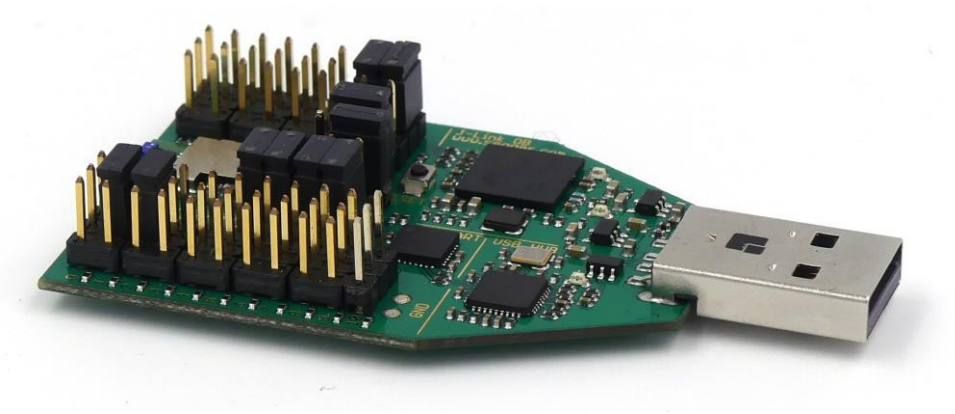
Panasonic

PAN1760A

Bluetooth Low Energy Module

Application Note – Doorbell

Rev. 1.0



Wireless Modules

By purchase of any of the products described in this document the customer accepts the document's validity and declares their agreement and understanding of its contents and recommendations. Panasonic Industrial Devices Europe GmbH (Panasonic) reserves the right to make changes as required at any time without notification.

This Application Note does not lodge the claim to be complete and free of mistakes.

The information contained herein is presented only as guidance for Product use. No responsibility is assumed by Panasonic for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.

Description of hardware, software, and other information in this document is only intended to illustrate the functionality of the referred Panasonic product. It should not be construed as guaranteeing specific functionality of the product as described or suitable for a particular application.

Any provided (source) code shall not be used or incorporated into any products or systems whose manufacture, use or sale is prohibited under any applicable laws or regulations.

Any outlined or referenced (source) code within this document is provided on an “as is” basis without any right to technical support or updates and without warranty of any kind on a free of charge basis according to § 516 German Civil Law (BGB) including without limitation, any warranties or conditions of title, non-infringement, merchantability, or fitness for a particular purpose. Customer acknowledges that (source) code may bear defects and errors.

The third-party tools mentioned in this document are offered by independent third-party providers who are solely responsible for these products. Panasonic has no responsibility whatsoever for the performance, product descriptions, specifications, referenced content, or any and all claims or representations of these third-party providers. Panasonic makes no warranty whatsoever, neither express nor implied, with respect to the goods, the referenced contents, or any and all claims or representations of the third-party providers.

To the maximum extent allowable by Law Panasonic assumes no liability whatsoever including without limitation, indirect, consequential, special, or incidental damages or loss, including without limitation loss of profits, loss of opportunities, business interruption, and loss of data.

Table of Contents

1 About This Document

1.1 Purpose and Audience

This Application Note is intended to explain how to setup and use the Doorbell application running on the PAN1760A USB stick.

The document is intended for software engineers.

1.2 Revision History

Revision	Date	Modifications/Remarks
1.0	2019-05-28	Initial version

1.3 Use of Symbols

Symbol	Description
	**Note** Indicates important information for the proper use of the product. Non-observance can lead to errors.
	**Attention** Indicates important notes that, if not observed, can put the product's functionality at risk.
	**Tip** Indicates useful information designed to facilitate working with the PAN1760A.
⇨ [chapter number] [chapter title]	**Cross reference** Indicates cross references within the document. **Example:** Description of the symbols used in this document ⇨ 1.3 Use of Symbols.
✓	**Requirement** Indicates a requirement that must be met before the corresponding tasks can be completed.
➔	**Result** Indicates the result of a task or the result of a series of tasks.
This font	**GUI text** Indicates fixed terms and text of the graphical user interface. **Example:** Click **Save**.

Symbol	Description
`This font`	**File names, messages, user input** Indicates file names or messages and information displayed on the screen or to be selected or entered by the user. **Examples:** `pan1760A.c` contains the actual module initialization. The message `Failed to save your data` is displayed. Enter the value `Product 123`.
Key	**Key** Indicates a key on the keyboard, e. g. **F10**.

1.4 Related Documents

[1] PAN1760A Software Guide

Please refer to the Panasonic website for more information as well as related documents
⇨ 9.1.2 Product Information.

2 Introduction

The PAN1760A USB stick is a development platform for the Bluetooth® Low Energy (LE) PAN1760A module.

It is based on the Bluetooth LE chipset TC35678 from Toshiba.

For further information please visit https://toshiba.semicon-storage.com/eu/product/wireless-communication/bluetooth/tc35678.html.

This Application Note explains the so-called Doorbell application which contains applications for both: a custom peripheral and a central device running on the PAN1760A in standalone mode using a custom Doorbell service.

The project can be used as a quick start for both peripheral and central custom projects.

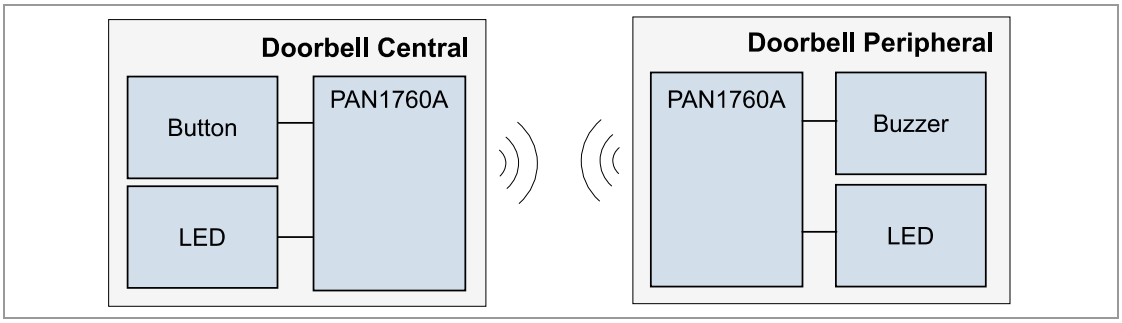
This Application Note refers to the Doorbell package release version 1.0.

To download the latest version, please refer to ⇨ 9.1.2 Product Information.

3 Doorbell Application Overview

The Doorbell application consists of two devices:

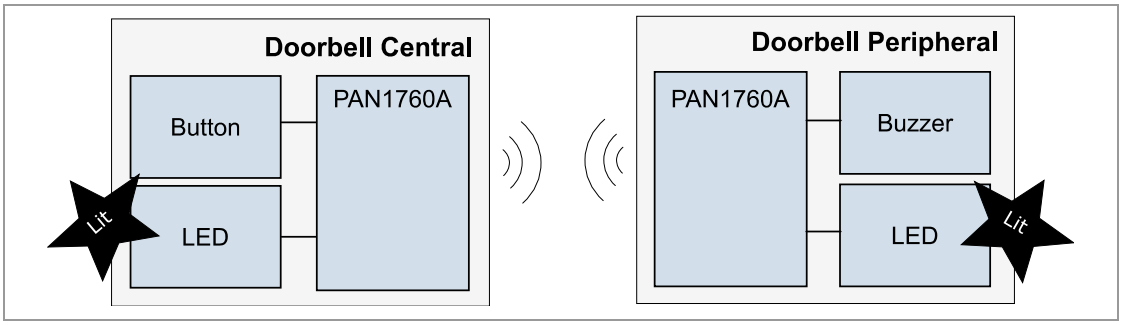
- The doorbell button device (implemented as a Bluetooth LE central)
- The doorbell bell device (implemented as a Bluetooth LE peripheral)



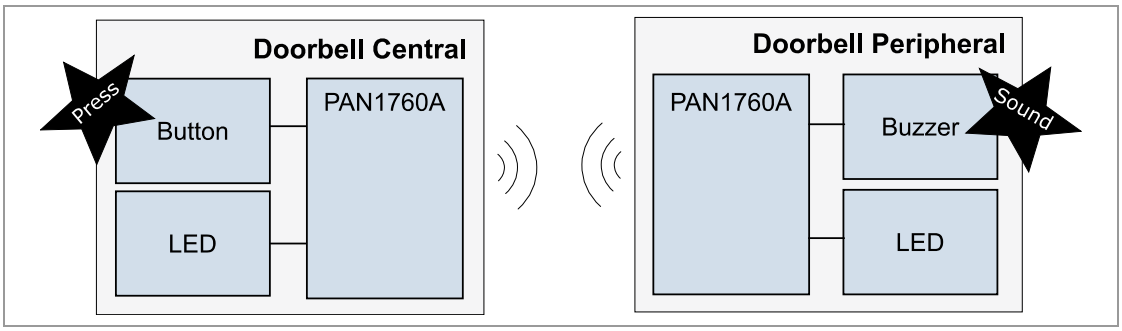
Both devices use the PAN1760A USB stick as a basis.

The doorbell button device has a simple switch. It replaces the traditional doorbell that is usually outside near the main entrance.

The doorbell bell device has a piezo buzzer attached. It replaces the traditional bell that is usually inside. The piezo buzzer has some kind of bell that rings when somebody pressed the doorbell button.



The devices will automatically connect to each other based on their device name. The LEDs will turn on to indicate that a connection has been established successfully.



Afterwards the button on the central device can be pressed and the currently selected ringtone tune is played.

4 Preparations

Toshiba provides a Bluetooth software development kit (SDK) that is available after registration. For further information please visit https://apps.toshiba.de/web/SDKRegistration/. After registration the Toshiba Bluetooth SDK can be downloaded from the “Developer Zone” https://apps.toshiba.de/extranet/display/TBTSW/.

The Toshiba Bluetooth SDK does only include a detailed example how to set up a peripheral device or central device in standalone mode, using a custom profile on the PAN1760A USB stick. Panasonic provides the PAN1760A Software Guide, which showcases how to use the SPP over Bluetooth LE profile for replacing a UART connection between two devices.

The PAN1760A Software Guide contains implementations for both peripheral and central devices in standalone mode, but does not show how to use a custom profile either.

In order to circumvent this, the Doorbell application tries to show how to implement a custom “Doorbell” profile for the peripheral device and central device.

A Simple off-the-shelf hardware including a button, a buzzer, and LEDs is used to show an appealing demo using a real-world use case, which can be used as a starting point for customer projects.

4.1 Prerequisites

The PAN1760A Software Guide already gives a good introduction into the following topics:

- Bluetooth Low Energy Basics
- Standalone Mode
- **EasyStandalone** tool
- Mode Selection via jumpers

It is assumed that these topics are known to the reader and are thus not explained in this Application Note any more.

If in doubt, please refer to the PAN1760A Software Guide for an introduction.

Furthermore it is assumed that Toshiba Bluetooth SDK version 4.2 is available, because it also includes the **EasyStandalone** tool that is used for programming the applications.

If in doubt, please refer to the PAN1760A Software Guide for an introduction how to use **EasyStandalone.**

4.2 Installation

The Doorbell application, shown in this document, is based on the Toshiba Bluetooth SDK version 4.2. It is mandatory to have that version downloaded before proceeding.

When the PAN1760A Doorbell application ZIP file is unpacked, the following directories are present:

- **binaries**: contains pre-compiled applications for immediate evaluation
- **pan-doorbell**: contains the project file and source code

Furthermore the following file is present in the current directory:
`PAN1760A Application Note - Doorbell.pdf` (this document)

5 Setup

In order to have audible output from the doorbell peripheral device, a simple piezo buzzer will be used. A simple tactile switch is used to simulate pressing a doorbell button on the doorbell central device.

On both devices LEDs are used to visually indicate the connection state of the system.

5.1 Hardware Prerequisites

Piezo buzzer, LEDs, and switches may be found in a spare parts collection.

The PAN1760A USB stick has multiple pin headers where peripherals can be attached to.

It is necessary to solder any peripherals to 2-pin female headers first so that they can be attached easily.

If none of these parts are available, then the following parts can be ordered and used for assembly:

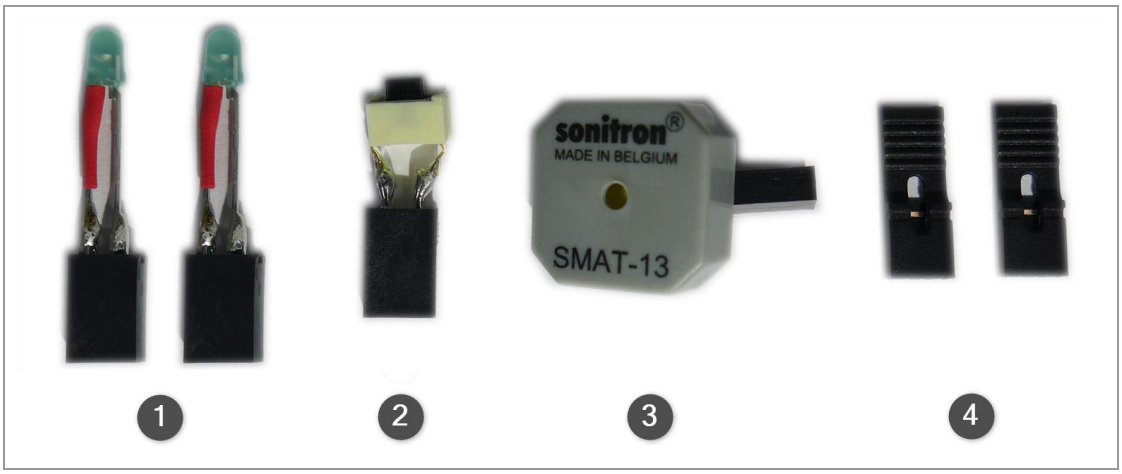
Item	Amount	Manufacturer	Part Number
Standard LED	2	Broadcom/Avago	HLMP-1790
Piezo Buzzer	1	Sonitron	SMAT-13-P7.5
Tactile Switch	1	Panasonic	EVQ-PE505K
2-pin 2.54 mm Female Header	4	Sullins Connector Solutions	PPTC021LFBN-RC

It is advisable to replace the programming selection jumper with a different jumper that has an attached latch or lug. This makes changing the jumper for programming much easier, especially when the other additional hardware is mounted.

Item	Amount	Manufacturer	Part Number
2-pin 2.54 mm Jumper with lug	2	Various	Various

5.2 Assembly

The LED, the buzzer, and the button need to be soldered to a 2-pin female header.



Number	Item
1	Standard LED
2	Button
3	Piezo Buzzer
4	2-pin 2.54 mm Jumper with lug

LED Assembly

Solder the LED to the 2-pin female header.

Make sure to mark the anode of the LED, for example with a small piece of heat shrink tube, because the orientation does matter when mounting the LED.

Button Assembly

Solder the button to the 2-pin female header in any orientation.

Buzzer Assembly

Pins can break!
Beware of squeezing both pins of the buzzer.

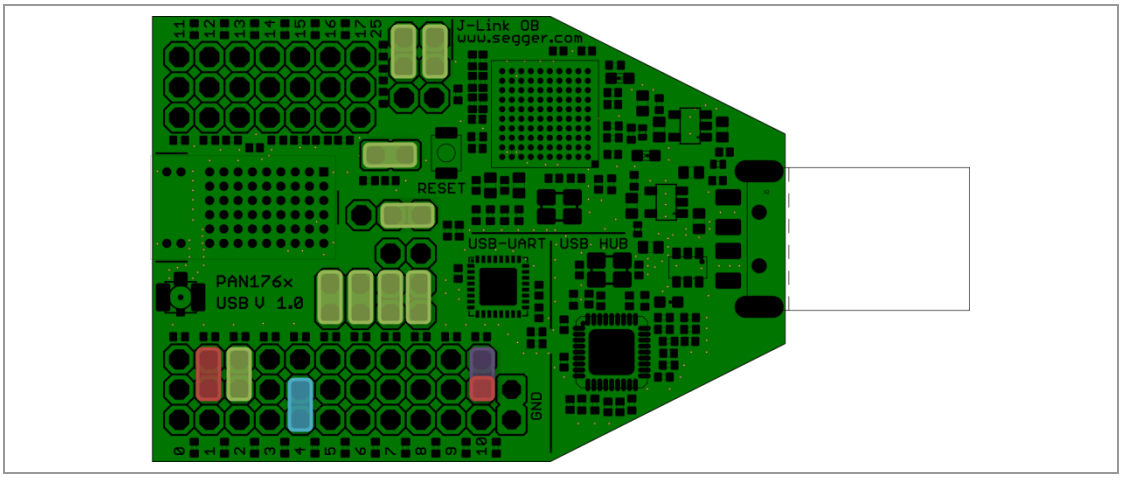
It is possible to bend one pin with a plier and shorting the other pin with a side cutter, then the 2-pin female header can be soldered orthogonal to the buzzer.

5.3 Peripheral Device

The following chapters will explain how to setup the hardware for the peripheral device and how to do the necessary programming.

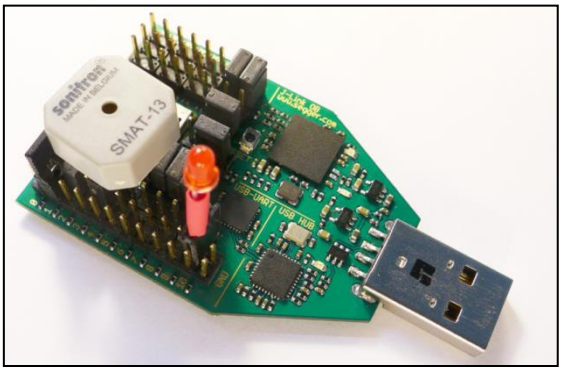
5.3.1 Hardware

The following steps need to be done on the PAN1760A USB stick that should become the peripheral device:



1. Replace the jumper (red marking).
2. Install the buzzer assembly (blue marking). The orientation does not matter.
3. Install the LED assembly (violet marking). The orientation does matter.

➔ The end result looks like this:



5.3.2 Programming

A pre-compiled peripheral device application can be found under `binaries/peripheral.hex`.

Use **EasyStandalone** and follow the usual procedure to program a custom application into the flash memory.

If in doubt, please refer to the `PAN1760A Software Guide` (chapter “Programming the Firmware”).

Please remember that the position of the programming selection jumper must be changed after programming.

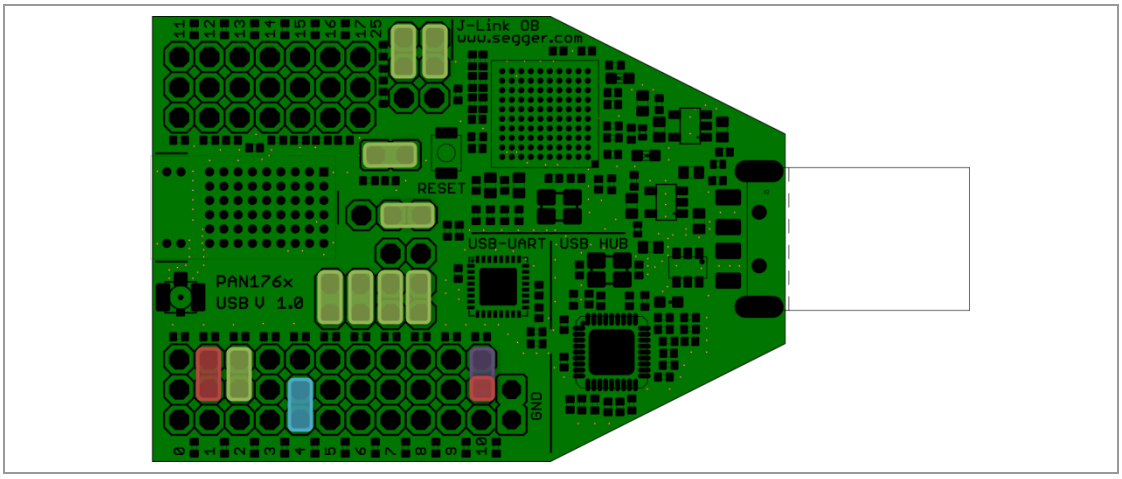
Otherwise the application will not be run.

5.4 Central Device

The following chapters will explain how to setup the hardware for the central device and how to do the necessary programming.

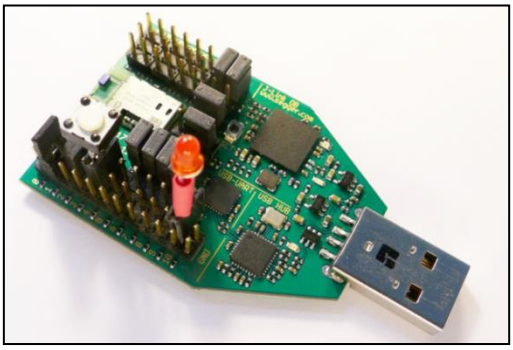
5.4.1 Hardware

The following steps need to be done on the PAN1760A USB stick that should become the peripheral device:



1. Replace the jumper (red marking).
2. Install the button assemble (blue marking). The orientation does not matter.
3. Install the LED assembly (violet marking). The orientation does matter.

➔ The end result looks like this:



5.4.2 Programming

A pre-compiled central device application can be found under `binaries/central.hex`.

Please use **EasyStandalone** and follow the usual procedure to program a custom application into the flash memory.

If in doubt, please refer to the `PAN1760A Software Guide` (chapter “Programming the Firmware”).

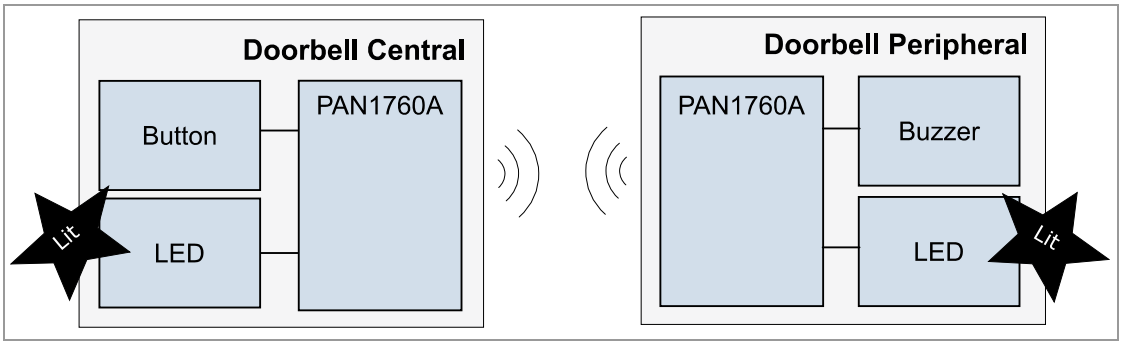
Please remember that the position of the programming selection jumper must be changed after programming.

Otherwise the application will not be run.

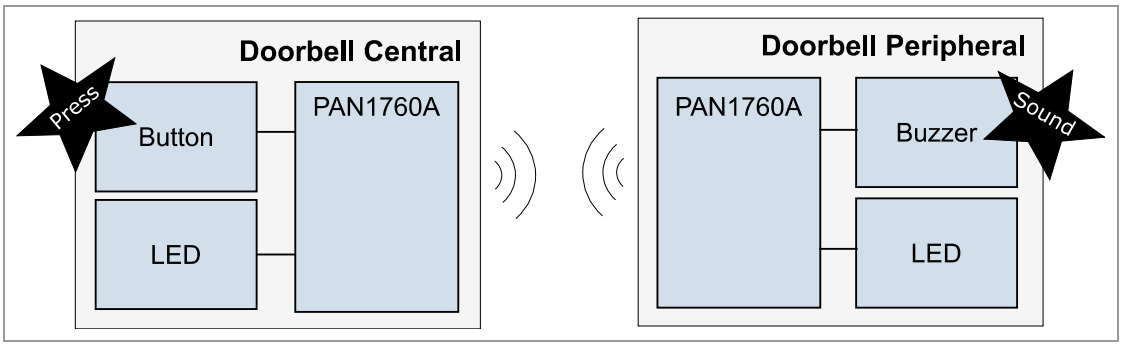
6 Basic Usage

Both the central and the peripheral devices use the standalone mode of PAN1760A.

After programming they are fully functional on their own, so it is also sufficient to plug them into a USB socket or a USB power adapter to start them.



1. Power up the devices.
   - ➔ The devices will automatically connect to each other (based on their device name).
   - ➔ The LEDs will turn on to indicate that a connection has been established successfully.
2. Press the button on the central device.
   - ➔ The currently selected ringtone tune is played.



7 Advanced Usage

The peripheral device and the central device will output their current status and actions via their UART to the PC, which can be visualized using a Terminal application.

If you do not have any terminal application yet, you can try application **Termite**.

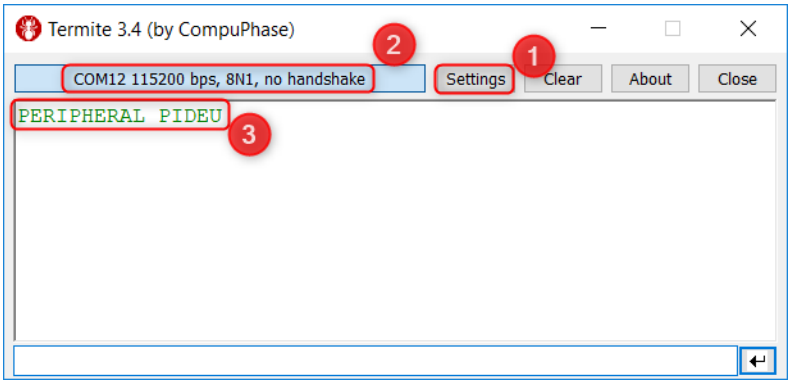
In the following steps **Termite** is used.

7.1 Setup

The following requirements must be met:

✓ Both PAN1760A USB sticks are properly programmed as central and peripheral device, jumpered to “standalone mode” and are attached to the PC.

1. Open **Termite**.
   ➔ A connection to the communication COM port of one of the PAN1760A USB sticks will be established.
2. Click **Settings** (1) to choose the communication COM port of one of the PAN1760A USB stick.



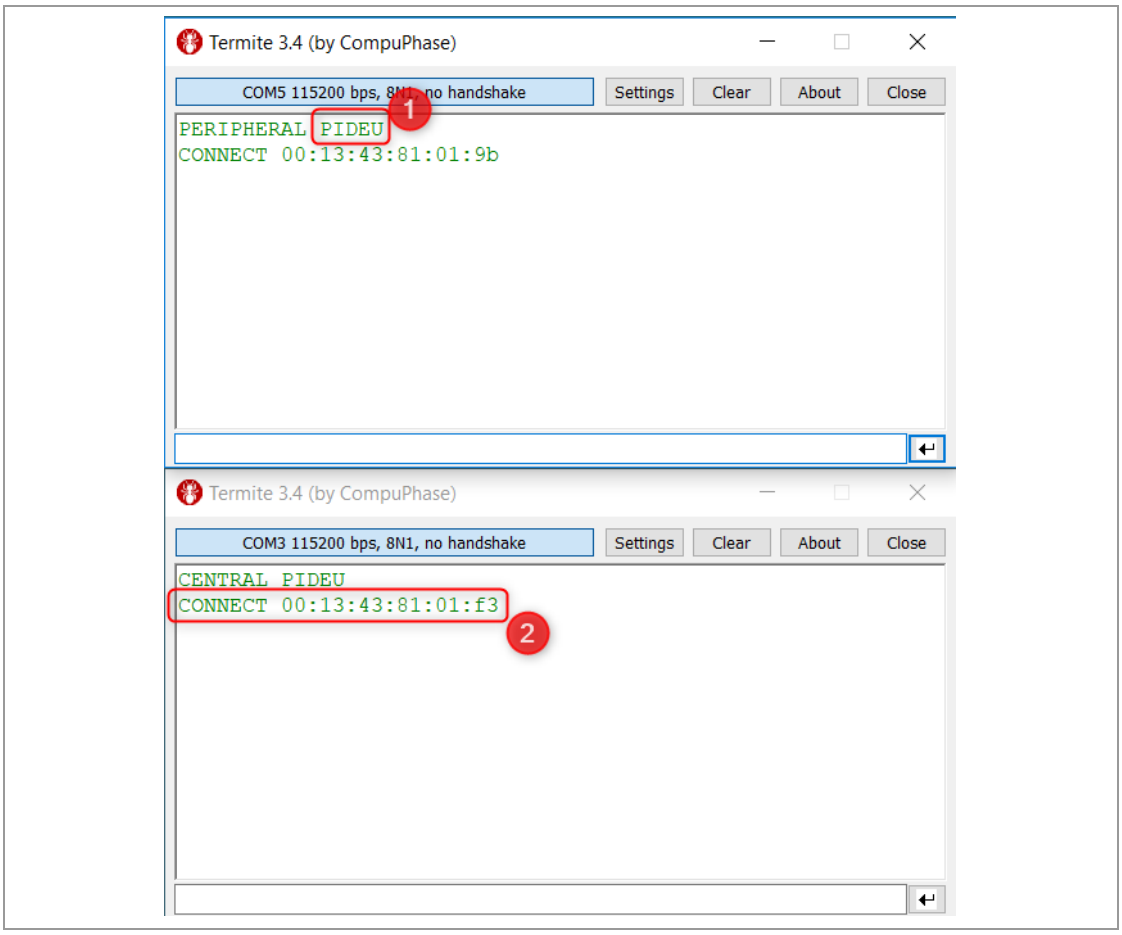
   ➔ The chosen COM port will be displayed (2).
3. Press the reset button on the PAN1760A USB stick.
   ➔ It will identify itself by printing either `CENTRAL` or `PERI PHERAL` to the terminal window (3).
4. Repeat the steps above with the other PAN1760A USB stick, so that the output of both devices is shown in the individual Termite windows.

7.2 Connection Establishment

After on both of the PAN1760A USB sticks the reset button, is pressed, a connection is automatically established, based on the device name.

Both devices report their role (either peripheral or central) and the device name after reset (1).

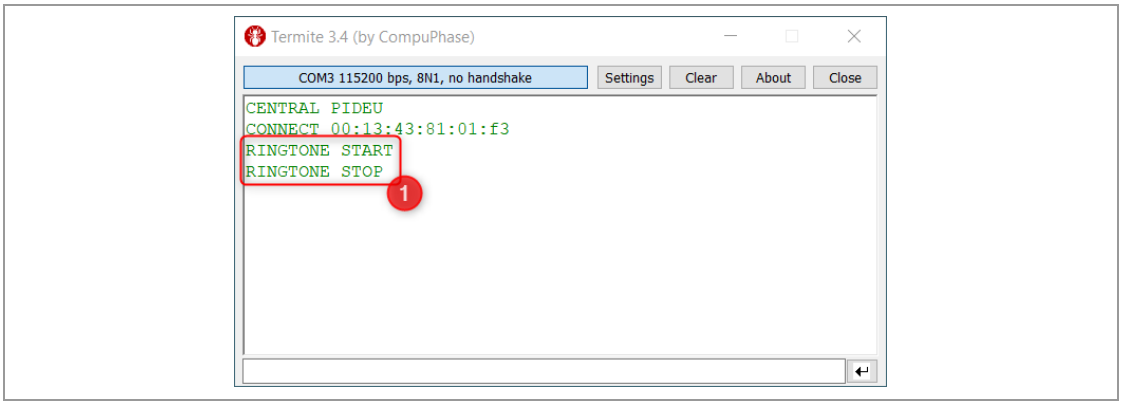
If a connection is established, a connect message (2) is shown and the LEDs on both devices should be lit.



7.3 Doorbell Action

When the button is pressed on the central device, the peripheral device starts playing a tune.

The information if the ringtone is actually playing or not, is transmitted from the peripheral device to the central device and shown on the terminal (1).

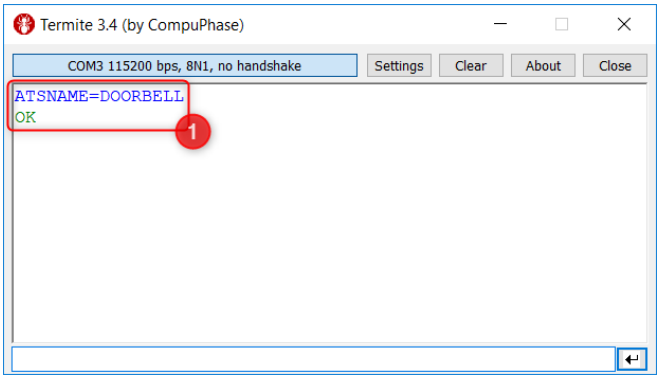


7.4 Device Name Configuration

The peripheral device and the central device establish a connection based on their device name.

Sometimes it is desirable to change the default device name, for example when multiple demos are operating in the same room.

1. Use the AT command `ATSNAME` to change the name on the peripheral device and on the central device.
   The syntax of the command is: `ASNAME=<name>` (1).

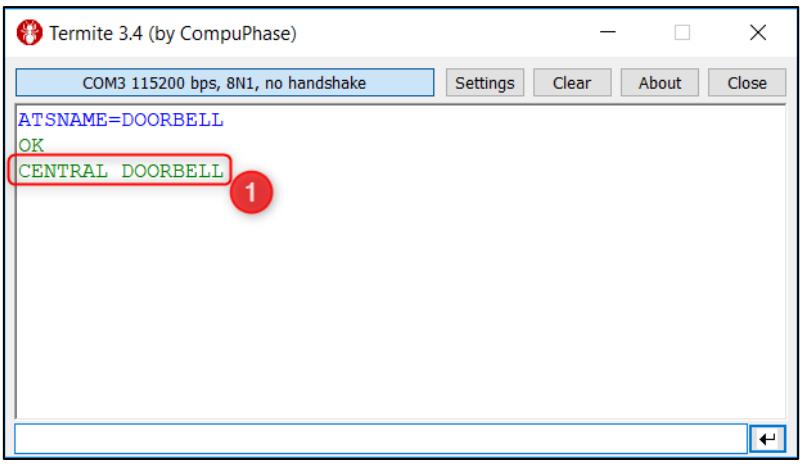


➔ If the name of the device was changed successfully, the command is acknowledged by the string OK on the terminal.

If the name has been changed on the peripheral device and on the central device, the devices need to be reset for the setting to take effect.

2. Press the reset button for the setting to take effect.

➔ The new device name is shown in the welcome message (1).



7.5 Tune Selection

The peripheral device features multiple tunes that can be selected.

The tune selection must be done on the central device by using the AT command `ATSTUNE`.

This is counterintuitive, but the main reason is to implement and showcase more Bluetooth Low Energy functionality in the Doorbell application.

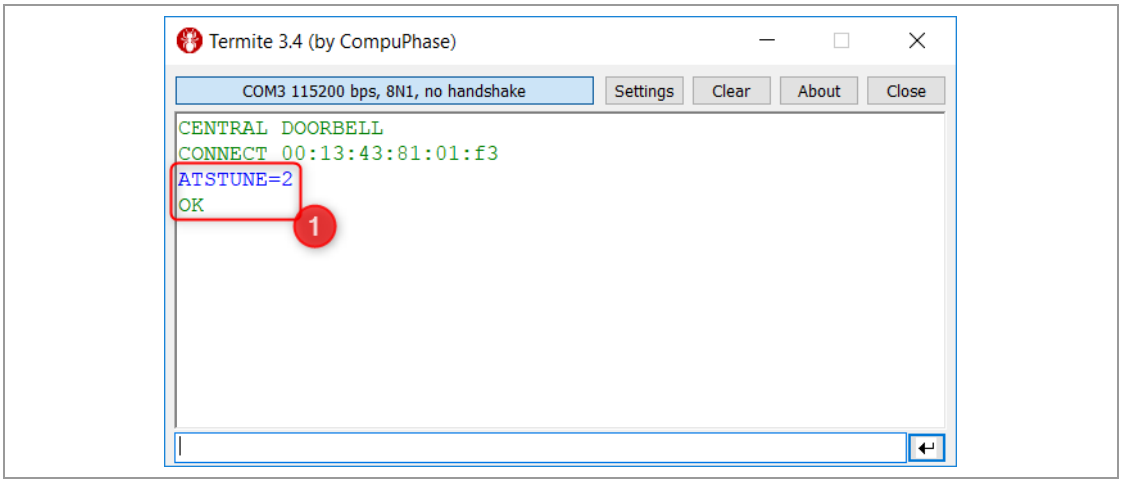
With the chosen design the tune selection must be forwarded from the central device to the peripheral device via Bluetooth LE.

> The AT command `ATSTUNE` only works when a connection is established between the peripheral device and the central device.

The syntax of the command is: `ASTUNE=<tune number>`.

At the moment the following tune numbers are available:

- 0 = “Für Elise” from Ludwig van Beethoven
- 1 = “Kleine Nachtmusik” from Wolfgang Amadeus Mozart
- 2 = “Ode an die Freude“ from Ludwig van Beethoven



If the tune of the peripheral was changed successfully, the command is acknowledged by the string OK on the terminal of the central device.

If something goes wrong, FAI LED is printed instead. If this happens, please check the following conditions:

- ✓ A connection is established between the two devices.
- ✓ The number lies in the allowed range.

8 Details “Under the Hood”

When the PAN1760A Doorbell Application ZIP file is unpacked, the following directories are present in the current directory:

- **binaries**: contains pre-compiled applications for immediate evaluation
- **pan-doorbell**: contains the project file and source code

8.1 Prerequisites

The Doorbell application uses the Toshiba Bluetooth SDK version 4.2, which needs to be unpacked to the same directory.

The resulting directory is named `BT_SDK_v_4_2` and must be renamed to `sdk` before continuing.

8.2 Organization

The Doorbell application project is based on **IAR Embedded Workbench**.

For further information please visit https://www.iar.com/iar-embedded-workbench/.

The project solution file is located under `project\project.eww`.

The project handling is completely similar to the handling described in the PAN1760A Software Guide (chapter “Standalone Mode”) and is not further discussed here.

8.3 File Structure

pan-doorbell	
| project.eww	IAR Embedded Workbench project file
+---application	Application directory
| | main.c	Shared main application files
| | peripheral.c	Peripheral implementation file
| | central.c	Central implementation file
| | music.c	PWM music implementation
| | mini-printf.c	Small printf implementation
| | compat.c	Compatibility code for other Panasonic modules
| | stub-functions.c	Stub function for non-used Bluetooth SDK
| | uart1_function.c	Custom UART driver
| \---include	Include file directory
| application.h	Main application header file
| compat.h	Compatibility code for other Panasonic modules
| debug.h	Header file containing debug routines
| mini-printf.h	Small printf implementation header file
| music.h	PWM music header file
| uart1_function.h	Custom UART driver header file
| user_configuration.h	Toshiba Bluetooth SDK configuration
\---project	Application-specific directory
[...]	Application and project-specific files

8.4 Project Setup

The project contains two configurations:

- “Debug – Peripheral”: contains the configuration for the Doorbell peripheral device.
- “Debug – Central”: contains the configuration for the Doorbell central device.

`main.c` contains the common start code that is shared between the configurations and that does not differ between the peripheral and central applications.

Right before the main processing loop is entered, `application_init_callback()` is called, which will handle the device-specific initialization of either the peripheral device or central device.

The complete device-specific implementations can be found in a single file each:

- The individual implementation of the peripheral device can be found in `peripheral.c`.
- The implementation of the central device can be found in `central.c`.

Both completely implement a custom Doorbell profile including a custom Doorbell service.

8.5 Custom Doorbell Profile

The custom Doorbell Profile consists of three services:

- Generic Access Service
- Device Information Service
- Doorbell Service

The Generic Access Service is mandatory for every peripheral device.

The Device Information Service includes the following characteristics:

- Model Number String
- Manufacturer Identifier
- Manufacturer Name String
- Organizationally Unique Identifier
- Hardware Revision String

Both services are taken as-is from the Toshiba Bluetooth SDK implementation and only provide dummy values.

The Doorbell Service consists of the following characteristics:

- Ring
- Current Tune

8.5.1 Ring Characteristic

The Ring characteristic has the following properties defined: read, write and indicate.

The read property can be used to read the current state of music tune playback. It is the value `0x00` if currently no music tune is playing and the value `0x01` if a music tune is currently playing.

The write property can be used to start the playback of a music tune. Any value (including the value `0x00`) will start the playback.

The indicate property is used by the peripheral device to transmit the current state of the music tune playback. When indications are properly enabled by the central device, the value `0x01` is transmitted when the music is just about to start. The value `0x00` is transmitted when the playback has just finished.

8.5.2 Current Tune Characteristic

The Current Tune characteristic has the read and writes properties defined.

The read property can be used to read back the currently selected tune.
The write property can be used to change the currently selected tune.

8.6 Peripheral Implementation

The necessary music playback is realized by using the built-in PWM controller of PAN1760A. The necessary code is contained in `music.c`, the minimalistic API is defined in `music.h`.

The complete implementation of the peripheral Doorbell profile can be found in `peripheral.c`.

It starts with a couple of structures that are used to store runtime application data and configuration data in non-volatile memory.

8.6.1 Ring Characteristic

The first section contains all information about the Ring characteristic. The characteristic value is stored in the `is_ringing` variable.

Next follows the definitions of the callback functions that are needed for initialization, reading, and writing of the Ring characteristic.

When the characteristic is read or written, the actual value is read from or stored to the `is_ringing` variable. When the characteristic value is written, `write_characteristic_ring()` is called and the music playback is unconditionally started.

There is no need for a special handling of the client characteristic configuration descriptor, so all the callback functions are mostly empty.

Afterwards the Ring characteristic is defined. It consists of the `characteristic_ring_definition` structure, which mostly consists of the callback functions.

It uses a custom UUID which was generated using https://www.uuidgenerator.net/. If you intend to use the same tool, please make sure to read the disclaimer carefully.

8.6.2 Current Tune Characteristic

The next section contains everything about the Current Tune characteristic which follows a very similar pattern.

Because it supports the properties read and write as well, it includes definitions for the necessary callback functions.

There is no dedicated storage for the Current Tune characteristic value, because it is implicitly stored in the device configuration data.

Currently Selected Tune

When `write_characteristic_current_tune()` is called, the `write_config()` helper function is used to store the currently selected tune in non-volatile memory.

Follow the definition of the Current Tune characteristic in `characteristic_current_tune_definition()` which again includes a custom UUID.

Afterwards the definition of the Doorbell Service is done in `service_doorbell`, using a custom UUID as well.

Started or Stopped Music Playback

Whenever the music playback is started or stopped, the “music subsystem” will call the callback functions: `music_started_callback()` and `music_stopped_callback()`. Because these callbacks may be called asynchronously, no calls into the Bluetooth SDK should be done.

These callback functions just track the current and set a flag to indicate if the music playback has started or stopped.

Bluetooth Related Processing

The function `application_process_callback()` is regularly called by the main processing loop, so it is an ideal place to handle all Bluetooth related processing.

When this function is called for the very first time, it will finish the device configuration by calling `ble_api_mng_config_connection_count()`. When processing of that command has finished, `ble_application_mng_config_connection_count_response_callback()` is called and the peripheral role can be finally started by starting the advertising.

Music Playack State

During normal operation, the function `application_process_callback()` checks if the current state of the music playback has changed and act accordingly.

If the music playback has been started or stopped, an indication may be sent if sending of indications has been enabled by the remote central device.

Necessary Functions

Afterwards some necessary functions for the Doorbell profile are defined.

First, connected and disconnected callback functions are defined which will toggle the LED according to the current connection status. If a connection was terminated, advertising is started again.

Finally, the definition of the Doorbell profile is done in `profile_doorbell`.

The following helper functions `read_config()` and `write_config()` help to read and write the current configuration from and to non-volatile memory.

Initialization

Afterwards `application_init_callback()` is defined which is called during device initialization. It makes sure that the configuration is read from non-volatile memory and then initializes all GPIO for the LED handling.

Then the “music subsystem” is initialized and the services of the Doorbell profile are configured. Finally the Bluetooth SDK is initialized by calling `ble_api_srv_profile_init()`.

Invoked AT Command

The `application_at_command_callback()` function is called whenever an AT command is invoked on the UART. For the peripheral device only the command `ATSNAME` is supported.

Stub Functions

At the very end some stub functions for the central application are defined. This is necessary, because the main code is shared and some callback functions of the Bluetooth SDK needs to be available, even if they are never used.

8.7 Central Implementation

The complete implementation of the central Doorbell profile can be found in `central.c`.

It starts with a couple of structures that are used to store runtime application data and configuration data in non-volatile memory.

8.7.1 Ring Characteristic

The first section contains all information about the Ring characteristic. It starts with the callback functions of the client characteristic configuration descriptor that is associated with the characteristic value of the Ring characteristic.

Callback Functions

Follow the necessary callback functions for the Ring characteristic.

When an indication is received that shows if the music playback has started or stopped on the peripheral device, `characteristic_ring_indication()` is called. The implementation will output a diagnostic message to the UART.

The current central application will never issue read of the Ring characteristic, so the corresponding callback functions exist just for completeness. A write is issued in order to start the music playback when a button is pressed, but the end result is not further checked in the callback function in this implementation.

Definition

Afterwards the Ring characteristic is defined. It consists of the `characteristic_ring_definition` structure, which mostly consists of the callback functions. Please note that it uses the same custom UUID as the peripheral device.

8.7.2 Current Tune Characteristic

The next section contains everything about the Current Tune characteristic which follows a very similar pattern.

Write and Read Characteristics

A write to the characteristic is issued when the command `ATSTUNE` is processed. The result of the operation is output on the UART as either “OK” or “FAILED”.

A read of that characteristic is never done by the central application, so the callback functions exist just for completeness.

Definition

The Current Tune characteristic is actually defined. It consists of the `characteristic_current_tune_definition` structure, which mostly consists of the callback functions. Please note that it uses the same custom UUID as the peripheral device.

Doorbell Service Definition

The definition of the Doorbell Service is done in `service_doorbell`, using a custom UUID as well and referencing the Ring and Current Tune characteristics.

When a connection has been established from the central to the peripheral device, the Bluetooth SDK will automatically scan the peripheral device for the services that the peripheral device offers.

The callback function `discover_done_callback()` is called when all services as required by the Doorbell profile have been found on the peripheral device.

The function needs to check if the required functionality with regard to characteristics and descriptors are available, too.

Established Connection

If all checks are successful, the LED is lit to indicate that a connection has been established successfully. Finally indications on the Ring characteristic are enabled, so that start and stop of the music playing on the peripheral device is properly noticed.

Helper Functions

When scanning for peripheral devices is done once after start up, the helper function `scan_start()` is called. Additionally it is called by `disconnected_callback()` which also makes sure that the LED is turned off after a disconnect.

The definition of the Doorbell profile is done in `profile_doorbell`.

The following helper functions `read_config()` and `write_config()` help to read and write the current configuration from and to non-volatile memory.

`button_interrupt()` is called directly from the interrupt service routine, so it will forward the information to the main processing loop, which will call `button_handler_callback()` in the right context. Only there it is possible to call `ble_api_cli_characteristic_write_value_start()` on the Ring characteristic to start the music playback on the peripheral device.

Initialization

Afterwards `application_init_callback()` is defined which is called during device initialization. It makes sure that the configuration is read from non-volatile memory and then initializes all GPIOs for the LED and button handling.

Bluetooth Related Processing

The function `application_process_callback()` is regularly called by the main processing loop, so it is an ideal place to handle all Bluetooth related processing.

When this function is called for the very first time, it will finish the device configuration by calling `ble_api_mng_config_connection_count()`. No other processing is necessary for the central application.

Scanning

When processing of that command has finished, the function `ble_application_mng_config_connection_count_response_callback()` is called. The central role can be finally started by “starting the scanning for peripheral devices”.

All of the operations of the central device are asynchronous and so is the scan functionality. `ble_application_master_scan_status_update_callback()` is called whenever scanning is stopped. Scanning is only stopped when a suitable device to connect to has been found, so this function tries to initiate a connection.

Whenever some advertising data is received by the central device during scanning `ble_application_master_advertise_report_event_callback()` is called.

By extracting a 128-bit UUID and a device name from the advertising the central device tries to find out if a suitable Doorbell peripheral device is found. If so it will stop the scanning.

Invoked AT Command

The `application_at_command_callback()` function is called whenever an AT command is invoked on the UART. For the central device the command `ATSNAME` and `ASTUNE` are supported.

When the command `ATSTUNE` is encountered, a write to the Current Tune characteristic value of the peripheral device is started.

When the command `ATSNAME` is encountered the device name is changed in the configuration data and the configuration data is written to the non-volatile memory.

8.8 Further Readings

If you need more details concerning inner workings of the Toshiba Bluetooth SDK or the underlying chipset, please refer to the documentation from the Toshiba Bluetooth SDK.

The Bluetooth SDK contains extensive documentation, for example the “Bluetooth SDK Developer’s Guide”.

All documentation from Toshiba can be found in the Bluetooth SDK directory `BT_SDK_v_4_2\documentation`.

9 Appendix

9.1 Contact Details

9.1.1 Contact Us

Please contact your local Panasonic Sales office for details on additional product options and services:

For Panasonic Sales assistance in the **EU**, visit

https://eu.industrial.panasonic.com/about-us/contact-us

Email: wireless@eu.panasonic.com

For Panasonic Sales assistance in **North America**, visit the Panasonic website "Sales & Support" to find assistance near you at

https://na.industrial.panasonic.com/distributors

Please visit the **Panasonic Wireless Technical Forum** to submit a question at

https://forum.na.industrial.panasonic.com

9.1.2 Product Information

Please refer to the Panasonic Wireless Connectivity website for further information on our products and related documents:

For complete Panasonic product details in the **EU**, visit

http://pideu.panasonic.de/products/wireless-modules.html

For complete Panasonic product details in **North America**, visit

http://www.panasonic.com/rfmodules