_IB_MANUF_FRAGMENT_DELAY = 0x080000DC

This manufacturer attribute is used to set a delay between 6LowPAN fragments at originator. The intention of such delay is to avoid collisions when fragments are retransmitted through subsequent hops.

The effect of this attribute depends on other related ones as follows:

When 9.12.67. ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_ENABLED = 0x080000DD is false, it is the delay between fragments.

When 9.12.67. ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_ENABLED = 0x080000DD is set, this IB sets a ceiling value to apply to the formula used by 9.12.68. ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_FACTOR = 0x080000DE.

If this IB is set to 0, no delay is applied in any case.

Attribute size is 2 bytes. The default value is 0.

9.12.67 ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_ENABLED = 0x080000DD

This manufacturer attribute is used to enable dynamic 6LowPAN fragments delay. When true, dynamic delay between fragments is used, otherwise static delay is applied.

Attribute size is 1 byte. The default value is 0 (false).

9.12.68 ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_FACTOR = 0x080000DE

This manufacturer attribute is used to set a factor in order to calculate the dynamic 6LowPAN fragments delay.

When 9.12.67. ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_ENABLED = 0x080000DD is set, delay between fragments is calculated depending on number of hops to destination:

*Delay = (ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_FACTOR * num_hops_to_destination)*, when *num_hops_to_destination > 2*. Otherwise *Delay = 0*.

This IB has no effect when 9.12.67. ADP_IB_MANUF_DYNAMIC_FRAGMENT_DELAY_ENABLED = 0x080000DD is false.

Attribute size is 2 bytes. The default value is 800.

9.12.69 ADP_IB_MANUF_BLACKLIST_TABLE_COUNT = 0x080000DF

This read-only manufacturer attribute can be used to get the number of active entries in the Blacklist table 9.12.31. ADP_IB_BLACKLIST_TABLE = 0x0000001E. The size of the attribute is 2 bytes. The value represents a 16 bits integer encoded little endian (LSB first).

9.12.70 ADP_IB_MANUF_BROADCAST_LOG_TABLE_COUNT = 0x080000E0

This read-only manufacturer attribute can be used to get the number of active entries in the Broadcast Log table 9.12.12. ADP_IB_BROADCAST_LOG_TABLE = 0x0000000B. The size of the attribute is 2 bytes. The value represents a 16 bits integer encoded little endian (LSB first).

9.12.71 ADP_IB_MANUF_CONTEXT_INFORMATION_TABLE_COUNT = 0x080000E1

This read-only manufacturer attribute can be used to get the number of active entries in the Context Information table 9.12.8. ADP_IB_CONTEXT_INFORMATION_TABLE = 0x00000007. The size of the attribute is 2 bytes. The value represents a 16 bits integer encoded little endian (LSB first).

9.12.72 ADP_IB_MANUF_GROUP_TABLE_COUNT = 0x080000E2

This read-only manufacturer attribute can be used to get the number of active entries in the Group table 9.12.15. ADP_IB_GROUP_TABLE = 0x0000000E. The size of the attribute is 2 bytes. The value represents a 16 bits integer encoded little endian (LSB first).

9.12.73 ADP_IB_MANUF_ROUTING_TABLE_ELEMENT = 0x080000E3

This object has the same content as the Routing table, but it is retrieved using the short address of the node, instead of the index inside the table. To do so, set the short address in the index field in the Get IB function.

This attribute is Read-only.

9.12.74 ADP_IB_MANUF_SET_PHASEDIFF_PREQ_PREP = 0x080000E4

This manufacturer attribute is used to define whether Phase Differential information is added to Hop Descriptor information on Path Request and Path Reply frames.

Attribute size is 1 byte. The default value is 0 (false) on PLC-only projects, and 1 (true) on projects supporting the Hybrid Profile.

9.12.75 ADP_IB_MANUF_HYBRID_PROFILE = 0x080000E5

This read-only manufacturer attribute is used to read from upper layers whether the underlying G3 stack implements the Hybrid Profile or not.

The attribute size is 1 byte. The Value is set to '`0`' (false) on PLC-only projects, and to '`1`' (true) on projects supporting the Hybrid Profile.

9.12.76 ADP_IB_LOW_LQI_VALUE_RF = 0x000000D0

The low LQI value defines the LQI value, used in metric computation, below which a link to a neighbour is considered an unreliable RF link. This value will be lower than adpHighLQIValueRF. The size of the attribute is 1 byte. The default value is 0x00.

9.12.77 ADP_IB_HIGH_LQI_VALUE_RF = 0x000000D1

The low LQI value defines the LQI value, used in metric computation, below which a link to a neighbour is considered a reliable RF link. This value will be higher than adpLowLQIValueRF and lower than 255. The size of the attribute is 1 byte. The default value is 254.

9.12.78 ADP_IB_KQ_RF = 0x000000D2

This attribute defines a weight factor for LQI to calculate link cost. The length of the attribute is 1 byte. Default value: 10.

9.12.79 ADP_IB_KH_RF = 0x000000D3

This attribute defines a weight factor for Hop Count to calculate link cost. The length of the attribute is 1 byte. Default value: 4.

9.12.80 ADP_IB_KRT_RF = 0x000000D4

This attribute defines a weight factor for Number of Active Routes to calculate link cost. The length of the attribute is 1 byte. Default value: 0.

9.12.81 ADP_IB_KDC_RF = 0x000000D5

This attribute defines a weight factor for Duty Cycle to calculate link cost. The length of the attribute is 1 byte. Default value: 10.

9.12.82 ADP_IB_USE_BACKUP_MEDIA = 0x000000D6

This attribute controls if retransmission can use the backup medium in case of transmission failure. The length of the attribute is 1 byte (boolean). Default value: True.

9.13 ADP Status Codes Description

Following is the list of status codes and a brief description of each one, as reported by ADP:

9.13.1 G3_SUCCESS (0x00)

Returned when the last requested operation succeeded, may it be a data request, a get or set request, etc.

9.13.2 G3_INVALID_REQUEST (0xA1)

This value is reported in the following situations:

- Data request is called before device has joined a network. Suggested action: Wait for Bootstrap process to complete before calling data request
- AdpNetworkJoinRequest is called when the device is already part of a network. Suggested action: Call AdpNetworkLeaveRequest and wait for the end of this process before trying to join a network again
- AdpNetworkLeaveRequest is called when the device is not part of a network. Suggested action: Assume device is not part of a network, even before leave request was called

9.13.3 G3_FAILED (0xA2)

This value is reported in the following situations:

- When setting ADP_IB_MANUF_REGISTER_DEVICE and the information is correctly set at the ADP layer, but not at the MAC level. Suggested action: Reset G3 stack and configure the MAC and ADP layers again
- Data request is correctly handled by ADP layer, but MAC layer has no resources in such moment to handle it. Suggested action: wait a random time in application level to let the MAC layer free some resources and try sending again
- On PAN coordinator, AdpNetworkStartConfirm returns this value when AdpNetworkStartRequest is not able to request information from the MAC layer (short and extended addresses), or when the short address at MAC layer does not match the value in ADP_IB_COORD_SHORT_ADDRESS. Suggested action: Reset G3 stack and configure MAC and ADP layers again

- On the device, AdpNetworkJoinConfirm returns this value when AdpNetworkJoinRequest is not able to request information from the MAC layer (extended address and PAN ID). Suggested action: Reset G3 stack and configure MAC and ADP layers again
- On the device, AdpNetworkJoinConfirm returns this value when setting MAC parameters during the joining process fails in some intermediate state while exchanging data with coordinator. Suggested action: Reset G3 stack and configure the MAC and ADP layers again. Then start the network join process again
- On the device, AdpDiscoveryConfirm returns this value when a scan confirm is received from the MAC layer, but the ADP layer was not expecting it. Suggested action: Ignore confirm and start the discovery process again if necessary

9.13.4 G3_INVALID_IPV6_FRAME (0xA3)

This value is returned when the buffer passed to AdpDataRequest for sending does not contain a valid IPv6 frame. Suggested action: Check the correctness of headers at the IP level and try sending again.

9.13.5 G3_NOT_PERMITED (0xA4)

This value is returned when a device is not accepted in the network it is trying to join. It may be due to a DECLINE frame reception or to a lack or wrong reception of the ACCEPTED frame. Suggested action: Check correctness of values used in network join, or try to join another network in case the device is banned in the current one.

9.13.6 G3_ROUTE_ERROR (0xA5)

This value is returned in the following situations:

- When AdpPathDiscoveryRequest primitive is called, and there is no route for the intended destination. Suggested action: Call AdpRouteDiscoveryRequest first to create the route, and then call AdpPathDiscoveryRequest to get the path
- When AdpDataRequest is called for a destination to which there is no route, and one of the following conditions is met:
  – Route discovery is not allowed due to request parameters or IB configuration. Suggested action: Allow route discovery, or perform AdpRouteDiscoveryRequest from upper layer before calling AdpDataRequest
  – Route discovery is allowed, and such discovery fails to find a route. Suggested action: Wait for a route to be available (or explicitly look for it using AdpRouteDiscoveryRequest primitive), and then retry sending data

9.13.7 G3_TIMEOUT (0xA6)

This value is returned in the following situations:

- When a join process is not completed after the time-out specified in ADP_IB_MAX_JOIN_WAIT_TIME parameter. Suggested action: Increase time-out
- When a path discover process is not completed after the time-out specified in ADP_IB_PATH_DISCOVERY_TIME parameter. Suggested action: Increase time-out

9.13.8 G3_INVALID_INDEX (0xA7)

This value is returned when trying to set/get an ADP IB object with an index out of range of the requested object. Suggested action: Check object range and verify the requested index.

9.13.9 G3_INVALID_PARAMETER (0xA8)

This value is returned when trying to set an ADP IB object with a different length than expected. Suggested action: Check that IB length is according to set request.

Note: A particular case is ADP_IB_SNIFFER_MODE, which is not implemented and this value is returned when trying to get/set it.

9.13.10 G3_NO_BEACON (0xA9)

Not used in the current stack.

9.13.11 G3_READ_ONLY (0xB0)

This value is returned when trying to set an ADP IB object which cannot be written. Suggested action: Check IB access rights.

9.13.12 G3_UNSUPPORTED_ATTRIBUTE (0xB1)

This value is returned when trying to set/get an ADP IB object with an ID which is not part of the available set. Suggested action: Check available IDs and verify the requested one.

9.13.13 G3_INCOMPLETE_PATH (0xB2)

This value is returned when the path returned after an AdpPathDiscoveryRequest has not the complete information, because a problem was encountered at some point of the path. Suggested action: Call AdpRouteDiscoveryRequest first to create the route, and then call AdpPathDiscoveryRequest to get the path.

9.13.14 G3_BUSY (0xB3)

This value is returned in the following situations:

When the AdpDiscoveryRequest primitive is called when a previous discovery process is already in progress. Suggested action: Wait for previous discovery process to finish (reception of corresponding AdpDiscoveryConfirm) to request a new one.

9.13.15 G3_NO_BUFFERS (0xB4)

This value is returned when the AdpDataRequest primitive is called, but the stack has no free resources to handle it. It can be due to the fact that all transmission buffers are being used, or because a route discovery is needed before data sending and there are no resources to perform the discovery (other discovery processes running are using the resources). Suggested action: wait a random time in application level to let the stack free some resources and try sending again.

9.13.16 G3_ERROR_INTERNAL (0xFF)

This value is returned in the following situations:

- When trying to set a MAC IB using a length greater than the attribute length. Suggested action: Check that attribute length is according to set length
- When trying to get a MAC IB and its length is greater than the container to return it. Suggested action: Report this issue, as it is a stack internal error that has to be fixed in the library

9.13.17 MAC_WRP_STATUS_ALTERNATE_PANID_DETECTION (0x80)

reported by 9.11.22. ADPM-NETWORK-STATUS.indication when a frame is received from a PAN Id different than the network which the node belongs to. Actions to be taken when an alternate PAN Id is detected are left to the application.

9.13.18 MAC_WRP_STATUS_COUNTER_ERROR (0xDB)

This value is reported when Frame Counter reaches 0xFFFFFFFF. Suggested action: Reset Device and Reset Frame Counter, as security has to be reinitialized.

9.13.19 MAC_WRP_STATUS_UNSUPPORTED_SECURITY (0xDF)

This value is reported when there is an Encryption Error. Suggested action: Check that ADP_IB_SECURITY_LEVEL has a correct value, otherwise reset Device, as security has to be reinitialized.

9.13.20 MAC_WRP_STATUS_CHANNEL_ACCESS_FAILURE (0xE1)

This value is reported when CSMA retires are exhausted. Suggested action: Retry transmission after a random time.

9.13.21 MAC_WRP_STATUS_DENIED (0xE2)

This value is reported when Tx is aborted by MAC layer. Suggested action: Retry transmission after a random time.

9.13.22 MAC_WRP_STATUS_SECURITY_ERROR (0xE4)

This value is reported when there is an Encryption Error. Suggested action: Reset Device, as security has to be reinitialized.

9.13.23 MAC_WRP_STATUS_FRAME_TOO_LONG (0xE5)

This value is reported when MAC frame is too long (should not happen as ADP layer will split frames correctly). Suggested action: Report error to Microchip providing App layer frame transmitted, as ADP layer has to be debugged.

9.13.24 MAC_WRP_STATUS_NO_ACK (0xE9)

This value is reported in case ACK is not received and route discovery is not allowed when calling AdpDataRequest. If discovery is allowed, a route will be created and data sent, or in case route is not found, a G3_ROUTE_ERROR will be reported. Suggested action: Trigger a Route Discovery process from application, as it has not been automatically triggered by the stack.

9.13.25 MAC_WRP_STATUS_TRANSACTION_OVERFLOW (0xF1)

This value is reported when Device is already transmitting. Suggested action: Retry transmission after a random time.

9.13.26 MAC_WRP_STATUS_UNAVAILABLE_KEY (0xF3)

This value is reported when there is an Encryption Error. Suggested action: Check that ADP_IB_ACTIVE_KEY_INDEX has a correct value, otherwise reset Device, as security has to be reinitialized.

9.13.27 MAC_WRP_STATUS_LIMIT_REACHED (0xFA)

This value is reported when MAC layer ran out of transmitting resources. Suggested action: Retry transmission after a random time.

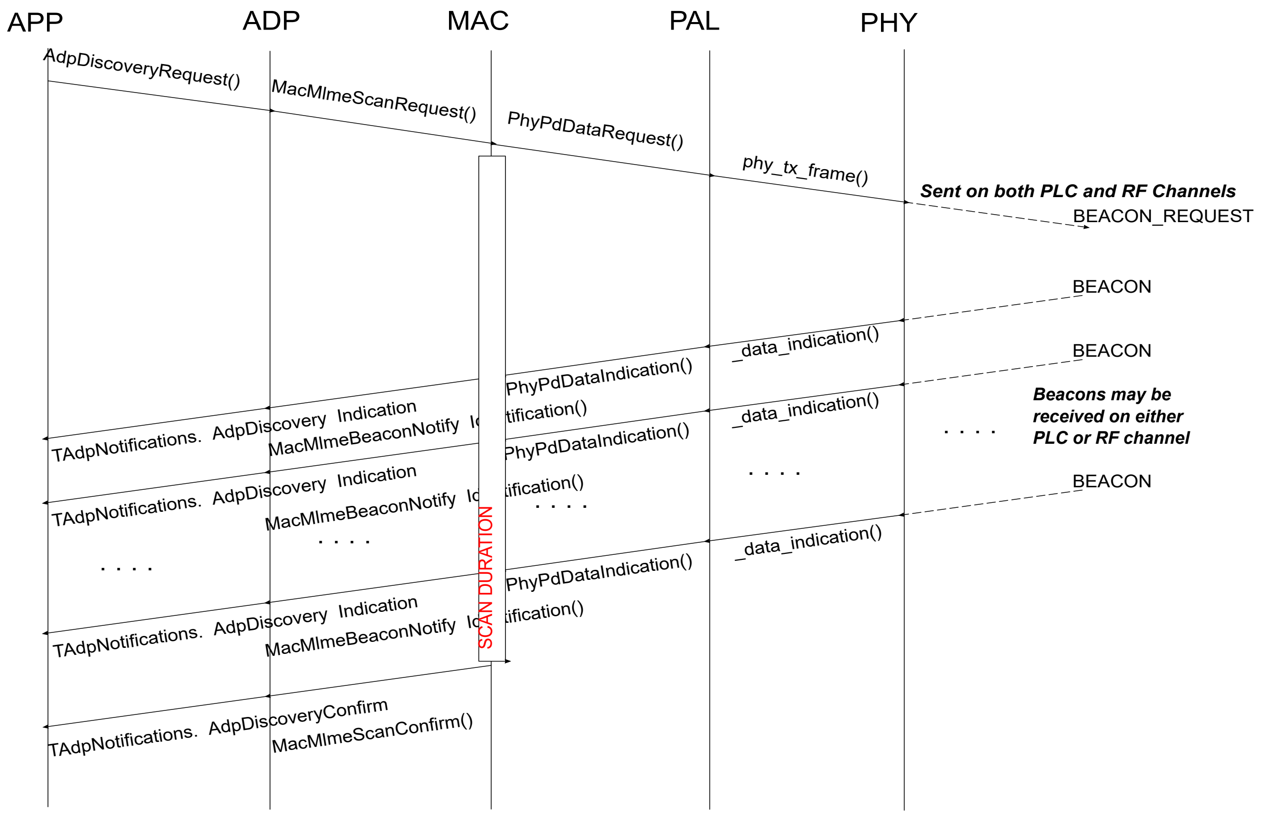
10. Data Link Processes

This chapter shows flow diagrams for different Data Link layer processes to help the user understand the flow of messages through the stack and the behavior of a G3-PLC or Hybrid network.

10.1 Network Discovery Request

The ADPM-DISCOVERY.request primitive allows the upper layer to request the ADPM to scan for networks operating in its POS. The ADPM-DISCOVERY.confirm primitive is generated by the ADP layer upon completion of a previous ADPM-DISCOVERY.request.

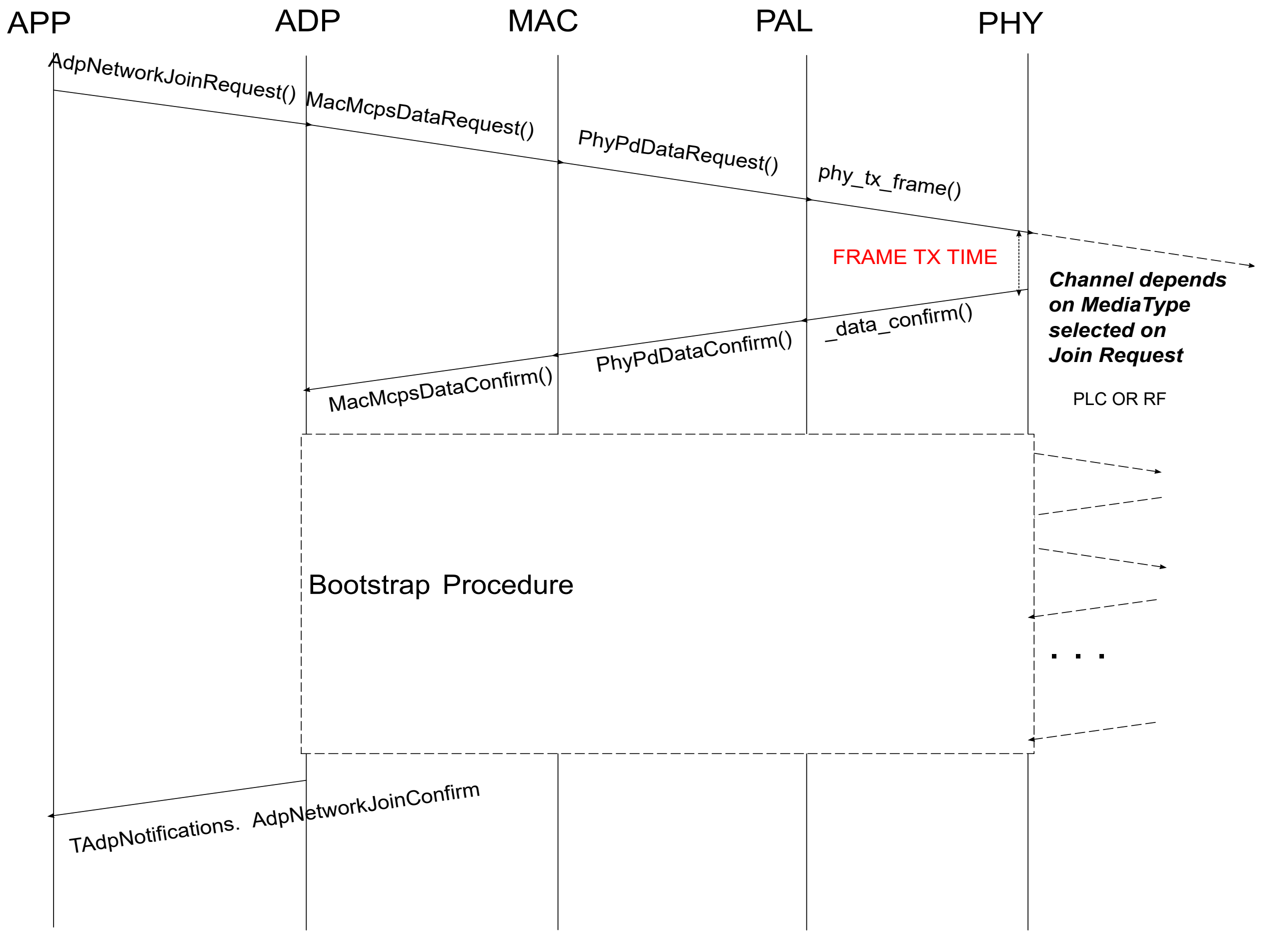
Figure 10-1. Network Discovery Request Flow Chart


APP
ADP
MAC
PAL
PHY
AdpDiscoveryRequest()
MacMlmeScanRequest()
PhyPdDataRequest()
phy_tx_frame()
Sent on both PLC and RF Channels
BEACON_REQUEST
BEACON
_data_indication()
PhyPdDataIndication()
BEACON
TAdpNotifications. AdpDiscovery Indication
MacMlmeBeaconNotify Indication()
Beacons may be received on either PLC or RF channel
SCAN DURATION
TAdpNotifications. AdpDiscoveryConfirm
MacMlmeScanConfirm()
. . . .

10.2 Network Join Request

The ADPM-NETWORK-JOIN.request primitive allows the upper layer to join an existing network. The ADPM-NETWORK-JOIN.confirm primitive is generated by the ADP layer to indicate the completion status of a previous ADPM-NETWORK-JOIN.request.

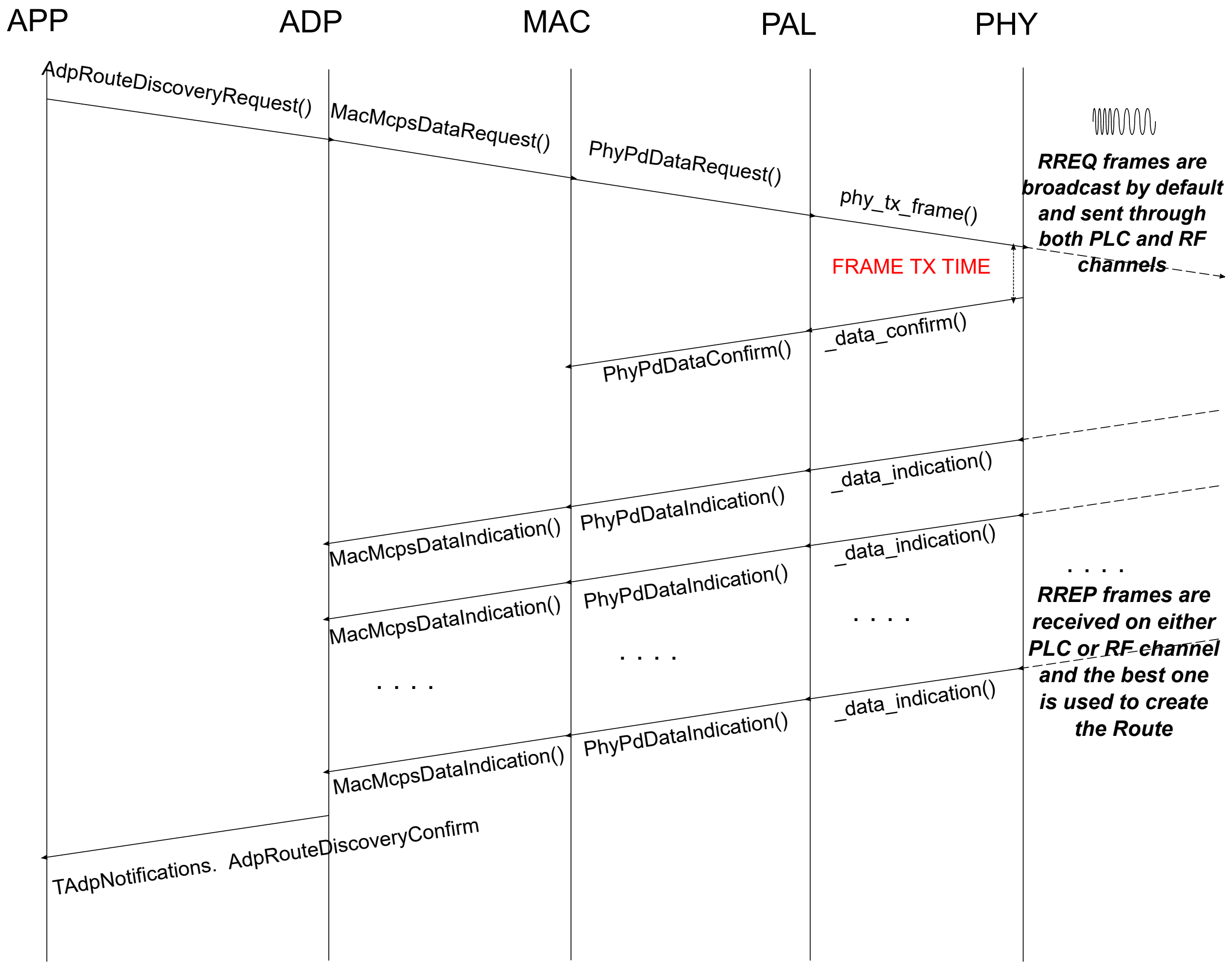
Figure 10-2. Network Join Request Flow Chart



10.3 Route/Path Discovery Request

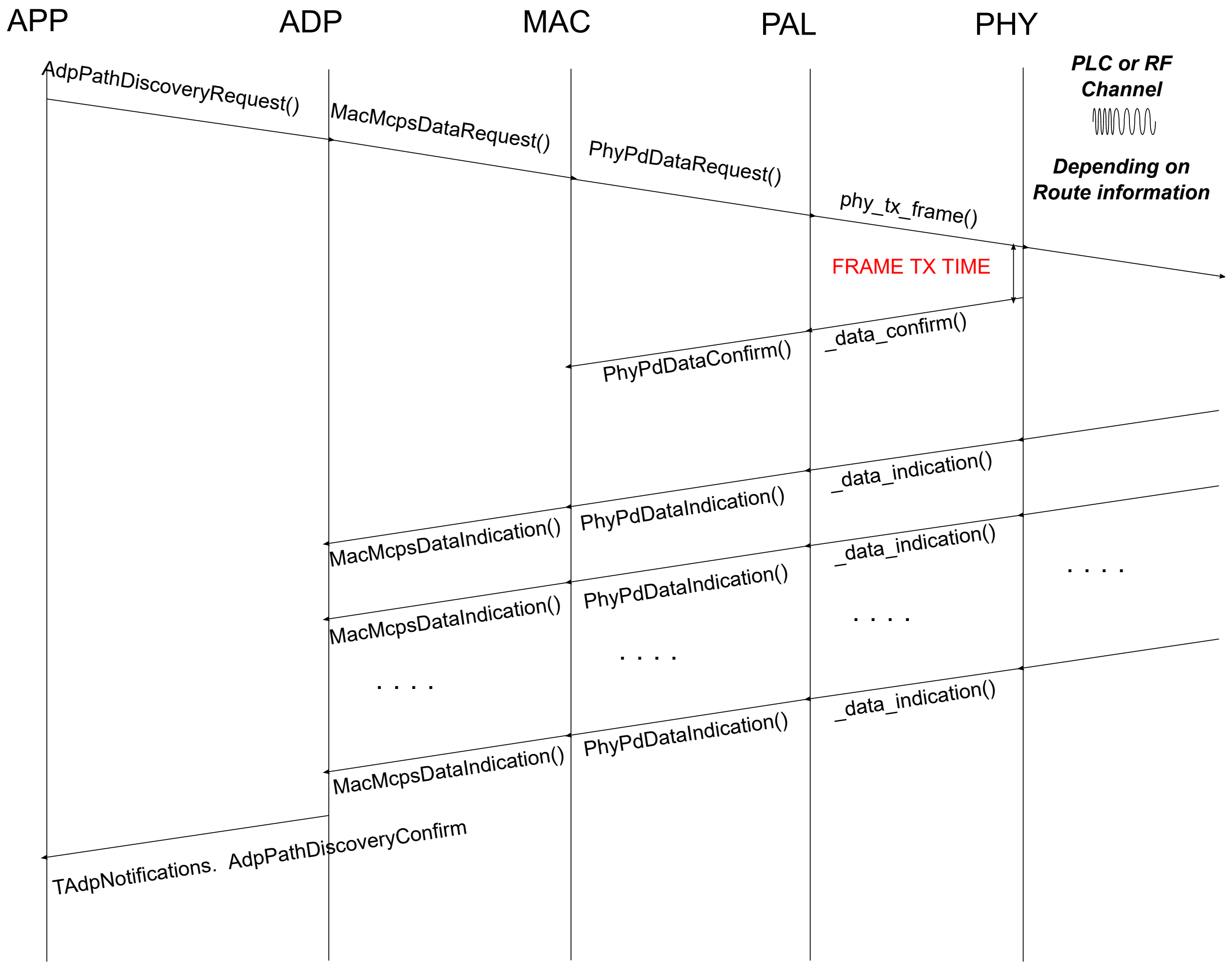
The ADPM-ROUTE-DISCOVERY.request primitive allows the upper layer to initiate a route discovery. The ADPM-ROUTE-DISCOVERY.confirm primitive allows the upper layer to be informed about a previous ADPM-ROUTE-DISCOVERY.request primitive.

Figure 10-3. Route Discovery Request Flow Chart



The ADPM-PATH-DISCOVERY.request primitive allows the upper layer to initiate a path discovery. The ADPM-PATH-DISCOVERY.confirm primitive allows the upper layer to be informed of the completion of a path discovery (ADPM-PATH-DISCOVERY.request).

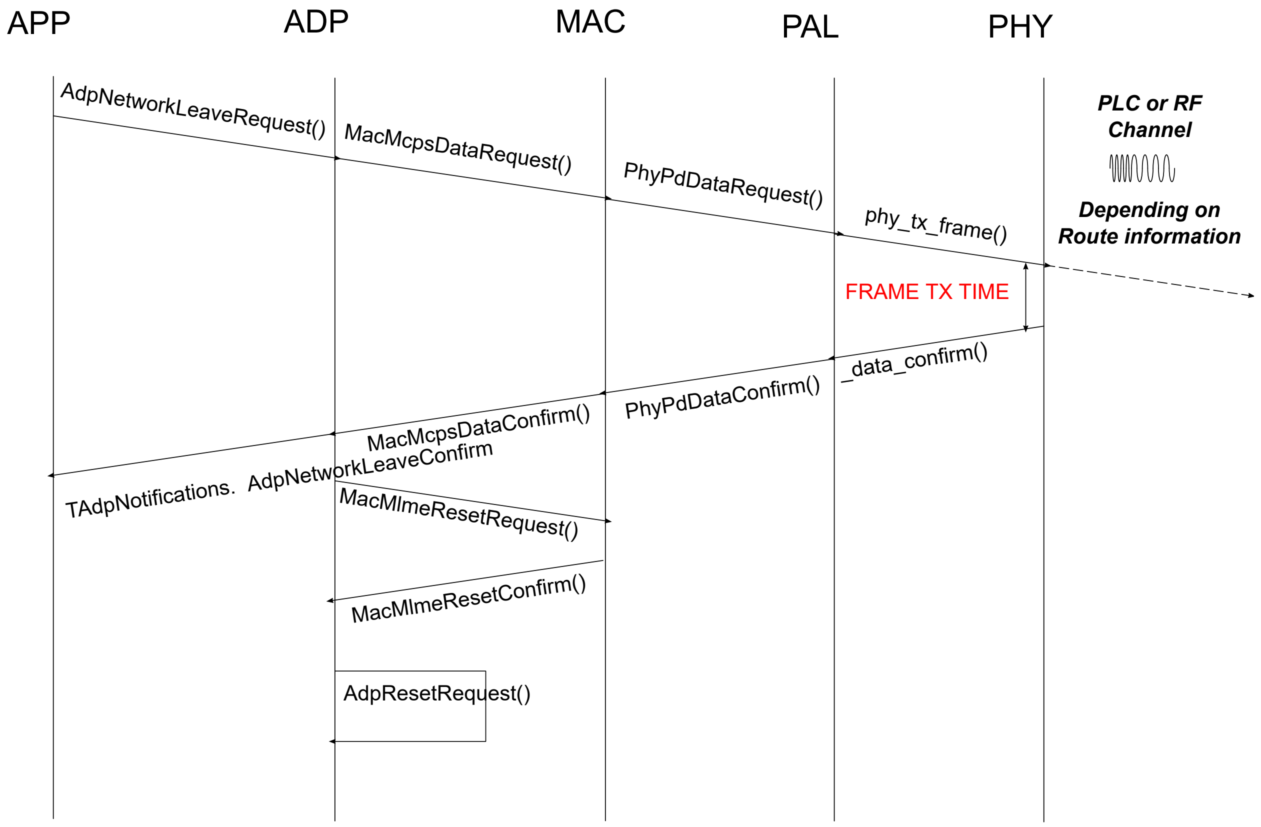
Figure 10-4. Path Discovery Request Flow Chart



10.4 Network Leave

The ADPM-NETWORK-LEAVE.request primitive allows a non-coordinator device to remove itself from the network. The ADPM-NETWORK-LEAVE.confirm primitive allows the upper layer to be informed of the status of its previous ADPM-NETWORK-LEAVE.request.

Figure 10-5. Network Leave Flow Chart



10.5 Data Request

The ADPD-DATA.request primitive requests the transfer of an application PDU (i.e., IPv6 packet) to another device or multiple devices. The ADPD-DATA.confirm primitive reports the result of a previous ADPD-DATA.requestprimitive. The ADPD-DATA.indication primitive is used to transfer received data from the adaptation sublayer to the upper layer.

Figure 10-6. ADP Data Request Flow Chart

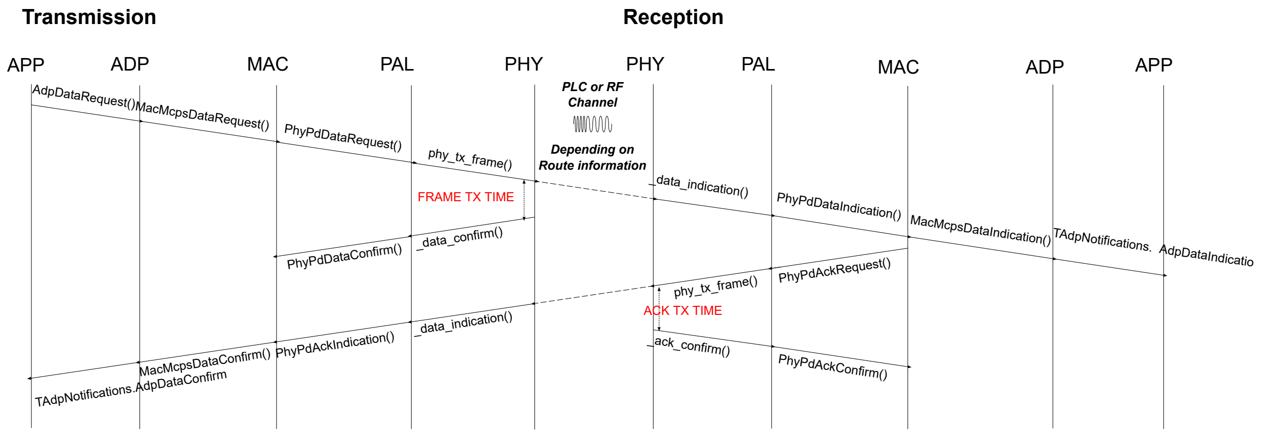
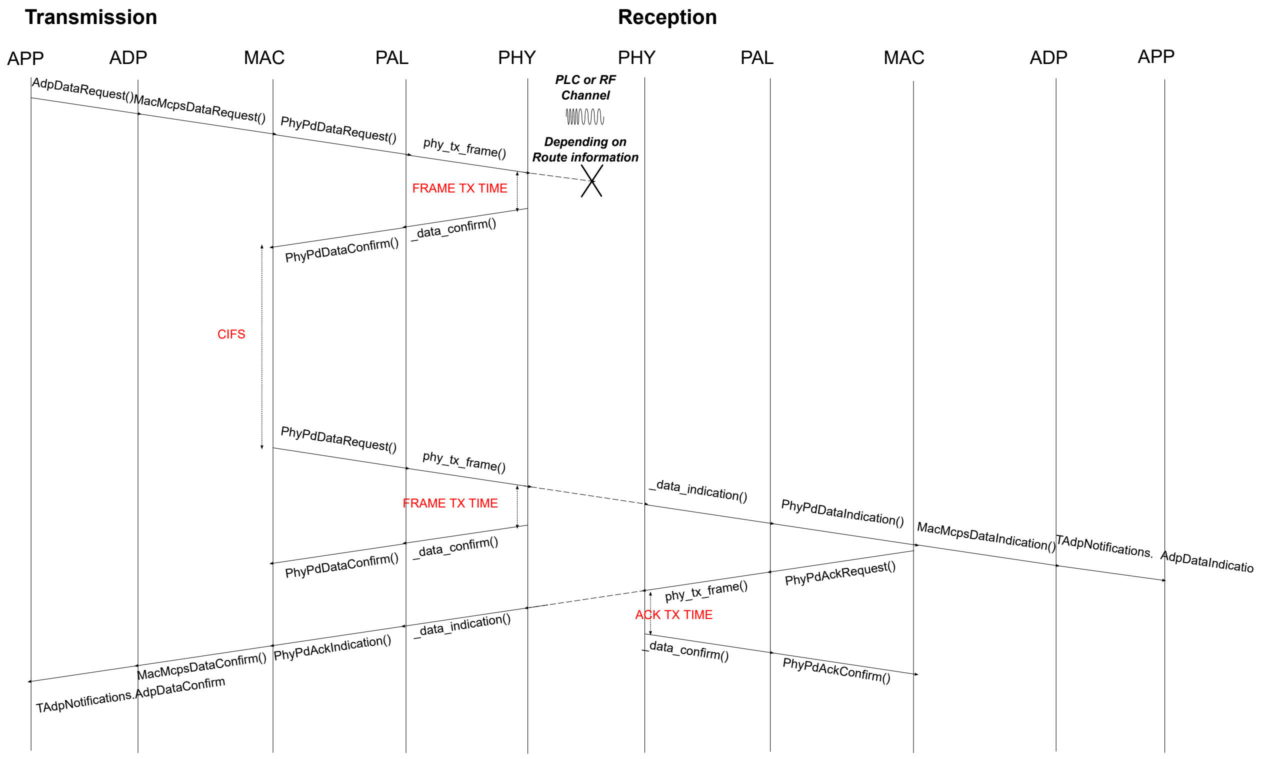


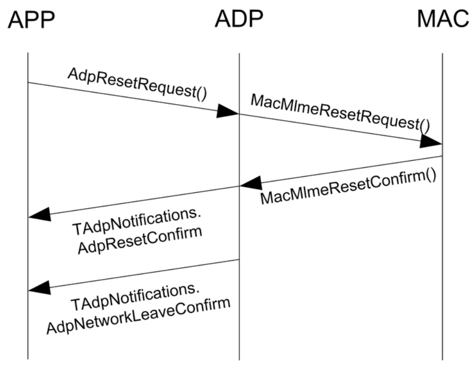
Figure 10-7. ADP Data Request Flow Chart. Retransmission due to ACK Fail



10.6 ADP Reset

The ADPM-RESET.request primitive performs a reset of the adaptation sublayer and allows the resetting of the MIB attributes. The ADPM-RESET.confirm primitive allows the upper layer to be notified of the completion of an ADPM-RESET.request primitive.

Figure 10-8. ADP Reset Flow Chart

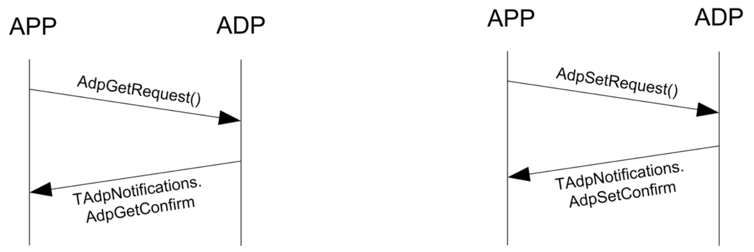

APP
ADP
MAC
AdpResetRequest()
MacMlmeResetRequest()
MacMlmeResetConfirm()
TAdpNotifications. AdpResetConfirm
TAdpNotifications. AdpNetworkLeaveConfirm

10.7 GET/SET MIBs

The ADPM-GET.request primitive allows the upper layer to get the value of an attribute from the ADP information base. The ADPM-GET.confirm primitive allows the upper layer to be informed of the status of a previously issued ADPM-GET.request primitive.

The ADPM-SET.request primitive allows the upper layer to set the value of an attribute in the ADP information base. The ADPM-SET.confirm primitive allows the upper layer to be informed about a previous ADPM-SET.request primitive.

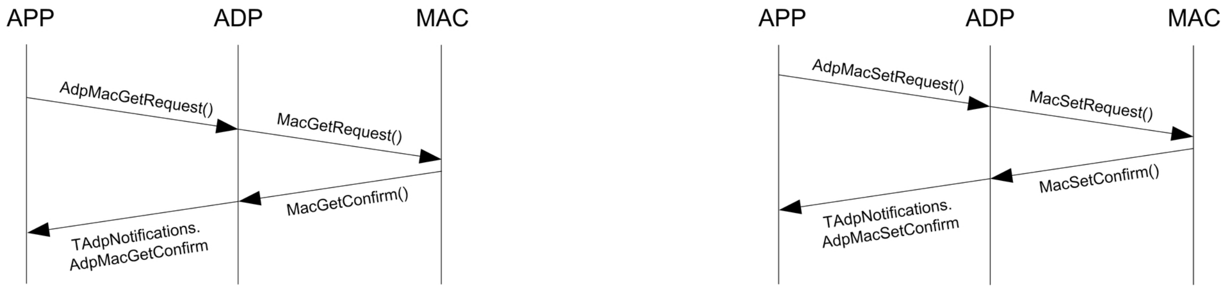
Figure 10-9. ADP GET Request and ADP SET Request Flow Charts



The ADPM-MAC.GET.request primitive allows the upper layer to get the value of an attribute from the MAC information base. The upper layer cannot access directly the MAC layer while ADP is running. The ADPM-MAC.GET.confirm primitive allows the upper layer to be informed of the status of a previously issued ADPM-MAC.GET.request primitive.

The ADPM-MAC.SET.request primitive allows the upper layer to set the value of an attribute in the MAC information base. The upper layer cannot access directly the MAC layer while ADP is running. The ADPM-MAC.SET.confirm primitive allows the upper layer to be informed about a previous ADPM-MAC.SET.request primitive.

Figure 10-10. ADPMAC GET Request and ADPMAC SET Request Flow Charts



11. Coordinator Module

This module is only present in the G3 coordinator projects. This section describes the API of the implementation of the coordinator functionality defined by the G3 standard to perform the bootstrapping procedure with joining devices (LoWPAN Bootstrapping Protocol, LBP).

11.1 Functional Description

The Coordinator module implements the functionality of responding to joining devices, in order to perform the network joining procedure (bootstrap) with them. It also maintains a list of devices that have joined the network, and manages the short address assigned to each one of them.

The module also offers the possibility to perform a re-keying procedure, by which the GMK is changed as defined in the G3 specification [ITU G.9903].

Finally, a primitive to start a coordinator-initiated kick is also offered by module, as well as a callback to process device-initiated kicks (which originate a leave indication).

11.2 API Listing

11.2.1 Coordinator Module Initialization

The module initialization function sets up all the variables and structures needed for this functionality. It must be called at system start.

Its prototype is:

```
void bs_init (void)
```

Parameters:

None

11.2.2 Re-keying

In order to launch a re-keying procedure, the following function must be called:

```
void bs_lbp_launch_rekeying (void)
```

Parameters:

None

See [ITU G.9903] for details about the procedure, and LBP_IB_GMK = 0x00000005 and LBP_IB_REKEY_GMK = 0x00000006 to set the current and new GMK.

11.2.3 Joined Devices Table Access

The table of devices that have joined the network can be accessed, as well as their number.

The number of devices that have successfully joined the network is returned by the following function:

```
uint16_t bs_lbp_get_lbds_counter (void)
```

Parameters:

None

And the table entries can be accessed through the function:

```
uint16_t bs_lbp_get_lbds_address (uint16_t i)
```

Parameters:

uint16_t i	Index of the entry to access

The function returns the short address assigned to the device stored in that entry of the table. The short addresses are assigned sequentially to the devices, in order of arrival, starting from the value defined in LBP_IB_INITIAL_SHORT_ADDRESS (see LBP_IB_INITIAL_SHORT_ADDRESS = 0x00000002 below).

Or through the function:

```
bool bs_lbp_get_lbds_ex_address (uint16_t us_short_address, uint8_t *puc_extended_address)
```

Parameters:

uint16_t us_short_address	Short address of the entry to access
*uint8_t *puc_extended_address*	Pointer to which extended address will be copied to

The function returns true in case the requested entry exists, and false otherwise.

11.2.4 Kick

As described in the G3 specification [ITU G.9903], the kick procedure can be initiated by the device or by the coordinator.

For the coordinator to start a kick procedure, the following function must be used:

```
void bs_lbp_kick_device (uint16_t us_short_address)
```

Parameters:

uint16_t us_short_address	Short address of the device to be kicked

11.2.5 IB Access

The Coordinator module offers a set of IBs which can be read and/or written using the following functions:

```
void bs_lbp_get_param (uint32_t ul_attribute_id, uint16_t us_attribute_idx, struct
t_bs_lbp_get_param_confirm *p_get_confirm)
```

Parameters:

uint32_t ul_attribute_id	ID of attribute to be read
uint16_t us_attribute_idx	Index to read (in case it is a list)
*struct t_bs_lbp_get_param_confirm *p_get_confirm*	Pointer to confirm structure where result will be returned

```
void bs_lbp_set_param (uint32_t ul_attribute_id, uint16_t us_attribute_idx, uint8_t
uc_attribute_len, const uint8_t *puc_attribute_value, struct t_bs_lbp_set_param_confirm
*p_set_confirm)
```

Parameters:

uint32_t ul_attribute_id	ID of attribute to be written
uint16_t us_attribute_idx	Index to write (in case it is a list)
uint8_t uc_attribute_len	Length of attribute to be written
*const uint8_t *puc_attribute_value*	Value to be written

*struct t_bs_lbp_set_param_confirm *p_set_confirm*	Pointer to confirm structure where result will be returned

11.2.6 Callback Integration with Upper and Lower Layers

When initializing the underlying ADP layer, all G3 stack callbacks have to be defined and set as parameter. In the provided implementation, functions to handle LBP callbacks (LbpConfirm and LbpIndication) are implemented in the Coordinator module; therefore, the upper layer which is in charge of initializing G3 stack needs to know the pointers to these functions in advance, before calling the ADPInitialize function. An example of usage can be found in provided example application for coordinator (described in DLMS App).

To get the callback pointers to link LBP notifications, the following function is used:

```
TBootstrapAdpNotifications* bs_get_not_handlers (void)
```

Parameters:

None

The callbacks are returned in the structure pointed by TBootstrapAdpNotifications*, defined as follows:

```
    typedef struct {
        AdpLbpConfirm fnctAdpLbpConfirm;
        AdpLbpIndication fnctAdpLbpIndication;
    }TBootstrapAdpNotifications;
```

The Coordinator module also offers the possibility to the upper layer of setting callbacks in order to be notified when a device joins or leaves the network, by means of the functions:

```
void bs_lbp_leave_ind_set_cb(pf_app_leave_ind_cb_t pf_handler)
```

Parameters:

pf_app_leave_ind_cb_t pf_handler	Pointer to function which will be called when a device leaves the network

```
void bs_lbp_join_ind_set_cb(pf_app_join_ind_cb_t pf_handler)
```

Parameters:

pf_app_join_ind_cb_t pf_handler	Pointer to function which will be called when a device joins the network

11.3 Coordinator Objects Specification and Access

In addition to the attributes defined in the G3 specification [ITU G.9903] and the MAC- and ADP-specific attributes (MIB Objects Specification and Access, ADPIB Objects Specification and Access), the Coordinator Module defines and manages the IB attributes listed here. The default endianness of all Coordinator IBs is little-endian, unless explicitly specified.

11.3.1 LBP_IB_IDS = 0x00000000

This object represents the Network Access Identifier of the EAP server. Its size depends on the used band (8 bytes for Cenelec and FCC, and 34 bytes for ARIB).

Access is write-only.

11.3.2 LBP_IB_DEVICE_LIST = 0x00000001

This IB provides access to the joined devices list. It allows the addition of an entry in this list, providing a pair short address – extended address, and to read one of the entries. If the entry to be written is already occupied by a valid entry, an error is returned.

The provided value when set has to be an array of size 10 bytes, where 2 first bytes are short address, and the following 8 bytes are extended address. The returned value follows this same format. The extended address is encoded in big endian and the short address is encoded in little endian to be coherent with ADP encodings.

The maximum number of entries in the table of joined devices is 500 for the lite coordinator and 2000 in the full one.

Access is read/write.

11.3.3 LBP_IB_INITIAL_SHORT_ADDRESS = 0x00000002

This IB provides access to the configurable initial short address that the coordinator uses to assign to the joining devices. Its size is 2 bytes.

If there are already joined devices, this parameter can no longer be changed, and an error is returned if this is attempted.

Access is read/write.

11.3.4 LBP_IB_ADD_DEVICE_TO_BLACKLIST = 0x00000003

This IB allows the addition of an extended address to the LBP blacklist. By doing so, this address will be banned from joining the network for the configured blacklist permanence time [ITU G.9903]. This IB's size is 8 bytes. The encoding of the extended address must be big endian.

Access is write-only.

11.3.5 LBP_IB_PSK = 0x00000004

With this IB, the PSK of the LBP procedure can be specified. Its size is 16 bytes.

Access is write-only.

11.3.6 LBP_IB_GMK = 0x00000005

With this IB, the GMK of the LBP procedure can be specified. Its size is 16 bytes.

Access is write-only.

11.3.7 LBP_IB_REKEY_GMK = 0x00000006

With this IB, the destination GMK of the LBP re-keying procedure can be specified. Its size is 16 bytes.

Access is write-only.

11.3.8 LBP_IB_SHORT_ADDRESS_FROM_EXTENDED = 0x00000007

If this IB is set, the short addresses are assigned based on the two last bytes of the extended address of the joining device. This may be useful if extended addresses are correlated, for easy identification of the devices. If not set, short addresses are assigned sequentially starting from the LBP_IB_INITIAL_SHORT_ADDRESS value. Parameter size is 1 byte, and default value is 0.

Access is read/write.

11.3.9 LBP_IB_MSG_TIMEOUT = 0x00000008

This IB contains a time-out, in seconds, to wait for the next bootstrap packet for a joining device. In case time-out expires, the slot for such joining device is cleared and a new joining device will be assigned to it. Parameter size is 2 bytes, and default value is 40.

Access is read/write.

11.4 Brief of Provided Coordinator Module Example

Microchip provides an example implementation of the Coordinator module to be used in the G3 PAN Coordinator in order to accept devices willing to join the network, and therefore make them accessible to upper layers for data exchange.

11.4.1 Initialization Function

When upper layer calls the LBP initialization function, the following steps are performed:

- Short address is set to 0 as coordinator
- Initial address to give to devices joining the network is set to 1
- LBD blacklist and LBD device list are cleared
- IdS parameter is initialized to constant values provided in the LBP module
- GMK key is set

All these values are available as either constants or initialized variables in the same Coordinator module, and can be changed by the user to fit specific needs.

11.4.2 Registered Devices List

A registered devices list is managed by the Coordinator module. This list has a number of elements defined by the constant MAX_LBDS. Each element represents a device that has joined the network, and the information contained in the list entry is the EUI64 of the joined device. The short address of the device is given by the result of the following operation: *(index in list) + g_current_context.initialShortAddr*. This initial short address is initialized to 1 by default, but can be changed to fit user needs by either changing the default value or by means of setting the LBP_IB_INITIAL_SHORT_ADDRESS parameter.

In the provided implementation, the Coordinator module assigns short addresses sequentially, starting with g_current_context.initialShortAddr and adding 1 each time a device wants to join the network. If a device bootstraps more than once, a new short address is given, even if it was already present in the network before and the old entry is removed from the list. In case the end of the list is reached, the algorithm looks for removed entries to reuse them. If the list is full, no additional devices are accepted in the network.

11.4.3 Coordinator Blacklist

A blacklist of devices is implemented in the module. When a device is present in the blacklist, it is not accepted when it tries to join the network. Entries can be added to the blacklist from the upper layers using LBP_IB_ADD_DEVICE_TO_BLACKLIST.

11.4.4 Processing of LBP Frames

An implementation of LBP frames is provided in the module following specification of IETF RFC 3748 and extended in G3 spec. Frame processing is available in the module. The encoding/decoding is done by a separate module (LBP module) as it is used by both Coordinator and Devices.

11.4.5 Re-keying

Re-keying functionality is implemented in the module.

After initial bootstrap using this module, all joined devices and the coordinator will be using key index 0 and GMK defined in the LBP_IB_GMK attribute. Launching the re-keying procedure by means of *bs_lbp_launch_rekeying* primitive will perform a network re-keying as defined in the G3 spec, by performing the next steps:

- Distribution of the new GMK, using the value stored in LBP_IB_REKEY_GMK attribute, and setting the key index not currently in use (as stated above, current index is 0 after initial bootstrap, so first re-keying will use index 1, and further processes will alternate key index value)
- Activation of new key in devices, by means of procedure defined in G3 spec. Devices will be commanded to set the distributed active index and the distributed GMK in the distribution process
- Activation of the new key in coordinator when activation in devices has finished
- Bootstrap module will then copy the value stored in LBP_IB_REKEY_GMK onto LBP_IB_GMK attribute, so this module is updated accordingly to the values set in ADP and MAC layers

When this process ends, all the network will be using the distributed key, and the module is ready for a new re-keying procedure by changing the value in LBP_IB_REKEY_GMK attribute and calling *bs_lbp_launch_rekeying* primitive again.

12. Serialization with Embedded USI

The Embedded USI is a wrapper that provides the interface between the G3 FW stack and the serial communications channel.

For serial transmissions from the G3 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 G3 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:

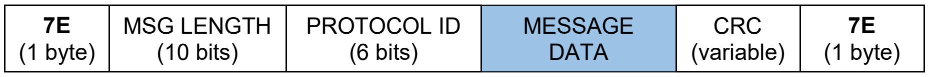
- USI frame format
- USI G3 protocols
- USI Configuration

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

12.1 USI Frame Format

The USI frame format is based on the HDLC specification used along with DLMS, and is shown in the following figure.

Figure 12-1. USI Frame Format



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 USI Protocols and Associated CRC Size table below)
- 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 USI Protocols and Associated CRC Size table below)

Table 12-1. USI Protocols and Associated CRC Size

Protocol	Protocol ID	CRC size (bits)
PROTOCOL_PHY_ATPL2X0[1]	0x22	16
PROTOCOL_SNIF_G3	0x23	16
PROTOCOL_MAC_G3	0x24	16
PROTOCOL_ADP_G3	0x25	16
PROTOCOL_COORD_G3	0x26	16
PROTOCOL_USER_DEFINED[2]	0xFE	Defined by the user

Notes:

1. The PROTOCOL_PHY_ATPL2X0 is used by the PLC PHY Tester PC tool that Microchip provides with the evaluation kit in order to serialize the API of the PHY layer. Despite this denomination, this protocol is also used in the PL360 platform for the same purpose.
2. Defined by the user for their own proprietary protocol, if necessary.

12.2 USI G3 Protocols

The USI is able to serialize the following G3 interfaces and services:

- G3 PHY layer. Used between Phy Tester Tool Embedded FW and PC Application.
- G3 Sniffer. Used between Phy Sniffer Tool Embedded FW and PC Application.
- G3 MAC layer. Used between ADP_MAC_Serialized Embedded FW and Example Scripts.
- G3 ADP layer. Used between ADP_MAC_Serialized Embedded FW and Example Scripts.
- G3 COORD layer. Used between ADP_MAC_Serialized Embedded FW and Example Scripts.
- User Application. Through the available User Defined USI Protocol.

12.3 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.

12.3.1 USI Ports Definition and Configuration

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, TX_UART_BUF0_SIZE, RX_UART_BUF0_SIZE)
#define PORT_1 CONF_PORT(USART_TYPE, 1, 230400, TX_USART_BUF1_SIZE, RX_USART_BUF1_SIZE)
```

`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 12-2. USI Port Configuration Parameters

Parameter	Description	Valid Values
Type	Type of link	UART_TYPE for UART USART_TYPE for USART USB_TYPE for USB
Channel	Instance	0: UART0/USART0 1: UART1/USART1 2: UART2/USART2
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.
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.

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.

12.3.2 Linking of G3 Sniffer

To link the embedded G3 sniffer to a USI port defined in the Embedded USI, file *conf_usi.h* must be edited.

An example of enabling embedded sniffer is provided below.

```
/* Enable Embedded Sniffer */
#define ENABLE_SNIFFER

#define USI_PORT_0      0

/* Port Communications configuration */
#define NUM_PORTS       1
#define PORT_0 CONF_PORT(UART_TYPE, USI_PORT_0, 230400, TX_UART_BUF0_SIZE, RX_UART_BUF0_SIZE)

/* Select PORT to sniffer phy iface */
#define PHY_SNIFFER_SERIAL_PORT                  USI_PORT_0
```

12.3.3 Linking of G3 API

To link any (or all) of the G3 APIs (MAC, ADP, COORD) to a USI port defined in the Embedded USI, file *conf_usi.h* must be edited.

An example of enabling all APIs Serialization, as done in the Adp_Mac_Serialized example application, is provided below.

```
#define USI_PORT_0      0

/* Port Communications configuration */
#define NUM_PORTS       1
#define PORT_0 CONF_PORT(UART_TYPE, USI_PORT_0, 230400, TX_UART_BUF0_SIZE, RX_UART_BUF0_SIZE)

/* Select PORT to serialize layers */
#define ADP_SERIAL_PORT                  USI_PORT_0
#define MAC_SERIAL_PORT                  USI_PORT_0
#define COORD_SERIAL_PORT                USI_PORT_0
```

In the case that a user wants to add Sniffer capabilities apart from controlling the G3 stack through the same Serial Channel, the Sniffer Port has to be defined also (as seen on previous section) by adding the following line:

```
#define PHY_SNIFFER_SERIAL_PORT          USI_PORT_0
```

13. Supported Platforms

This chapter describes which hardware platforms are currently supported in the Microchip G3-PLC software package. A platform is usually comprised of three major components:

- An MCU
- A transceiver chip. This may be integrated into the MCU (i.e.:PL485)
- A specific board or even several boards that contain the MCU and the transceiver chip

13.1 Supported MCU Families

Currently, the supported families are the SAM4C, SAME70, SAMG55 and PIC32CX platforms.

The dedicated code for each device of the family can be found in the corresponding sub-directories of the FW package.

13.2 Supported Devices

Currently, the supported devices are ATPL250, PL360, PL460 and PL485 for PLC, and AT86RF215 for RF.

13.3 Supported Boards

The currently supported boards and platforms are the following:

- PIC32CXMTG-EK + PL460-EK on Xplained Port
- PIC32CXMTSH-DB + PL460-EK on Xplained Port
- PL360MB
- SAM4CMS-DB + PL460-EK on Xplained Port
- SAME70 Xplained + PL460-EK on Xplained Port
- PL360BN
- SAMG55 Xplained + PL460-EK on Xplained Port
- SAMG55 Xplained + PL460-EK on Xplained Port + ATREB215-EK on Xplained Port
- PL360G55CF
- PL360G55CB
- PL360G55CF + ATREB215-EK on Xplained Port
- PL360G55CB + ATREB215-EK on Xplained Port
- PL485-EK
- PL485-EK + ATREB215-EK on Xplained Port
- ATPL250AMB
- ATPL250ABN

14. 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.

14.1 General Prerequisites

The following tools and toolchains are used for building the applications from this FW package:

- Microchip Studio (see www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices)
- IAR Systems® Embedded Workbench (see http://www.iar.com/)
- Keil® MDK-ARM Microcontroller Development Kit (see http://www.keil.com/arm/mdk.asp)

14.2 Building the Applications

14.2.1 Using Microchip Studio

Each application can be rebuilt with Microchip Studio directly, using version 7.

14.2.2 Using IAR Embedded Workbench®

Each application can be rebuilt with IAR Embedded Workbench directly, using version 8.32.

14.2.3 Using Keil®

Each application can be rebuilt with Keil µVision® directly, using version 5.

15. PICS

This chapter lists the conformance of the Microchip G3-PLC implementation with the requirements and optional features as defined by the G3-PLC specification document.

A Protocol Implementation Conformance Statement (PICS), as defined by ISO 9646 [2], 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 G3 specification and in the test suites used by the certification process.

When an implementation is ready for certification, the implementer must provide the PICS corresponding to the implementation capability.

The PICS values are included in the device certificate, in order to indicate the conditions of the certification in terms of implemented options.

15.1 Band-plan Support

This PICS entry is used to describe the band-plan supported by the device under test.

If a physical device is able to support several band-plans, it will be tested once for each band-plan, setting the following PICS value accordingly.

Current implementation supports Cenelec A-Band, Cenelec B-Band, ARIB and FCC band-plans.

15.2 Feature Implementation Statement

Microchip G3 PLC stack implements the following optional features:

- Coherent Modulation.
- Security profile F1 and F2.
- Routing: LOADng implementation supporting composite metrics and capacity metrics.

15.3 Other Features

Microchip provides a Bootstrap module with some basic PAN Coordinator features, such as creating the network and accepting all network join from the devices. This way, it is possible to build a network and exchange data.

This is just a basic implementation for demonstration purposes, and it should not be taken as the reference to build a full-featured PAN Coordinator.

16. Gateway Configuration

First, it is important to configure the G3 Gateway Coordinator to fit the user´s environment. Microchip provides two different kinds of Gateway implementations: a full G3 Coordinator for the ATPL250ABN, PL360BN and SAME70-XPLAINED + PL460EK boards (with Ethernet port), and another version for the rest of the available platforms (serial connection, PPP).

IPv6 connectivity between the host computer and the G3 Gateway Coordinator is required. This chapter shows how to configure and verify that the user´s environment is ready.

16.1 Ethernet G3 Gateway Coordinator Configuration

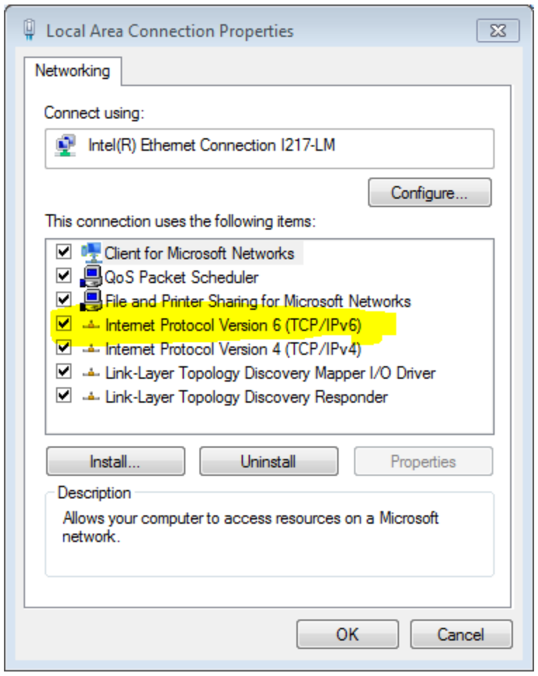
16.1.1 Environment Requirements

The PC must have the IPv6 protocol enabled.

Important: Contact your Network Administrator to learn the details of the IPv6 configuration in your LAN network.

To verify that the IPv6 protocol is enabled, check the configuration of the local network adapter. Go to *"Windows/ Control Panel/Network and Sharing Center/Change Adapter Settings"*, then select the "Properties" submenu (right click over the icon) of your network interface.

Figure 16-1. Local Area Connection Properties



Local Area Connection Properties

Networking

Connect using:

Intel(R) Ethernet Connection I217-LM

This connection uses the following items:

- [✓] Client for Microsoft Networks
- [✓] QoS Packet Scheduler
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] Internet Protocol Version 6 (TCP/IPv6)
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [✓] Link-Layer Topology Discovery Mapper I/O Driver
- [✓] Link-Layer Topology Discovery Responder

Description

Allows your computer to access resources on a Microsoft network.

The figure above shows the Properties window with the IPv6 protocol enabled. If the IPv6 protocol is not enabled in the LAN router, you will need to manually set up your system and the Gateway Coordinator (see Static Host and Gateway configuration).

16.1.2 Automatic Gateway Coordinator Configuration

By default, the G3 Gateway Coordinator firmware is configured to use the SLAAC (State Less Address Auto Configuration) protocol. This means that it will listen to the LAN traffic waiting for a Router Advertisement Message from the LAN router to auto configure itself. The figure below shows the network connections needed. This process might take up to several minutes, depending on the local network configuration.

Figure 16-2. Automatic Environment Configuration

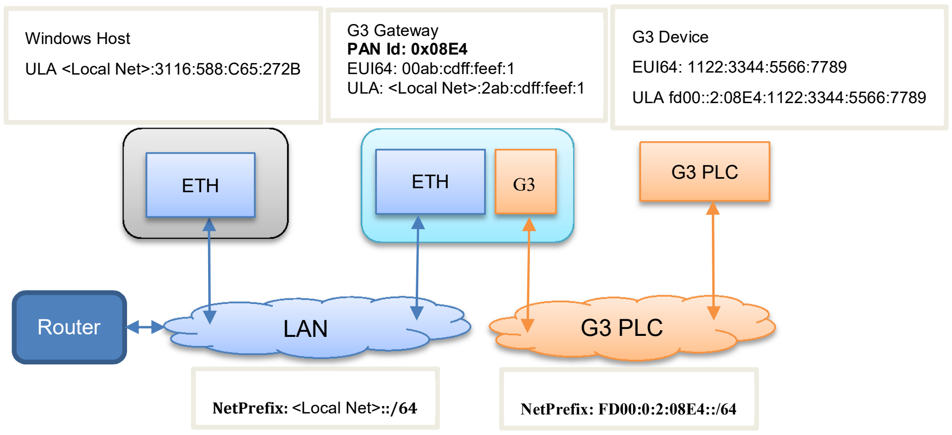
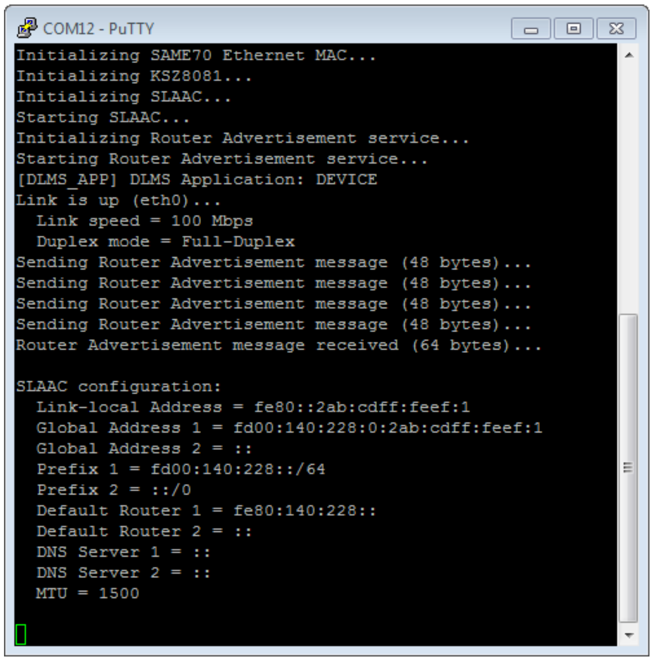

Windows Host
ULA <Local Net>:3116:588:C65:272B
G3 Gateway
PAN Id: 0x08E4
EUI64: 00ab:cdff:feef:1
ULA: <Local Net>:2ab:cdff:feef:1
G3 Device
EUI64: 1122:3344:5566:7789
ULA fd00::2:08E4:1122:3344:5566:7789
ETH
ETH
G3
G3 PLC
Router
LAN
G3 PLC
NetPrefix: <Local Net>::/64
NetPrefix: FD00:0:2:08E4::/64

Figure 16-3. G3 Gateway Coordinator Console, SLAAC Protocol Log



```
Initializing SAME70 Ethernet MAC...
Initializing KSZ8081...
Initializing SLAAC...
Starting SLAAC...
Initializing Router Advertisement service...
Starting Router Advertisement service...
[DLMS_APP] DLMS Application: DEVICE
Link is up (eth0)...
  Link speed = 100 Mbps
  Duplex mode = Full-Duplex
Sending Router Advertisement message (48 bytes)...
Sending Router Advertisement message (48 bytes)...
Sending Router Advertisement message (48 bytes)...
Sending Router Advertisement message (48 bytes)...
Router Advertisement message received (64 bytes)...

SLAAC configuration:
  Link-local Address = fe80::2ab:cdff:feef:1
  Global Address 1 = fd00:140:228:0:2ab:cdff:feef:1
  Global Address 2 = ::
  Prefix 1 = fd00:140:228::/64
  Prefix 2 = ::/0
  Default Router 1 = fe80:140:228::
  Default Router 2 = ::
  DNS Server 1 = ::
  DNS Server 2 = ::
  MTU = 1500
```

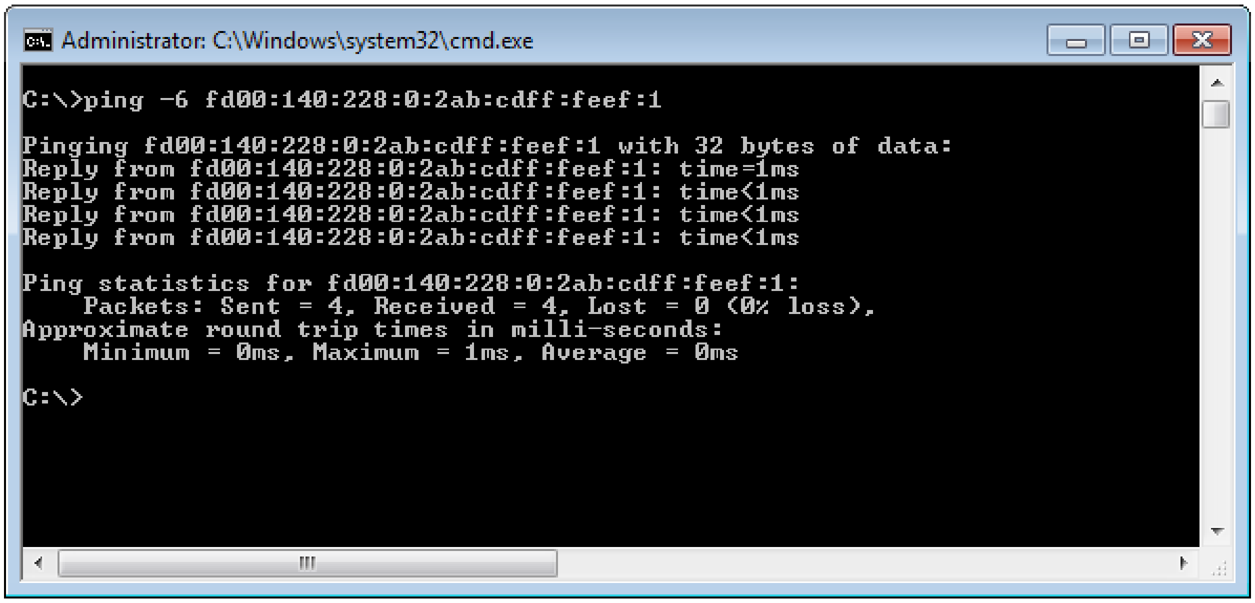
The figure above shows the output console from a G3 Gateway Coordinator receiving the Router Advertisement message. In this case, this device will assign itself two IPv6 addresses. The first address will be an LLA, and the second one will be a ULA (Global address 1). The ULA address is the one that the PC will need to know in order to talk to the gateway. In this case, the ULA address is *FD00:140:228:0:02ab:cdff:feef:1*, meaning that the device is connected to the network *"FD00:140:228:0::/64"* (as advertised by the LAN router in the previous figure).

The PC must have an address in the same network, or the local router in the LAN network must be configured to include a route to the device.

Also, the G3 Gateway Coordinator is configured to use the Router Advertisement service to publish the route to the PLC network. The console log snapshot also shows that the gateway device is sending its own Router Advertisement messages, publishing the route to the G3 PLC network. IPv6 routers on the LAN may use the published information to learn the new route to the PLC network.

Once the Gateway Coordinator has completed the SLAAC protocol, you can ping it to test the communication with your personal computer. The figure below shows the output of the ping command in a Windows 7 PC command console.

Figure 16-4. Windows Console, Ping to Gateway



```
C:\>ping -6 fd00:140:228:0:2ab:cdff:feef:1

Pinging fd00:140:228:0:2ab:cdff:feef:1 with 32 bytes of data:
Reply from fd00:140:228:0:2ab:cdff:feef:1: time=1ms
Reply from fd00:140:228:0:2ab:cdff:feef:1: time<1ms
Reply from fd00:140:228:0:2ab:cdff:feef:1: time<1ms
Reply from fd00:140:228:0:2ab:cdff:feef:1: time<1ms

Ping statistics for fd00:140:228:0:2ab:cdff:feef:1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\>
```

Important: Please contact your network administrator if you have any problems discovering your local IPv6 address and network prefix, or the G3 Gateway cannot be pinged.

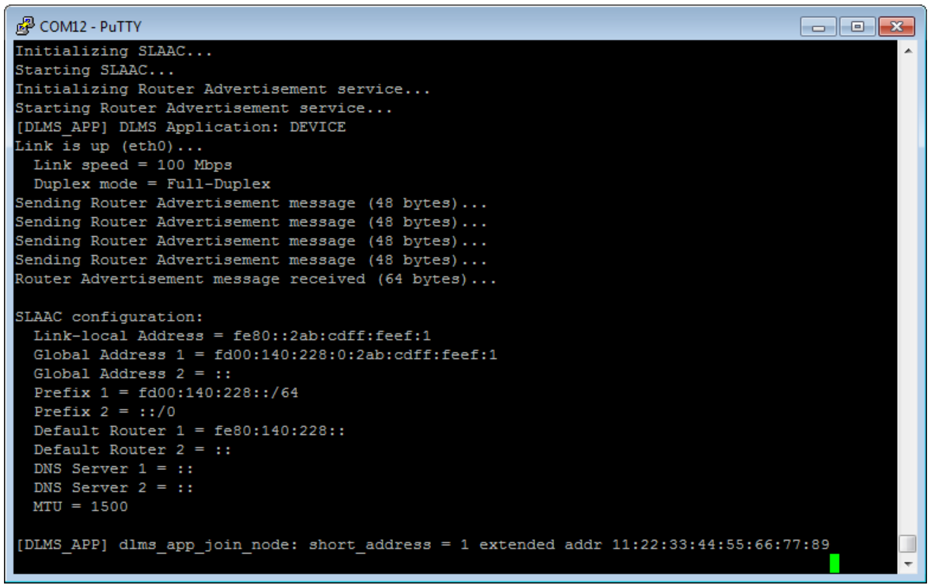
The PC also needs to know the network´s PAN Identifier that the Gateway Coordinator will create.

Important: The G3 Network´s PANID value is configured at compilation in the file “conf_bs.h”. The default value for an Ethernet Gateway Coordinator is 0x8e4.

16.1.3 Connecting to the PLC Network

Once there is connectivity with the Gateway Coordinator, the IPv6 route to the PLC network must be set up. Depending on your LAN router configuration, it may have already learned it from the Router Advertisement messages that the Gateway Coordinator generates. To check this, ping a node within the PLC network; if there is a response, the LAN router is already taking care of the routing, and the system is ready. Otherwise, you need to manually set up a route in the PC.

Figure 16-5. G3 Gateway Coordinator Console, Device Bootstrap Information

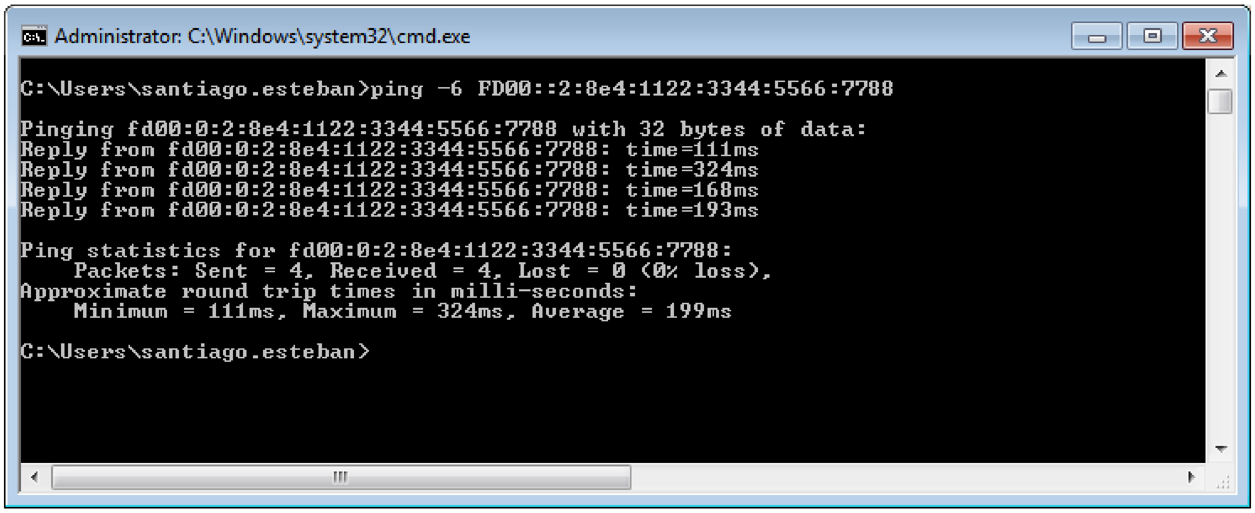


```
Initializing SLAAC...
Starting SLAAC...
Initializing Router Advertisement service...
Starting Router Advertisement service...
[DLMS_APP] DLMS Application: DEVICE
Link is up (eth0)...
  Link speed = 100 Mbps
  Duplex mode = Full-Duplex
Sending Router Advertisement message (48 bytes)...
Sending Router Advertisement message (48 bytes)...
Sending Router Advertisement message (48 bytes)...
Sending Router Advertisement message (48 bytes)...
Router Advertisement message received (64 bytes)...

SLAAC configuration:
  Link-local Address = fe80::2ab:cdff:feef:1
  Global Address 1 = fd00:140:228:0:2ab:cdff:feef:1
  Global Address 2 = ::
  Prefix 1 = fd00:140:228::/64
  Prefix 2 = ::/0
  Default Router 1 = fe80:140:228::
  Default Router 2 = ::
  DNS Server 1 = ::
  DNS Server 2 = ::
  MTU = 1500

[DLMS_APP] dlms_app_join_node: short_address = 1 extended addr 11:22:33:44:55:66:77:89
```

To ping a node, you need to know its extended address (EUI64). This information is stored inside the device; however, looking at the Gateway Console log, it will be printed at bootstrap completion. In the figure above, the last line shows the bootstrap information for a node with EUI64 *"1122:3344:5566:7788"* and a G3 MAC address *"1"*. This means that this node has two IPv6 addresses assigned, an LLA, *FE80::1122:3344:5566:7789*, and a ULA, *FD00::2:08E4:1122:3344:5566:7789*.

If the LAN router is configured to learn new routes, then, from the PC console, it will be able to ping the remote device using its ULA address. The figure below shows this. If there are no ping responses from the remote PLC network and the ping returns a "Destination net unreachable" error, a network route to the PLC network needs to be manually configured.

Figure 16-6. Windows Console. Ping to Remote PLC Network



```
C:\Users\santiago.esteban>ping -6 FD00::2:8e4:1122:3344:5566:7788

Pinging fd00:0:2:8e4:1122:3344:5566:7788 with 32 bytes of data:
Reply from fd00:0:2:8e4:1122:3344:5566:7788: time=111ms
Reply from fd00:0:2:8e4:1122:3344:5566:7788: time=324ms
Reply from fd00:0:2:8e4:1122:3344:5566:7788: time=168ms
Reply from fd00:0:2:8e4:1122:3344:5566:7788: time=193ms

Ping statistics for fd00:0:2:8e4:1122:3344:5566:7788:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 111ms, Maximum = 324ms, Average = 199ms

C:\Users\santiago.esteban>
```

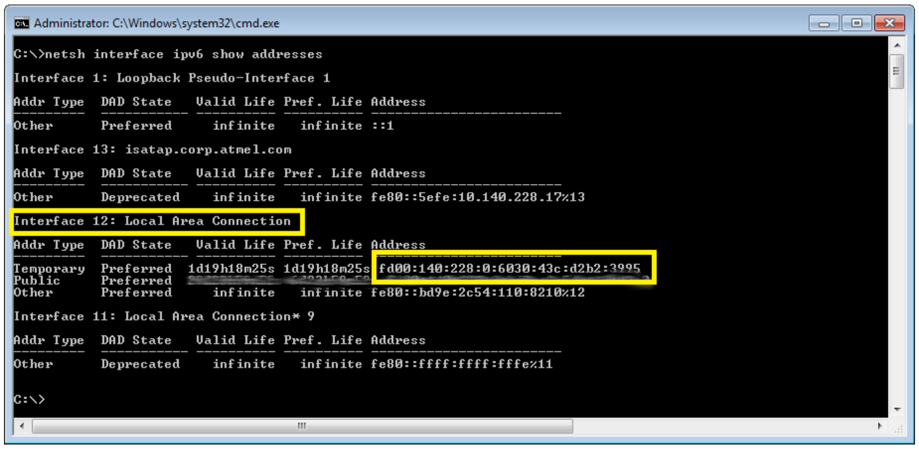
16.1.4 Configuring an IPv6 Route to a Remote PLC Network

If ping communication fails, a route to the G3-PLC network needs to be set up.

To do so, you first need to find out the network interface identifier. Execute the next command in a Windows console:

```
C:\>netsh interface ipv6 show addresses
```

Figure 16-7. Windows IPv6 Addresses



```
C:\>netsh interface ipv6 show addresses

Interface 1: Loopback Pseudo-Interface 1

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Preferred     infinite   infinite ::1

Interface 13: isatap.corp.atmel.com

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Deprecated    infinite   infinite fe80::5efe:10.140.228.17%13

Interface 12: Local Area Connection

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Temporary  Preferred  1d19h18m25s 1d19h18m25s fd00:140:228:0:6030:43c:d2b2:3995
Public     Preferred  [redacted]
Other      Preferred     infinite   infinite fe80::bd9e:2c54:110:8210%12

Interface 11: Local Area Connection* 9

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Deprecated    infinite   infinite fe80::ffff:ffff:fffe%11


C:\>
```

In the figure above, we can see that the *Interface 12, Local Area Connection* has a valid ULA address assigned. That ULA address also indicates in which network we are located: *FD00:140:228:0::/64*. Both the G3 Gateway coordinator and the PC must be in the same network.

To set up a new route to the PLC network, execute the next command on the host PC console:

```
    C:\>netsh interface ipv6 add route FD00:0:2:08E4::/64 interface=12 nexthop=
    FD00:140:228:0:02ab:cdff:feef:0001
```

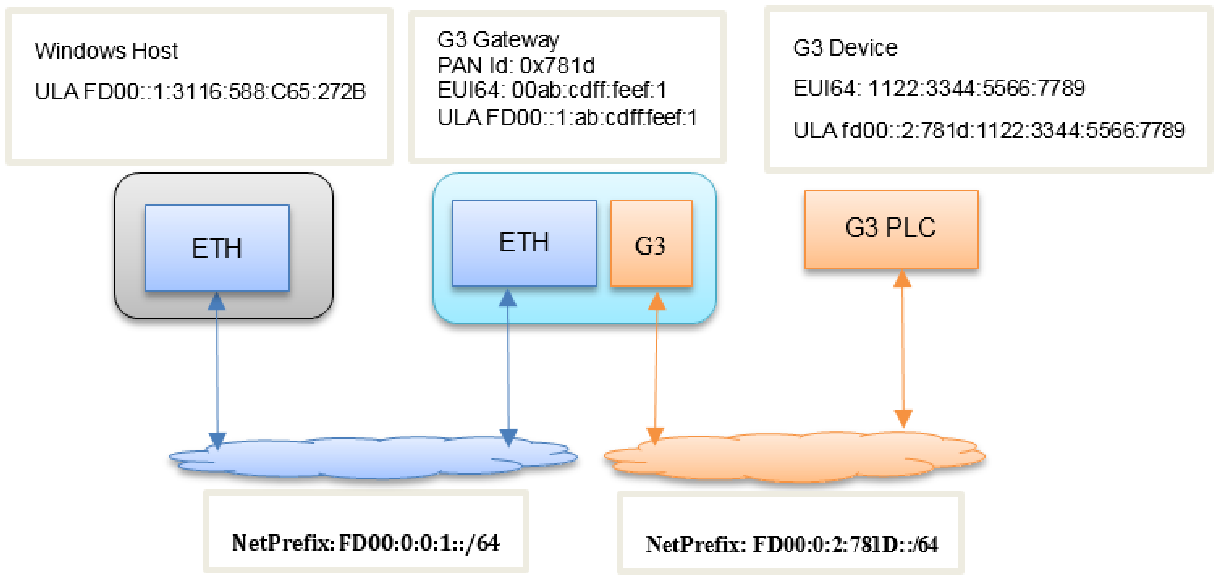
This command tells the host computer to redirect all messages to the network *"FD00:0:2:8e4::/64"* through Interface 12 and using gateway *FD00:140:228:0:02ab:cdff:feef:0001*. Now you should be able to ping the target network as in Figure 16-6.

If everything is correctly set up, the host PC is able to ping the Gateway Coordinator and the G3 PLC network devices.

16.1.5 Static Host and Gateway Configuration

If the LAN connecting the host PC and the G3 Gateway Coordinator does not support IPv6 (there is no IPv6 router on the network), all the network addresses will need to be manually configured. The host PC and G3 Gateway must be connected to the same Ethernet switch/hub or VLAN. A network setup as defined in the figure below has to be configured.

Figure 16-8. Manual Setup Environment Example


Windows Host
ULA FD00::1:3116:588:C65:272B
G3 Gateway
PAN Id: 0x781d
EUI64: 00ab:cdff:feef:1
ULA FD00::1:ab:cdff:feef:1
G3 Device
EUI64: 1122:3344:5566:7789
ULA fd00::2:781d:1122:3344:5566:7789
ETH
ETH
G3
G3 PLC
NetPrefix: FD00:0:0:1::/64
NetPrefix: FD00:0:2:781D::/64

In the host PC, configure the following:

1. Enable the IPv6 protocol.
2. Configure a valid ULA address.
3. Configure the network route to the target PLC network.

In the G3 Gateway Coordinator configure:

1. A valid ULA address (in the same network as the host PC).
2. The IP address to use as an IPv6 gateway; in this case, the IPv6 address of the host PC.

16.1.5.1 Host PC IPv6 Configuration

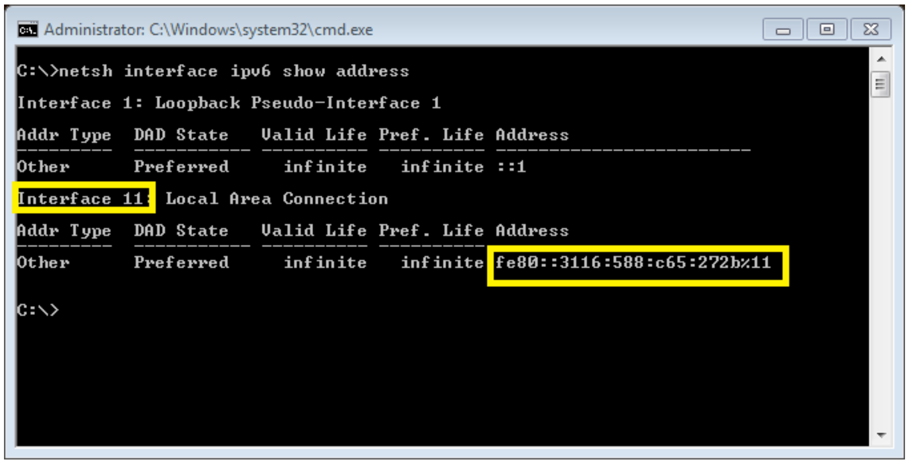
First, enable the IPv6 protocol. Look for the Local Area Connection Properties of your network adapter. To do so, go to *"Control Panel/Network and Sharing Center"*, click on *"Change Adapter Settings"* on the left side menu. Right click on your adapter´s icon and select the "Properties" option. Then, confirm that the IPv6 protocol is enabled (Internet Protocol Version 6 TCP/IPv6) and click OK. See Figure 16-1.

Once the IPv6 protocol is enabled, open a command console: "cmd.exe" and execute the command:

```
C:\>netsh interface ipv6 show address
```

You will see something similar to Figure 16-7, with at least a Local Link Address, *"FE80::<EUI64>"* assigned to your network adapter.

Figure 16-9. Windows Console IPv6 Configuration



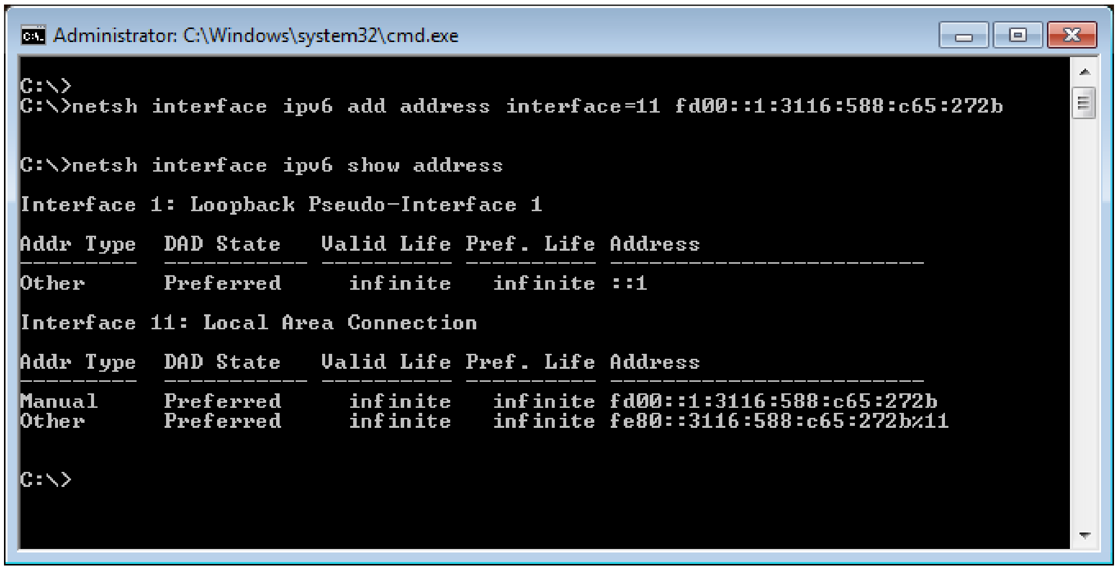
```
C:\>netsh interface ipv6 show address
Interface 1: Loopback Pseudo-Interface 1
Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Preferred     infinite   infinite ::1
Interface 11: Local Area Connection
Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Preferred     infinite   infinite fe80::3116:588:c65:272b%11

C:\>
```

The figure above shows the configuration for Interface 11 and its LLA. In this case, the host PC has a EUI64 *3116:588:c65:272b*. Next, add a new ULA address to be able to route messages:

```
C:\>netsh interface ipv6 add address interface=11 address=FD00::1:<Host EUI64>
```

The figure below shows the console output of adding a ULA address to this host.

Figure 16-10. Add ULA IPv6 Address to Host PC



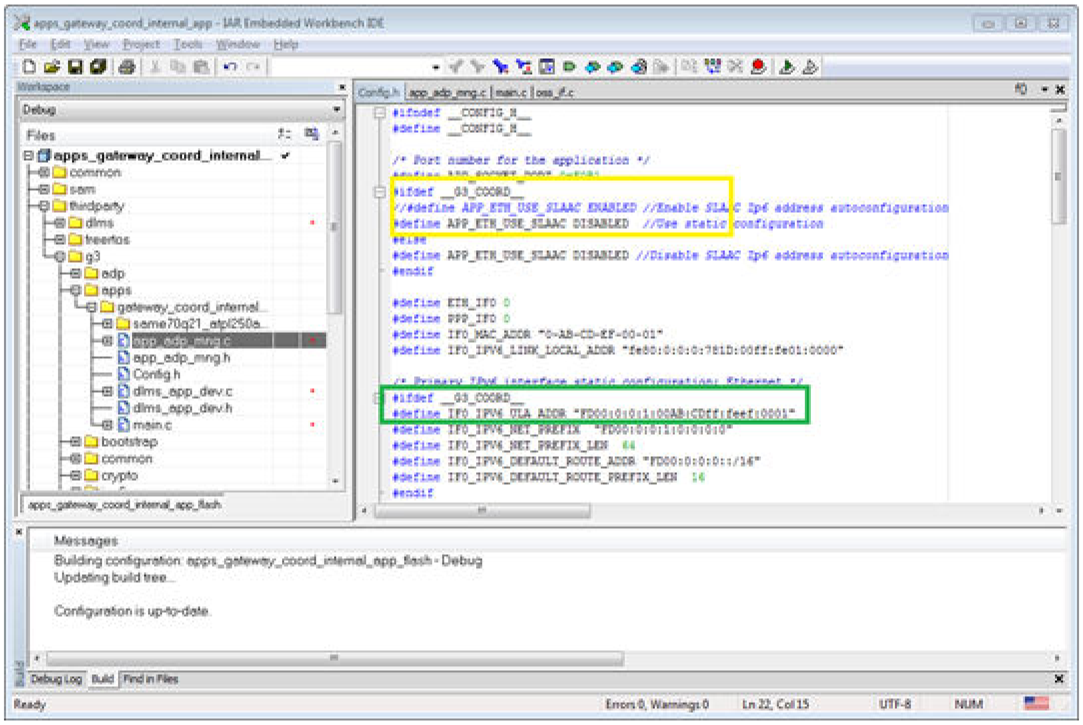
```
C:\>
C:\>netsh interface ipv6 add address interface=11 fd00::1:3116:588:c65:272b

C:\>netsh interface ipv6 show address

Interface 1: Loopback Pseudo-Interface 1

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Preferred     infinite   infinite ::1

Interface 11: Local Area Connection

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Manual     Preferred     infinite   infinite fd00::1:3116:588:c65:272b
Other      Preferred     infinite   infinite fe80::3116:588:c65:272b%11

C:\>
```

16.1.5.2 G3 Gateway Coordinator Configuration

Now that the host PC has an IPv6 address, configure and rebuild the G3 Gateway Coordinator Firmware to switch off the SLAAC protocol and configure the proper IPv6 addresses. Here, only the configuration parameters will be shown; to find out information about how to build the G3 Gateway firmware, refer to your user kit manual.

The figure below shows the G3 Gateway Coordinator project open with the IAR Workbench IDE. In order to disable the SLAAC protocol, edit the Config.h file (yellow box) to comment the macro APP_ETH_USE_SLAAC_ENABLED and uncomment the macro APP_ETH_USE_SLAAC_DISABLED. Rebuild and download the new firmware into the target hardware. Reboot the system.

Figure 16-11. IAR Workbench G3 Gateway Config.h Header File



```
#ifndef __CONFIG_H__
#define __CONFIG_H__

/* Port number for the application */
#define APP_SOCKET_PORT 0xF0B1
#ifdef __G3_COORD__
//#define APP_ETH_USE_SLAAC_ENABLED //Enable SLAAC Ip6 address autoconfiguration
#define APP_ETH_USE_SLAAC_DISABLED  //Use static configuration
#else
#define APP_ETH_USE_SLAAC_DISABLED //Disable SLAAC Ip6 address autoconfiguration
#endif

#define ETH_IFO 0
#define PPP_IFO 0
#define IF0_MAC_ADDR "0-AB-CD-EF-00-01"
#define IF0_IPV6_LINK_LOCAL_ADDR "fe80:0:0:0:781D:00ff:fe01:0000"

/* Primary IPv6 interface static configuration: Ethernet */
#ifdef __G3_COORD__
#define IF0_IPV6_ULA_ADDR "FD00:0:0:1:00AB:CDff:feef:0001"
#define IF0_IPV6_NET_PREFIX  "FD00:0:0:1:0:0:0:0"
#define IF0_IPV6_NET_PREFIX_LEN  64
#define IF0_IPV6_DEFAULT_ROUTE_ADDR "FD00:0:0:0::/16"
#define IF0_IPV6_DEFAULT_ROUTE_PREFIX_LEN  16
#endif
```

```
Messages
Building configuration: apps_gateway_coord_internal_app_flash - Debug
Updating build tree...

Configuration is up-to-date.
```

Now the G3 Gateway Coordinator is configured to use the default address defined by the macro IF0_IPV6_ULA_ADDR in Config.h (green box). We should be able to ping the target hardware from the host PC:

```
C:\>Ping -6 FD00::1:ab:cdff:feef:1
```

Important: If you cannot ping your hardware device from your host PC, contact your Network Administrator to learn about your local network configuration for the correct setup.

16.1.5.3 Host Routing Configuration

With the host PC and gateway network configured, it is time to set up the remote PLC network route on the host PC.

Similarly to the routing configuration explained in the 16.1.2. Automatic Gateway Coordinator Configuration, configure a new route to the target network:

```
C:\>netsh interface ipv6 add route FD00:0:2:08E4::/64 interface=11 nexthop=
FD00::1:0ab:cdff:feef:1
```

Remember to configure the interface number of your local host. Now it will be possible to ping nodes in the G3 network as shown in Figure 16-6:

```
C:\>ping -6 FD00:0:0:08E4:1122:3344:5566:7789
```

With this last step, you will be able to ping nodes in the remote PLC network through the G3 Gateway Coordinator. The environment is ready.

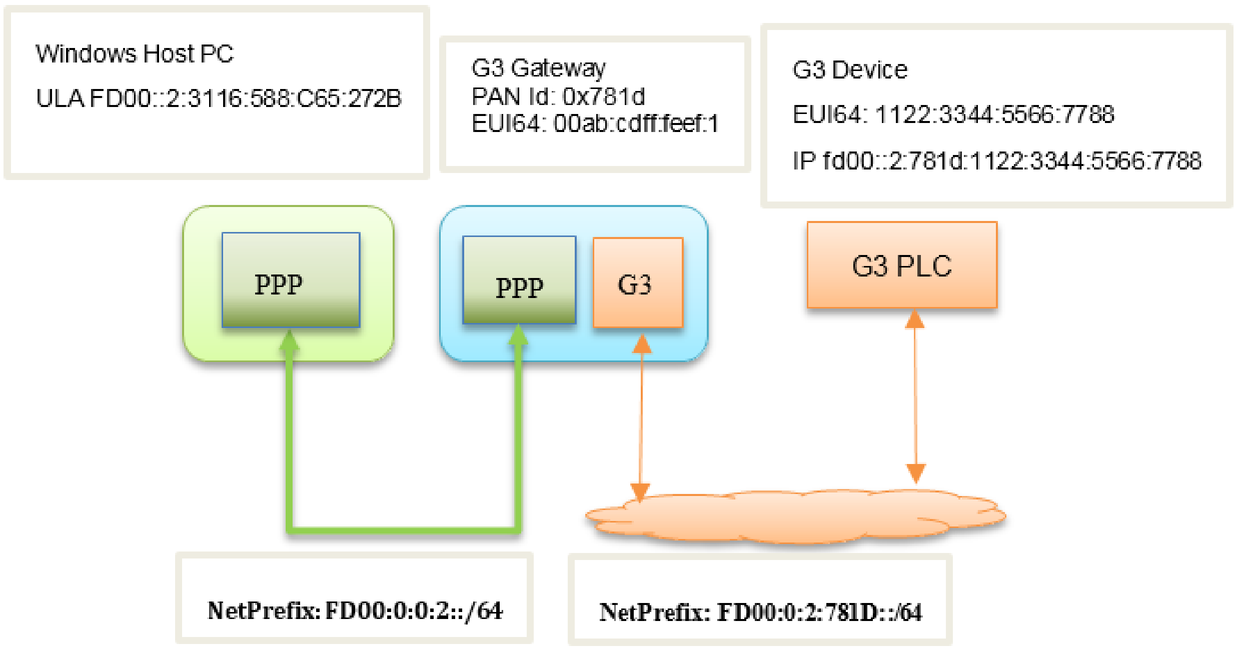
16.2 PPP G3 Gateway Coordinator Lite

The figure below displays the environment needed to use a G3 Gateway Coordinator through a PPP connection. It shows two different IPv6 networks: the G3-PLC network (orange), and the PPP network (green). The G3 Gateway Coordinator Lite serializes the IPv6 protocol over a serial, null-modem connection. This way it is possible to connect a G3 Coordinator to a host PC directly, having access to the PLC network at the IPv6 level.

The PPP connection is much slower than the Ethernet, but PLC networks are even slower; therefore, there is not a big drawback to connecting to a Gateway through a PPP link. The biggest disadvantage is that the configuration is a bit harder:

1. Create a null-modem device.
2. Configure IPv6 protocol.
3. Configure PPP.
4. Configure IPv6 routing.

Figure 16-12. PPP G3 Gateway Setup


Windows Host PC
ULA FD00::2:3116:588:C65:272B
G3 Gateway
PAN Id: 0x781d
EUI64: 00ab:cdff:feef:1
G3 Device
EUI64: 1122:3344:5566:7788
IP fd00::2:781d:1122:3344:5566:7788
PPP
PPP
G3
G3 PLC
NetPrefix: FD00:0:0:2::/64
NetPrefix: FD00:0:2:781D::/64

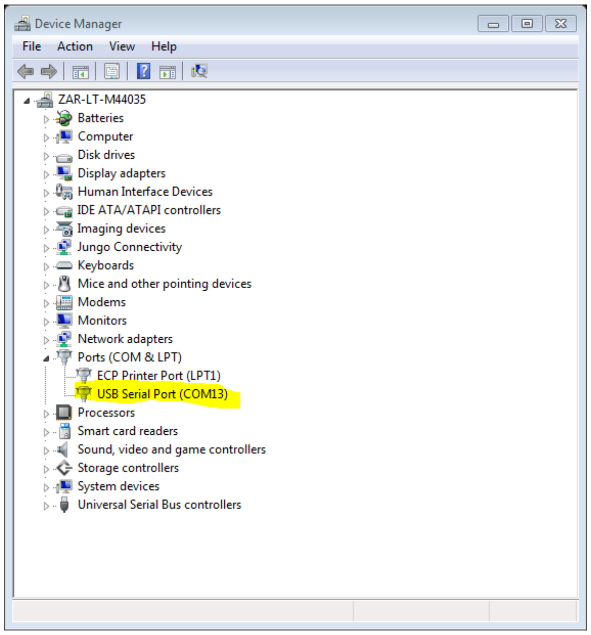
16.2.1 Setting Up a Null-Modem Interface in a Windows Host Computer

Here, a Microchip PL360MB board will be used as a G3 Gateway Coordinator Lite, which uses a certain Serial Port Driver. Other platforms will have a very similar configuration with only minor differences on the host PC due to the host serial port driver.

The PPP interface is configured by default to use the UART0 of the board. It is connected through a USB serial adapter that Windows is able to recognize as a virtual COM port.

1. Install the Microchip USB MCP2221 Windows serial driver.
2. Open the Windows Device Manager, then expand the “Port” tree and plug your Gateway board into your Windows computer. A new serial port will appear.

Figure 16-13. Windows Device Manager

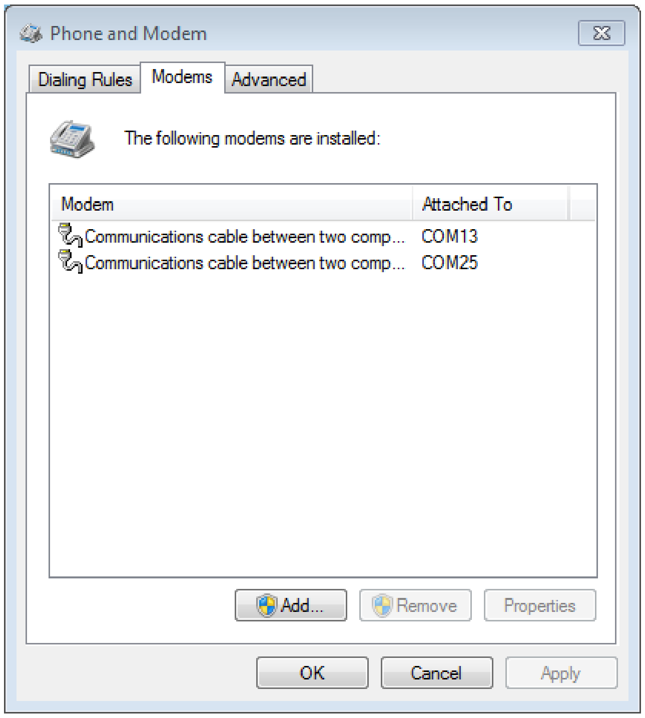


ZAR-LT-M44035
- Batteries
- Computer
- Disk drives
- Display adapters
- Human Interface Devices
- IDE ATA/ATAPI controllers
- Imaging devices
- Jungo Connectivity
- Keyboards
- Mice and other pointing devices
- Modems
- Monitors
- Network adapters
- Ports (COM & LPT)
  - ECP Printer Port (LPT1)
  - USB Serial Port (COM13)
- Processors
- Smart card readers
- Sound, video and game controllers
- Storage controllers
- System devices
- Universal Serial Bus controllers

In this case, COM13 will be used for the null-modem.

3. Open the Phone and Modem window in the Control Panel. Select the **Modems** tab.

Figure 16-14. Phone and Modem Window

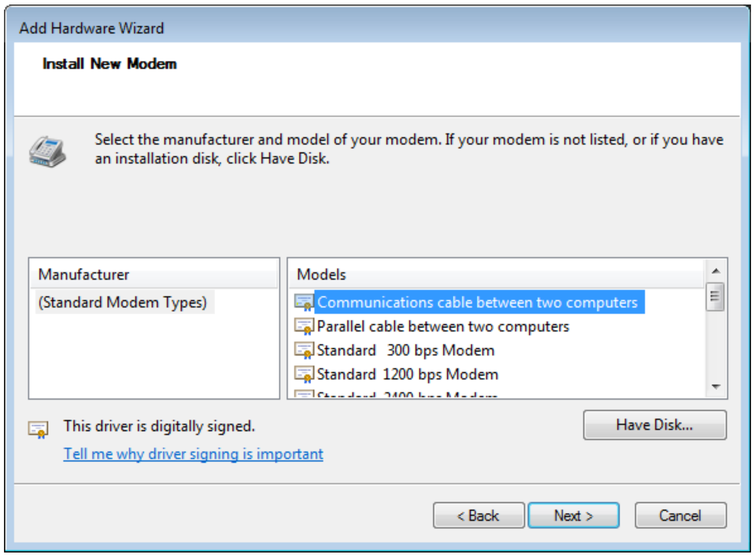


The following modems are installed:

Modem	Attached To
Communications cable between two comp...	COM13
Communications cable between two comp...	COM25

4. Add a new modem. Check **Don't detect my modem, I will select it from a list**, then click **Next**.
5. Under Manufacturer, select **Standard Modem Types**.
6. Under Model, select **Communications cable between two computers**.
7. Click **Next**.

Figure 16-15. Hardware Wizard: Add New Modem



Install New Modem

Select the manufacturer and model of your modem. If your modem is not listed, or if you have an installation disk, click Have Disk.

Manufacturer	Models
(Standard Modem Types)	Communications cable between two computers
	Parallel cable between two computers
	Standard 300 bps Modem
	Standard 1200 bps Modem

This driver is digitally signed.

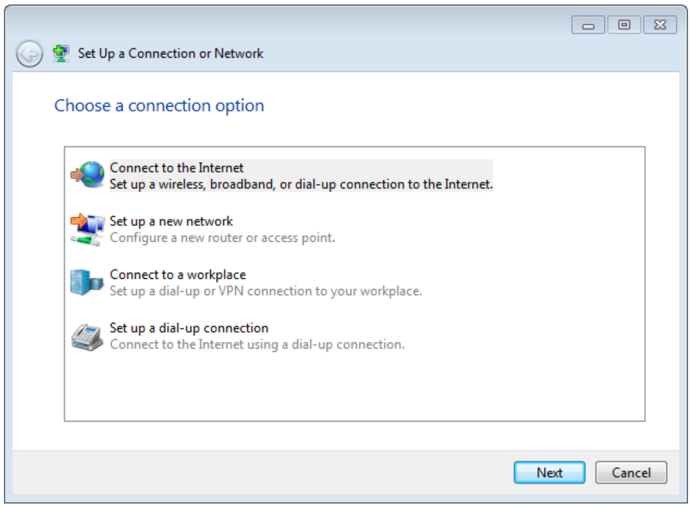
8. **Select your COM** (COM13 in this case) port , then click **Next**, then click **Finish**.

Important: Some Windows versions misbehave configuring the network speed. Use the option "Select all ports" if this happens.

We have installed a modem device correctly. Now, we will setup the PPP connection.

9. Start the **Network and Sharing Center** from the **Control Panel**.
10. Click **Set up a new connection or network**.

Figure 16-16. Set Up Connection Wizard



Set Up a Connection or Network

Choose a connection option

Connect to the Internet
Set up a wireless, broadband, or dial-up connection to the Internet.

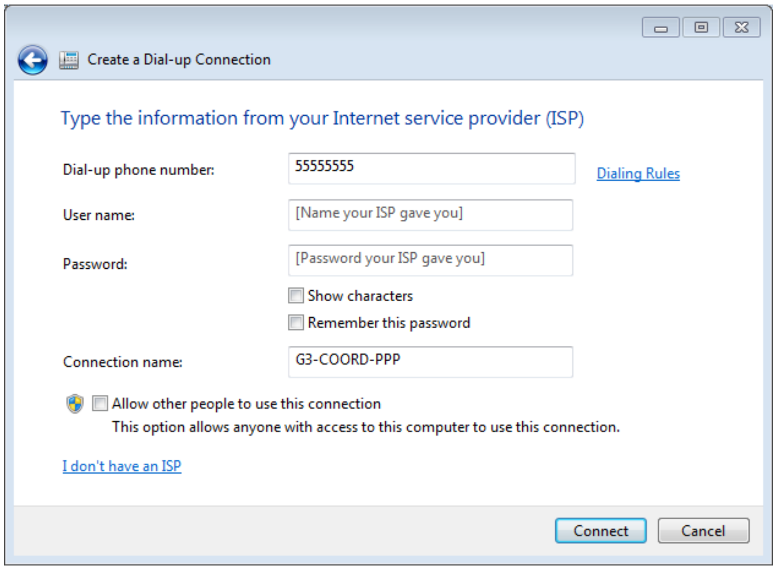
Set up a new network
Configure a new router or access point.

Connect to a workplace
Set up a dial-up or VPN connection to your workplace.

Set up a dial-up connection
Connect to the Internet using a dial-up connection.

11. Select a **Set up a dial-up connection**.
12. Select **Communications cable between two computers**.
13. Now, enter a dummy number in the Dial-up phone number field. It will not be used. Choose a **Connection name**, for example, "G3-COORD-PPP".

Figure 16-17. Dial-up Connection Settings



Create a Dial-up Connection

Type the information from your Internet service provider (ISP)

Dial-up phone number: 55555555

Dialing Rules

User name: [Name your ISP gave you]

Password: [Password your ISP gave you]

Show characters

Remember this password

Connection name: G3-COORD-PPP

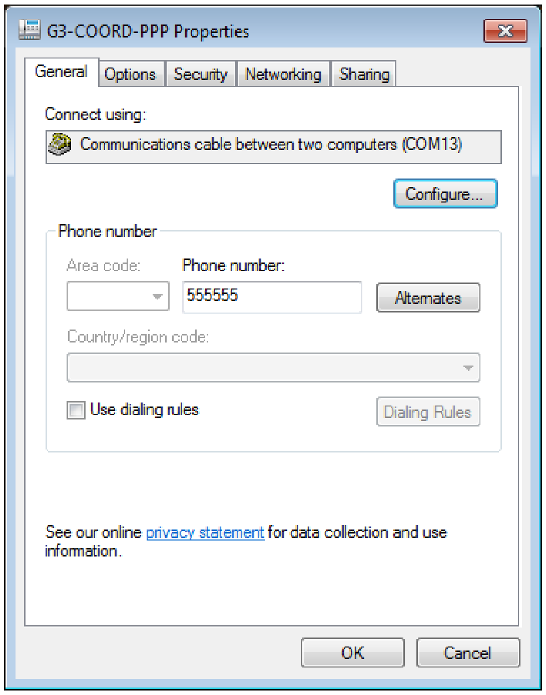
Allow other people to use this connection
This option allows anyone with access to this computer to use this connection.

I don't have an ISP

14. Click Connect, then **Skip**.
15. Now you have to configure the connection.
16. Start the **Network and Sharing Center** from the **Control Panel**.
17. Select **Change Adapter** Settings from the left hand side column menu.

18. Right click on the connection **G3-COORD-PPP** and click Properties.

Figure 16-18. Null Modem Interface Properties


G3-COORD-PPP Properties

Connect using:
Communications cable between two computers (COM13)

Phone number
Area code:
Phone number: 555555

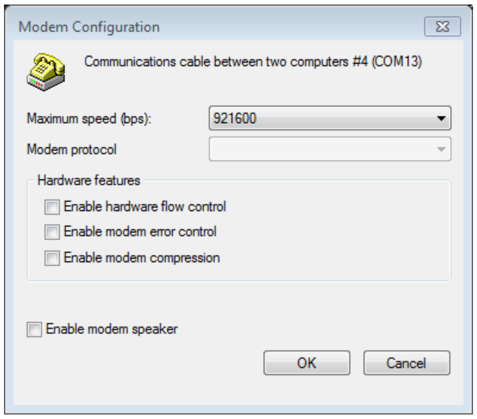
Country/region code:

Use dialing rules

See our online privacy statement for data collection and use information.

19. Select **Configure**.

Figure 16-19. Modem Configuration


Modem Configuration

Communications cable between two computers #4 (COM13)

Maximum speed (bps): 921600
Modem protocol

Hardware features
Enable hardware flow control
Enable modem error control
Enable modem compression

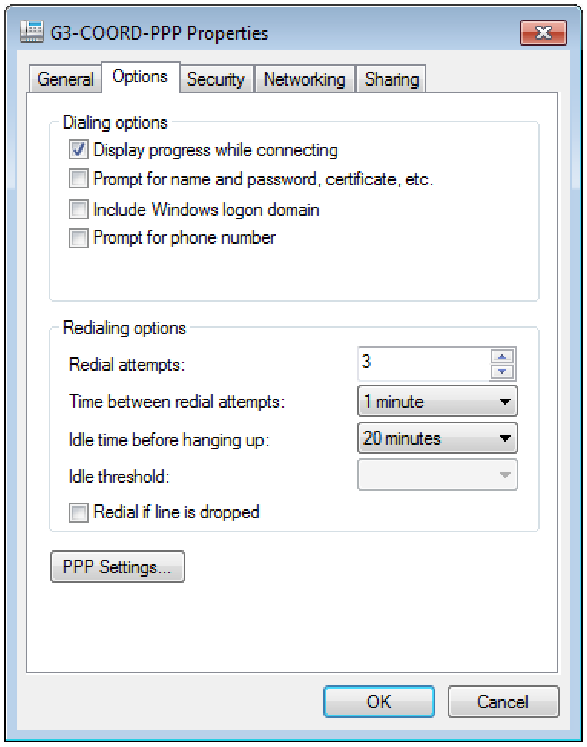
Enable modem speaker

Select the Maximum speed to **921600** bps. Disable the **flow control**, **error control** and **compression** check boxes. Click OK.

Important: Verify that Modem Options are configured correctly. Some Microsoft Windows versions misbehave when saving the values configured in this window. If speed and flow control are not correctly set up, the PPP link will not work.

20. Select the **Options** tab.

Figure 16-20. Modem Options



Dialing options

- [✓] Display progress while connecting
- [ ] Prompt for name and password, certificate, etc.
- [ ] Include Windows logon domain
- [ ] Prompt for phone number

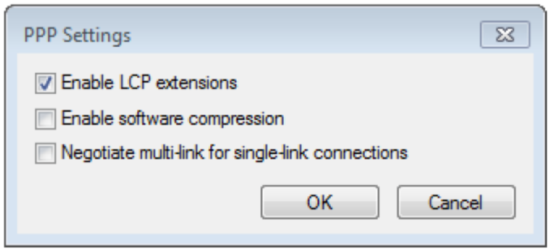
Redialing options

Redial attempts:	3
Time between redial attempts:	1 minute
Idle time before hanging up:	20 minutes
Idle threshold:	

- [ ] Redial if line is dropped

21. Check **Display progress while connecting** and uncheck the other options.
22. Select **PPP Settings**.

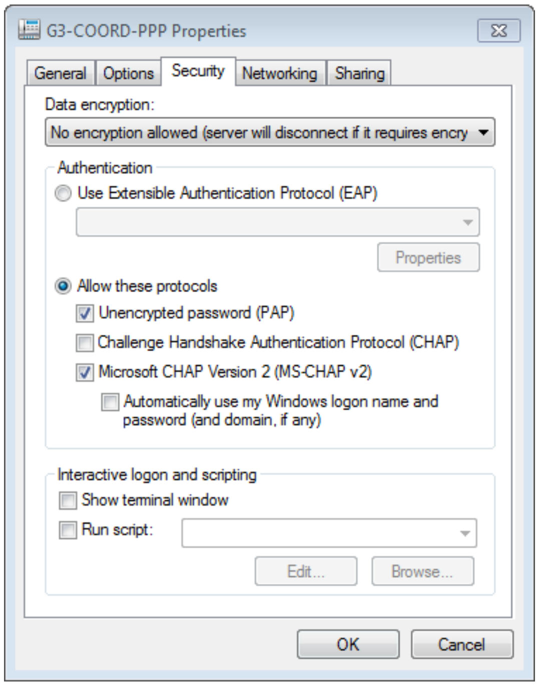
Figure 16-21. Modem PPP Settings



- [✓] Enable LCP extensions
- [ ] Enable software compression
- [ ] Negotiate multi-link for single-link connections

23. Select **Enable LCP extensions** and disable the software compression and multi-link options. Click OK.
24. Select the **Security** tab.

Figure 16-22. Modem Security Properties



Data encryption: No encryption allowed (server will disconnect if it requires encry

Authentication

Use Extensible Authentication Protocol (EAP)

[✓ Allow these protocols]

[✓] Unencrypted password (PAP)

[ ] Challenge Handshake Authentication Protocol (CHAP)

[✓] Microsoft CHAP Version 2 (MS-CHAP v2)

[ ] Automatically use my Windows logon name and password (and domain, if any)

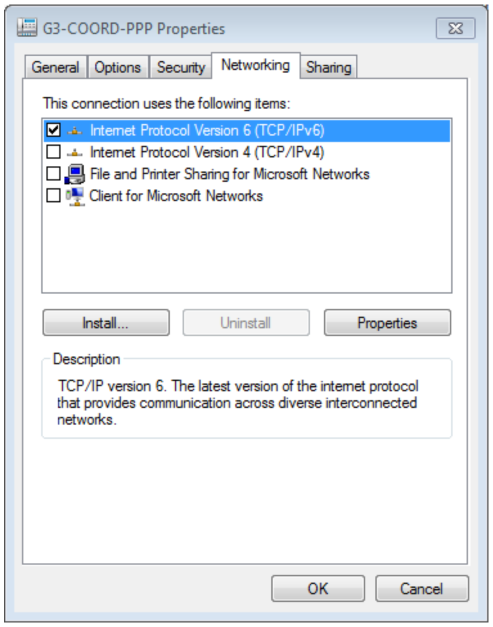
Interactive logon and scripting

[ ] Show terminal window

[ ] Run script:

25. Under **Data encryption**, select **No encryption allowed**. Click OK.
26. Select the **Networking** tab.

Figure 16-23. Modem Networking Properties



This connection uses the following items:

[✓] Internet Protocol Version 6 (TCP/IPv6)

[ ] Internet Protocol Version 4 (TCP/IPv4)

[ ] File and Printer Sharing for Microsoft Networks

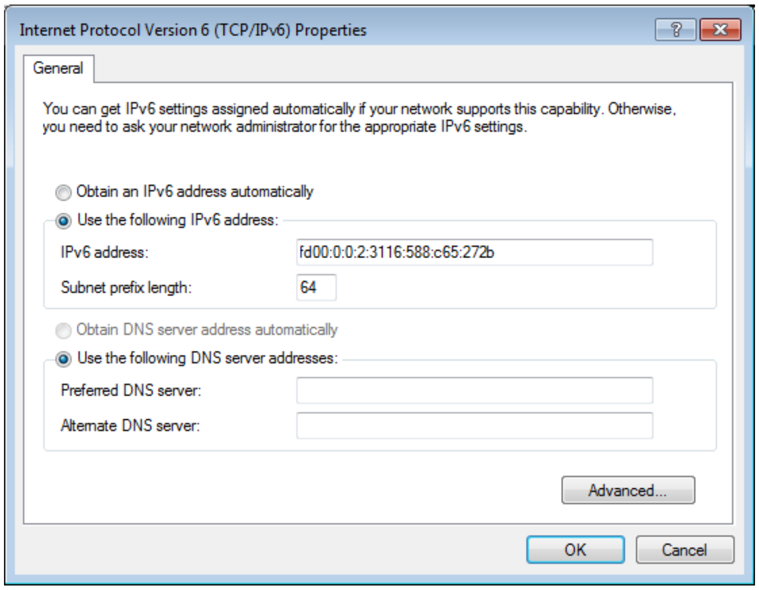
[ ] Client for Microsoft Networks

Description

TCP/IP version 6. The latest version of the internet protocol that provides communication across diverse interconnected networks.

27. Check **Internet Protocol Version 6** and uncheck the other options.
28. Select **Internet Protocol Version 6**, then click Properties.
29. Select **Use the following IPv6 address**. Configure the IPv6 address for this interface, in this example, *FD00::2:<Your EUI64>*. You can find out your EUI64 as explained in Host IPv6 Configuration. Leave the DNS options empty. By default, G3 Gateway Coordinator Lite is configured to use the network prefix “FD00::2::/64”; therefore, the host must use the same network prefix.

Figure 16-24. Modem IPv6 Configuration



Internet Protocol Version 6 (TCP/IPv6) Properties

General

You can get IPv6 settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IPv6 settings.

○ Obtain an IPv6 address automatically
● Use the following IPv6 address:

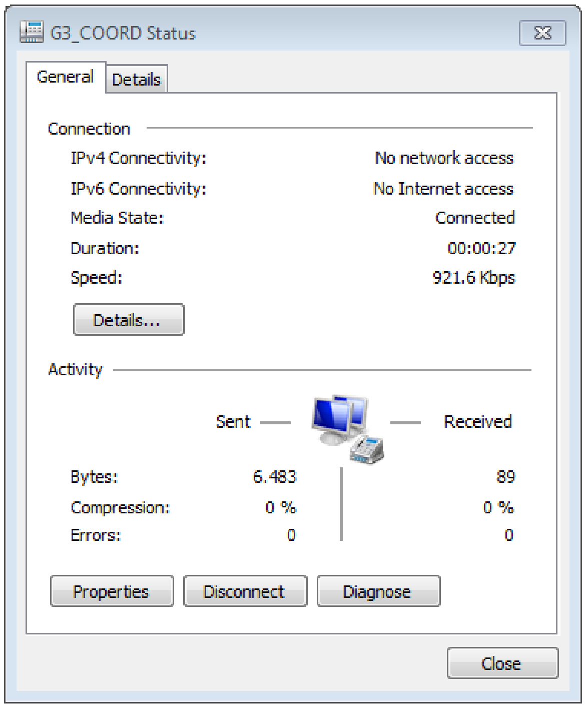
IPv6 address: fd00:0:0:2:3116:588:c65:272b
Subnet prefix length: 64

○ Obtain DNS server address automatically
● Use the following DNS server addresses:

Preferred DNS server:
Alternate DNS server:

30. Click OK. Close the **G3-COORD-PPP Properties** window. The network interface is ready.
31. Double click on the network interface. It will start the dial-up process. If everything is OK, right click on the G3-COORD-PPP connection and choose status:

Figure 16-25. Modem Status: Connected



G3_COORD Status

General | Details

Connection

IPv4 Connectivity:	No network access
IPv6 Connectivity:	No Internet access
Media State:	Connected
Duration:	00:00:27
Speed:	921.6 Kbps

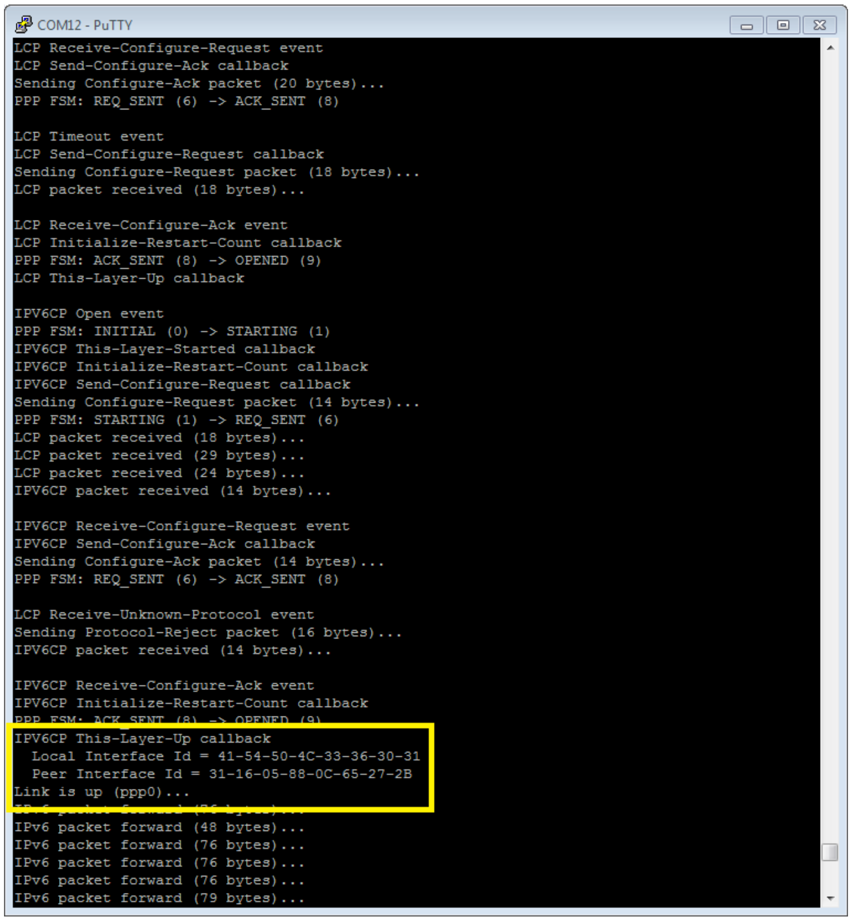
Activity

	Sent	Received
Bytes:	6.483	89
Compression:	0 %	0 %
Errors:	0	0

The figure below shows the output console from the G3 Gateway Coordinator Lite with the initial PPP/LCP protocols handshake between the host and the gateway. The gateway device executes the role of a PPP server, whereas, the host (Windows null-modem connection) is a PPP client.

At the end of the handshake, the host and gateway share the IPv6 information, such as the EUI64, needed to identify themselves at the IP level (figure below, yellow box). After the message “Link is up (ppp0)…”, the link switches to “interface mode”, and at that moment, the PPP interfaces in both the host and gateway are ready to perform communications as a regular network interface.

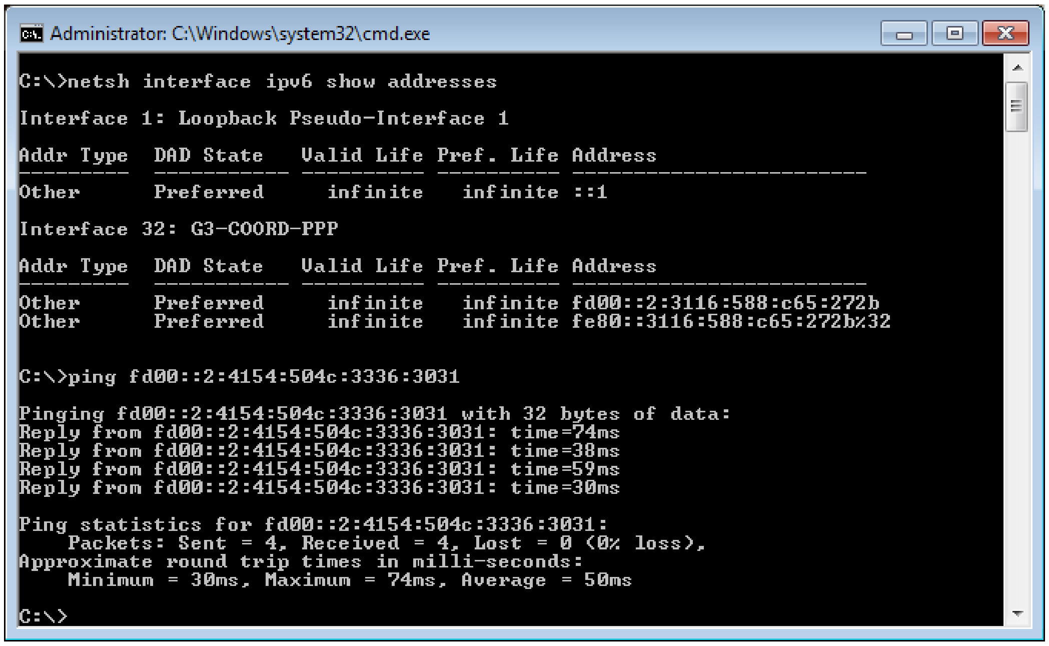
Figure 16-26. Gateway Coordinator Lite Console: PPP/LCP Protocol Completion



```
LCP Receive-Configure-Request event
LCP Send-Configure-Ack callback
Sending Configure-Ack packet (20 bytes)...
PPP FSM: REQ_SENT (6) -> ACK_SENT (8)

LCP Timeout event
LCP Send-Configure-Request callback
Sending Configure-Request packet (18 bytes)...
LCP packet received (18 bytes)...

LCP Receive-Configure-Ack event
LCP Initialize-Restart-Count callback
PPP FSM: ACK_SENT (8) -> OPENED (9)
LCP This-Layer-Up callback

IPV6CP Open event
PPP FSM: INITIAL (0) -> STARTING (1)
IPV6CP This-Layer-Started callback
IPV6CP Initialize-Restart-Count callback
IPV6CP Send-Configure-Request callback
Sending Configure-Request packet (14 bytes)...
PPP FSM: STARTING (1) -> REQ_SENT (6)
LCP packet received (18 bytes)...
LCP packet received (29 bytes)...
LCP packet received (24 bytes)...
IPV6CP packet received (14 bytes)...

IPV6CP Receive-Configure-Request event
IPV6CP Send-Configure-Ack callback
Sending Configure-Ack packet (14 bytes)...
PPP FSM: REQ_SENT (6) -> ACK_SENT (8)

LCP Receive-Unknown-Protocol event
Sending Protocol-Reject packet (16 bytes)...
IPV6CP packet received (14 bytes)...

IPV6CP Receive-Configure-Ack event
IPV6CP Initialize-Restart-Count callback
PPP FSM: ACK_SENT (8) -> OPENED (9)
IPV6CP This-Layer-Up callback
  Local Interface Id = 41-54-50-4C-33-36-30-31
  Peer Interface Id = 31-16-05-88-0C-65-27-2B
Link is up (ppp0)...
IPv6 packet forward (76 bytes)...
IPv6 packet forward (48 bytes)...
IPv6 packet forward (76 bytes)...
IPv6 packet forward (76 bytes)...
IPv6 packet forward (76 bytes)...
IPv6 packet forward (79 bytes)...
```

Now host and gateway can communicate at the IP level. The host is able to ping the remote gateway through the PPP interface. The figure below shows the host network status and a ping to the remote gateway. In the Gateway console log, the EUI64 associated to the gateway is visible (Local Interface Id, figure above, yellow box).

Figure 16-27. Host PPP Network Status and Remote Ping



Administrator: C:\Windows\system32\cmd.exe

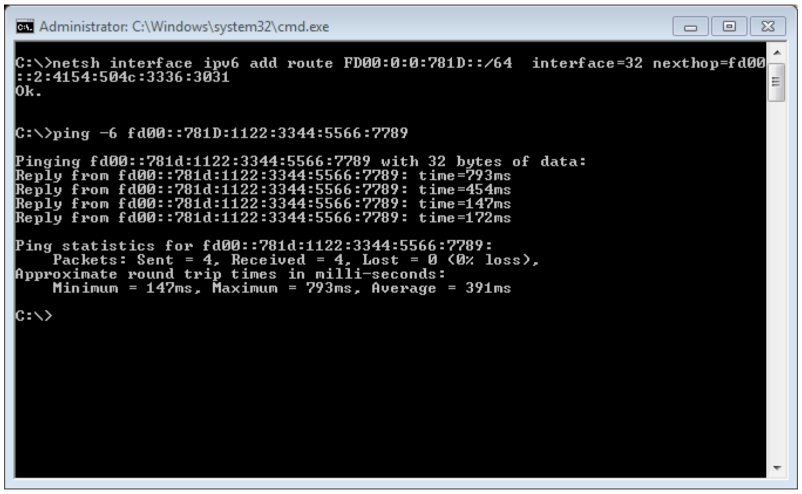
```
C:\>netsh interface ipv6 show addresses

Interface 1: Loopback Pseudo-Interface 1

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Preferred     infinite   infinite ::1

Interface 32: G3-COORD-PPP

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- ------------------------
Other      Preferred     infinite   infinite fd00::2:3116:588:c65:272b
Other      Preferred     infinite   infinite fe80::3116:588:c65:272b%32


C:\>ping fd00::2:4154:504c:3336:3031

Pinging fd00::2:4154:504c:3336:3031 with 32 bytes of data:
Reply from fd00::2:4154:504c:3336:3031: time=74ms
Reply from fd00::2:4154:504c:3336:3031: time=38ms
Reply from fd00::2:4154:504c:3336:3031: time=59ms
Reply from fd00::2:4154:504c:3336:3031: time=30ms

Ping statistics for fd00::2:4154:504c:3336:3031:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 30ms, Maximum = 74ms, Average = 50ms

C:\>
```

In the figure above, the first command, *“netsh interface ipv6 show addresses”*, shows the information for the G3-COORD-PPP network interface, like its interface number (32 in this case) and its IPv6 addresses.

Finally, a route needs to be set up for the remote G3 network through the new interface. The remote G3 PLC network is configured by default to use the PANID 0x781D. Nodes in the network will have a ULA address in the network *FD00:0:2:781D::/64*. The figure below shows the command to configure the route in this case.

```
C:\>netsh interface ipv6 add route FD00:0:2:781D::/64 interface=<G3-Coord-PPP>
nexthop=FD00:0:0:2:<GW EUI64>
```

Figure 16-28. Host Remote PLC Network Route Configuration



Administrator: C:\Windows\system32\cmd.exe

```
C:\>netsh interface ipv6 add route FD00:0:0:781D::/64  interface=32 nexthop=fd00
::2:4154:504c:3336:3031
Ok.


C:\>ping -6 fd00::781D:1122:3344:5566:7789

Pinging fd00::781d:1122:3344:5566:7789 with 32 bytes of data:
Reply from fd00::781d:1122:3344:5566:7789: time=793ms
Reply from fd00::781d:1122:3344:5566:7789: time=454ms
Reply from fd00::781d:1122:3344:5566:7789: time=147ms
Reply from fd00::781d:1122:3344:5566:7789: time=172ms

Ping statistics for fd00::781d:1122:3344:5566:7789:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 147ms, Maximum = 793ms, Average = 391ms

C:\>
```

Once the route to the remote PC is configured correctly, then the host PC will be able to ping devices on the PLC network, as shown in the figure above.

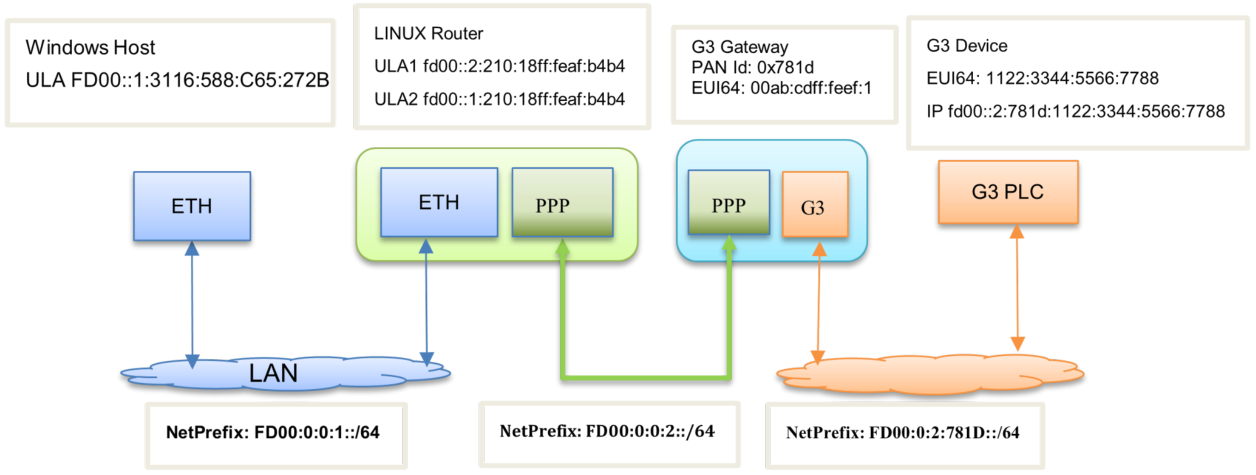
If the host can ping the G3 Gateway and the nodes in the PLC network, the system is ready.

16.2.2 Setting Up a Null-Modem Interface in a Linux Host

Microsoft Windows® Operating System (OS) behavior can be erratic when configuring the null-modem interface. Depending on your OS version, updates, firewall, antivirus configuration, etc., it might not let you configure your system properly. It is possible to connect your PPP G3 Gateway Coordinator Lite to a Linux box and make it a router for your local network.

The figure below shows this configuration. The Linux router must have two ULA addresses, one on each network that it has to connect, in this example case, the PPP network (FD00::2::/64) and the local Ethernet (FD00::1::/64). Also routes must be configured accordingly. But first, the PPP connection must be configured.

Figure 16-29. Linux Intermediate Router Configuration


Windows Host
ULA FD00::1:3116:588:C65:272B
LINUX Router
ULA1 fd00::2:210:18ff:feaf:b4b4
ULA2 fd00::1:210:18ff:feaf:b4b4
G3 Gateway
PAN Id: 0x781d
EUI64: 00ab:cdff:feef:1
G3 Device
EUI64: 1122:3344:5566:7788
IP fd00::2:781d:1122:3344:5566:7788
ETH
ETH
PPP
PPP
G3
G3 PLC
LAN
NetPrefix: FD00:0:0:1::/64
NetPrefix: FD00:0:0:2::/64
NetPrefix: FD00:0:2:781D::/64

16.2.2.1 Configure the Linux PPP Connection

First, verify that your Linux desktop has the PPPD package installed. Otherwise, download and install it with your package manager. Configuration parameters should work in other Linux distributions without any problem. Here, commands will be shown in the Debian/Ubuntu fashion.

```
$ sudo apt-get install pppd
```

Once PPPD is installed on your system, proceed with its configuration by doing the following:

1. Create a new configuration file in /etc/ppp/peers/ with the options below:

```
$ sudo vi /etc/ppp/peers/g3-coord-lite
# daemon
nodetach
debug
nopersist
noauth
noip
lcp-echo-interval 500
lcp-echo-adaptive

# compress
nodeflate
noccp
nopcomp

# common options
lock
nocrtscts
local
+ipv6

#null-modem call
connect 'chat TIMEOUT 20000 " " CLIENT CLIENTSERVER'
```

2. Create a new script in /etc/ppp/ipv6-up.d/g3-coord-up with the IPv6 and routing configuration:

```
$ sudo vi /etc/ppp/ipv6-up.d/g3-coord-up
#Add Unique Local Address to this interface
ip address add fd00::2:<your-eui64> dev ppp0
#Route to the G3-PLC network
```

```
    ip route add fd00:0:2:<your-panid>::/64 dev ppp0
    #Local default route
    ip route add fd00:0:0:2::/64 dev ppp0
```

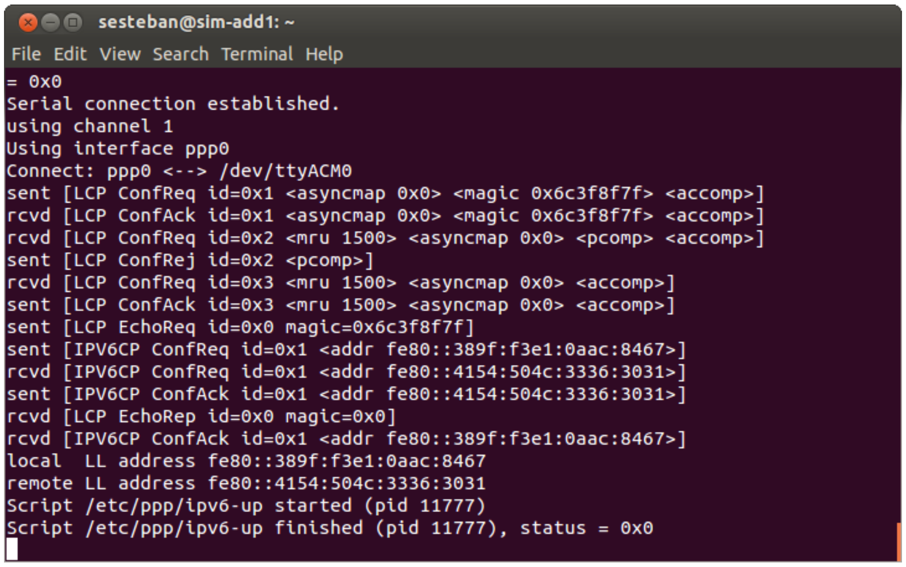
3. Give execution permissions to the new script:

```
    $ sudo chmod a+x /etc/ppp/ipv6-up.d/g3-coord-up
```

4. Now connect your USB cable to your Linux board.
5. Find out the USB device assigned to your Gateway Coordinator Lite device. Use the dmesg command to do so. Typically, it will be assigned to /dev/ttyACM0.
6. Now with everything configured and ready, execute the command *pppd* to bring up the network interface with the script name option and the tty name/speed configuration:

```
    $ sudo pppd call g3-coord-lite /dev/ttyACM0 921600
```

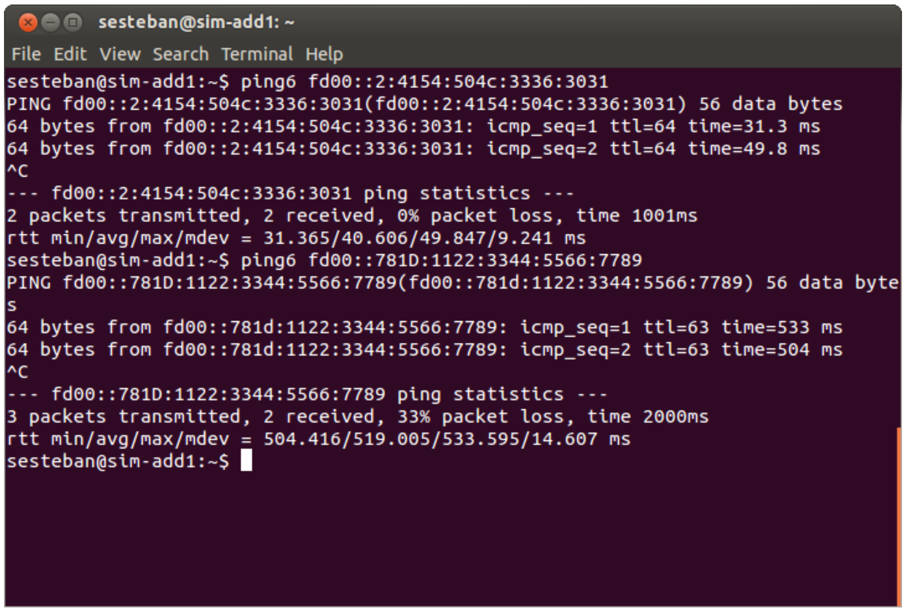
Figure 16-30. PPPd Command Output Showing LCP Protocol Handshake



```
= 0x0
Serial connection established.
using channel 1
Using interface ppp0
Connect: ppp0 <--> /dev/ttyACM0
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <magic 0x6c3f8f7f> <accomp>]
rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <magic 0x6c3f8f7f> <accomp>]
rcvd [LCP ConfReq id=0x2 <mru 1500> <asyncmap 0x0> <pcomp> <accomp>]
sent [LCP ConfRej id=0x2 <pcomp>]
rcvd [LCP ConfReq id=0x3 <mru 1500> <asyncmap 0x0> <accomp>]
sent [LCP ConfAck id=0x3 <mru 1500> <asyncmap 0x0> <accomp>]
sent [LCP EchoReq id=0x0 magic=0x6c3f8f7f]
sent [IPV6CP ConfReq id=0x1 <addr fe80::389f:f3e1:0aac:8467>]
rcvd [IPV6CP ConfReq id=0x1 <addr fe80::4154:504c:3336:3031>]
sent [IPV6CP ConfAck id=0x1 <addr fe80::4154:504c:3336:3031>]
rcvd [LCP EchoRep id=0x0 magic=0x0]
rcvd [IPV6CP ConfAck id=0x1 <addr fe80::389f:f3e1:0aac:8467>]
local  LL address fe80::389f:f3e1:0aac:8467
remote LL address fe80::4154:504c:3336:3031
Script /etc/ppp/ipv6-up started (pid 11777)
Script /etc/ppp/ipv6-up finished (pid 11777), status = 0x0
```

Now the interface is up and running. The Gateway Coordinator and devices on the G3-PLC network can be accessed at the IPv6 level.

Figure 16-31. Ping Gateway and Remote G3-PLC Device Through PPP (Linux Host)



```
sesteban@sim-add1:~$ ping6 fd00::2:4154:504c:3336:3031
PING fd00::2:4154:504c:3336:3031(fd00::2:4154:504c:3336:3031) 56 data bytes
64 bytes from fd00::2:4154:504c:3336:3031: icmp_seq=1 ttl=64 time=31.3 ms
64 bytes from fd00::2:4154:504c:3336:3031: icmp_seq=2 ttl=64 time=49.8 ms
^C
--- fd00::2:4154:504c:3336:3031 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 31.365/40.606/49.847/9.241 ms
sesteban@sim-add1:~$ ping6 fd00::781D:1122:3344:5566:7789
PING fd00::781D:1122:3344:5566:7789(fd00::781d:1122:3344:5566:7789) 56 data byte
s
64 bytes from fd00::781d:1122:3344:5566:7789: icmp_seq=1 ttl=63 time=533 ms
64 bytes from fd00::781d:1122:3344:5566:7789: icmp_seq=2 ttl=63 time=504 ms
^C
--- fd00::781D:1122:3344:5566:7789 ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2000ms
rtt min/avg/max/mdev = 504.416/519.005/533.595/14.607 ms
sesteban@sim-add1:~$
```

16.2.2.2 Set Up LAN Network Routing

Now that the Linux host is able to communicate with the Gateway and the PLC network, it is time to set up the communications between the Windows host and the G3 Gateway coordinator. First, verify that the Linux computer has the routing capabilities enabled. Execute the command sysctl as shown:

```
$ sysctl net.ipv6.conf.all.forwarding
net.ipv6.conf.all.forwarding = 0
```

The kernel configuration variable must be set to “1” in order to enable IPv6 capabilities:

```
$ sudo sysctl -w net.ipv6.conf.all.forwarding=1
net.ipv6.conf.all.forwarding = 1
```

Assuming that the LAN router does not have a working IPv6 router, a pair of IPv6 addresses will need to be set up to the Windows and Linux hosts on the network FD00::1::/64. On the Linux host, the command is:

```
$ sudo ip address add FD00::1:<Eth0_EUI64>/64 dev eth0
```

And on the Windows host (within a command console *cmd.exe*):

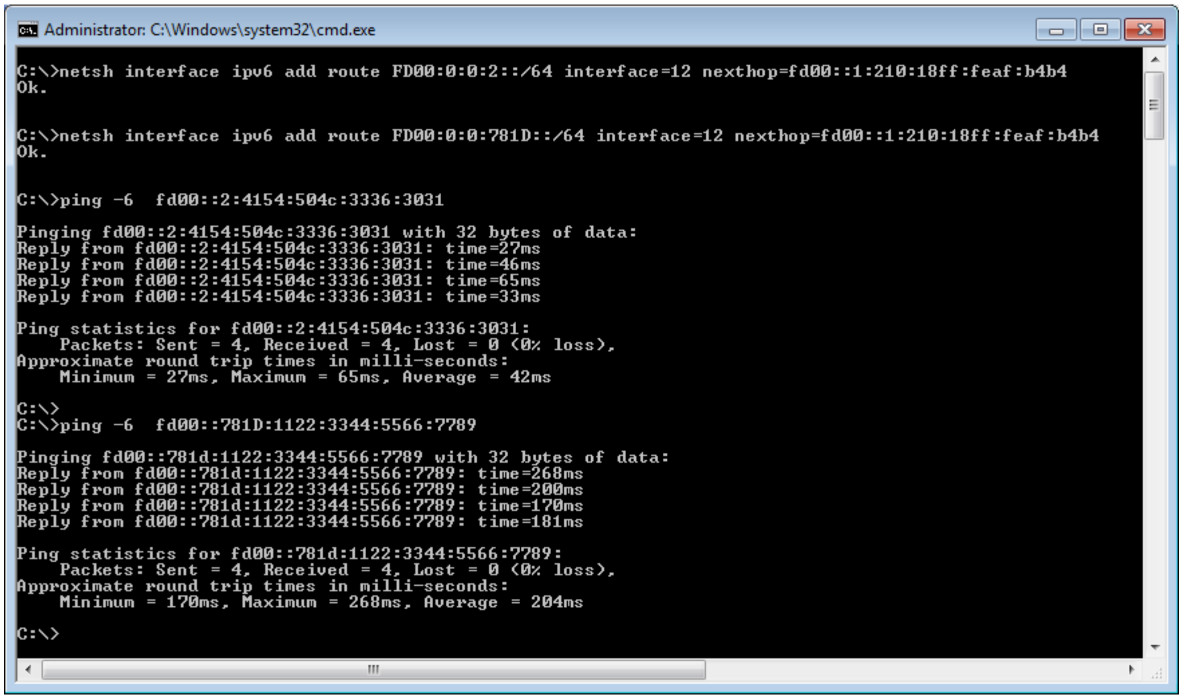
```
C:\>netsh interface ipv6 add address interface=<your interface id>
address=”FD00::1:<Your EUI64>
```

Then, we will need to manually set up the routing in the Windows host; in this case, two new routes need to be added. The first route sets the path to the FD00::2::/64 network (and to reach the G3 Gateway coordinator), and a second route to reach the G3 PLC network:

```
C:\>netsh interface ipv6 add route FD00:0:0:2::/64 interface=<your id>
nexthop=<FD00:0:0:1:<Linux Host>

C:\>netsh interface ipv6 add route FD00:0:2:781D::/64 interface=<your id>
nexthop=<FD00:0:0:1:<Linux Host>
```

The figure below shows the configuration commands for our environment and the ping results to the G3 Gateway Coordinator and to the G3-PLC network. This completes the system configuration defined in 16.2.2. Setting Up a Null-Modem Interface in a Linux Host.

Figure 16-32. Windows to Linux, to PPP Routing



Administrator: C:\Windows\system32\cmd.exe

```
C:\>netsh interface ipv6 add route FD00:0:0:2::/64 interface=12 nexthop=fd00::1:210:18ff:feaf:b4b4
Ok.


C:\>netsh interface ipv6 add route FD00:0:0:781D::/64 interface=12 nexthop=fd00::1:210:18ff:feaf:b4b4
Ok.


C:\>ping -6  fd00::2:4154:504c:3336:3031

Pinging fd00::2:4154:504c:3336:3031 with 32 bytes of data:
Reply from fd00::2:4154:504c:3336:3031: time=27ms
Reply from fd00::2:4154:504c:3336:3031: time=46ms
Reply from fd00::2:4154:504c:3336:3031: time=65ms
Reply from fd00::2:4154:504c:3336:3031: time=33ms

Ping statistics for fd00::2:4154:504c:3336:3031:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 27ms, Maximum = 65ms, Average = 42ms

C:\>
C:\>ping -6  fd00::781D:1122:3344:5566:7789

Pinging fd00::781d:1122:3344:5566:7789 with 32 bytes of data:
Reply from fd00::781d:1122:3344:5566:7789: time=268ms
Reply from fd00::781d:1122:3344:5566:7789: time=200ms
Reply from fd00::781d:1122:3344:5566:7789: time=170ms
Reply from fd00::781d:1122:3344:5566:7789: time=181ms

Ping statistics for fd00::781d:1122:3344:5566:7789:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 170ms, Maximum = 268ms, Average = 204ms

C:\>
```

17. Abbreviations

ADP	Adaptation Sublayer
ADPD	Adaptation Sublayer Data Service
ADPIB	Adaptation Sublayer Information Base
ADPM	Adaptation Sublayer Management Service
API	Application Programming Interface
APP	Application
ASF	Advanced Software Framework
FCS	Frame Check Sequence
FW	Firmware
HAL	Hardware Abstraction Layer
MAC	Medium Access Control
MAC RT	MAC Real Time
MCPS	MAC Common Part Sub-layer
MCU	Microcontroller Unit
MIB	MAC Information Base
MIC	Message Integrity Code
MLME	MAC Sub-layer Management Entity
MSDU	MAC Service Data Unit
OSS	Operating System Support
PAL	PHY Abstraction Layer
PAN	Personal Area Network
PDU	Physical layer Data Unit
PIB	Protocol Information Base
PLME	Physical Layer Management Entity
POS	Personal Operating Space
SPI	Serial Peripheral Interface
TRX	Transceiver

18. References

1. Smart Energy Products: http://www.microchip.com/design-centers/smart-energy-products/overview
2. Power Line Communications: http://www.microchip.com/design-centers/smart-energy-products/power-line-communications/overview
3. Microchip Design Support: http://www.microchip.com/support/hottopics.aspx
4. ITU G.9903: https://www.itu.int/rec/T-REC-G.9903/en
5. ITU G.9901: http://www.itu.int/rec/T-REC-G.9901/en
6. G3-PLC Alliance: http://www.g3-plc.com/
7. Advanced Software Framework: http://www.microchip.com/development-tools/atmel-studio-7/advanced-software-framework-(asf)
8. Microchip Studio: www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices

19. Revision History

Note: This document starts from G3-PLC Firmware Stack rev. C (04/2020).

19.1 Rev A - 11/2021

1. General Architecture	Added/Modified sections to reflect Hybrid Profile.
2.1. G3-PLC Workspace Structure	Added/Modified sections to reflect Hybrid Profile.
3.3. Configuration Files	Added and described new files. Review conf_clock.h and conf_pplc_if.h files description.
5. Example Applications	Added Meter/Modem Apps description. Added RF PHY Examples.
5.4. Clock Configuration Example for Low Requirements Module	Added this section.
6.7. Voltage and Thermal Monitor for PLC Driver	Added this section.
9. API of DATA LINK Layer	Added MAC SAP and MAC Primitives chapters for RF MAC. Added RF MAC MIB Objects definition. Introduced Hybrid Abstraction Layer. Updated ADPM-DISCOVERY.indication, ADPM-NETWORK-JOIN.request and ADPM-NETWORK-STATUS.indication to reflect changes introduced on Hybrid Profile. Added New ADP IB Objects definition.
9.7. MIB Objects Specification and Access	Added MAC_WRP_PIB_MANUF_ACK_TX_DELAY_RF, MAC_WRP_PIB_MANUF_ACK_RX_WAIT_TIME_RF, MAC_WRP_PIB_MANUF_ACK_CONFIRM_WAIT_TIME_RF, MAC_WRP_PIB_MANUF_DATA_CONFIRM_WAIT_TIME_RF.
10. Data Link Processes	Review Diagrams to add RF channel.
13. Supported Platforms	Added new supported platforms.
14. Toolchain	Introduced Microchip Studio.
16. Gateway Configuration	Imported Topic.

The Microchip Website

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

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

Product Change Notification Service

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

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

Customer Support

Users of Microchip products can receive assistance through several channels:

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

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

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

Microchip Devices Code Protection Feature

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

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

Legal Notice

Trademarks

ISBN: 978-1-5224-9371-6

Quality Management System

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

Worldwide Sales and Service

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