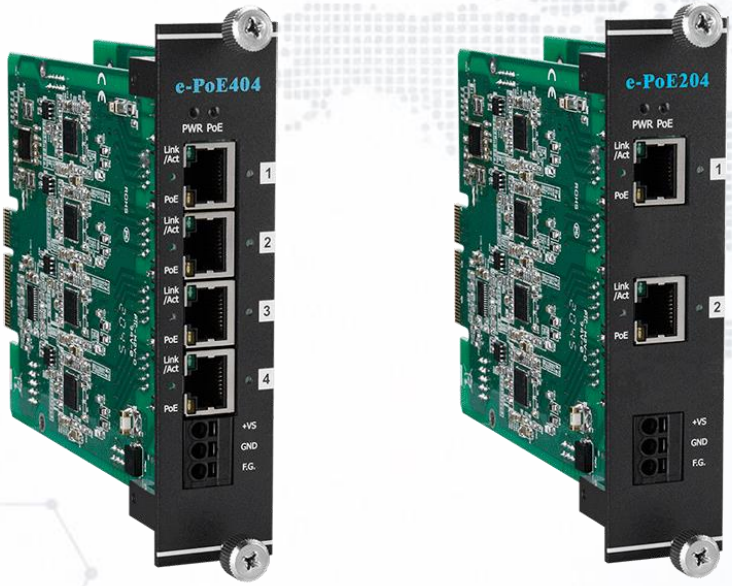
e-PoE204/e-PoE404

User Manual

V1.0.1 July 2026



2/4 port 10/100/1000 Mbps

PoE (PSE) Ethernet Expansion Module

Written by Tommy Hong

Edited by Anna Huang

Warranty

All products manufactured by ICP DAS are under warranty regarding defective materials for a period of one year, beginning from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for any damage resulting from the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Copy right © 2018 by ICP DAS Co., Ltd. All rights are reserved.

Trademarks

Names are used for identification purposes only and may be registered trademarks of their respective companies.

Contact Us

If you have any problems, please feel free to contact us.
You can count on us for a quick response.
Email: service@icpdas.com

Contents

Contents 3

1. Overview 6
- 1.1. Features 8
- 1.2. Applications 9
- 1.3. Specifications 10
- 1.4. Hardware Overview 11
- 1.5. Dimension 14

2. Installation 15
- 2.1. Hardware Installation 16
- 2.2. Software Installation 18
  - 2.2.1. Obtaining PoE Manager 18
  - 2.2.2. Installing Required Components 20
  - 2.2.3. Launching PoE Manager 22
- 2.3. SDK Preparation 24
  - 2.3.1. Obtaining the SDK 24
  - 2.3.2. SDK Packages 26
  - 2.3.3. Preparing the SDK 26

3. PoE Manager 27
- 3.1. Overview 27
- 3.2. Main Window 28
- 3.3. User Interface 30
  - 3.3.1. Ports 31
  - 3.3.2. Port Details 35
  - 3.3.3. Power Management 37
  - 3.3.4. Controller Status 41
  - 3.3.5. Priority 44

4. SDK API Reference 46
- 4.1. Overview 47
  - 4.1.1. SDK Relationship 47
  - 4.1.2. e-PoE SDK Libraries 48
  - 4.1.3. Supported Development Environment 49
- 4.2. Programming Workflow 50
  - 4.2.1. Development Architecture 51
  - 4.2.2. SDK Initialization 52
- 4.3. Function Categories 53

4.3.1. System Functions 53
4.3.2. Controller Functions 54
4.3.3. Port Management Functions 55
4.3.4. Power Management Functions 56
4.3.5. Monitoring Functions 57
4.3.6. Watchdog Functions 58
4.3.7. Interrupt Functions 59
4.3.8. Register Access Functions 60
4.4. API Reference 61
4.4.1. EBUS_SLOT 63
4.4.2. iPOE_Init 64
4.4.3. iPOE_Close 65
4.4.4. GetSDKVersion 66
4.4.5. GetNetVersion 67
4.4.6. GetFirmwareVersion 68
4.4.7. Reset 70
4.4.8. Set_PSE_config 72
4.4.9. Set_SYS_SW_config 74
4.4.10. Get_SYS_SW_config 77
4.4.11. Get_PSE_status 80
4.4.12. Get_PSE_Event 83
4.4.13. Clear_PSE_Event 85
4.4.14. Set_POE_Port 86
4.4.15. Set_POE_port_HiPower 88
4.4.16. Get_POE_port_HiPower 90
4.4.17. Set_POE_port_force_power 92
4.4.18. Set_port_priority 94
4.4.19. Get_port_priority 96
4.4.20. Set_POE_port_power_limit 98
4.4.21. Get_POE_port_power_limit 100
4.4.22. Set_power_banks 102
4.4.23. Get_power_banks 104
4.4.24. Get_total_power 106
4.4.25. Get_power_supply_voltage_current 108
4.4.26. Get_port_status 110
4.4.27. Get_port_Laststatus 113
4.4.28. Get_port_power_measurements 115
4.4.29. Get_port_event_status 117
4.4.30. Clear_port_event_status 120
4.4.31. SetWDTConfig 122

4.4.32. GetWDTConfig 124
4.4.33. GetWDTStatus 126
4.4.34. ResetWDT 128
4.4.35. Set_interrupt_setting 129
4.4.36. Get_interrupt_setting 131
4.4.37. Clear_interrupt 132
4.4.38. Set_register_value 133
4.4.39. Get_register_value 134

Appendix 135

A. Revision History 135

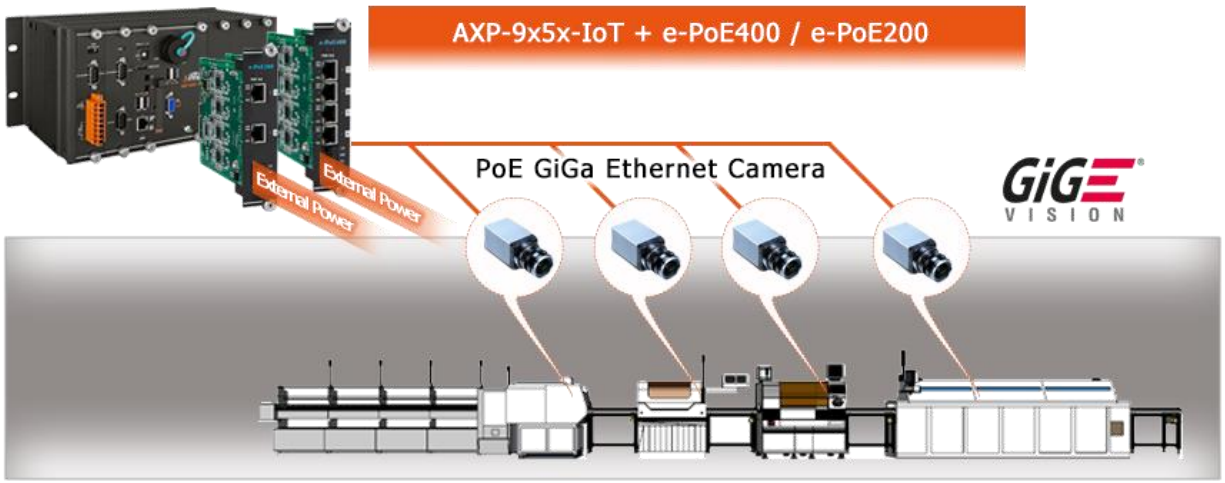
1. Overview

The e-PoE204 and e-PoE404 are Industrial Power over Ethernet (PoE) expansion modules designed for the ICP DAS PAC platform. They provide high-speed Gigabit Ethernet connectivity together with IEEE 802.3af/802.3at compliant PoE functionality, enabling both power and data transmission over a single Ethernet cable.

Unlike standalone Ethernet adapters, the e-PoE204/e-PoE404 operate as PAC SDK Extension Modules. Installed into an expansion slot of a supported PAC controller, the modules integrate seamlessly with the PAC SDK and extend the controller with programmable PoE management capabilities. Through the provided SDK and PoE Manager utility, developers can configure PoE parameters, monitor operating status, retrieve diagnostic information, and control individual Ethernet ports without directly accessing hardware registers.

The e-PoE204 provides two independent Gigabit Ethernet PoE ports, while the e-PoE404 provides four independent Gigabit Ethernet PoE ports. Each port supports up to 30 W of PoE output and complies with IEEE 802.3af/802.3at standards. Advanced functions including per-port power control, power budget management, port monitoring, event reporting, and watchdog support are available through the SDK APIs.

The modules are specifically designed for bandwidth-intensive Industrial Ethernet applications requiring reliable network communication and centralized power management. Typical applications include machine vision systems, automated optical inspection (AOI), industrial automation, intelligent transportation, surveillance systems, robotics, and medical imaging systems utilizing PoE GigE Vision cameras.



The e-PoE204/e-PoE404 are supported by the PoE Manager configuration utility and the e-PoE SDK, which includes native Visual C++ and .NET libraries. The SDK follows the PAC SDK programming model, allowing developers to rapidly integrate PoE functionality into Windows-based industrial applications while maintaining a consistent software architecture across ICP DAS PAC products.

The modules are installed into the PCIe expansion slots of supported PAC controllers, including:

- AXP-9251-IoT
- AXP-9651-IoT
- EMP-9251-IoT
- EMP-9651-IoT

By combining Industrial Gigabit Ethernet communication, IEEE-compliant PoE technology, centralized software management, and PAC SDK integration, the e-PoE204/e-PoE404 provide a scalable and flexible platform for developing Industrial IoT and machine automation solutions.

1.1. Features

This section summarizes the major features of the e-PoE204/e-PoE404 modules. In addition to providing Gigabit Ethernet connectivity and IEEE 802.3af/802.3at compliant Power over Ethernet (PoE), the modules support software-based power management and monitoring through the e-PoE SDK and PoE Manager utility. These features enable developers to build reliable, centrally managed Industrial Ethernet applications with simplified deployment and maintenance.

Ethernet Features

- Two (e-PoE204) or four (e-PoE404) independent Gigabit Ethernet ports
- Intel® I211AT Gigabit Ethernet Controller
- 10/100/1000 Mbps auto-negotiation
- Supports IEEE 1588 hardware time synchronization
- Supports Jumbo Frame (up to 9 KB)
- Supports Link Aggregation for increased bandwidth and redundancy

Power over Ethernet (PoE) Features

- IEEE 802.3af and IEEE 802.3at compliant
- Up to 30 W PoE output per port
- Automatic Powered Device (PD) detection and classification
- Intelligent PoE power budget management
- Individual port power ON/OFF control
- Per-port power limit configuration
- Real-time voltage, current, and power monitoring
- Comprehensive PoE protection, including over-current, over-voltage, under-voltage, over-temperature, and power budget protection

Software Features

- Fully integrated with the PAC SDK Extension architecture
- Native SDK libraries for Visual C/C++ and .NET applications
- Supports PoE Manager for graphical configuration and monitoring
- Software APIs for controller configuration, port management, power management, monitoring, watchdog, interrupt handling, and register access
- Supports firmware version and SDK version management
- Simplifies application development through a unified programming interface

Industrial Design

- Designed for ICP DAS PAC platforms
- PCIe expansion module architecture
- Industrial operating temperature: –25 °C to +60 °C
- Compact mechanical design for industrial control systems
- CE, RoHS, and WEEE compliant

1.2. Applications

This section outlines typical application scenarios for the e-PoE204/e-PoE404, showcasing its versatility in automation and monitoring environments.

- Machine Vision Inspection
- Automated Optical Inspection
- Surveillance and Security
- Medical Imaging

1.3. Specifications

This section provides the complete hardware, software, Ethernet, PoE, power, mechanical, and environmental specifications of the module.

Model	e-PoE204	e-PoE404
General		
Type	e-Busx4 (4.0 GB/s total bandwidth)	
System LED Indicator	1 x PoE Power, 1 x System Power, 2 x Link Status, 2 x PoE Status	
Software		
SDK	Power Management SDK, Windows 10 IoT OS compatible, VC++ dll	
Utility	PoE Power Management Uility (Voltage, current Power monitor, PoE Power ON/OFF)	
Ethernet		
Ports	2 x RJ-45, Intel I211AT Controller, 10/100/1000 Mbps	4 x RJ-45, Intel I211AT Controller, 10/100/1000 Mbps
Protocol	IEEE 1588	
Jumbo Frame	9 KB	
Link Aggregation	Yes	
PoE Technology		
PoE Compliance	IEEE802.3af Compliance, IEEE802.3at Compliance	
PoE Power	Up to 30 W per port (Total 60 W)	Up to 30 W per port (Total 120 W)
PoE Operation	Automatic detection and power management	
PoE Pin Assignments	B Type V+(4, 5), V- (7, 8)	
PoE Output Voltage	+53 VDC	
PoE Disconnect Mode	DC Disconnect	
PoE Protection	Yes, undervoltage, overvoltage, overcurrent, overheating, and budget overages (Setting by software)	
Power		
Consumption	0.15 A @ 24 VDC without PD loading, 6 A @ 24 VDC with PD full loading (30 W per ports) 0.10 A @ 48 VDC without PD loading, 3 A @ 48 VDC with PD full loading (30 W per ports)	
Input Range	24 ~ 52 VDC Input	
Reverse Polarity	Yes	
Mechanical		
Dimensions (W x L x H)	31 mm x 134 mm x 136 mm	
Environment		
Operating Temperature	-25 ~ +60 °C	
Storage Temperature	-40 ~ +85 °C	
Humidity	10 ~ 90 % RH, Non-condensing	

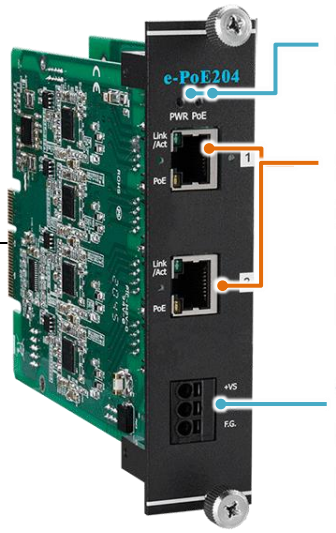
1.4. Hardware Overview

This section describes the physical layout and hardware components of the e-PoE204 and e-PoE404 modules. Understanding the function of each connector and indicator helps ensure proper installation, wiring, operation, and maintenance.
Although the e-PoE204 and e-PoE404 provide different numbers of Ethernet ports, both modules share the same hardware architecture, management functions, and interface design.
Front Panel Overview

The front panel provides Ethernet communication ports, PoE status indicators, power terminals, and system status LEDs. These components allow users to monitor module operation and connect external powered devices.
Major hardware components include:

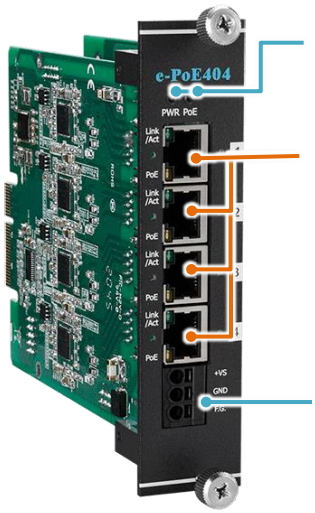
- Power LED Indicator
- PoE Power LED Indicator
- Port Status LEDs
- Link/Activity LEDs
- RJ-45 Gigabit Ethernet Ports
- External Power Terminal Block

e-PoE204



PWR: Power LED indicator
PoE: PoE Power LED Indicator
1: PoE port 1
2: PoE port 2
Link/ACT: Ethernet link status LED indicator
PoE: PoE output status LED indicator
+VS: Power Input
GND: Power supply ground
F.G. : Frame ground

e-PoE404



PWR: Power LED indicator
PoE: PoE Power LED Indicator
1: PoE port 1 **2: PoE port 2**
3: PoE port 3 **4: PoE port 4**
Link/ACT: Ethernet link status LED indicator
PoE: PoE output status LED indicator
+VS: Power Input
GND: Power supply ground
F.G. : Frame ground

LED Indicators

The LED indicators provide real-time information about the operating status of the module and each Ethernet port.

Indicator	Description
PWR	Indicates the module power status. ON indicates that the module is powered correctly.
PoE	Indicates that PoE power is available and the PoE controller is operating normally.
Port PoE Status	Indicates the PoE output status of the corresponding Ethernet port. The LED turns ON when PoE power is supplied to the connected Powered Device (PD).
Link/Act	Indicates Ethernet link status and network activity. Solid ON indicates a valid Ethernet connection. Blinking indicates network traffic.

RJ-45 Gigabit Ethernet Ports

The RJ-45 connectors provide Gigabit Ethernet communication and IEEE 802.3af/802.3at compliant PoE power output.

Each Ethernet port supports:

- 10/100/1000 Mbps auto-negotiation
- Automatic MDI/MDIX
- IEEE 802.3af/802.3at PoE
- Up to 30 W PoE output per port
- Powered Device (PD) detection and classification

Depending on the module model:

- e-PoE204: Two independent Gigabit Ethernet PoE ports
- e-PoE404: Four independent Gigabit Ethernet PoE ports

External Power Terminal Block

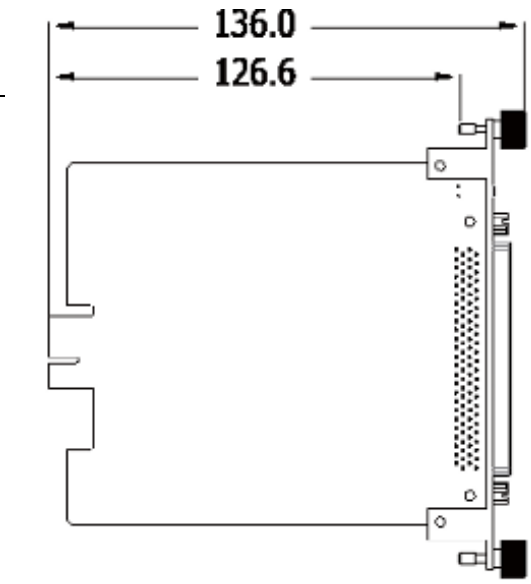
The terminal block supplies external DC power required for PoE operation.

Terminal	Description
+VS	External DC power input (24–52 VDC).
GND	Power supply ground.
F.G.	Frame ground. Connect to protective earth (PE) to improve EMC performance and reduce electrical noise.

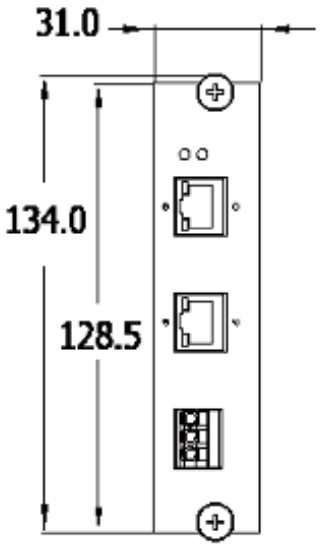
1.5. Dimension

This section provides the dimension diagrams of the e-PoE204 and e-PoE404 modules, helping users plan mechanical design and installation.

e-PoE204

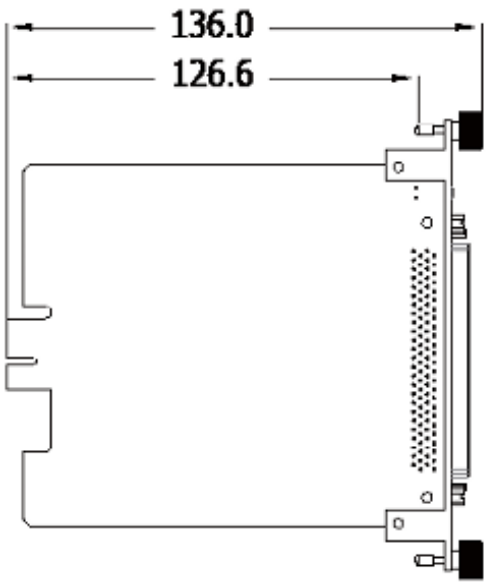


Left View

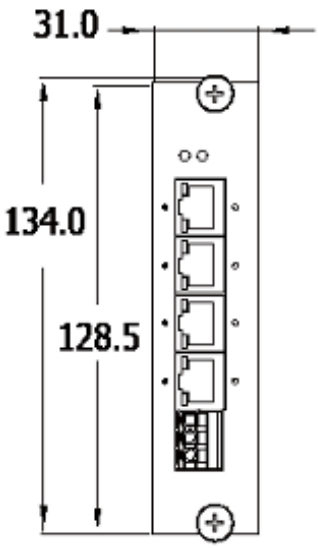


Front View

e-PoE404



Left View



Front View

2. Installation

This chapter describes the procedures required to install and prepare the e-PoE204/e-PoE404 for operation. It covers hardware installation, software installation, and SDK setup to ensure the device and management software are installed correctly before configuration and operation.

Before proceeding, ensure that all required hardware components, software packages, and development resources are available.

Before using the e-PoE204/e-PoE404, complete the following installation tasks:

- Install the hardware module.
- Connect the external power supply.
- Verify module detection.
- Install PoE Manager.
- Install the required runtime components.
- Prepare the SDK for application development.

After completing this chapter, proceed to:

Chapter 3. PoE Manager
or
Chapter 4. SDK API Reference

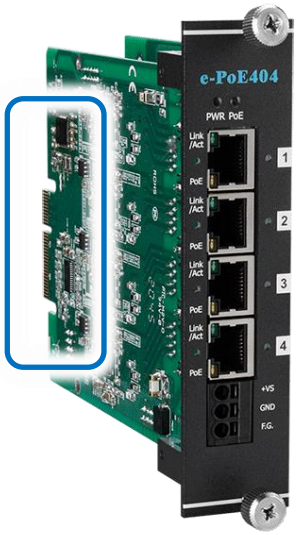
2.1. Hardware Installation

This section describes how to install the e-PoE204/e-PoE404 module into a compatible PAC system and connect the external power supply. After the installation is complete, use the PAC Utility to verify that the module has been detected successfully.

Installation Procedure

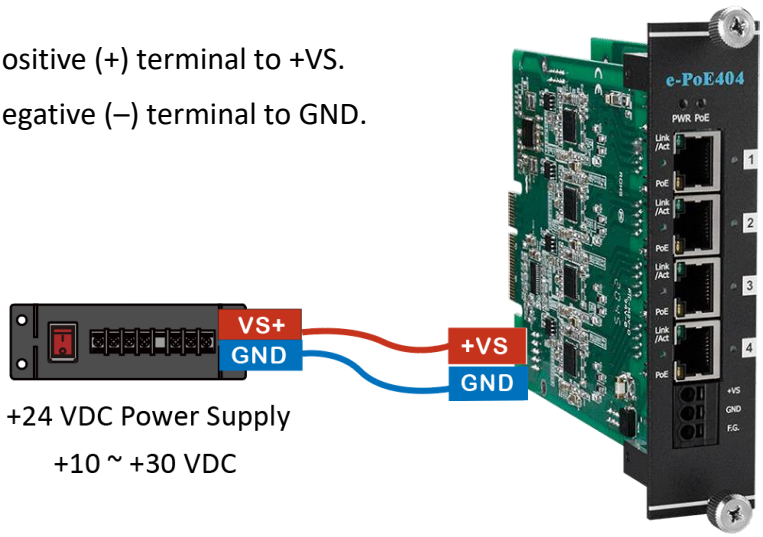
1. Power off the PAC before installing the module.
2. Align the module with the designated eBus slot, and press firmly until the module is fully seated in the connector.





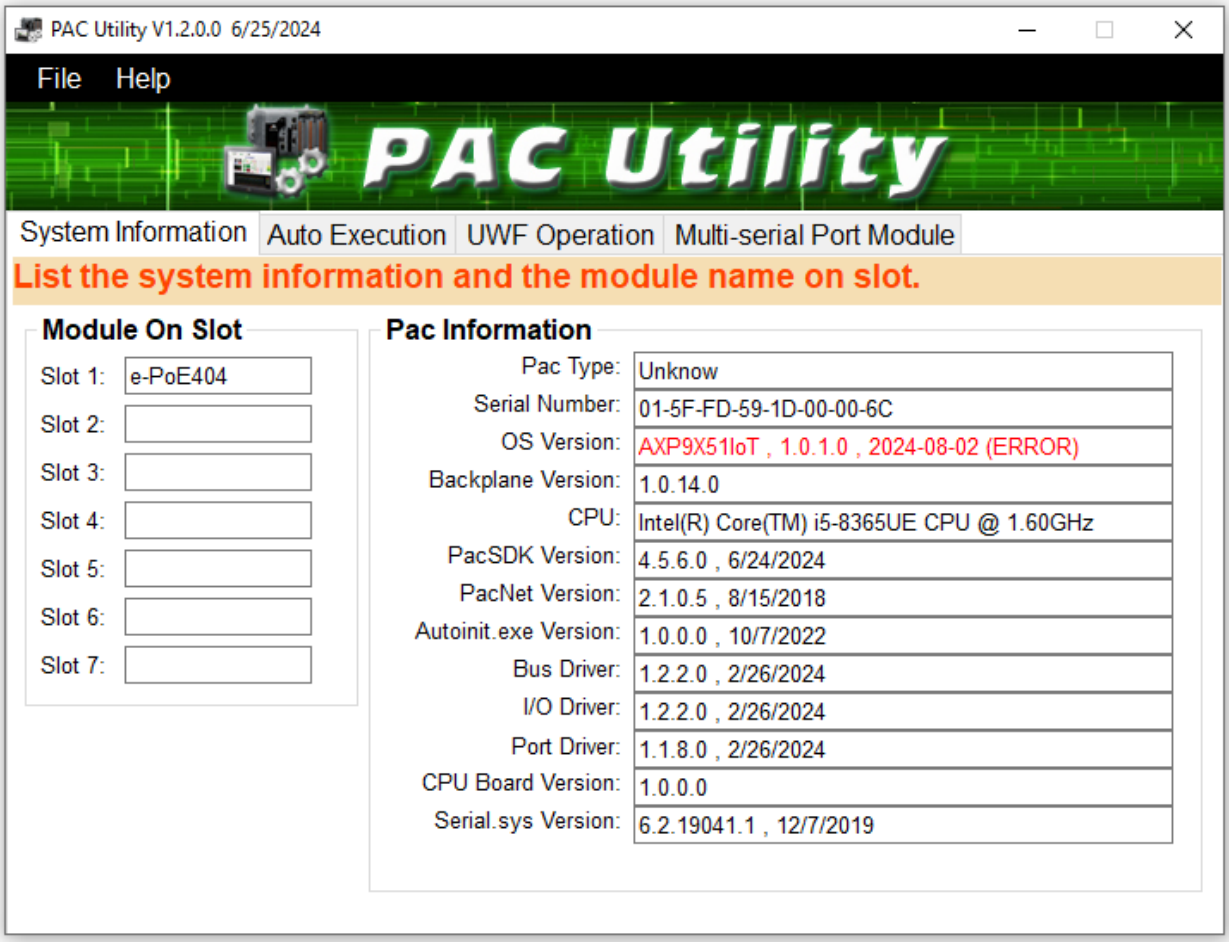
3. Connect the external DC power supply.

- Connect the positive (+) terminal to +VS.
- Connect the negative (–) terminal to GND.



4. Power on the PAC.

5. Launch PAC Utility and verify that the e-PoE204/e-PoE404 module is detected correctly.



Tips

- Always power off the PAC before installing or removing the module.
- Ensure that the module is fully inserted into the connector before applying power.
- Verify that the external power supply meets the specifications listed in the product datasheet.
- After installation, confirm that the module appears correctly in PAC Utility before proceeding with software configuration.

2.2. Software Installation

This section describes how to install and prepare PoE Manager for managing the e-PoE204/e-PoE404 devices. Before launching the utility, ensure that all required software components have been installed and that the PoE Manager package has been downloaded successfully.

PoE Manager is a Windows-based utility for configuring and monitoring the e-PoE204/e-PoE404. Before using the utility, complete the software preparation described in the following sections.

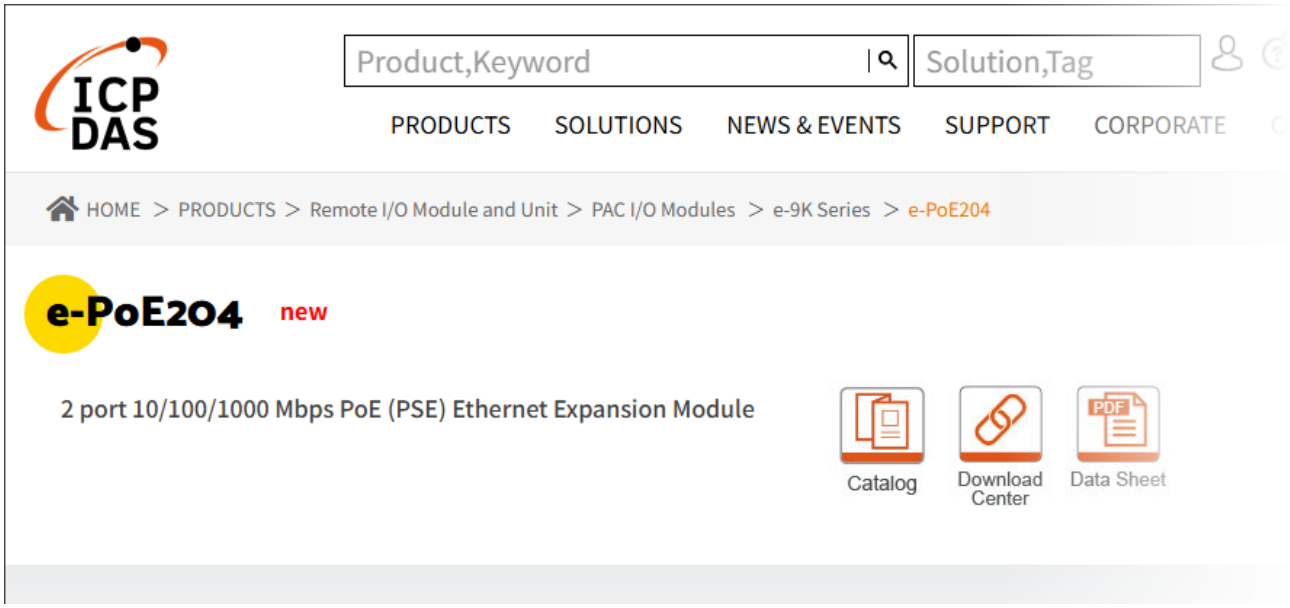
2.2.1. Obtaining PoE Manager

Download the latest PoE Manager package from the ICP DAS website.



Procedure

1. Open the product page for the e-PoE204 or e-PoE404.
2. Click Download Center



3. Locate the Utility & Tools section.

	FILE NAME	DESCRIPTION	MODEL	LAST UPDATE
	e-9K (PAC IO)	Quick Start	e-PoE204	2023-09-01

Utility & Tools

	FILE NAME	DESCRIPTION	MODEL	LAST UPDATE
	e-PoE204, e-PoE404	Utility & Tools (for Windows)	e-PoE204	2026-04-07

SDK

4. Download the latest PoE Manager package.

5. Extract the downloaded archive to a local directory.

2.2.2. Installing Required Components

Before launching PoE Manager, install the required third-party runtime components. These installation files are preloaded on the PAC and can be found in the default installation directory.

The required installation files are located in the following directory on the PAC:
C:\Program Files (x86)\Iocomp\Installer\

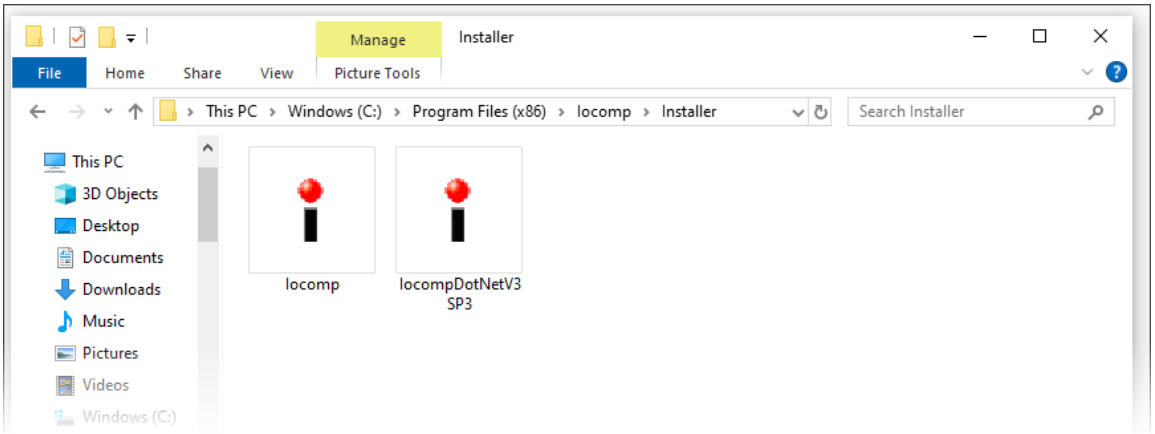
The directory contains the following installation files:

Software	Description
IocompDotNetV3SP3.exe	Installs the required Iocomp .NET runtime components.
IocompDotNetGettingStartedExamplesV4SP0VS2008CSharp.msi	Installs the required supporting components for PoE Manager.

Installation Procedure

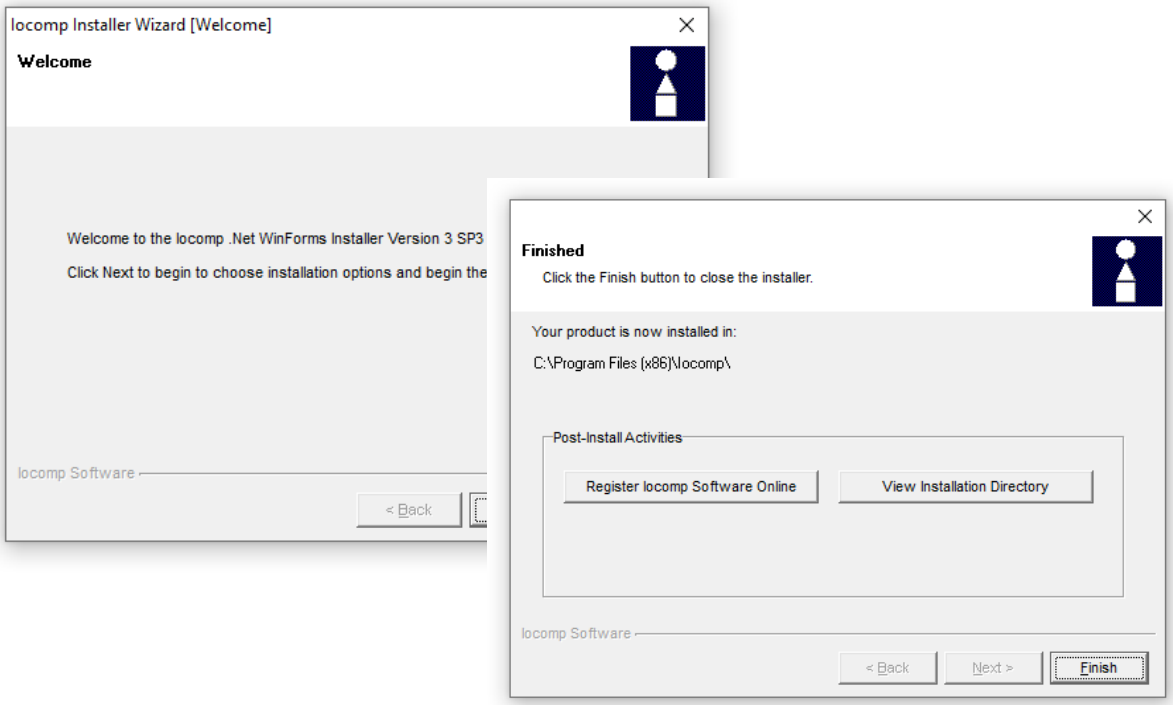
To install the required components:

1. Open the following directory:
   C:\Program Files (x86)\Iocomp\Installer\



2. Right-click IocompDotNetV3SP3.exe, and then select Run as administrator.

3. Follow the on-screen instructions to complete the installation.



4. Repeat the same procedure for IocompDotNetGettingStartedExamplesV4SP0VS2008CSharp.msi.

5. After both components have been installed successfully, launch PoE Manager.

2.2.3. Launching PoE Manager

After the required software components have been installed, PoE Manager can be launched.

Procedure

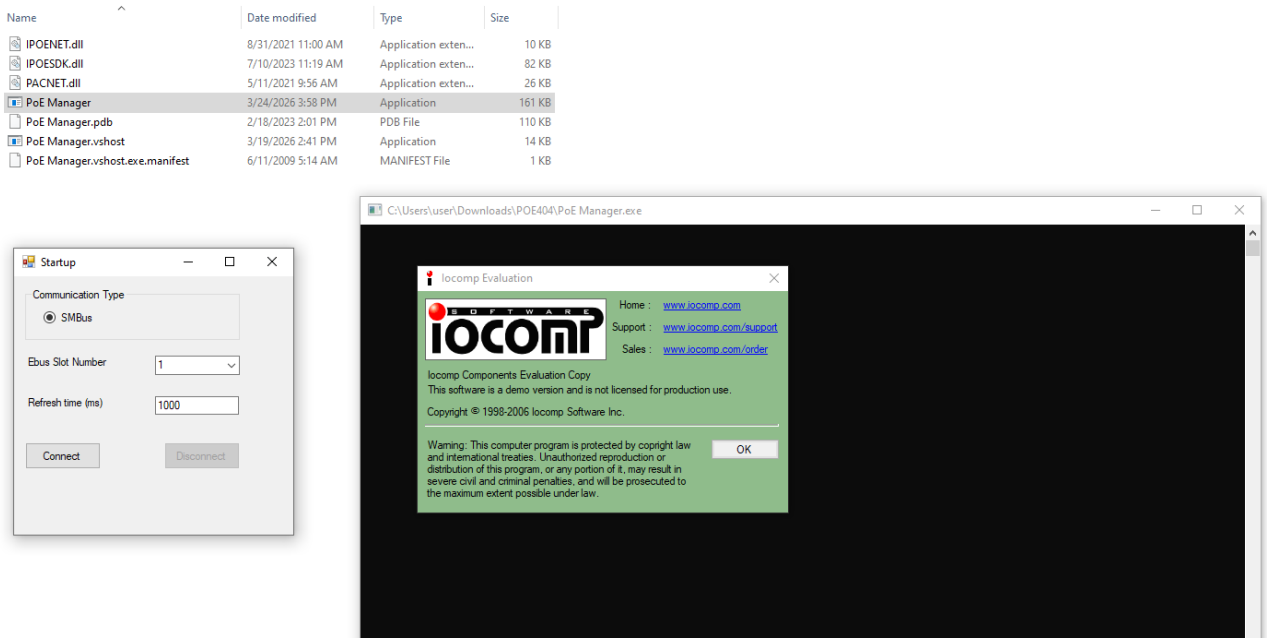
1. Open the folder containing the extracted PoE Manager package.

2. Double-click PoE Manager.exe.

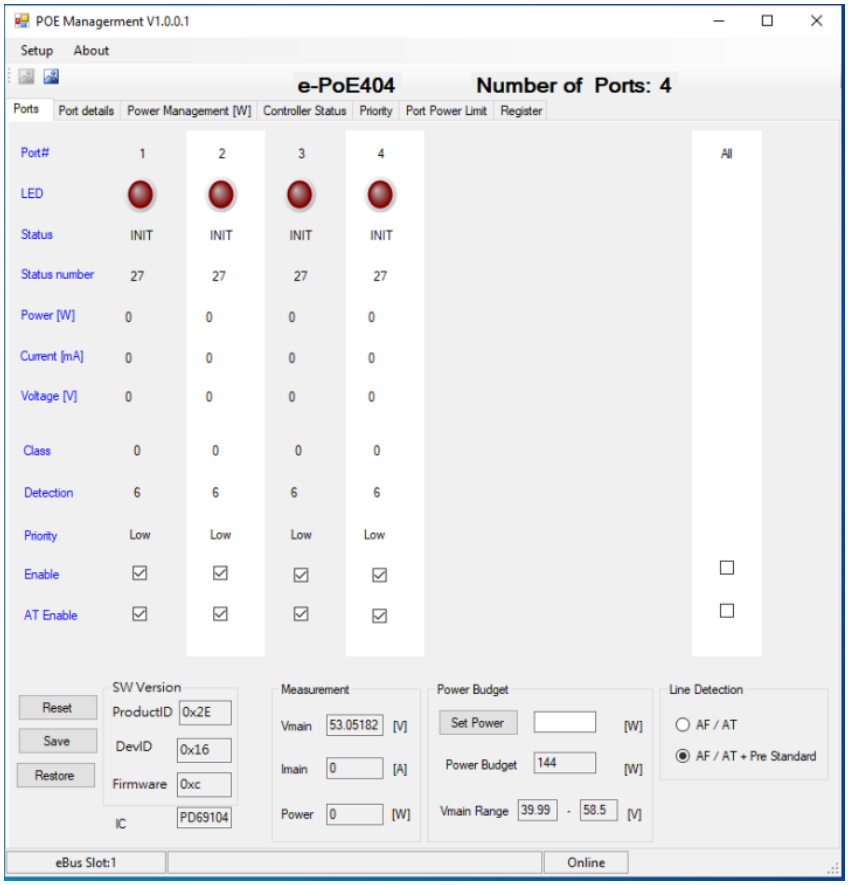
3. In the Startup dialog box, configure the communication settings:

- Enter the eBus Slot Number.
- Specify the Refresh Time if required.

4. Click Connect.



5. After a successful connection, the PoE Manager main window is displayed.



For detailed descriptions of each function page, refer to Chapter 3. PoE Manager.

2.3. SDK Preparation

This section describes how to prepare the Software Development Kit (SDK) for application development using the e-PoE204/e-PoE404 module.

The e-PoE SDK is an extension module of the PAC SDK. It provides dedicated APIs for PoE configuration, monitoring, and power management while utilizing the PAC SDK communication framework.

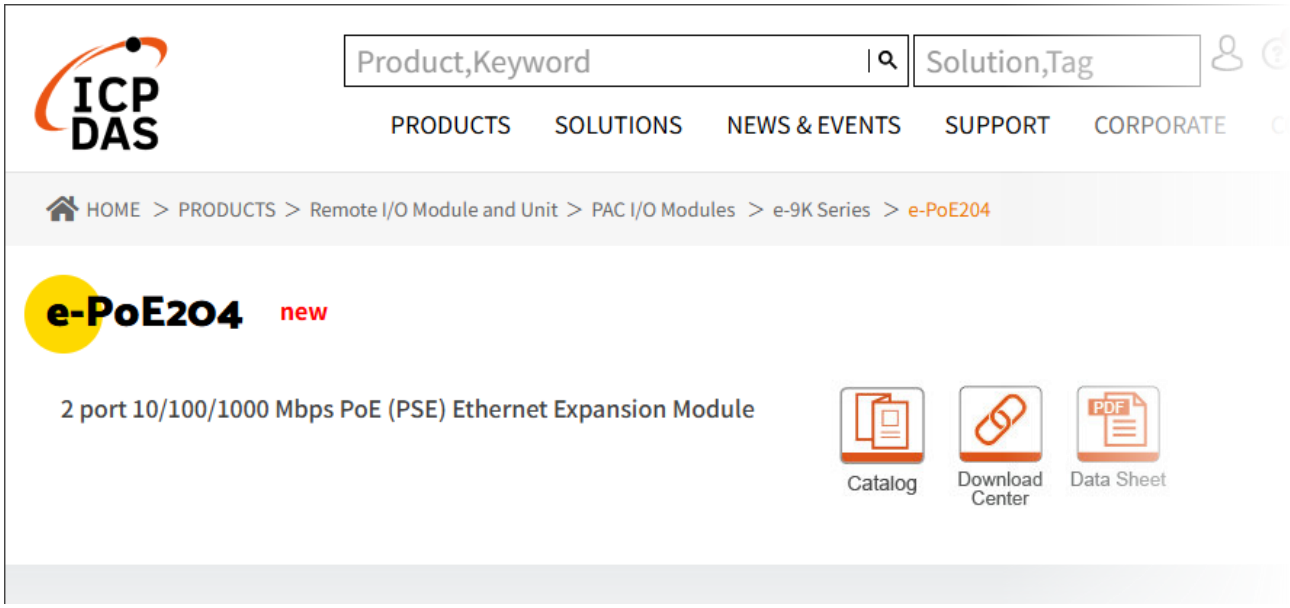
After completing this section, you can proceed to Chapter 4. SDK API Reference for detailed programming information.

2.3.1. Obtaining the SDK

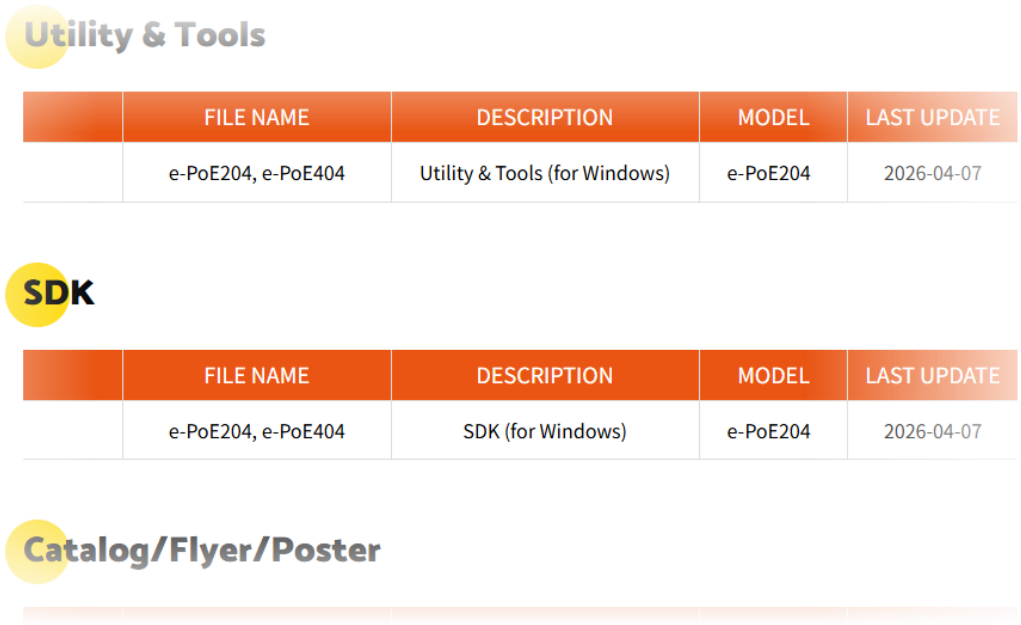
Download the SDK package from the ICP DAS product website.

Procedure

1. Open the product page for the e-PoE204 or e-PoE404.
2. Click Download Center



3. Locate the SDK section.



Utility & Tools

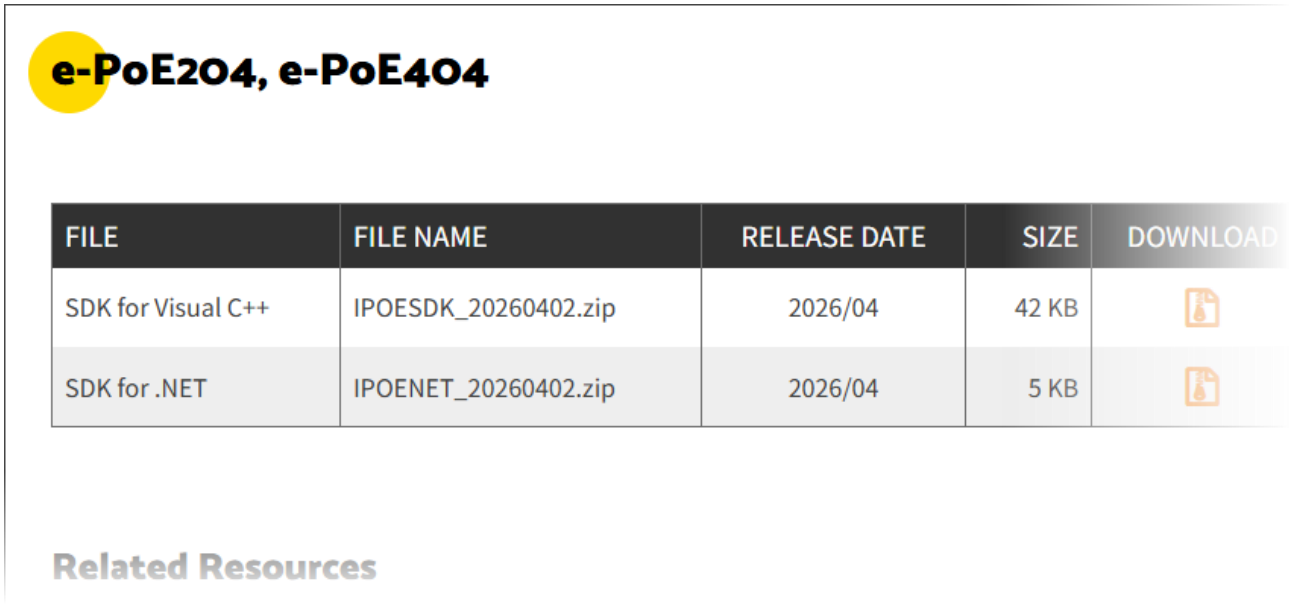
	FILE NAME	DESCRIPTION	MODEL	LAST UPDATE
	e-PoE204, e-PoE404	Utility & Tools (for Windows)	e-PoE204	2026-04-07

SDK

	FILE NAME	DESCRIPTION	MODEL	LAST UPDATE
	e-PoE204, e-PoE404	SDK (for Windows)	e-PoE204	2026-04-07

Catalog/Flyer/Poster

4. Download the SDK package corresponding to your development platform.



e-PoE204, e-PoE404

FILE	FILE NAME	RELEASE DATE	SIZE	DOWNLOAD
SDK for Visual C++	IPOESDK_20260402.zip	2026/04	42 KB	
SDK for .NET	IPOENET_20260402.zip	2026/04	5 KB	

Related Resources

5. Extract the downloaded archive to a local directory.

2.3.2. SDK Packages

Separate SDK packages are provided for different programming environments.

SDK Package	Description
SDK for Visual C++	Provides IPOESDK.dll for native Windows application development.
SDK for .NET	Provides IPOENET.dll for .NET application development.

2.3.3. Preparing the SDK

After downloading the SDK package, prepare your development environment before calling any e-PoE APIs.

Procedure

1. Extract the downloaded SDK package.
2. Select the SDK package corresponding to your development environment.
3. Copy the required DLL into your project or application directory.
4. Add a reference to the DLL in your development environment.
5. Build the application.
6. Verify that the DLL is loaded successfully before calling any e-PoE APIs.

After the SDK has been prepared successfully, refer to Chapter 4. SDK API Reference for API descriptions, programming workflow, function reference, return values, and programming examples.

3. PoE Manager

This chapter describes the graphical user interface (GUI) of PoE Manager and its configuration functions.

PoE Manager provides a centralized interface for monitoring and configuring the e-PoE204/e-PoE404. Each configuration page focuses on a specific management function, allowing users to monitor port status, configure operating parameters, and manage PoE power efficiently.

The following sections describe each configuration page, its parameters, operating procedures, and related notes.

3.1. Overview

PoE Manager is a Windows-based management utility designed for configuring and monitoring the e-PoE204/e-PoE404.

After connecting to the target device, users can monitor the operating status of each PoE port, configure power management parameters, adjust port priorities, set power limits, monitor controller status, and access register information through an intuitive graphical user interface.

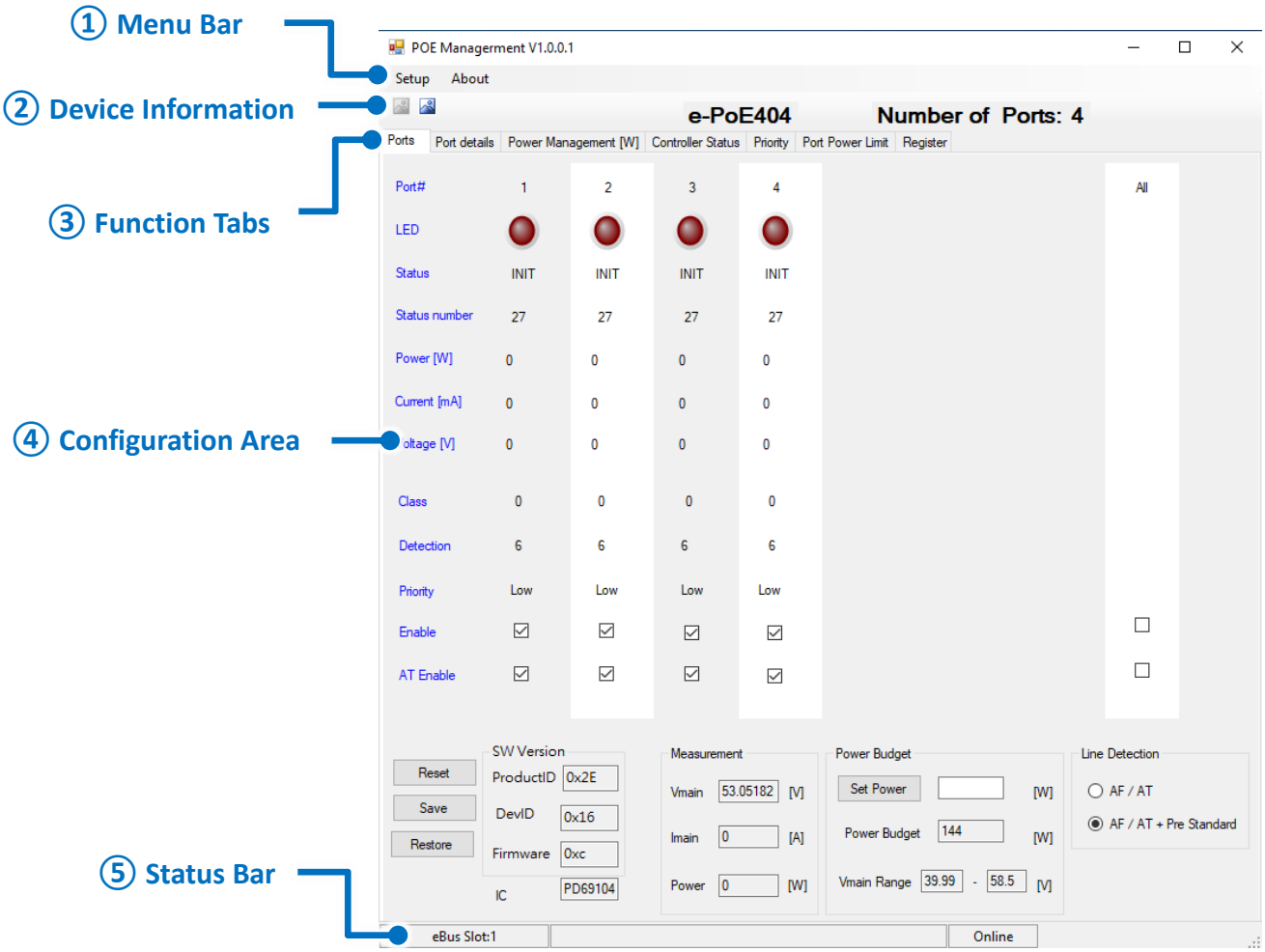
The main functions provided by PoE Manager include:

- Monitoring PoE port status
- Configuring PoE parameters
- Managing power allocation
- Configuring port priorities
- Configuring port power limits
- Monitoring controller status
- Reading register information

3.2. Main Window

After a successful connection, the PoE Manager main window is displayed.

The main window provides access to all monitoring and configuration functions through a tab-based interface. Users can switch between different pages to monitor port status, configure power management settings, and view controller information.



Main Window Components

Item	Description
① Menu Bar	Displays the available menu commands.
② Device Information	Displays the connected device model and the number of available PoE ports.
③ Function Tabs	Provides access to different monitoring and configuration pages.
④ Configuration Area	Displays the settings and operating information for the selected function page.
⑤ Status Bar	Displays the connection status and current eBus slot information.

Navigation

Select the desired function tab to access the corresponding configuration page.

The available function tabs include:

- Ports
- Port Details
- Power Management
- Controller Status
- Priority
- Port Power Limit
- Register

Detailed descriptions of each function page are provided in the following sections.

3.3. User Interface

The PoE Manager user interface is organized into multiple function tabs, each designed for a specific monitoring or configuration task. This tab-based layout enables users to quickly switch between different management functions while maintaining a consistent operating experience.

Each function page displays related operating information and configuration parameters for the e-PoE204/e-PoE404. Depending on the selected page, users can monitor PoE port status, configure power management settings, adjust port priorities, set power limits, monitor controller status, or access register information.

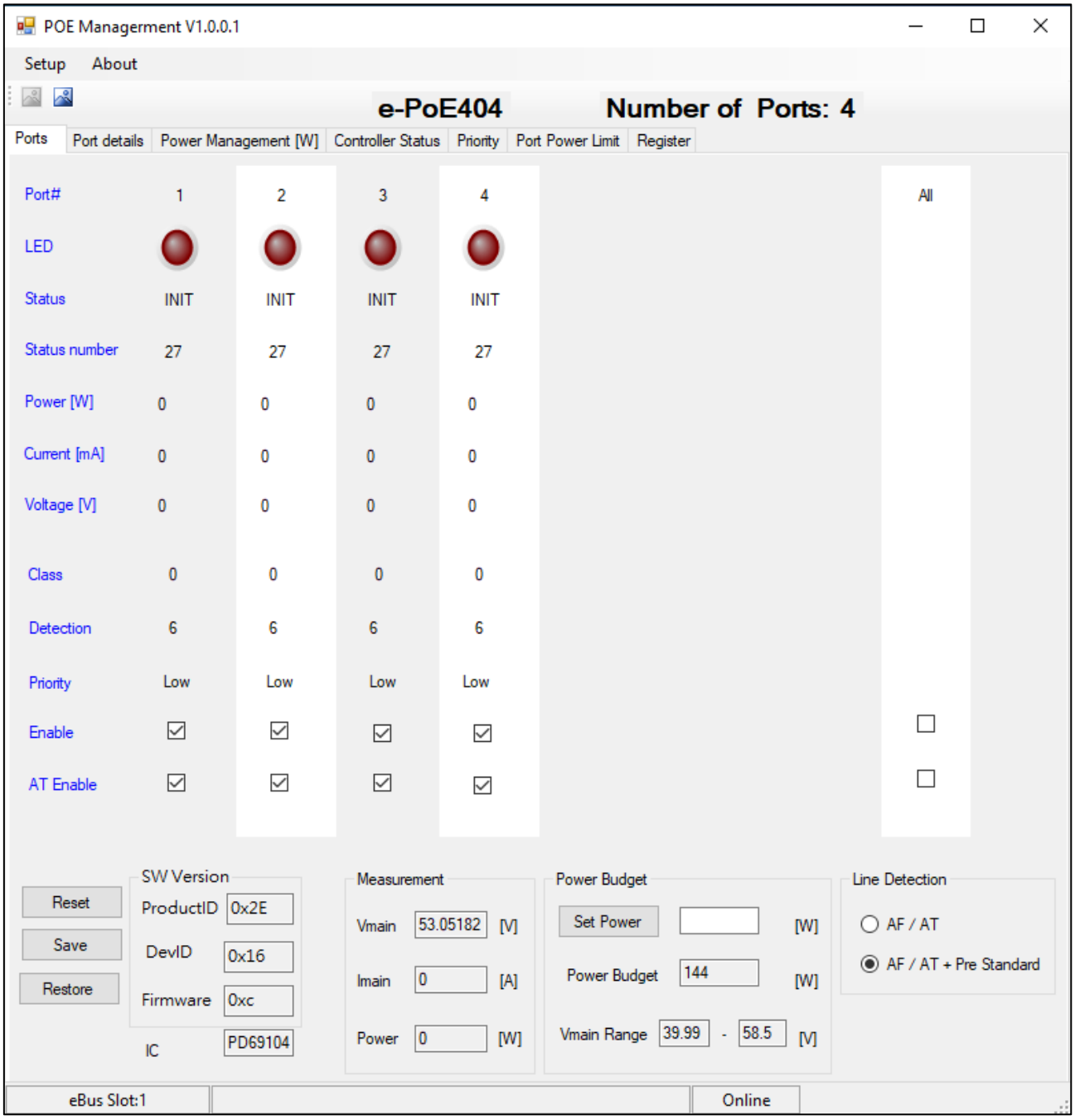
The following sections describe each function page, including its purpose, available parameters, operating procedures, and related notes.

Available Function Pages

Function Page	Description
Ports	Displays the operating status and power information for all PoE ports.
Port Details	Displays detailed operating information for the selected PoE port.
Power Management	Configures system power management parameters.
Controller Status	Displays the operating status of the PoE controller.
Priority	Configures the priority level for each PoE port.
Port Power Limit	Configures the maximum power output for each PoE port.
Register	Displays register information for monitoring and diagnostic purposes.

3.3.1. Ports

The Ports page provides a centralized view for monitoring and configuring all PoE ports. Users can view the operating status, electrical measurements, power configuration, and controller information for each port. The page also provides functions for enabling or disabling PoE output, configuring the system power budget, and selecting the PD detection mode.



The Ports page consists of the following functional areas:

- Port Status
- Device Information
- Measurement
- Power Configuration
- Line Detection

Port Status

The Port Status area displays the real-time operating information of each PoE port.

Item	Description
Port#	Displays the PoE port number.
LED	Indicates the current LED status of the port.
Status	Displays the current operating status of the PoE port.
Status Number	Displays the controller status code.
Power [W]	Displays the output power.
Current [mA]	Displays the output current.
Voltage [V]	Displays the output voltage.
Class	Displays the detected IEEE 802.3 power class.
Detection	Displays the PD detection result.
Priority	Displays the configured priority level.
Enable	Enables or disables PoE output.
AT Enable	Enables IEEE 802.3at mode.

Device Information

The SW Version area displays the controller identification and firmware information.

Item	Description
ProductID	Product identification code.
DevID	Device identification code.
Firmware	Firmware version.
IC	PoE controller model.

Measurement

The Measurement area displays the electrical measurements of the PoE controller.

Item	Description
Vmain	Main input voltage.
Imain	Main input current.
Power	Total input power.

Power Configuration

The Power Budget area is used to configure the maximum available PoE power.

Item	Description
Set Power	Sets the maximum system output power.
Power Budget	Displays the configured power budget.
Vmain Range	Displays the supported input voltage range.

Line Detection

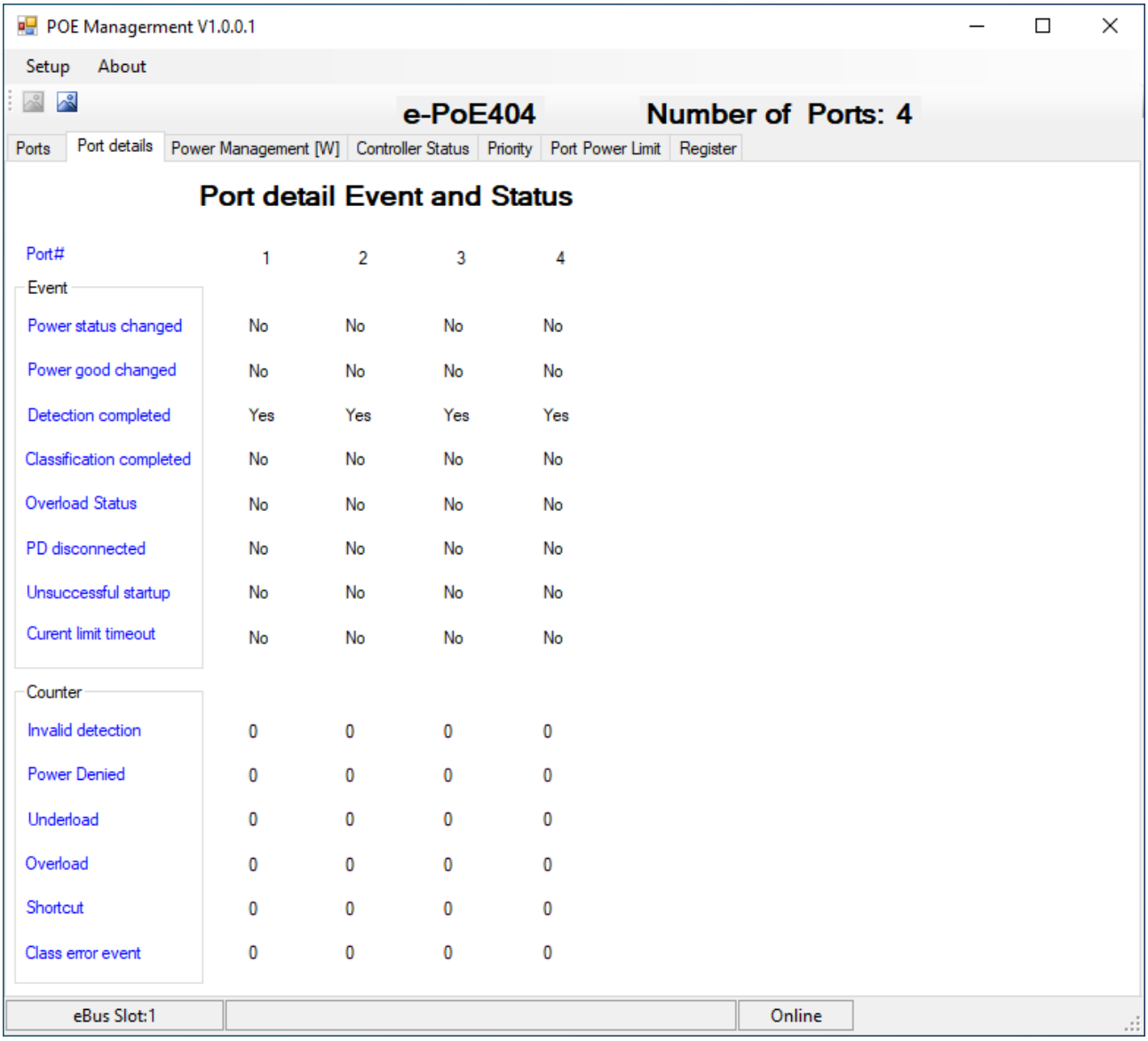
The Line Detection area selects the Powered Device (PD) detection mode.

Item	Description
AF / AT	Detects IEEE 802.3af/at devices only.
AF / AT + Pre Standard	Detects IEEE 802.3af/at and legacy pre-standard devices.

3.3.2. Port Details

The Port Details page provides detailed event status and fault statistics for each PoE port. It allows users to monitor port events, identify abnormal operating conditions, and review fault counters for diagnostic and troubleshooting purposes.

The displayed information is updated periodically according to the configured refresh interval.



The Port Details page consists of two information sections:

- Event – Displays the current event status for each PoE port.
- Counter – Displays the accumulated event counters for diagnostic purposes.

Each column represents a PoE port, and each row represents a specific event or counter.

Event Status

The Event section displays the current operating events detected for each PoE port.

Item	Description
Power Status Changed	Indicates whether the port power status has changed.
Power Good Changed	Indicates whether the Power Good status has changed.
Detection Completed	Indicates whether PD detection has completed successfully.
Classification Completed	Indicates whether PD classification has completed successfully.
Overload Status	Indicates whether an overload condition is present.
PD Disconnected	Indicates whether the connected Powered Device (PD) has been disconnected.
Unsuccessful Startup	Indicates whether PoE startup failed.
Current Limit Timeout	Indicates whether the port entered current limit protection.

Event Counters

The Counter section records the accumulated number of specific fault events for each PoE port.

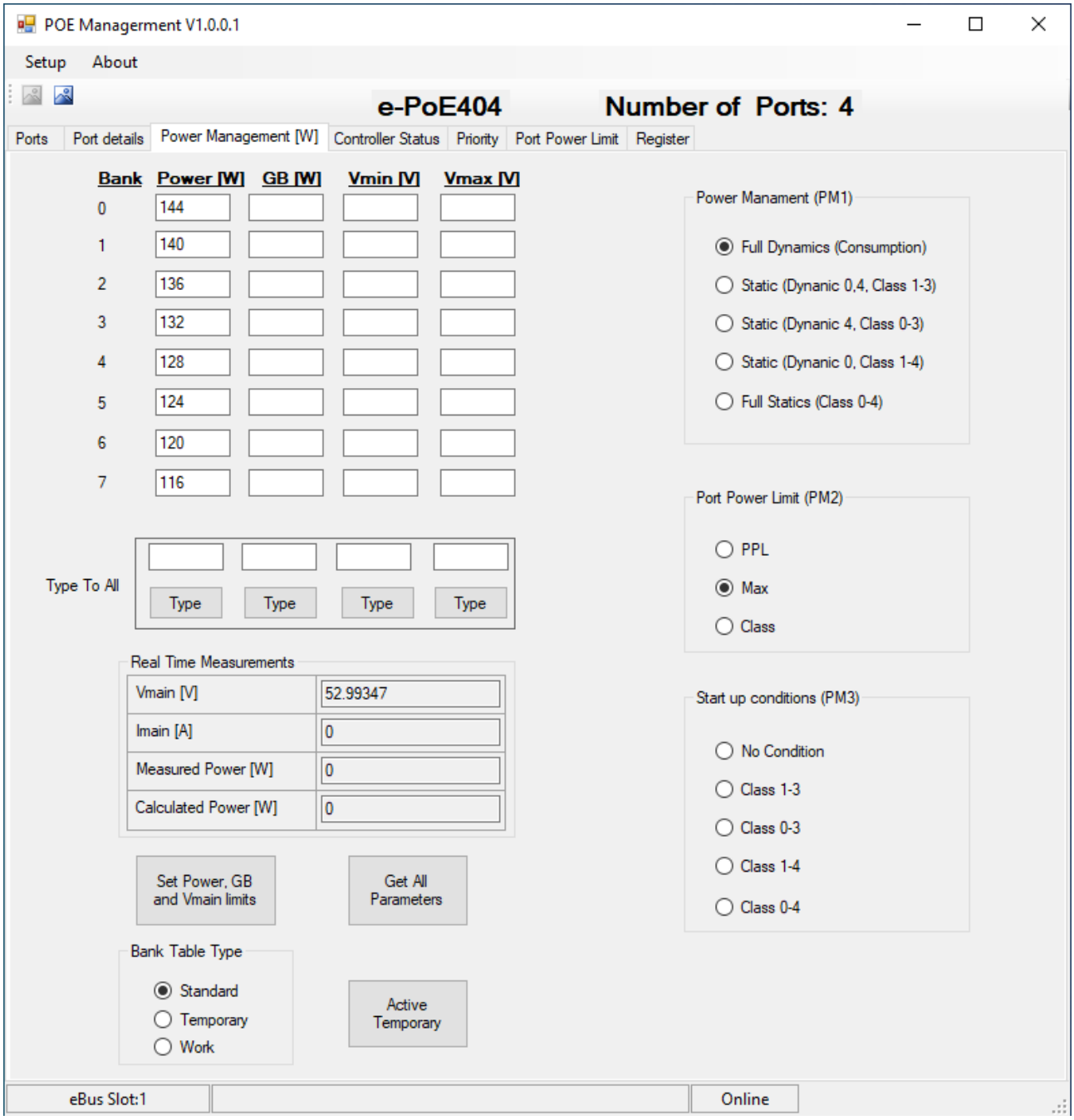
These counters help identify recurring issues and assist in system diagnostics.

Item	Description
Invalid Detection	Number of invalid PD detection events.
Power Denied	Number of times power delivery was denied.
Underload	Number of underload events.
Overload	Number of overload events.
Shortcut	Number of short-circuit events.
Class Error Event	Number of PD classification errors.

3.3.3. Power Management

The Power Management page is used to configure the global power management parameters of the e-PoE204/e-PoE404. It provides functions for configuring power budget banks, selecting the power management mode, defining port power limit behavior, configuring startup conditions, and monitoring real-time input power measurements.

These settings determine how the PoE controller allocates available power to connected Powered Devices (PDs) under different operating conditions.



The Power Management page consists of the following functional areas:

- Power Bank Configuration
- Power Management Mode (PM1)
- Port Power Limit Mode (PM2)
- Startup Conditions (PM3)
- Real-Time Measurements

Power Bank Configuration

The Power Bank Configuration section is used to configure multiple power management profiles. Each bank stores an independent set of power-related parameters.

Item	Description
Bank	Displays the Power Bank number.
Power [W]	Configures the available power budget for the selected bank.
GB [W]	Configures the Guard Band value for the selected bank.
Vmin [V]	Specifies the minimum input voltage threshold.
Vmax [V]	Specifies the maximum input voltage threshold.

Power Management Mode (PM1)

The PM1 section selects the power allocation strategy used by the PoE controller.
Available modes include:

Item	Description
Full Dynamics (Consumption)	Allocates power according to actual power consumption.
Static (Dynamic 0.4, Class 1-3)	Uses a predefined allocation strategy based on PD classification.
Static (Dynamic 4, Class 0-3)	Uses a fixed allocation strategy for supported PD classes.
Static (Dynamic 0, Class 1-4)	Uses predefined static allocation parameters.
Full Statics (Class 0-4)	Allocates power entirely according to the configured power classes.

Port Power Limit Mode (PM2)

The PM2 section defines how the maximum output power of each PoE port is determined.

Item	Description
PPL	Uses the configured Port Power Limit.
Max	Uses the maximum available output power.
Class	Determines the output power according to the detected PD class.

Startup Conditions (PM3)

The PM3 section defines the startup conditions for PoE power allocation.

Item	Description
No Condition	No startup restriction.
Class 1-3	Applies startup conditions for Class 1–3 devices.
Class 0-3	Applies startup conditions for Class 0–3 devices.
Class 1-4	Applies startup conditions for Class 1–4 devices.
Class 0-4	Applies startup conditions for Class 0–4 devices.

Real-Time Measurements

The Real-Time Measurements section displays the current electrical measurements of the input power source.

Item	Description
Vmain [V]	Displays the input voltage.
Imain [A]	Displays the input current.
Measured Power [W]	Displays the measured input power.
Calculated Power [W]	Displays the calculated input power.

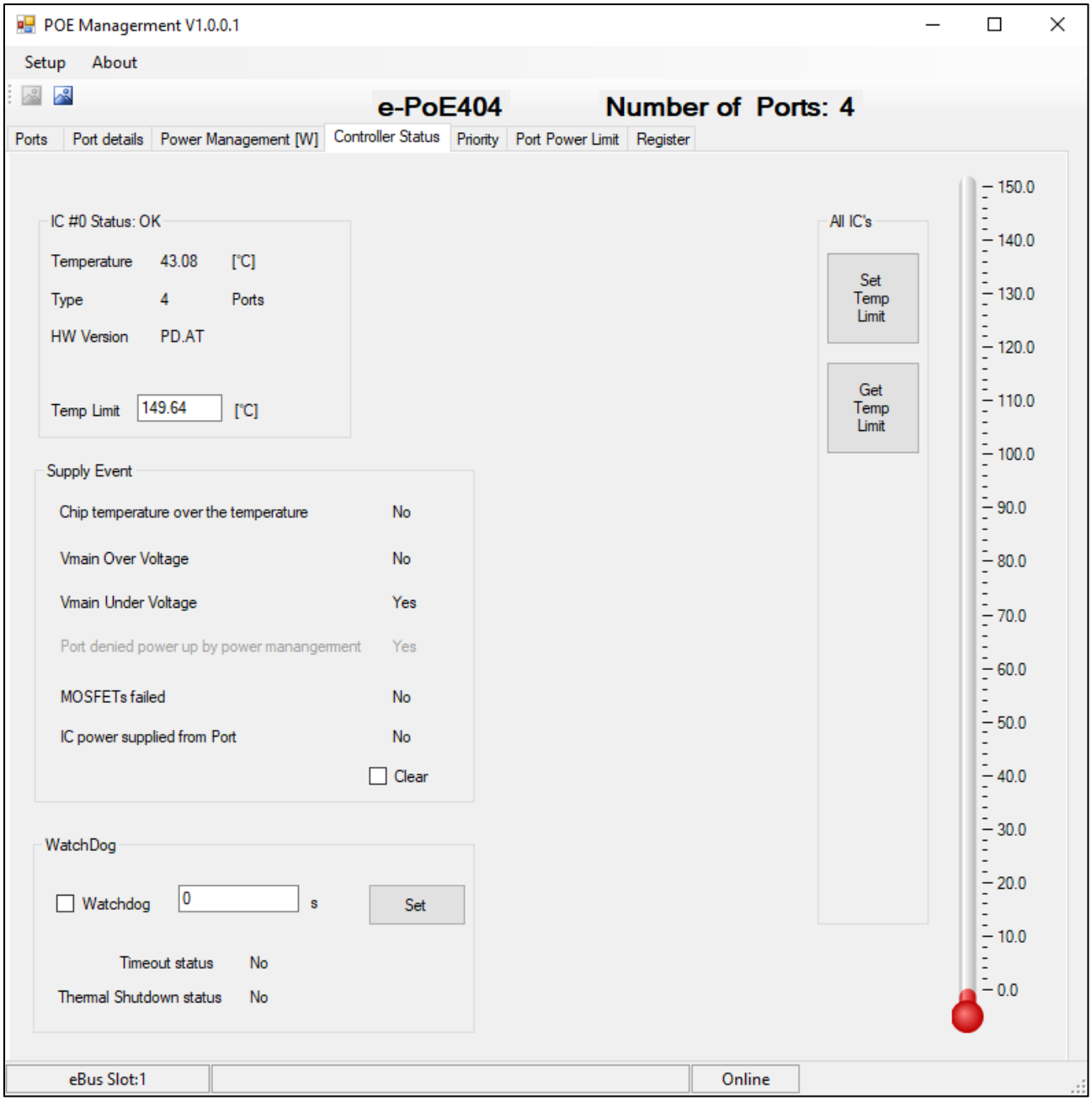
Operation Buttons

Button	Description
Set Power, GB and Vmain Limits	Applies the configured Power, Guard Band, and voltage limit settings.
Get All Parameters	Reads the current power management configuration from the device.
Active Temporary	Activates the temporary Power Bank configuration.

3.3.4. Controller Status

The Controller Status page displays the operating status and health information of the PoE controller. It provides controller identification, temperature monitoring, power supply event status, and watchdog information to assist users in monitoring system operation and diagnosing abnormal conditions.

Users can also configure the controller temperature limit and monitor controller protection events through this page.



The Controller Status page consists of the following functional areas:

- Controller Information
- Supply Event
- Watchdog
- Temperature Limit

Controller Information

The Controller Information section displays the current operating information of the PoE controller.

Item	Description
IC Status	Displays the current controller operating status.
Temperature [°C]	Displays the controller temperature.
Type	Displays the controller type or supported port configuration.
HW Version	Displays the hardware version of the controller.
Temp Limit [°C]	Displays or configures the controller temperature limit.

Supply Event

The Supply Event section displays the current power-related event status reported by the controller.

Item	Description
Chip Temperature Over the Temperature	Indicates whether the controller temperature exceeds the configured limit.
Vmain Over Voltage	Indicates whether the input voltage exceeds the allowable range.
Vmain Under Voltage	Indicates whether the input voltage is below the allowable range.
Port Denied Power Up by Power Management	Indicates whether PoE power-up was prevented by the configured power management policy.
MOSFETs Failed	Indicates whether a MOSFET fault has been detected.
IC Power Supplied from Port	Indicates whether the controller is powered through a PoE port.
Clear	Clears the displayed event status.

Watchdog

The Watchdog section is used to configure and monitor the controller watchdog timer.

Item	Description
Watchdog	Enables or disables the watchdog timer.
Timeout (s)	Specifies the watchdog timeout period in seconds.
Set	Applies the watchdog configuration.
Timeout Status	Displays whether a watchdog timeout has occurred.
Thermal Shutdown Status	Displays whether the controller has entered thermal shutdown

Temperature Limit

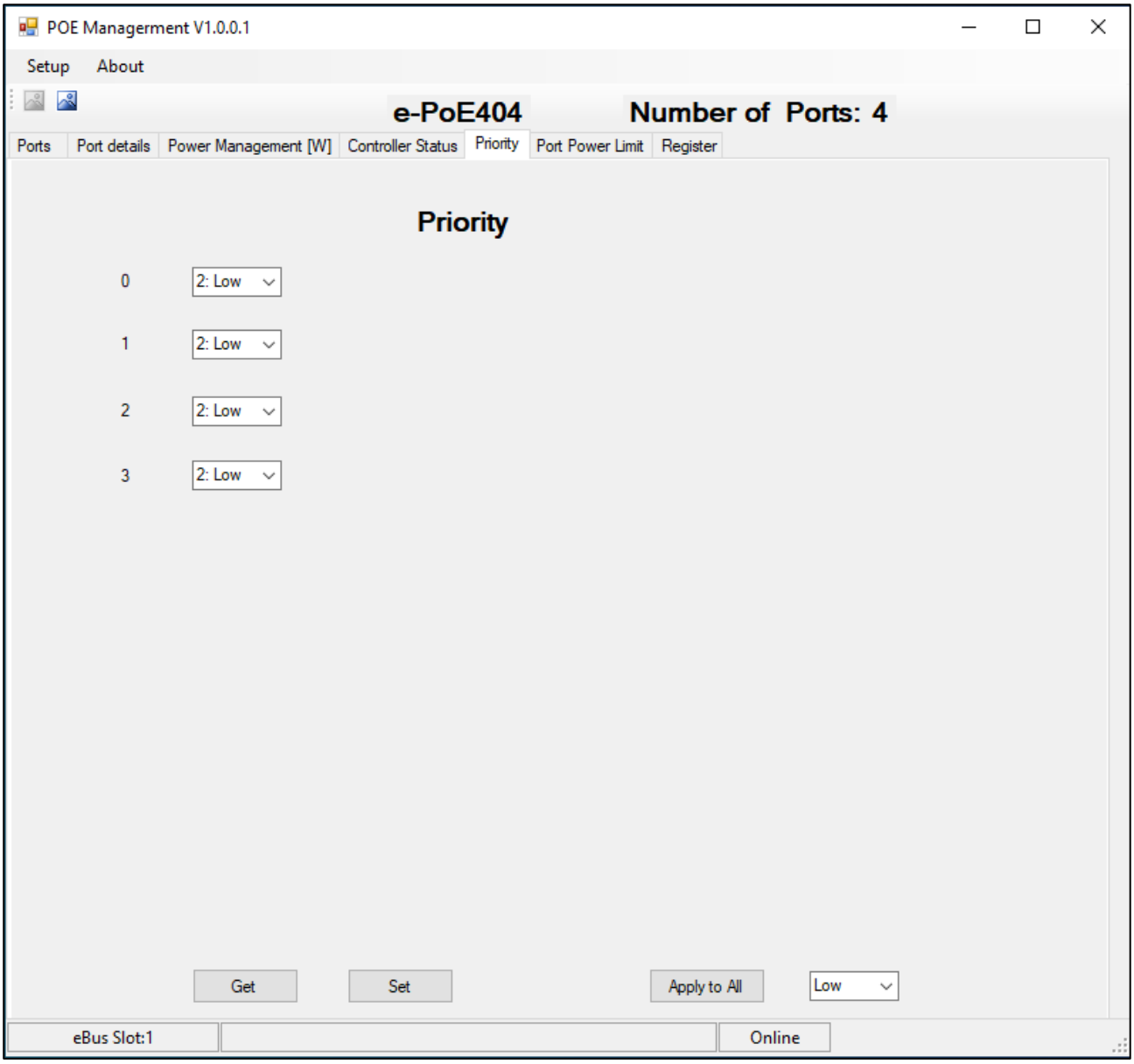
The Temperature Limit section is used to configure or retrieve the controller temperature threshold.

Item	Description
Set Temp Limit	Applies the configured temperature limit.
Get Temp Limit	Reads the current temperature limit from the controller.

3.3.5. Priority

The Priority page is used to configure the power priority for each PoE port. When the available power is insufficient to supply all connected Powered Devices (PDs), the PoE controller allocates power according to the configured priority levels.

Users can configure each port individually or apply the same priority level to all ports simultaneously.



The Priority page provides individual priority settings for each PoE port and supports applying the same priority level to all ports.

The configured priority affects the power allocation order when the available power budget is limited.

Port Priority Configuration

Each PoE port can be assigned an independent priority level.

Item	Description
Port	Displays the PoE port number.
Priority	Selects the priority level for the specified port.

Global Priority Configuration

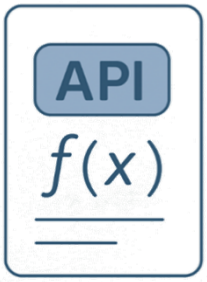
The Apply to All function applies the selected priority level to all PoE ports simultaneously.

Item	Description
Apply to All	Applies the selected priority level to every PoE port.
Priority List	Selects the priority level to be applied.

Operation Buttons

Item	Description
Get	Reads the current priority settings from the device.
Set	Writes the configured priority settings to the device.

4. SDK API Reference



This chapter describes the programming interfaces provided by the e-PoE SDK for developing applications that communicate with e-PoE204 and e-PoE404 modules installed on an ICP DAS PAC controller.

The e-PoE SDK extends the PAC Development SDK by providing dedicated APIs for configuring PoE functions, controlling power delivery, monitoring operating status, and accessing controller information. These APIs enable developers to integrate PoE management capabilities into custom applications using standard programming languages such as C/C++ and Microsoft .NET.

This chapter introduces the SDK architecture, recommended programming workflow, API function categories, detailed API descriptions, and return values. Each API includes its function prototype, parameter descriptions, return value, remarks, and programming examples to facilitate application development.

Before using the APIs described in this chapter, ensure that the PAC Development SDK and the e-PoE SDK have been installed and configured according to the instructions in Section 2.3 SDK Preparation.

Chapter Contents

Section	Description
4.1 Overview	Introduces the SDK relationship, runtime libraries, and supported development environments.
4.2 Programming Workflow	Describes the recommended API initialization and programming sequence.
4.3 Function Categories	Lists all API functions grouped by functional category.
4.4 API Reference	Provides detailed descriptions of each API, including syntax, parameters, return values, remarks, and examples.

4.1. Overview

Unlike a standalone Software Development Kit (SDK), the e-PoE SDK is designed as an extension library for the PAC Development SDK. It works together with the PAC SDK to provide PoE management capabilities, including controller configuration, port management, power control, status monitoring, watchdog management, and register access.

The e-PoE SDK supports both native C/C++ and Microsoft .NET application development through dedicated runtime libraries. Developers can integrate PoE control functions into their own applications by calling the provided APIs.

Before using the APIs described in this chapter, ensure that the PAC Development SDK and the e-PoE SDK have been prepared according to the instructions in Section 2.3 SDK Preparation.

4.1.1. SDK Relationship

The e-PoE SDK extends the functionality of the PAC Development SDK by providing APIs specifically designed for controlling e-PoE modules.

User applications communicate with the PAC controller through the PAC SDK, while the e-PoE SDK provides the software interface required to access PoE-related functions.

4.1.2. e-PoE SDK Libraries

The e-PoE SDK provides runtime libraries for different programming environments.

SDK Package	Description
IPOESDK.dll	Native C/C++ runtime library for e-PoE application development.
IPOENET.dll	.NET runtime library for C# and VB.NET application development.

4.1.3. Supported Development Environment

The e-PoE SDK supports the following programming environments.

Programming Language	Runtime Library
C / C++	IPOESDK.dll
C#	IPOENET.dll
VB.NET	IPOENET.dll

4.2. Programming Workflow

This section describes the recommended programming workflow when developing applications with the e-PoE SDK.

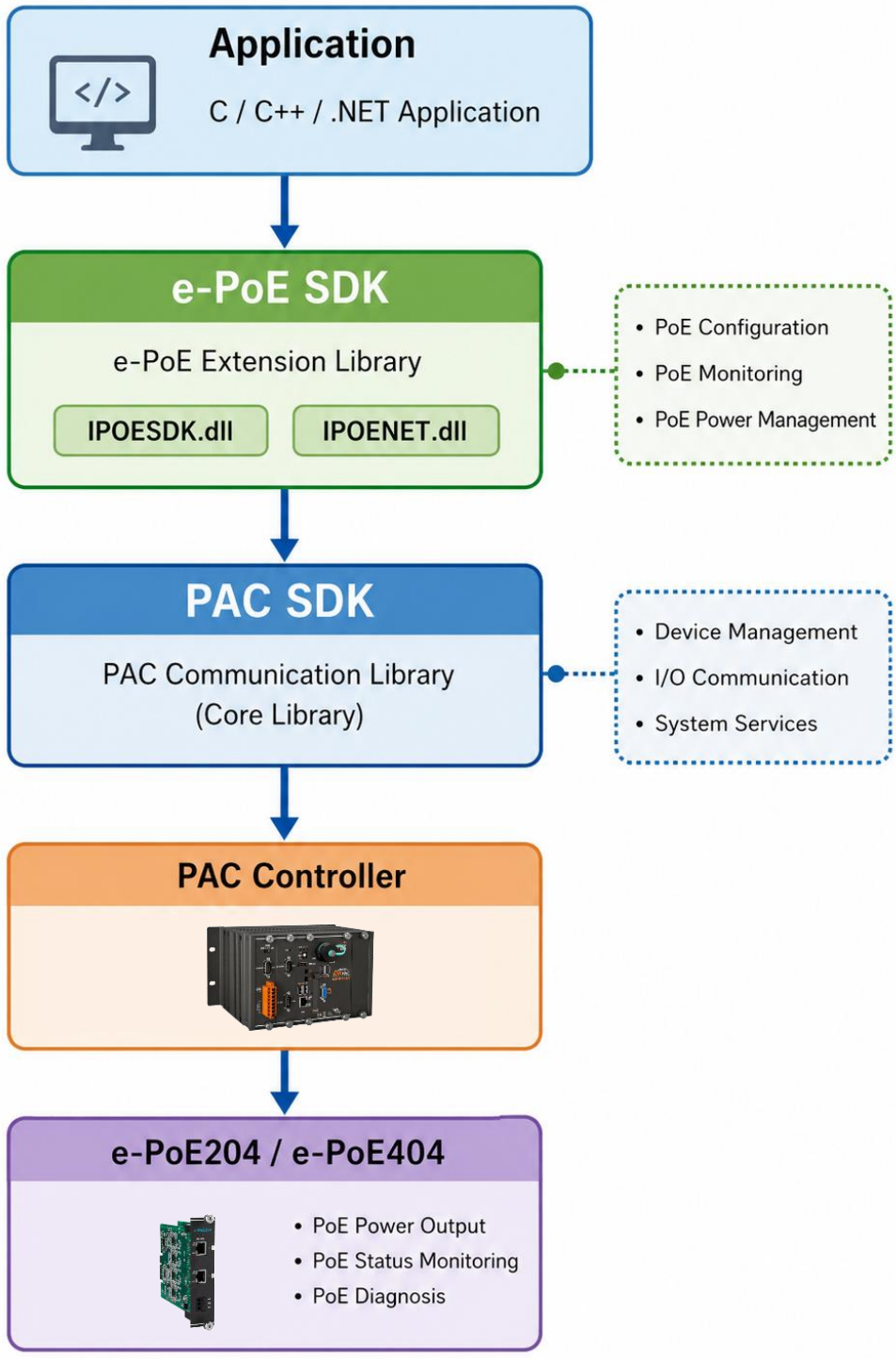
Since the e-PoE SDK is an extension library for the PAC Development SDK, applications should first establish communication with the PAC controller before accessing e-PoE-specific APIs.

Following the recommended programming sequence helps ensure proper initialization, reliable communication, and stable operation of the application.

4.2.1. Development Architecture

The e-PoE SDK operates as an extension layer of the PAC Development SDK.

Applications access the PAC controller through the PAC SDK, while the e-PoE SDK provides dedicated APIs for configuring and monitoring e-PoE modules.



4.2.2. SDK Initialization

Before calling any e-PoE API, the application should complete the following initialization procedure.

1. Initialize the PAC Development SDK.
2. Open communication with the PAC controller.
3. Initialize the e-PoE SDK.
4. Verify communication with the e-PoE module.
5. Begin PoE configuration and monitoring.

Failure to complete the initialization procedure may cause subsequent API calls to fail.

4.3. Function Categories

The e-PoE SDK organizes API functions into functional categories according to their intended purpose. This classification enables developers to quickly locate the appropriate APIs for controller configuration, PoE management, monitoring, and system maintenance.

Detailed descriptions of each API are provided in Section 4.4 API Reference.

4.3.1. System Functions

System functions provide basic operations for SDK initialization, resource management, version information, and controller identification.

Typical functions include:

- SDK initialization and termination
- SDK version information
- Firmware version retrieval
- Controller reset

These functions provide SDK initialization, resource management, version information, and basic controller operations.

API	Description
EBUS_SLOT()	Converts the slot number to the eBus slot identifier.
POE_Init()	Initializes the e-PoE SDK.
POE_Close()	Releases SDK resources.
GetSDKVersion()	Retrieves the SDK version.
GetNetVersion()	Retrieves the .NET library version.
GetFirmwareVersion()	Retrieves the controller firmware version.
Reset()	Resets the e-PoE controller.

4.3.2. Controller Functions

Controller functions are used to configure and manage the operating parameters of the e-PoE controller.

Typical functions include:

- Controller configuration
- System configuration
- Controller status
- Controller event management

These functions configure the controller and retrieve controller status.

API	Description
Set_PSE_config()	Sets controller operating parameters.
Set_SYS_SW_config()	Sets system and software settings.
Get_SYS_SW_config()	Retrieves system and software settings.
Get_PSE_status()	Retrieves controller operating status.
Get_PSE_Event()	Retrieves controller events.
Clear_PSE_Event()	Clears controller events.

4.3.3. Port Management Functions

Port management functions configure and control individual PoE ports.

Typical functions include:

- Port enable/disable
- PoE mode configuration
- Port priority
- High-power configuration
- Force power control

These functions configure and control individual PoE ports.

API	Description
Set_POE_Port()	Enables or disables a PoE port.
Set_POE_port_HiPower()	Sets High Power mode.
Get_POE_port_HiPower()	Retrieves High Power configuration.
Set_POE_port_force_power()	Controls force power mode.
Set_port_priority()	Sets port priority.
Get_port_priority()	Retrieves port priority.

4.3.4. Power Management Functions

Power management functions configure power allocation and retrieve power-related information.

Typical functions include:

- Power limit configuration
- Power bank configuration
- Total power information
- Voltage and current monitoring

These functions configure power allocation and monitor power usage.

API	Description
Set_POE_port_power_limit()	Sets port power limit.
Get_POE_port_power_limit()	Retrieves port power limit.
Set_Power_banks()	Sets power bank settings.
Get_Power_banks()	Retrieves power bank settings.
Get_total_Power()	Retrieves total power consumption.
Get_Power_supply_voltage_current()	Retrieves input voltage and current.

4.3.5. Monitoring Functions

Monitoring functions retrieve the operating status of the controller and PoE ports.

Typical functions include:

- Port status
- Port measurements
- Power consumption
- Controller operating status

These functions monitor controller and PoE port status.

API	Description
Get_port_status()	Retrieves the current operating status of a PoE port.
Get_port_Laststatus()	Retrieves the previous operating status of a PoE port.
Get_port_power_measurements()	Retrieves port voltage, current, and power measurements.
Get_port_event_status()	Retrieves the current event status of a PoE port.
Clear_port_event_status()	Clears the current event status of a PoE port.

4.3.6. Watchdog Functions

Watchdog functions configure and monitor the watchdog mechanism to improve system reliability.

Typical functions include:

- Watchdog configuration
- Watchdog status
- Watchdog reset

These functions configure and monitor the watchdog timer.

API	Description
SetWDTConfig()	Sets the watchdog timer.
GetWDTConfig()	Retrieves watchdog configuration.
GetWDTStatus()	Retrieves watchdog status.
ResetWDT()	Clears the watchdog timeout status.

4.3.7. Interrupt Functions

Interrupt functions configure interrupt behavior and retrieve interrupt events generated by the controller.

Typical functions include:

- Interrupt configuration
- Interrupt status
- Interrupt clear

These functions configure and monitor interrupt events.

API	Description
Set_interrupt_setting()	Sets interrupt settings.
Get_interrupt_setting()	Retrieves interrupt configuration.
Clear_interrupt()	Clears interrupt events.

4.3.8. Register Access Functions

Register access functions provide direct read and write access to controller registers for advanced configuration and debugging.

Typical functions include:

- Register read
- Register write

Register access functions are intended for advanced users. Incorrect register settings may cause unexpected controller behavior.

These functions provide direct register access.

API	Description
Set_register_value()	Writes a register value.
Get_register_value()	Reads a register value.

4.4. API Reference

This section provides detailed descriptions of all APIs supported by the e-PoE SDK.

Each API is organized according to the functional categories introduced in Section 4.3 Function Categories. The APIs are presented in alphabetical order within each category to facilitate quick reference.

For each API, the following information is provided:

- Function description
- Function prototype (Syntax)
- Parameter descriptions
- Return value
- Remarks
- Programming examples

Before calling any API, ensure that the PAC Development SDK and the e-PoE SDK have been initialized successfully. Unless otherwise specified, all APIs require a valid controller connection established through the PAC SDK.

API Documentation Format

Each API is documented using the following format.

Item	Description
Description	Briefly describes the purpose of the function.
Syntax	Shows the function prototype.
Parameters	Describes the input and output parameters.
Return Value	Defines the return value of the function.
Example	Demonstrates how to use the function in C/C++ and C#.

API Organization

The APIs described in this chapter are grouped into the following functional categories.

- System Functions
- Controller Functions
- Port Management Functions
- Power Management Functions
- Monitoring Functions
- Watchdog Functions
- Interrupt Functions
- Register Access Functions

Detailed descriptions of each API are provided in the following sections.

4.4.1. EBUS_SLOT

Converts the slot number to the eBus slot identifier.

Syntax

```
int EBUS_SLOT(
  int islot
);
```

Parameter

i_slot

[IN] slot number.

Return Value

Return value for SDK use.

Example

```
int slot = 1;
iPOE_Init (IntPtr hPort, EBUS_SLOT (slot));
```

4.4.2. iPOE_Init

Initializes the e-PoE SDK.

Syntax

```
BOOL iPOE_Init (
  HANDLE hPort,
  int slot
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command.

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
iPOE_Init (IntPtr hPort, EBUS_SLOT (slot));
```

[C#]

```
IPOENET.Sys. POE_Init (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot));
// Initilize SDK and check device for use.
```

4.4.3. iPOE_Close

Releases SDK resources.

Syntax

```
BOOL iPOE_Close(
  HANDLE hPort,
  int slot
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

---

Return Value

Return true if success, otherwise false.Set the communication timeout between packets send and receive.

Example

[C++] Windows

```
int slot = 1;
iPOE_ Close (IntPtr hPort, EBUS_SLOT (slot));
```

[C#]

```
IPOENET.Sys. POE_ Close (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot));  // Uninitilize SDK.
```

4.4.4. GetSDKVersion

Retrieves the SDK version.

Syntax

```
BOOL iPOE_GetSDKVersion (
  LPSTR version
);
```

Parameter

version

[in] Specifies IPOESDK version.

Return Value

Return TRUE, if the version is correctly. Otherwise, return false.

Example

[C++] Windows

```
char version[32];
iPOE_ GetSDKVersion (version);
```

[C#]

```
string version;
IPOENET.Sys. GetSDKVersion (version);
Console.WriteLine("The IPOESDK.dll version is : " + version);
//The example displays the following output to the console:
// The IPOESDK.dll version is : 1.0.0.0
```

4.4.5. GetNetVersion

Retrieves the .NET library version.

Syntax

```
BOOL iPOE_GetNetVersion (
  LPSTR version
);
```

Parameter

version

[in] Specifies IPOENET version.

Return Value

Return TRUE, if the version is correctly. Otherwise, return false.

Example

[C++] Windows

```
char version[32];
iPOE_ GetNetVersion (version);
```

[C#]

```
string version;
IPOENET.Sys. GetNetVersion (version);
Console.WriteLine("The IPOENET.dll version is : " + version);
//The example displays the following output to the console:
// The IPOENET.dll version is : 1.0.0.0
```

4.4.6. GetFirmwareVersion

Retrieves the controller firmware version.

Syntax

```
int iPOE_GetFirmwareVersion(
  HANDLE hPort,
  int slot,
  LPSTR strName
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

strName

[out] Retrieves the firmware version of the PSE device.

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
LPSTR strVer;
iPOE_ GetFirmwareVersion(IntPtr hPort, EBUS_SLOT (slot), strVer);
```

[C#]

```
string strVer =
IPOENET.Sys.POE_GetFirmwareVersion(Program.hPort,IPOENET.Sys.EBUS_SLOT(Program
.intSlot));
// Get device firmware version.
```

4.4.7. Reset

Resets the e-PoE device.

Syntax

```
BOOL iPOE_Reset (
  HANDLE hPort,
  int slot
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

---

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
iPOE_Reset (IntPtr hPort, EBUS_SLOT (slot));
```

[C#]

```
IPOENET.Sys. POE_Reset (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot));
// Can check return value is Success or not, true is Success, false is Fail.Enable
/ disable the handle of watchdog by library.
```

4.4.8. Set_PSE_config

Sets controller operating parameters.

Syntax

```
int iPOE_Set_PSE_config (
  HANDLE hPort,
  int slot ,
  float temp_alarm
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

temp_alarm

[in] set the temperature alarm of PSE device

Return Value

Returns 0 on success or the following error codes in the event of an error.

Example

[C++] Windows

```
int slot = 1;
float temp_alarm = 70;
iPOE_ Set_PSE_config (IntPtr hPort, EBUS_SLOT (slot), temp_alarm);
```

[C#]

```
float temp_alarm = 70;
IPOENET.POE. Set_PSE_config (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot), temp_alarm);
//if IC temperature exceeds 70℃, an interrupt is triggered with the user-defined
temperature event.
```

4.4.9. Set_SYS_SW_config

Sets system and software settings.

Syntax

```
int iPOE_Set_SYS_SW_config (
  HANDLE hPort,
  int slot ,
  unsigned char SYS_CFG[4],
  unsigned char SW_CFG[7]
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

SYS_CFG

[in] 4 bytes system configuration array

[0]: MSCC capacitor detection enable for all ports. Default value is Cap pin state.

1 – Resistor detection only

0 – Resistor & legacy detection

[1]: Set PS_PG pin source or communication

0 – PS_PG state is set only by PS_PG[3:0] pins

1 – PS_PG state is set by communication

[2]: Pin PS_PG3 pin state overwrite

1 – PS_PG3 pin is '1' (*)

0 – PS_PG3 pin is '0' (*)

[3]: Set ports startup current limit to AF (IEEE standard compliant) or set startup current

limit at startup to high current (not compliant with IEEE standard)

0 – i lim AF during startup

1– i lim AT during startup

SW_CFG

[in] 7 bytes software configuration array

[0]:

0: Set Icut according to TPPL

1: Set Icut to max according to ICUT_AT_MAX (0x99/0x9A)

[1] Set calculation method of power management

‘0’ – Static (according to class or PPL)

‘1’ – Dynamic (according to real port consumption)

[2] When class 0 is detected enable high power port.

In Auto mode and Semi-Auto mode reset value is 1.

In MSCC Extended Auto mode reset value is 0.

‘0’ – Class 0 is detected as high power (*)

‘1’ – Class 0 is detected as AF (*)

[3] When port is set to be high power enable class 123 as high power.

In Auto mode and Semi-Auto mode reset value is 1.

In MSCC Extended Auto mode reset value is 0.

‘0’ – Class 123 detected as high power (*)

‘1’ – Class 123 detected as AF (*)

[4] Class error configuration bit. If the result of the second class event is not equal to the first event then fail the detection or set is as valid detection.

‘0’ – Two different results are invalid

‘1’ – Two different results are valid

[5] If class current is higher than class 4 limit set class result as either error, class0 or class4.

0: 00/11 – Set class as error

1: 01 – Set high current class as class 4

2: 10 – Set high current class as class 0

[6] When startup power is not enough for current port, check whether the following ports can be started up or mark them all as PM

'0' – Don't try to start the following ports

'1' – Check whether startup power is enough for lower priority ports

---

Return Value

Returns 0 on success or the following error codes in the event of an error.

Example

[C++] Windows

```
int slot = 1;
unsigned char SYS_CFG[4];
unsigned char SW_CFG[7];
iPOE_ Set_SYS_SW_config (IntPtr hPort, EBUS_SLOT (slot), SYS_CFG, SW_CFG);
```

[C#]

```
byte[] SYS_CFG = new byte[]{0,0,0,0};
byte[] SW_CFG[7] = new byte[]{0,0,0,0,0,0,0};
IPOENET.POE. Set_SYS_SW_config (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , SYS_CFG, SW_CFG);
```

4.4.10. Get_SYS_SW_config

Retrieves system and software settings.

Syntax

```
int iPOE_Get_SYS_SW_config (
  HANDLE hPort,
  int slot ,
  unsigned char SYS_CFG[4],
  unsigned char SW_CFG[7]
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

SYS_CFG

[out] 4 bytes system configuration array

[0]: MSCC capacitor detection enable for all ports. Default value is Cap pin state.

‘1’ – Resistor detection only

‘0’ – Resistor & legacy detection

[1] Set PS_PG pin source or communication

‘0’ – PS_PG state is set only by PS_PG[3:0] pins

‘1’ – PS_PG state is set by communication

[2] Pin PS_PG3 pin state overwrite

‘1’ – PS_PG3 pin is '1' (*)

‘0’ – PS_PG3 pin is '0' (*)

[3] Set ports startup current limit to AF (IEEE standard compliant) or set startup current limit

at startup to high current (not compliant with IEEE standard)

‘0’ – i lim AF during startup

‘1’– i lim AT during startup

SW_CFG

[out] 7 bytes software configuration array

[0]

0: Set Icut according to TPPL

1: Set Icut to max according to ICUT_AT_MAX (0x99/0x9A)

[1] Set calculation method of power management

‘0’ – Static (according to class or PPL)

‘1’ – Dynamic (according to real port consumption)

[2] When class 0 is detected enable high power port.

In Auto mode and Semi-Auto mode reset value is 1.

In MSCC Extended Auto mode reset value is 0.

‘0’ – Class 0 is detected as high power (*)

‘1’ – Class 0 is detected as AF (*)

[3] When port is set to be high power enable class 123 as high power.

In Auto mode and Semi-Auto mode reset value is 1.

In MSCC Extended Auto mode reset value is 0.

‘0’ – Class 123 detected as high power (*)

‘1’ – Class 123 detected as AF (*)

[4] Class error configuration bit. If the result of the second class event is not equal to the first event then fail the detection or set is as valid detection.

‘0’ – Two different results are invalid

‘1’ – Two different results are valid

[5] If class current is higher than class 4 limit set class result as either error, class0 or class4.

0: 00/11 – Set class as error

1: 01 – Set high current class as class 4

2: 10 – Set high current class as class 0

[6] When startup power is not enough for current port, check whether the following ports can be started up or mark them all as PM

‘0’ – Don’t try to start the following ports

‘1’ – Check whether startup power is enough for lower priority ports

---

Return Value

Returns 0 on success or the following error codes in the event of an error.

Example

[C++] Windows

```
int slot = 1;
unsigned char SYS_CFG[4];
unsigned char SW_CFG[7];
iPOE_ Get _SYS_SW_config (IntPtr hPort, EBUS_SLOT (slot), SYS_CFG, SW_CFG);
```

[C#]

```
byte[] SYS_CFG = new byte[]{0,0,0,0};
byte[] SW_CFG[7] = new byte[]{0,0,0,0,0,0,0};
IPOENET.POE. Get _SYS_SW_config (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , SYS_CFG, SW_CFG);
```

4.4.11. Get_PSE_status

Retrieves controller operating status.

Syntax

```
bool iPOE_Get_PSE_status (
  HANDLE hPort,
  int slot ,
  unsigned char *mode,
  unsigned char *hw_type,
  float *temp,
  float *temp_alarm,
  unsigned char *chip_type,
  unsigned char *status1
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

mode

[out] AF/AT mode

mode:

AF Mode :0

AT Mode :1 (default)

hw_type

[out] Number of ports verified by the internal communication

0 = Invalid/non-existing PoE device

2 = 2-port PoE device

4 = 4-port PoE device.

8 = 8-port PoE device.

temp

[out] Temperature measured by the PSE device.

If the PSE device does not exist, the response is 0xFF. This field can return a negative temperature (down to -40 °C).

If the reported temperature value is greater than 205, then the real temperature will be 256 minus the reported temperature value.

Units are in degrees Celsius.

temp_alarm

[out] get the upper temperature limit of this PoE device.

chip_type

[out] get the chip type of the PSE device.

PD69104 = 0x00

status1

[out] reserverd

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned char mode;
unsigned char hw_type;
float temp;
float temp_alarm;
unsigned char chip_type;
unsigned char status1;
iPOE_ Get _SYS_SW_config (IntPtr hPort, EBUS_SLOT (slot), &mode, &hw_type, &temp,
&temp_alarm, &chip_type, &status1);
```

[C#]

```
byte mode;
byte hw_type;
float temp;
float temp_alarm;
byte chip_type;
byte status1;
IPOENET.POE. Get _SYS_SW_config (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot), ref mode, ref hw_type, ref temp, ref
temp_alarm, ref chip_type, ref status1);
```

4.4.12. Get_PSE_Event

Retrieves controller events.

Syntax

```
BOOL iPOE_Get_PSE_Event (
  HANDLE hPort,
  int slot ,
  unsigned char *extersupPOWfPort,
  unsigned char *overTEMP,
  unsigned char *OVLO,
  unsigned char *UVLO,
  unsigned char *deniedpowerbyPM,
  unsigned char *FETfail,
  unsigned char *thermalshutdown
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

extersupPOWfPort

[out] IC power is externally supplied from one of the ports – MSCC proprietary

overTEMP

[out] Chip temperature is over the temperature alarm threshold – MSCC proprietary

OVLO

[out] Over Voltage Lock Out (OVLO) on main supply (V MAIN) – MSCC proprietary

UVLO

[out] Under Voltage Lock Out (UVLO) on main supply

deniedpowerbyPM

[out] One port or more were denied to power up due to power management– MSCC proprietary

FETfail

[out] If this bit is set one or more MOSFETs may have failed.

thermalshutdown

[out] This bit is set in case of a thermal shutdown. All ports are turned off.

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned char extersupPOWfPort;
unsigned char overTEMP;
unsigned char OVLO;
unsigned char UVLO;
unsigned char deniedpowerbyPM;
unsigned char FETfail;
unsigned char thermalshutdown;
iPOE_ Get_PSE_Event (IntPtr hPort, EBUS_SLOT (slot), &extersupPOWfPort,
&overTEMP, &OVLO, &UVLO, &deniedpowerbyPM, &FETfail, &thermalshutdown);
```

[C#]

```
byte extersupPOWfPort;
byte overTEMP;
byte OVLO;
byte UVLO;
byte deniedpowerbyPM;
byte FETfail;
byte thermalshutdown;
IPOENET.Sys. POE_ Get_PSE_Event (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , ref extersupPOWfPort, ref overTEMP,
ref OVLO, ref UVLO, ref deniedpowerbyPM, ref FETfail, ref thermalshutdown);
```

4.4.13. Clear_PSE_Event

Clears controller events.

Syntax

```
BOOL iPOE_Clear_PSE_Event (
  HANDLE hPort,
  int slot
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
iPOE_ Clear_PSE_Event (IntPtr hPort, EBUS_SLOT (slot));
```

[C#]

```
IPOENET.Sys. POE_ Clear_PSE_Event (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot));
```

4.4.14. Set_POE_Port

Enables or disables a PoE port.

Syntax

```
BOOL iPOE_Set_POE_Port (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char cmd,
  unsigned char port_mode
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module.

0=port1,

1=port2,

2=port3,

3= port4

cmd

[in] Set enable/disable for POE port

port_mode

[in] Set AF/AT operation.

0=AF operation

1=AT operation (Default)

2=AT operation high power

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char cmd = 1; //Enable
unsigned char port_mode = 1;//AT opration
iPOE_ Set_POE_Port (IntPtr hPort, EBUS_SLOT (slot), port, cmd, port_mode);
```

[C#]

```
uint port = 0;
byte cmd = 1;
byte port_mode = 1;
IPOENET.Sys. Set_POE_Port (Program.hPort, IPOENET.Sys.EBUS_SLOT(Program.intSlot) ,
port, cmd, port_mode);
//port 1 Enable and set AT operation mode.
```

4.4.15. Set_POE_port_HiPower

Sets High Power mode.

Syntax

```
BOOL iPOE_Set_POE_port_HiPower (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char hipw_mode,
  float Acutoffthreshold,
  int current_limit
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

hipw_mode

[in] Set the high power mode.

Must be one of:

0x00(disables 2-event classification + disables large capacitor class)

0x01(enable 2-event classification + disables large capacitor class)

0x02(disables 2-event classification + enable large capacitor class)

0x03(enable 2-event classification + enable large capacitor class)

Acutoffthreshold

[in] Sets cutoff current threshold (I CUT) on Port, 37.5mA/count when value=0, and 18.75mA/count when value= 1.

current_limit

[in] Sets the current limit setting for port.

When 80h -> set Current Limiting reference to AF

When C0h -> take Current Limiting reference from I_LIM_AT_MAX register (9Bh)

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char hipw_mode = 1; // enable 2-event classification + disables large capacitor class
float Acutoffthreshold = 1;// Conversion scale is 18.75mA/count
int current_limit =0x80 ; // set Current Limiting reference to AF
iPOE_ Set_POE_port_HiPower (IntPtr hPort, EBUS_SLOT (slot), port, hipw_mode, Acutoffthreshold, current_limit);
```

[C#]

```
uint port = 0;
byte hipw_mode= 1;
float Acutoffthreshold= 1;
int current_limit = 0x80;
IPOENET.Sys.Set_POE_port_HiPower (Program.hPort, IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, hipw_mode, Acutoffthreshold, current_limit);
```

4.4.16. Get_POE_port_HiPower

Retrieves High Power configuration.

Syntax

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

hipw_mode

[out] Get the high power mode

Acutoffthreshold

[out] Gets cutoff current threshold (I CUT) on Port

current_limit

[out] Gets the current limit setting for port

hpstate

[out] Get the current limit setting for port

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char *hipw_mode = 0; // init value
float *Acutoffthreshold = 0;// init value
int *current_limit =0 ; // init value
int *hpstate =0 ; // init value
iPOE_ Get_POE_port_HiPower (IntPtr hPort, EBUS_SLOT (slot), port, hipw_mode,
Acutoffthreshold, current_limit, hpstate);
```

[C#]

```
uint port = 0; //port 1
byte hipw_mode= 0;
float Acutoffthreshold= 0;
int current_limit = 0;
int hpstate =0 ;
IPOENET.Sys.Get_POE_port_HiPower (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, ref hipw_mode, ref
Acutoffthreshold, ref current_limit, ref hpstate);
```

4.4.17. Set_POE_port_force_power

Controls force power mode.

Syntax

```
BOOL iPOE_ Set_POE_port_force_power (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  char force_power_action
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

force_power_action

[in] Force power action (1:turn on/0: turn off)

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
char force_power_action = 0; // turn off the port force power
iPOE_ Set_POE_port_force_power (IntPtr hPort, EBUS_SLOT (slot), port,
force_power_action);
```

[C#]

```
uint port = 0; //port 1
byte force_power_action = 0; // turn off the port force power
IPOENET.Sys.Set_POE_port_force_power (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, force_power_action);
```

4.4.18. Set_port_priority

Sets port priority.

Syntax

```
BOOL iPOE_Set_POE_port_priority (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char priority
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

priority

[in] Set the POE port priority

PRIORITY_HIGH = 0

PRIORITY_MEDIUM = 1

PRIORITY_LOW = 2

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char priority = 0;// priority set hight
iPOE_Set_POE_port_priority (IntPtr hPort, EBUS_SLOT (slot), port, priority);
```

[C#]

```
uint port = 0; //port 1
byte priority = 0;// priority set hight
IPOENET.Sys.Set_port_priority (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, priority);
```

4.4.19. Get_port_priority

Retrieves port priority.

Syntax

```
BOOL iPOE_Get_POE_port_priority (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char* priority
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

priority

[out] Get the POE port priority

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char *priority = 0;//init value
iPOE_Get_POE_port_priority (IntPtr hPort, EBUS_SLOT (slot), port, priority);
```

[C#]

```
uint port = 0; //port 1
byte priority = 0;
IPOENET.Sys.Get_port_priority (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, ref priority);
```

4.4.20. Set_POE_port_power_limit

Sets port power limit

Syntax

```
BOOL iPOE_Set_POE_port_power_limit (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  int pre_defined_power_limit,
  unsigned int temp_port_power_limit
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

pre_defined_power_limit

[in] Set pre-defined power limint.

DVDD – 44W

Open – 36W

GND – 20W

temp_port_power_limit

[in] Set temperature power limint

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
int pre_defined_power_limit = 44; // set DVDD
unsigned int temp_port_power_limit = 60;//set High Power
iPOE_ Set_POE_port_HiPower (IntPtr hPort, EBUS_SLOT (slot), port,
pre_defined_power_limit, temp_port_power_limit);
```

[C#]

```
uint port = 0; //port 1
int pre_defined_power_limit = 44; // set DVDD
uint temp_port_power_limit = 60; //set High Power
IPOENET.Sys.Set_POE_port_HiPower (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, pre_defined_power_limit,
temp_port_power_limit);
```

4.4.21. Get_POE_port_power_limit

Retrieves port power limit.

Syntax

```
BOOL iPOE_Get_POE_port_power_limit (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned int *pre_defined_power_limit,
  unsigned int *temp_port_power_limit
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

pre_defined_power_limit

[out] Get pre-defined power limint

temp_port_power_limit

[out] Get temperature power limint

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned int *pre_defined_power_limit = 0; // init value
unsigned int *temp_port_power_limit = 0;// init value
iPOE_Get_POE_port_power_limit (IntPtr hPort, EBUS_SLOT (slot), port,
pre_defined_power_limit, temp_port_power_limit);
```

[C#]

```
uint port = 0; //port 1
int pre_defined_power_limit = 0; // init value
uint temp_port_power_limit = 0; // init value
IPOENET.Sys. Get_POE_port_power_limit (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, ref pre_defined_power_limit, ref
temp_port_power_limit);
```

4.4.22. Set_power_banks

Sets power bank settings.

Syntax

```
BOOL iPOE_Set_power_banks (
  HANDLE hPort,
  int slot ,
  unsigned char bank,
  float power_limit,
  float max_shutdown_voltage,
  float min_shutdown_voltage,
  unsigned char guard_bank
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

bank

[in] The bank number

power_limit

[in] Set power budget

max_shutdown_voltage

[in] Reserved for future use. Must be set to 0.

min_shutdown_voltage

[in] Reserved for future use. Must be set to 0.

guard_bank

[in] Reserved for future use. Must be set to 0.

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char bank = 0;
float power_limit = 80;
float max_shutdown_voltage = 0;
float min_shutdown_voltage = 0;
unsigned char guard_bank = 0;
iPOE_Set_power_banks (IntPtr hPort, EBUS_SLOT (slot), port, bank, power_limit,
max_shutdown_voltage, min_shutdown_voltage, guard_bank);
```

[C#]

```
uint port = 0; //port 1
byte bank = 0;
float power_limit = 80;
float max_shutdown_voltage = 0;
float min_shutdown_voltage = 0;
byte guard_bank = 0;
IPOENET.Sys. Set_power_banks (IntPtr hPort, EBUS_SLOT (slot), port, bank,
power_limit, max_shutdown_voltage, min_shutdown_voltage, guard_bank);
```

4.4.23. Get_power_banks

Retrieves power bank settings.

Syntax

```
BOOL iPOE_Get_power_banks (
  HANDLE hPort,
  int slot ,
  unsigned char bank,
  float *power_limit,
  float *max_shutdown_voltage,
  float *min_shutdown_voltage,
  unsigned char *guard_bank
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

bank

[in] The bank value passed to the power_bank parameter of iPOE_Get_total_power().

power_limit

[out] Set power budget

max_shutdown_voltage

[out] Reserved

min_shutdown_voltage

[out] Reserved

guard_bank

[out] Reserved

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char bank = 0;
float *power_limit = 0;
float *max_shutdown_voltage = 0;
float *min_shutdown_voltage = 0;
unsigned char *guard_bank = 0;
iPOE_Get_power_banks (IntPtr hPort, EBUS_SLOT (slot), port, bank, power_limit,
max_shutdown_voltage, min_shutdown_voltage, guard_bank);
```

[C#]

```
uint port = 0; //port 1
byte bank = 0; //init value
float power_limit = 0;
float max_shutdown_voltage = 0;
float min_shutdown_voltage = 0;
byte guard_bank = 0;
IPOENET.Sys.Get_power_banks (IntPtr hPort, EBUS_SLOT (slot), port, bank, ref
power_limit, ref max_shutdown_voltage, ref min_shutdown_voltage, ref
guard_bank);
```

4.4.24. Get_total_power

Retrieves total power consumption.

Syntax

```
BOOL iPOE_Get_total_power (
  HANDLE hPort,
  int slot ,
  float *power_consumption,
  float *calculated_power,
  float *available_power,
  float *vmain_measure_voltage,
  unsigned char *power_bank,
  unsigned char *bank_number
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

power_consumption

[out] Get total power consumption

calculated_power

[out] Get total calculated power consumption

available_power

[out] Get total available power consumption

vmain_measure_voltage

[out] Get main power supply voltage

power_bank

[out] Get the number of bank for PSE

bank_number

[out] Get the current bank number

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
float *power_consumption = 0; //init value
float *calculated_power = 0;
float *available_power = 0;
float *vmain_measure_voltage = 0;
unsigned char *power_bank = 0;
unsigned char *bank_number = 0;
iPOE_Get_total_power (IntPtr hPort, EBUS_SLOT (slot), port, power_consumption,
calculated_power, available_power, vmain_measure_voltage, power_bank,
bank_number);
```

[C#]

```
uint port = 0; //port 1
float power_consumption = 0; //init value
float calculated_power = 0;
float available_power = 0;
float vmain_measure_voltage = 0;
byte power_bank = 0;
byte bank_number = 0;
IPOENET.Sys. Get_total_power (IntPtr hPort, EBUS_SLOT (slot), port, ref
power_consumption, ref calculated_power, ref available_power, ref
vmain_measure_voltage, ref power_bank, ref bank_number);
```

4.4.25. Get_power_supply_voltage_current

Retrieves input voltage and current.

Syntax

```
BOOL iPOE_get_power_supply_voltage_current (
  HANDLE hPort,
  int slot ,
  float *vmain_max_th_voltage,
  float *vmain_low_th_voltage,
  float *icut_current
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

vmain_max_th_voltage

[out] Get Vmain High threshold setting

vmain_low_th_voltage

[out] Get Vmain low threshold setting

icut_current

[out] Get Icut Maximum current

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
float *vmain_max_th_voltage = 0;
float *vmain_low_th_voltage = 0;
float *icut_current = 0;
iPOE_Get_ power_supply_voltage_current (IntPtr hPort, EBUS_SLOT (slot), port,
vmain_max_th_voltage, vmain_low_th_voltage, icut_current);
```

[C#]

```
uint port = 0; //port 1
float vmain_max_th_voltage = 0;
float vmain_low_th_voltage = 0;
float icut_current = 0;
IPOENET.Sys.Get_ power_supply_voltage_current (IntPtr hPort, EBUS_SLOT (slot),
port, ref vmain_max_th_voltage, ref vmain_low_th_voltage, ref icut_current);
```

4.4.26. Get_port_status

Retrieves the current operating status of a PoE port.

Syntax

```
BOOL iPOE_Get_POE_port_status (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char *mode,
  unsigned char *en_dis,
  unsigned char *bPowerenabled,
  unsigned char *bstatePowerGood,
  unsigned char *portstatus,
  unsigned char counter[6]
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

mode

[out] Get port detection status

0: AF mode

1: AT mode

2: AT mode high power

en_dis

[out] Get the POE class

00:shutdown

01:Manual

10:semi-auto

11:auto

bPowerenabled

[out] Get the statu of power enable/disable

bstatePowerGood

[out] Get the power good status

portstatus

[out] Get the port status

portstatus &0x1: MOSFET over temperature event.

portstatus &0x2: OFF Due to PM.

portstatus &0x4: MSCC cap detection passed.

portstatus &0x8: failed in detection/class/startup.

counter

[out] Get 6 bytes counter array

counter[0] : Invalid Detection Counter

counter[1] : Power Denied Counter

counter[2] : Overload Event Counter

counter[3] : Under-load Event Counter

counter[4] : Short Circuit Event Counter

counter[5] : Class Error Event Counter

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char *mode = 0;//init value
unsigned char *en_dis = 0;
unsigned char *bPowerenabled = 0;
unsigned char *bstatePowerGood = 0;
unsigned char *portstatus = 0;
unsigned char counter[6] = 0;
iPOE_Get_POE_port_status (IntPtr hPort, EBUS_SLOT (slot), port, mode, en_dis,
bPowerenabled, bstatePowerGood, portstatus, counter);
```

[C#]

```
uint port = 0; //port 1
byte mode = 0;
byte en_dis = 0;
byte bPowerenabled = 0;
byte bstatePowerGood = 0;
byte portstatus = 0;
byte[] counter = new byte[]{0,0,0,0,0,0};
IPOENET.Sys.Get_POE_port_status (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, ref mode, ref en_dis, ref
bPowerenabled, ref bstatePowerGood, ref portstatus, ref counter);
```

4.4.27. Get_port_Laststatus

Retrieves the previous operating status of a PoE port.

Syntax

```
BOOL iPOE_Get_POE_port_Laststatus (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char *port_detect,
  unsigned char *poe_class,
  unsigned char *force_power_en
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

port_detect

[out] Get port detection status

poe_class

[out] Get the POE class

force_power_en

[out] Get the statu of force power

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char *port_detect = 0;//init value
unsigned char *poe_class = 0;
unsigned char *force_power_en = 0;
iPOE_Get_POE_port_Laststatus (IntPtr hPort, EBUS_SLOT (slot), port,
port_detect, poe_class, force_power_en);
```

[C#]

```
uint port = 0; //port 1
byte port_detect = 0;
byte poe_class = 0;
byte force_power_en = 0;
IPOENET.Sys.Get_port_Laststatus (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot) , port, ref port_detect, ref poe_class,
ref  force_power_en);
```

4.4.28. Get_port_power_measurements

Retrieves port voltage, current, and power measurements.

Syntax

```
BOOL iPOE_Get_ port_power_measurements (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  float *cal_current,
  float *port_power_consumption,
  float *port_voltage
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

cal_current

[out] Get port current measurement

port_power_consumption

[out] Get port power consumption measurement

port_voltage

[out] Get port voltage measurement

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
float *cal_current = 0; //init value
float *port_power_consumption = 0;
float *port_voltage = 0;
iPOE_ Get _POE_port_power_measurements (IntPtr hPort, EBUS_SLOT (slot), port,
cal_current, port_power_consumption, port_voltage);
```

[C#]

```
uint port = 0; //port 1
float cal_current = 0; //init value
float port_power_consumption = 0;
float port_voltage = 0;
IPOENET.Sys.Get_POE_port_power_measurements (IntPtr hPort, EBUS_SLOT (slot),
port, ref cal_current, ref port_power_consumption, ref port_voltage);
```

4.4.29. Get_port_event_status

Retrieves the current event status of a PoE port.

Syntax

```
BOOL iPOE_Get_port_event_status (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char *powerstatuschange,
  unsigned char *powergoodchange,
  unsigned char *detetcomplet,
  unsigned char *classcomplet,
  unsigned char *overloadstatus,
  unsigned char *underloadstatus,
  unsigned char *unsuccessstartup,
  unsigned char *curentlimittimeout
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

port

[in] The port of PSE module

powerstatuschange

[out] occur when a port is turned on or off

powergoodchange

[out] power good status changes

detetcomplet

[out] port has completed a detection cycle

classcomplet

[out] port has completed classifying the PD

overloadstatus

[out] port has been turned off due to load current above ICUT or ILIM for longer than tCUT

underloadstatus

[out] port has been turned off due to PD was disconnected

unsuccessstartup

[out] port has been turned off due to an unsuccessful startup

curentlimittimeout

[out] port has been turned off due to current-limit for longer than tLIM

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char *powerstatuschange = 0; //init value
unsigned char *powergoodchange = 0;
unsigned char *detetcomplet = 0;
unsigned char *classcomplet = 0;
unsigned char *overloadstatus = 0;
unsigned char *underloadstatus = 0;
unsigned char *unsuccessstartup = 0;
unsigned char *curentlimittimeout = 0;
iPOE_Get_port_event_status (IntPtr hPort, EBUS_SLOT (slot), port,
powerstatuschange, powergoodchange, detetcomplet, classcomplet,
overloadstatus, underloadstatus, unsuccessstartup, curentlimittimeout);
```

[C#]

```
uint port = 0; //port 1
byte powerstatuschange = 0; //init value
byte powergoodchange = 0;
byte detetcomplet = 0;
byte classcomplet = 0;
byte overloadstatus = 0;
byte underloadstatus = 0;
byte unsuccessstartup = 0;
byte curentlimittimeout = 0;
IPOENET.Sys.Get_port_event_status (IntPtr hPort, EBUS_SLOT (slot), port, ref
powerstatuschange, ref powergoodchange, ref detetcomplet, ref classcomplet, ref
overloadstatus, ref underloadstatus, ref unsuccessstartup, ref
curentlimittimeout);
```

4.4.30. Clear_port_event_status

Clears the current event status of a PoE port.

Syntax

```
BOOL iPOE_Clear_port_event_status (
  HANDLE hPort,
  int slot ,
  unsigned char PWREVN_cor,
  unsigned char DETEVN_cor,
  unsigned char FLTEVN_cor,
  unsigned char TSEVN_cor
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

PWREVN_cor

[in] Clear Power Event register

0: not clear

1: clear

DETEVN_cor

[in] Clear Detect and Classification Event register

0: not clear

1: clear

FLTEVN_cor

[in] Clear Overload & Under-load Time Out register

0: not clear

1: clear

TSEVN_cor

[in] Clear Overload During Start Up Time Out register

0: not clear

1: clear

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char PWREVN_cor = 0; //not clear
unsigned char DETEVN_cor = 1; //clear
unsigned char FLTEVN_cor = 0; //not clear
unsigned char TSEVN_cor = 1; //clear
iPOE_Clear_port_event_status (IntPtr hPort, EBUS_SLOT (slot), port, PWREVN_cor,
DETEVN_cor, FLTEVN_cor, TSEVN_cor);
```

[C#]

```
uint port = 0; //port 1
byte PWREVN_cor = 1; //clear
byte DETEVN_cor = 0; //not clear
byte FLTEVN_cor = 0; //not clear
byte TSEVN_cor = 0; //not clear
IPOENET.Sys. Clear_port_event_status (IntPtr hPort, EBUS_SLOT (slot), port,
PWREVN_cor, DETEVN_cor, FLTEVN_cor, TSEVN_cor);
```

4.4.31. SetWDTConfig

Sets the watchdog timer.

Syntax

```
BOOL iPOE_SetWDTConfig (
  HANDLE hPort,
  int slot,
  short enStatus,
  unsigned long wdtTimeout
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

enStatus

[in] The watchdog status sets for PSE device

wdtTimeout

[in] The watchdog timeout value sets for PSE device

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
short enStatus = 0;
unsigned long wdtTimeout = 0;
iPOE_SetWDTConfig (IntPtr hPort, EBUS_SLOT (slot), enStatus, wdtTimeout);
```

[C#]

```
ushort enStatus = 0;
uint wdtTimeout = 0;
IPOENET.POE.SetWDTConfig(Program.hPort, IPOENET.Sys.EBUS_SLOT(Program.intSlot),
enStatus, wdtTimeout);
```

4.4.32. GetWDTConfig

Retrieves watchdog configuration.

Syntax

```
BOOL iPOE_GetWDTConfig (
  HANDLE hPort,
  int slot,
  short *enStatus,
  unsigned long *wdtTimeout
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

enStatus

[out] The watchdog status of PSE device

wdtTimeout

[out] The watchdog timeout value of PSE device

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
short enStatus = 0;
unsigned long wdtTimeout = 0;
iPOE_GetWDTConfig (IntPtr hPort, EBUS_SLOT (slot), &enStatus, &wdtTimeout);
```

[C#]

```
ushort enStatus = 0;
uint wdtTimeout = 0;
IPOENET.POE.GetWDTConfig(Program.hPort, IPOENET.Sys.EBUS_SLOT(Program.intSlot),
ref enStatus, ref wdtTimeout);
```

4.4.33. GetWDTStatus

Retrieves watchdog status.

Syntax

```
BOOL iPOE_GetWDTStatus (
  HANDLE hPort,
  int slot,
  unsigned char *timeoutstatus,
  unsigned int *overtempstatus
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

timeoutstatus

[out] The watchdog timeout status of PSE device

overtempstatus

[out] The thermal shutdown status of PSE device

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned char timeoutstatus = 0;
unsigned int overtempstatus = 0;
iPOE_GetWDTStatus (IntPtr hPort, EBUS_SLOT (slot), &timeoutstatus,
&overtempstatus);
```

[C#]

```
byte timeoutstatus = 0;
int overtempstatus = 0;
IPOENET.POE.GetWDTStatus (Program.hPort, IPOENET.Sys.EBUS_SLOT(Program.intSlot),
ref timeoutstatus, ref overtempstatus);
```

4.4.34. ResetWDT

Clears the watchdog timeout status.

Syntax

```
BOOL iPOE_ResetWDT (
  HANDLE hPort,
  int slot
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function.

slot

[in] The slot in which PSE module is to receive the command. Default is local.

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
iPOE_ ResetWDT (IntPtr hPort, EBUS_SLOT (slot));
```

[C#]

```
IPOENET.POE. ResetWDT (Program.hPort,
IPOENET.Sys.EBUS_SLOT(Program.intSlot));
```

4.4.35. Set_interrupt_setting

Sets interrupt settings.

Syntax

```
BOOL iPOE_set_interrupt_setting (
  HANDLE hPort,
  int slot ,
  unsigned int port,
  unsigned char enable,
  unsigned char parameters
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

port

[in] The port of PSE module

enable

[in] Enable/Disable PSE interrupt function

parameters

[in] Parameters refers to an interrupt event

Return Value

Return true if success, otherwise false.

Example

[C++] Windows

```
int slot = 1;
unsigned int port = 0; //port 1
unsigned char enable = 0;
unsigned char parameters = 0;
iPOE_Set_interrupt_setting (IntPtr hPort, EBUS_SLOT (slot), port, bank,
power_limit, max_shutdown_voltage, min_shutdown_voltage, guard_bank);
```

[C#]

```
uint port = 0; //port 1
byte bank = 0;
float power_limit = 80;
float max_shutdown_voltage = 0;
float min_shutdown_voltage = 0;
byte guard_bank = 0;
IPOENET.Sys. Set_power_banks (IntPtr hPort, EBUS_SLOT (slot), port, bank,
power_limit, max_shutdown_voltage, min_shutdown_voltage, guard_bank);
```

4.4.36. Get_interrupt_setting

Retrieves interrupt configuration.

Syntax

```
BOOL iPOE_get_interrupt_setting (
  HANDLE hPort,
  int slot ,
  unsigned int port ,
  unsigned char *enable,
  unsigned char *parameters
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

port

[in] The port of PSE module

enable

[out] Enable/Disable PSE interrupt function

parameters

[out] Get the interrupt event settings of the controller

Return Value

Return true if success, otherwise false.

4.4.37. Clear_interrupt

Clears interrupt events.

Syntax

```
BOOL iPOE_get_interrupt_setting (
  HANDLE hPort,
  int slot ,
  unsigned int port
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

port

[in] The port of PSE module

Return Value

Return true if success, otherwise false.

4.4.38. Set_register_value

Writes a register value.

Syntax

```
BOOL iPOE_set_register_value (
  HANDLE hPort,
  int slot ,
  unsigned char addr,
  unsigned char data
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

addr

[in] The register of PSE device

data

[in] Set the value of the register

Return Value

Return true if success, otherwise false.

4.4.39. Get_register_value

Reads a register value.

Syntax

```
BOOL iPOE_get_register_value (
  HANDLE hPort,
  int slot ,
  unsigned char addr,
  unsigned char *data
);
```

Parameter

hPort

[in] The HANDLE opened by the interface open function

slot

[in] The slot in which PSE module is to receive the command. Default is local

addr

[in] The register of PSE device

data

[out] Get the value of the register

Return Value

Return true if success, otherwise false.

Appendix

A. Revision History

This appendix provides the revision history of the USB-2255-32/USB-2255-64 series user manual.

The table below shows the revision history.

Revision	Date	Description
1.0.1	July 2026	Initial issue