WinPAC Standard API User Manual

(WinCE Based ((eVC & .NET))

Version 2.0.0, August 2010

Service and usage information for

WinPAC-8000 **ViewPAC-2000** **WinPAC-5000**

Written by Sean

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, not for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only may be registered trademarks of their respective companies.

Contact US

If you have any problem, please feel free to contact us.
You can count on us for quick response.

Email: service@icpdas.com

Contents

Perface

This guide introduces WinPAC Software Development Kit (SDK). It provides an overview of what you can do with the SDK and the technologies that are available to you through the SDK.

Software Development Tool

Microsoft eMbedded Visual C++ Visual Basic.net Visual C#

Requirements

The WinPAC SDK only supports NET Framework 2.0 or above.

Installation Path

After installing the WinPAC SDKs, a number of functions can be installed on the Host PC, and this installation puts the header files, libraries into the following public places so they are easily changed by update the WinPAC SDKs.

Header files:

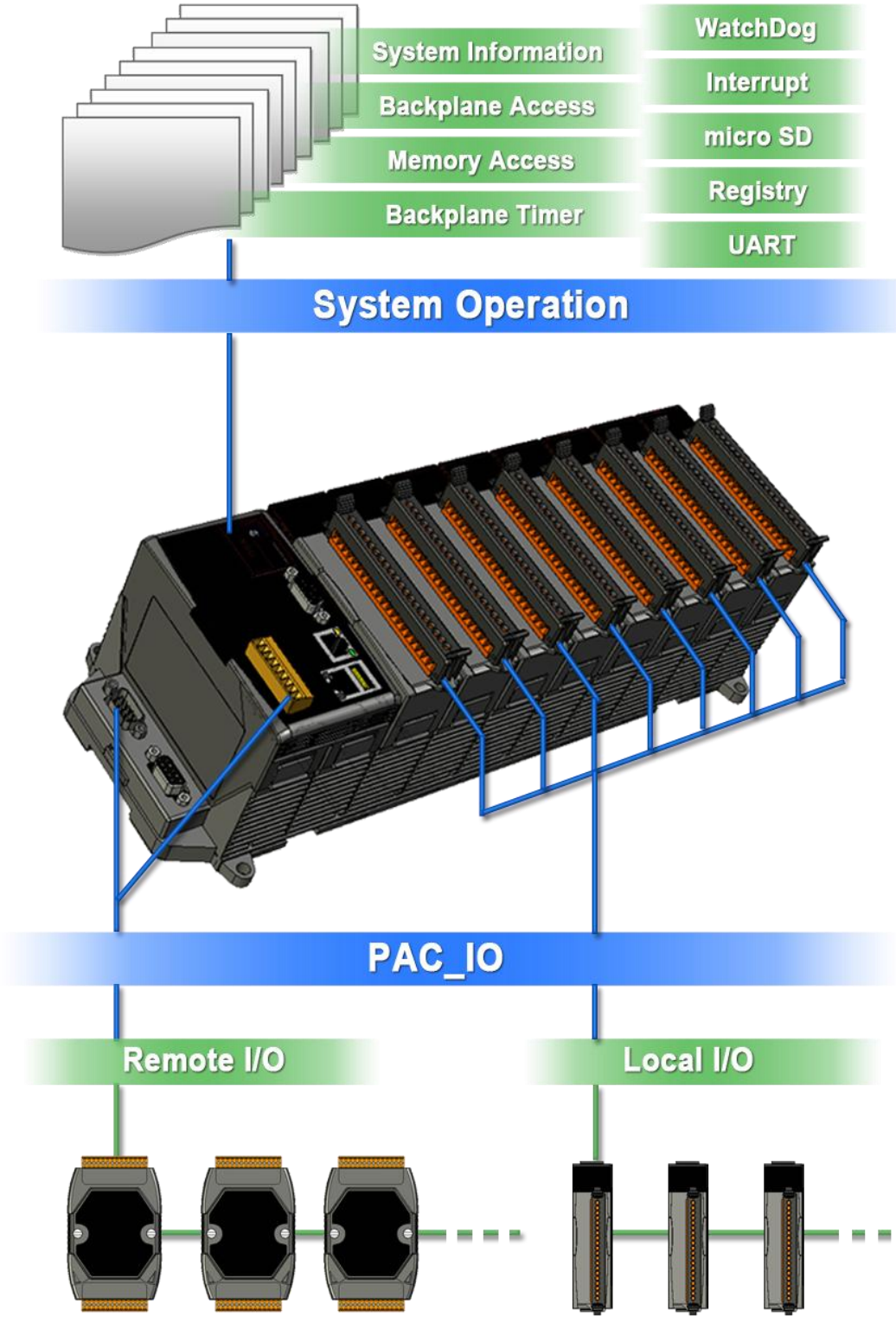
C:\Program Files\Windows CE Tools\wce500\PAC270\Icpdas\Include\ARMV4I\

Libraries:

C:\Program Files\Windows CE Tools\wce500\PAC270\Icpdas\Lib\ARMV4I\

Overview of WinPAC API

The WinPAC API enables applications to exploit the power of WinPAC. The WinPAC API consists of the following APIs and functional categories:



System Information API

Provides reference information for the system status.

Backplane API

Provides reference information for the backplane access APIs, including Hot Plug and backplane information.

Interrupt API

Provides reference information for the Interrupt APIs.

Memory Access API

Provides reference information for the memory R/W APIs, including EEPROM and SRAM.

Watchdog API

Provides reference information for the watchdog APIs, including hardware watchdog and OS watchdog.

microSD Management API

Provides reference information for the microSD Manager.

Registry API

Provides reference information for the registry.

UART API

Provides reference information for the Uart APIs.

PAC_IO API

Provides reference information for IO APIs, including local and remote

In additions, no matter 8K or 87K modules use the same API.

Backplane timer API

Provides hardware timerout/timer/tickcout functions.

Error Handling API

Provides reference information for error handling.

1. System Information API

System operations include basic operation, such as reboot and changing slot and version display, including OS, Eboot, SDK, Serial Number, and Mac address. The following topics describe how you can show the system information, or other basic operation programmatically using the system functions.

Supported Modules

The following shows the overview of the system functions which are available with WinPAC.

Functions\Models	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_BackwardCompatible	Y	Y	Y	-	-	-	-
pac_ChangeSlot	Y	Y	Y	-	Y	Y	-
pac_EnableLED	Y	Y	Y	Y	Y	Y	Y
pac_GetCPUVersion	Y	Y	Y	Y	Y	Y	Y
pac_GetEbootVersion	Y	Y	Y	Y	Y	Y	Y
pac_GetMacAddress	Y	Y	Y	Y	Y	Y	Y
pac_GetMapCOM	Y	Y	Y	-	-	-	-
pac_GetModuleName	Y	Y	Y	Y	Y	Y	Y
pac_GetModuleType	Y	Y	Y	Y	Y	Y	Y
pac_GetOSVersion	Y	Y	Y	Y	Y	Y	Y
pac_GetRotaryID	Y	Y	Y	Y	Y	Y	Y
pac_GetSerialNumber	Y	Y	Y	Y	Y	Y	Y
pac_GetWinPacNetVersion	Y	Y	Y	Y	Y	Y	Y
pac_GetWinPacSDKVersion	Y	Y	Y	Y	Y	Y	Y
pac_Reboot	Y	Y	Y	Y	Y	Y	Y

Function List

The following functions are used to retrieve or set system information.

Function	Description
pac_BackwardCompatible	This function retrieves the WinPAC270 running in the backward compatible mode or not.
pac_ChangeSlot	This function specifies the slot from one to another.
pac_EnableLED	This function decides the LED turning on or not.
pac_GetCPUVersion	This function retrieves the CPU version.
pac_GetEbootVersion	This function retrieves the eboot version.
pac_GetMacAddress	This function retrieves the Mac address.
pac_GetMapCOM	This function retrieves the index mapping of COM port.
pac_GetModuleName	This function is used to retrieve the name of an 8000 series I/O module, which is plugged into a specific I/O slot in the WinPac system.
pac_GetOSVersion	This function retrieves the OS version.
pac_GetRotaryID	This function retrieves the rotary switch ID.
pac_GetSerialNumber	This function retrieves the serial number.
pac_GetWinPacNetVersion	This function retrieves the version of the WinPacNet.dll.
pac_GetWinPacSDKVersion	This function retrieves the SDK version.
pac_Reboot	This function reboots the device.

1.1. pac_BackwardCompatible

This function retrieves the WinPAC270 running in the backward compatible mode or not.

Syntax

```
bool pac_BackwardCompatible();
```

Parameters

None

Return Values

Return true if the WinPAC is running in a backward compatible mode, otherwise false.

Examples

[eVC]

```
bool bBC;
bBC = pac_BackwardCompatible;
```

[C#]

```
bool bBC;
bBC = WinPAC.pac_BackwardCompatible();
```

1.2. pac_ChangeSlot

This function specifies the slot from one to another.

Syntax

```
void pac_ChangeSlot(
   BYTE slotNo
);
```

Parameters

slotNo

[in] Specifies the slot number.

Return Values

None

Examples

[eVC]

```
HANDLE hPort;
bool ret;
char buf[10];
hPort = uart_Open("COM0:");
pac_ChangeSlot(0);      // Change to the slot 0 which the 87k module plug in
ret = uart_SendCmd(hPort,"$00M", buf);      // $00M: ask the device name
uart_Close(hPort);
```

1.3. pac_EnableLED

This function decides the LED turning on or not.

Syntax

```
void pac_EnableLED(
   bool bFlag
);
```

Parameters

bFlag

[in] Boolean value that specifies whether to enable or disable the keyboard.

Value	Description
TRUE	Enable the RUN LED
FALSE	Disable the RUN LED

Return Values

None

Examples

[eVC]

```
pac_EnableLED(True);
```

[C#]

```
WinPAC.pac_EnableLED(True);
```

1.4. pac_GetCPUVersion

This function retrieves the CPU version.

Syntax

```
void pac_GetCPUVersion(
   LPSTR cpu_version
);
```

Parameters

cpu_version

[out] Retrieves the CPU version of the WinPAC.

Return Values

None

Examples

[eVC]

```
char CPU[32];
pac_GetCPUVersion(CPU);
```

[C#]

```
string CPU;
CPU = WinPAC.pac_GetCPUVersion();
```

1.5. pac_GetEbootVersion

This function retrieves the eboot version.

Syntax

```
void pac_GetEbootVersion(
   LPSTR eboot_version
);
```

Parameters

Eboot_version

[out] Retrieves the Eboot version.

Return Values

None

Examples

[eVC]

```
char Eboot[32];
pac_GetEbootVersion(Eboot);
```

[C#]

```
string Eboot;
Eboot = WinPAC.pac_GetEbootVersion();
```

1.6. pac_GetMacAddress

This function retrieves the Mac address.

Syntax

```
void pac_GetMacAddress(
    BYTE LAN,
    LPSTR MacAddr
);
```

Parameters

LAN

[in] Specifies the LAN number.

MacAddr

[out] Retrieves the MAC address of the specified LAN port.

Return Values

None

Examples

[eVC]

```
byte MAC = 1;
pac_GetMacAddress(MAC);
```

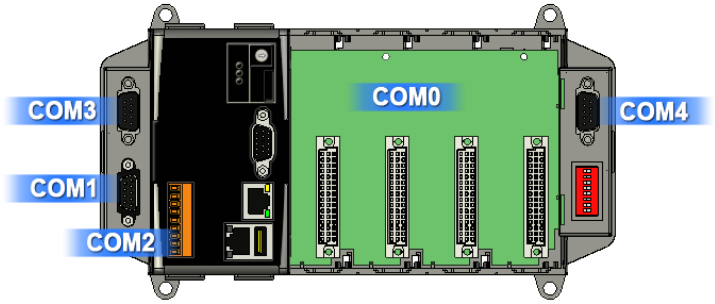
[C#]

```
byte MAC = 1;
String LAN;
LAN = WinPac.pac_GetMacAddress (MAC);
```

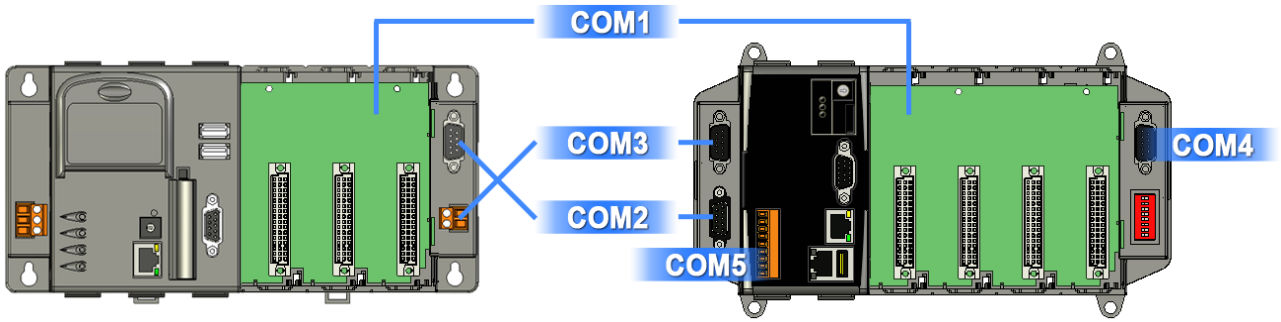
1.7. pac_GetMapCOM

This function retrieves the index mapping of COM port.

[Normal mode]



[Backward-Compatible mode]



Syntax

```
int pac_GetMapCom(
    int ndx
);
```

Parameters

ndx

[in] Specifies which slot COM port map between backward compatible or not.

Return Values

Return the index of COM port under the backward compatible, if backward compatible is running. Otherwise, return the index of COM port in the normal mode.

Examples

[eVC]

```
int currentCOM;     // current com port
int normalCOM;      // com port index in normal mode
currentCOM = pac_GetMapCom(normalCOM);
// If the device is running on normal mode, then the return value, currentCOM,
//equals normalCOM
// ex: normalCOM = 0; after this API, the currentCOM = 0, too.
// Otherwise, if the device is running on backward compatible mode, then the
//return value, currentCOM, is backward compatible mapping index
// ex: normalCOM = 0; after this API, the currentCOM = 1
```

[C#]

```
int currentCOM;
int normalCOM;
currentCOM = WinPAC.pac_GetMapCom(normalCOM);
```

1.8. pac_GetModuleName

This function is used to retrieve the name of an 8000 series I/O module, which is plugged into a specific I/O slot in the WinPac system. This function supports the collection of system hardware configurations.

Syntax

```
int pac_GetModuleName(
    BYTE slot,
    LPSTR strName
);
```

Parameters

slot

[in] Specify the slot number where the I/O module is plugged into.

strName

[out] The pointer to a buffer to receive the name of the I/O module

Return Values

Return 255 if the module is I-8K series module, otherwise, undefined.

Examples

[eVC]

```
byte slot = 1;
char strName[10];
pac_GetModuleName(slot, strName);
```

[C#]

```
byte slot = 1;
string strName;
int ModuleType = 0;
ModuleType = WinPac.pac_GetModuleName(slot, ref strName);
//Because of calling by reference, you should add “ref” keyword.

//For this API, there are two overloading for .NET.
//First is above,
//Another is below whose return value is Module Name, not Module type.

byte slot = 1;
string strName;
strName = WinPac.pac_GetModuleName(slot);
```

1.9. pac_GetModuleType

This function is used to retrieve the type of I/O modules.

Syntax

```
int pac_GetModuleType(
    int slot,
);
```

Parameters

slot

[in] Specify the slot number where the I/O module is plugged into.

Return Values

None

Examples

[eVC]

```
int slot = 1;
char strName[10];
pac_GetModuleName(slot, strName);
```

[C#]

```
byte slot = 1;
string strName;
int ModuleType = 0;
ModuleType = WinPac.pac_GetModuleName(slot, ref strName);
//Because of calling by reference, you should add “ref” keyword.

//For this API, there are two overloading for .NET.
//First is above,
//Another is below whose return value is Module Name, not Module type.

byte slot = 1;
string strName;
strName = WinPac.pac_GetModuleName(slot);
```

1.10. pac_GetOSVersion

This function retrieves the OS version.

Syntax

```
void pac_GetOSVersion(
   LPSTR os_version
);
```

Parameters

os_version

[out] Retrieves the OS version of the WinPAC.

Return Values

Return a nonzero value if successful, otherwise false.

Examples

[eVC]

```
char OS[32];
pac_GetOSVersion(OS);
```

[C#]

```
string OS;
OS = WinPAC.pac_GetOSVersion();
```

1.11. pac_GetRotaryID

This function retrieves the rotary switch ID.

Syntax

```
int pac_GetRotaryID();
```

Parameters

None

Return Values

Return the position of the rotary switch.

Examples

[eVC]

```
int RotaryID;
RotaryID = pac_GetRotaryID();
```

[C#]

```
int RotaryID;
RotaryID = WinPAC.pac_GetRotaryID();
```

1.12. pac_GetSerialNumber

This function retrieves the serial number.

Syntax

```
void pac_GetSerialNumber(
   LPSTR SerialNumber
);
```

Parameters

SerialNumber

[out] Retrieves the serial number of the WinPAC.

Return Values

None

Examples

[eVC]

```
char SN[32];
pac_GetSerialNumber(SN);
```

[C#]

```
string SN;
SN = WinPAC.pac_GetSerialNumber();
```

1.13. pac_GetWinPacNetVersion

This function retrieves the version of the WinPacNet.dll.

Syntax

```
void pac_GetWinPacNetVersion(
   LPSTR dll_version
);
```

Parameters

dll_version

[out] Retrieves the version of the WinNetVersion.

Return Values

None

Examples

[eVC]

```
char DLL[32];
pac_GetSWinPacNetVersion(SDK);
```

[C#]

```
string WinPacNet;
WinPacNet = WinPac.pac_GetWinPacNetVersion();
```

1.14. pac_GetWinPacSDKVersion

This function retrieves the SDK version.

Syntax

```
void pac_GetWinPacSDKVersion(
   LPSTR sdk_version
);
```

Parameters

sdk_version

[out] Retrieves the version of the WinPacSDK.

Return Values

None

Examples

[eVC]

```
char SDK[32];
pac_GetWinPacSDKVersion(SDK);
```

[C#]

```
string WinPacSDK;
WinPacSDK = WinPAC.pac_GetWinPacSDKVersion();
```

1.15. pac_Reboot

This function reboots the device.

Syntax

```
void pac_Reboot(void);
```

Parameters

None

Return Values

None

Examples

[eVC]

```
pac_Reboot();
```

[C#]

```
WinPAC.pac_Reboot();
```

2. Backplane API

Backplane operations include hot plug, interrupt and backplane information, such as NET ID and backplane version. The following topics describe how you can show the backplane information, hot plug, or interrupt operation programmatically using the backplane functions.

Note:

The functions on the chapter are not applicable for WP-5xxx series.

Supported Modules

The following shows the overview of the backplane functions which are available with WinPAC.

Functions / Modules	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_EnableRetrigger	Y	Y	Y	-	Y	Y	-
pac_GetBackplaneID	Y	Y	Y	-	Y	Y	Y
pac_GetBatteryLevel	Y	Y	Y	-	Y	Y	-
pac_GetDIPSwitch	Y	Y	Y	-	-	-	-
pac_GetSlotCount	Y	Y	Y	-	Y	Y	Y
pac_RegistryHotPlug (Beta testing)	-	-	-	-	-	-	-
pac_UnregistryHotPlug (Beta testing)	-	-	-	-	-	-	-

Function List

The following functions are used to retrieve or set backplane functions.

Function	Description
pac_EnableRetrigger	This function decides the retrigger turning on or not.
pac_GetBackplaneID	This function retrieves the backplane ID.
pac_GetBatteryLevel	This function retrieves the backplane battery status.
pac_GetDIPSwitch	This function retrieves the dip switch.
pac_GetSlotCount	This function retrieves the number of slot.
pac_RegistryHotPlug (Beta testing)	This function registers the registry after turning on the hot plug.
pac_UnregistryHotPlug (Beta testing)	This function deletes the registry key after turning off the hot plug.

2.1. pac_EnableRetrigger

This function decides the retrigger turning on or not.

Syntax

```
void pac_EnableRetrigger(
   BYTE iValues
);
```

Parameters

iValues

[in] Decide turning on or not.

Return Values

None

Examples

None

2.2. pac_GetBackplaneID

This function retrieves the backplane ID.

Syntax

```
void pac_GetBackplaneID(
   LPSTR backplane_version
);
```

Parameters

backplane_version

[out] the pointer to a buffer to receive the backplane version.

Return Values

None

Examples

[eVC]

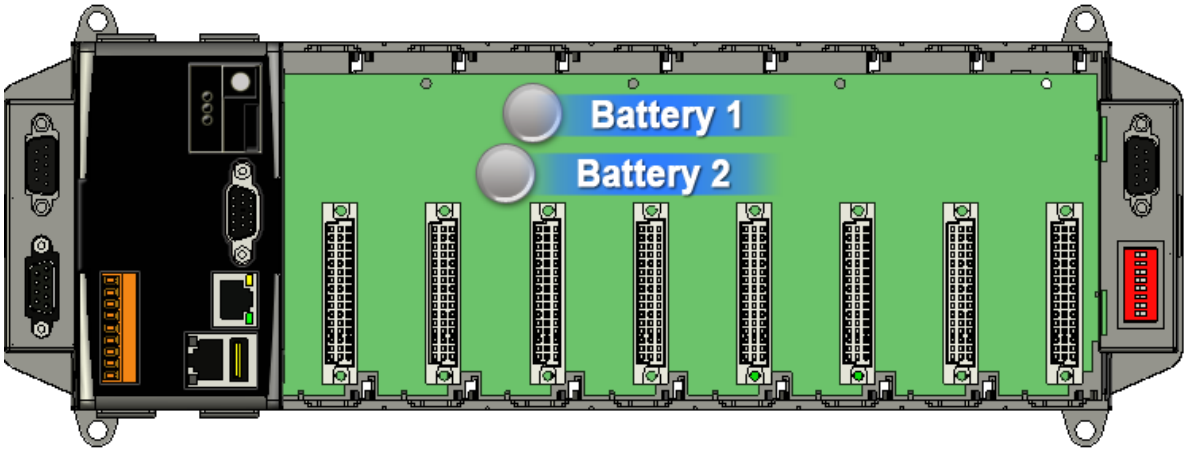
```
char Backplane[32];
pac_GetBackplaneID(Backplane);
```

[C#]

```
string Backplane;
Backplane = WinPAC.pac_GetBackplaneID();
```

2.3. pac_GetBatteryLevel

This function retrieves the backplane battery status.



Syntax

```
int pac_ GetBatteryLevel(
    int nBattery
);
```

Parameters

nBattery

[in] Specifies the index of battery.

1 means first battery

2 means second battery

Return Values

1 means high voltage

2 means low voltage

Examples

[eVC]

```
int nBattery;
int index = 1;
nBattery = pac_GetBatteryLevel(index);
```

[C#]

```
int nBattery;
int index = 1;
nBattery = WinPAC.pac_GetBatteryLevel(index);
```

2.4. pac_GetDIPSwitch

This function retrieves the dip switch.

Syntax

```
int pac_GetDIPSwitch();
```

Parameters

None

Return Values

Return Values of the DIP switch.

Examples

[eVC]

```
int iDipSwitch;
iDipSwitch = pac_GetDIPSwitch();
```

[C#]

```
int iDipSwitch;
iDipSwitch = WinPAC.pac_GetDIPSwitch();
```

2.5. pac_GetSlotCount

This function retrieves the number of slot.

Syntax

```
WORD pac_GetSlotCount();
```

Parameters

None

Return Values

Return the number of slots available.

Examples

[eVC]

```
WORD wSlot;
wSlot = pac_GetSlotCount();
```

[C#]

```
int wSlot;
wSlot = WinPAC.pac_GetSlotCount();
```

2.6. pac_RegistryHotPlug (Beta testing)

This function registers the registry after turning on the hot plug.

Syntax

```
void pac_RegistryHotPlug(
   DWORD hWnd,
   DWORD msgID
);
```

Parameters

hWnd

[in] User specifies the handle ID.

Return Values

None

Examples

None

2.7. pac_UnregistryHotPlug (Beta testing)

This function deletes the registry key after turning off the hot plug.

Syntax

```
void pac_UnregistryHotPlug(
   DWORD hWnd
);
```

Parameters

hWnd

[in] User specifies the handle ID.

Return Values

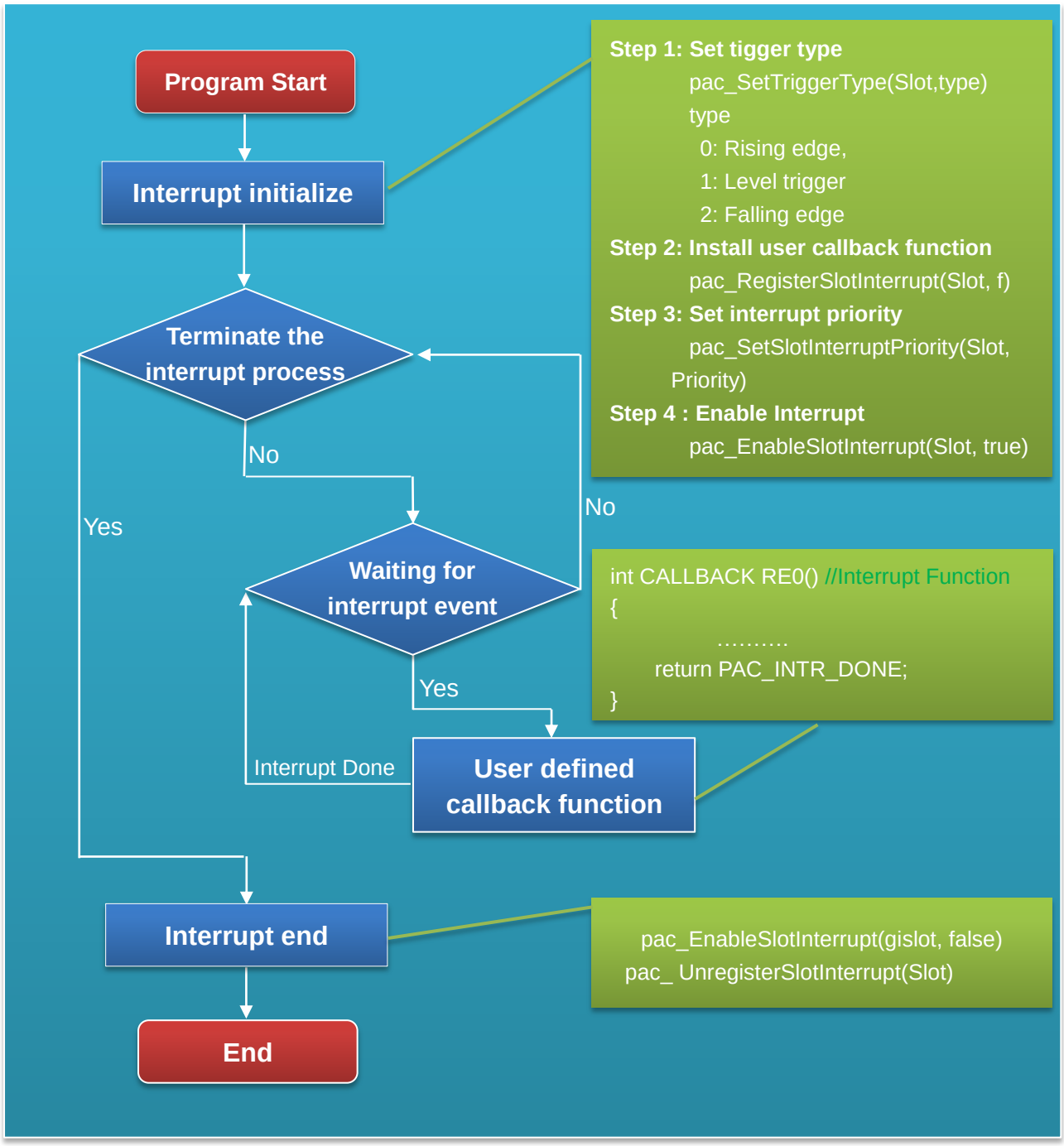
None

Examples

None

3. Interrupt API

Interrupt operations include basic management operations, such as interrupt done, enable and disable interrupt. The following topics describe how you can operate interrupt programmatically using the interrupt functions.



Supported Modules

The following shows the overview of the interrupt functions which are available with WinPAC.

Functions\Models	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_EnableSlotInterrupt	Y	Y	Y	Y	Y	Y	-
pac_GetSlotInterruptEvent	Y	Y	Y	Y	Y	Y	-
pac_GetSlotInterruptID	Y	Y	Y	Y	Y	Y	-
pac_InterruptDone	Y	Y	Y	Y	Y	Y	-
pac_InterruptInitialize	Y	Y	Y	Y	Y	Y	-
pac_RegisterSlotInterrupt	Y	Y	Y	Y	Y	Y	-
pac_SetSlotInterruptEvent	Y	Y	Y	Y	Y	Y	-
pac_SetSlotInterruptPriority	Y	Y	Y	Y	Y	Y	-
pac_SetTriggerType	Y	Y	Y	Y	Y	Y	-
pac_UnregisterSlotInterrupt	Y	Y	Y	Y	Y	Y	-

Function List

The following functions are used to retrieve or set interrupt functions.

Function	Description
pac_EnableSlotInterrupt	This function performs hardware operations necessary to enable the specified hardware interrupt.This function performs hardware operations necessary to enable the specified hardware interrupt.This function performs hardware operations necessary to enable the specified hardware interrupt.This function performs hardware operations necessary to enable the specified hardware interrupt.
pac_GetSlotInterruptEvent	This function retrieves the slot event handle which registered by pac_InterruptInitialize.
pac_GetSlotInterruptID	This function retrieves the slot interrupt ID.
pac_InterruptDone	This function signals to the kernel that interrupt processing has been completed.
pac_InterruptInitialize	This function initializes a slot interrupt with the kernel. This initialization allows the slot to register an event and enable the interrupt.
pac_RegisterSlotInterrupt	This function registers slot interrupt service route and turns on slot interrupt.
pac_SetSlotInterruptEvent	This function allows a device driver to assign the slot event handle.
pac_SetSlotInterruptPriority	This function sets the priority for a real-time thread on a thread by thread basis.
pac_SetTriggerType	This function can assign the pulse trigger type.
pac_UnregisterSlotInterrupt	This function unregisters slot interrupt service route and diables a hardware interrupt as specified by its interrupt identifier.

3.1. pac_EnableSlotInterrupt

This function performs hardware operations necessary to enable the specified hardware interrupt.

Syntax

```
void pac_EnableSlotInterrupt(
    BYTE slot,
    bool bEnable
);
```

Parameters

slot

[in] Specify the index of slot to enable interrupt or disable.

bEnable

[in] Specify the Slot interrupt turning on or not. Return Values

Return Values

None

Examples

[eVC]

```
int slot = 3;      // if slot is 3
int CALLBACK slot_callback_proc()
{
// do something
Pac_InterruptDone(slot);
  return 1;
// if return 0, SDK will do pac_InterruptDone automatically,
// else return 1, users should do pac_InterruptDone by themselves if nedded
}

void CIntrDlg::OnButton1()
{
  pac_RegisterSlotInterrupt(slot, slot_callback_proc);
  pac_EnableSlotInterrupt(slot, true);      // enable slot interrupt
}

void CIntrDlg::OnButton2()
{
  pac_EnableSlotInterrupt(slot, false);      // disable slot interrupt
  pac_UnregisterSlotInterrupt(slot);      // unregister slot interrupt
}
```

3.2. pac_GetSlotInterruptEvent

This function retrieves the slot event handle which registered by pac_InterruptInitialize.

Syntax

```
HANDLE pac_GetSlotInterruptEvent(
   BYTE slot
);
```

Parameters

slot

[in] Specify the index of slot to retrieve the event handle.

Return Values

Return a handle to the event object if success, otherwise, false.

Examples

None

3.3. pac_GetSlotInterruptID

This function retrieves the slot interrupt ID.

Syntax

```
DWORD pac_GetSlotInterruptID(
   BYTE Slot
);
```

Parameters

Slot

[in] Specify the slot.

Return Values

Return the slot interrupt ID.

Examples

None

3.4. pac_InterruptDone

This function signals to the kernel that interrupt processing has been completed.

Syntax

```
void pac_InterruptDone(
   BYTE slot
);
```

Parameters

Slot

[in] Specify the slot.

Return Values

Return the slot interrupt ID.

Examples

[eVC]

```
HANDLE hIntr;
bool bExit = false;
BYTE slot=0;

DWORD INTP_Thread(PVOID pContext)
{
   while (bExit)
{
WaitForSingleObject(hIntr, INFINITE);

   // do something

     pac_InterruptDone(slot);
   }
   pac_EnableSlotInterrupt(slot, false);
   pac_SetSlotInterruptEvent( slot, NULL);
   CloseHandle(pac_GetSlotInterruptEvent(slot));
   return 0;
}
void CInterruptDlg::OnButton1()
{
   bExit = true;

   pac_InterruptInitialize(slot);
   pac_EnableSlotInterrupt(slot, true);
   hIntr = pac_GetSlotInterruptEvent(slot);

   CreateThread(NULL, 0, INTP_Thread, &slot, 0, NULL);
}
```

3.5. pac_InterruptInitialize

This function initializes a slot interrupt with the kernel. This initialization allows the slot to register an event and enable the interrupt.

Syntax

```
bool pac_InterruptInitialize(
   BYTE slot
);
```

Parameters

slot

[in] Specify the index of slot to initialize.

Return Values

Return true if success, otherwise false.

Examples

None

Remarks

If you want to get the registered event handle, please call this API, pac_GetSlotInterruptEvent.

3.6. pac_RegisterSlotInterrupt

This function registers slot interrupt service route and turns on slot interrupt.

Syntax

```
bool pac_RegisterSlotInterrupt(
    BYTE slot,
    pac_CALLBACK_FUNC f
);
```

Parameters

slot

[in] Specify the index of slot.

f

A callback function.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
int slot = 3;      //if slot is 3
int CALLBACK slot_callback_proc()
{
// do something
  return 0;
// if return 0, SDK will do pac_InterruptDone automatically, else return 1,
// users should do pac_InterruptDone by themselves if nedded.
}

void CIntrDlg::OnButton1()
{
  pac_RegisterSlotInterrupt(slot, slot_callback_proc);
  pac_EnableSlotInterrupt(slot, true);      // enable slot interrupt
}

void CIntrDlg::OnButton2()
{
  Pac_EnableSlotInterrupt(slot, false);      //disable slot interrupt
  pac_UnregisterSlotInterrupt(slot);      // unregister slot interrupt
}
```

Remarks

To see more information, please reference user manual - Chapter 5 API and Demo Reference.

3.7. pac_SetSlotInterruptEvent

This function allows a device driver to assign the slot event handle.

Syntax

```
void pac_SetSlotInterruptEvent(
   BYTE slot,
   HANDLE hEvent
);
```

Parameters

slot

[in] Specify the index of slot to retrieve the event handle.

hEvent

[in] Event to be signaled.

Return Values

None

Examples

None

3.8. pac_SetSlotInterruptPriority

This function sets the priority for a real-time thread on a thread by thread basis.

Syntax

```
bool pac_SetSlotInterruptPriority(
    BYTE slot,
    int nPriority
);
```

Parameters

slot

[in] Specify the index of slot to set priority.

nPriority

[in] Priority to set for the thread.

This value can range from 0 through 255, with 0 as the highest priority.

Return Values

Return true if success, otherwise false.

Examples

None

3.9. pac_SetTriggerType

This function can assign the pulse trigger type.

Syntax

```
void pac_SetTriggerType(
    int iType
);
```

Parameters

iType

[in] Specify the pulse trigger type.

0: Rising edge trigger(default)

1: Level trigger

2: Falling edge trigger

Return Values

None

Examples

None

3.10. pac_UnregisterSlotInterrupt

This function unregisters slot interrupt service route and diables a hardware interrupt as specified by its interrupt identifier.

Syntax

```
bool pac_UnregisterSlotInterrupt(
    BYTE slot
);
```

Parameters

slot

[in] Specify the index of slot.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
int slot = 3; //if slot is 3
int CALLBACK slot_callback_proc()
{
// do something
  Pac_InterruptDone(slot);
  return 1;
// if return 0, SDK will do pac_InterruptDone automatically,
// else return 1, users should do pac_InterruptDone by themselves if nedded
}
void CIntrDlg::OnButton1()
{
  pac_RegisterSlotInterrupt(slot, slot_callback_proc);
  pac_EnableSlotInterrupt(slot, true);      // enable slot interrupt
}
void CIntrDlg::OnButton2()
{
pac_EnableSlotInterrupt(slot, false);     // disable slot interrupt
  pac_UnregisterSlotInterrupt(slot);      // unregister slot interrupt
}
```

Remarks

To see more information, please reference user manual - Chapter 5 API and Demo Reference

4. Memory Access API

Memory operations include basic management operations, such as reading from and writing to the EEPROM or SRAM. The following topics describe how you can read, or write data programmatically using the memory functions.

Supported Modules

The following shows the overview of the memory access functions which are available with WinPAC.

Functions / Modules	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_GetMemorySize	Y	Y	Y	Y▲	Y	Y	Y
pac_ReadMemory	Y	Y	Y	Y▲	Y	Y	Y
pac_WriteMemory	Y	Y	Y	Y▲	Y	Y	Y
pac_EnableEEPROM	Y	Y	Y	Y	Y	Y	Y

▲ WP-5xxx only supports the memory type 1 (EEPROM), not type 0 (SRAM).

Function List

The following functions are used to retrieve or set memory access functions.

Function	Description
pac_GetMemorySize	This function retrieves the size of the specified memory.
pac_ReadMemory	This function retrieves data from the specified memory.
pac_WriteMemory	This function stores data in the specified memory.
pac_EnableEEPROM	This function turns on/off EEPROM.

4.1. pac_GetMemorySize

This function retrieves the size of the specified memory.

Syntax

```
DWORD pac_GetMemorySize(
   int mem_type
);
```

Parameters

mem_type

[in] Handle to a currently type memory.

WINpac_MEM_SRAM 0

WINpac_MEM_EEPROM 1

Note: WP-5xxx series doesn't support SRAM

Return Values

The return value specifies the memory size.

Examples

[eVC]

```
DWORD mem_size;
mem_size = pac_GetMemorySize(WINpac_MEM_SRAM);
```

[C#]

```
uint mem_size;
mem_size = WinPAC.pac_GetMemorySize(0);
```

4.2. pac_ReadMemory

This function retrieves data from the specified memory.

Syntax

```
bool pac_ReadMemory(
    DWORD address,
    LPBYTE lpBuffer,
    DWORD dwLength,
    int mem_type
);
```

Parameters

address

[in] Specifies the memory address where read from.

lpBuffer

[in] A pointer to a buffer that receives the memory data.

dwLength

[in] Number of characters to be read.

mem_type

[in] Handle to a currently type memory.

WINpac_MEM_SRAM 0

WINpac_MEM_EEPROM 1

Note: WP-5xxx series doesn't support SRAM.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
DWORD address = 0;
BYTE Buffer[2];
ret = pac_ReadMemory(address, Buffer, 2, WINpac_MEM_SRAM);
```

[C#]

```
bool ret;
uint address = 0;
byte[] Buffer = new byte[2];
ret = WinPAC.pac_ReadMemory(address, Buffer, 2, 0);
```

Remarks

The input range size of EEPROM doesn't exceed 8K.

The architecture of EEPROM is as following:

0 ~0x1FFF (8k) for users

0x2000~0 x 3FFF(8k) for reserve

The input range size of SRAM only 0 ~0x6FFFF (448K)

Another 64K of SRAM is system reserved.

4.3. pac_WriteMemory

This function stores data in the specified memory.

Syntax

```
bool pac_WriteMemory(
    DWORD address,
    LPBYTE lpBuffer,
    DWORD dwLength,
    int mem_type
);
```

Parameters

address

[in] Specifies the memory address where write from.

lpBuffer

[in] A pointer to the buffer containing the data to be written to the memory.

dwLength

[in] Number of characters to be written.

mem_type

[in] Handle to a currently type memory.

WINpac_MEM_SRAM 0

WINpac_MEM_EEPROM 1

Note: WP-5xxx series doesn't support SRAM

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
DWORD address = 0;
BYTE Buffer[2];
Buffer[0] = 10;
Buffer[1] = 20;
ret = pac_WriteMemory(address, Buffer, 2, WINpac_MEM_SRAM);
```

[C#]

```
bool ret;
uint address = 0;
byte[] Buffer = new byte[2] { 10, 20 };
ret = WinPAC.pac_WriteMemory(address, Buffer, 2, 0);
```

Remarks

The input range size of EEPROM doesn't exceed 8K.

The architecture of EEPROM is as following:

0 ~0x1FFF (8k) for users

0x2000~0x3FFF(8k) for reserve

The input range size of SRAM only 0 ~0x6FFFF (448K)

Another 64K of SRAM is system reserved.

4.4. pac_EnableEEPROM

This function turns on/off EEPROM.

Syntax

```
void pac_EnableEEPROM(
    bool bEnable
);
```

Parameters

bEnable

[in] To turn on or protect EEPROM.

True: To turn on EEPROM.

False: To protect EEPROM.

Return Values

None

Examples

[eVC]

```
int ret;
DWORD address = 0;
BYTE Buffer[2];
Buffer[0] =0xAB;
Buffer[1] =0xCD;
pac_EnableEEPROM(true);
ret = pac_WriteMemory(address, Buffer, 2, WINpac_MEM_EEPROM);
pac_EnableEEPROM(false) ;
```

[C#]

```
bool ret;
uint address = 0;
byte[] Buffer = new byte[2] {0xAB,0xCD };
WinPAC.pac_EnableEEPROM(true);
ret = WinPAC.pac_WriteMemory(address, Buffer, (uint)Buffer.Length, 0);
WinPAC.pac_EnableEEPROM(false);
```

Remarks

Before writing EEPROM, you must enable the EEPROM first; after written EEPROM, you have to disable EEPROM.

5. Watchdog API

Watchdog operations include basic management operations, such as turning on and refreshing. The following topics describe how you can operate watchdog programmatically using the watchdog functions.

Supported Modules

The following shows the overview of the watchdog functions which are available with WinPAC.

Functions / Modules	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_DisableWatchDog	Y	Y	Y	Y	Y	Y	Y
pac_EnableWatchDog	Y	Y	Y	Y	Y	Y	Y
pac_GetWatchDogState	Y	Y	Y	Y	Y	Y	Y
pac_GetWatchdogTime	Y	Y	Y	Y	Y	Y	Y
pac_RefreshWatchDog	Y	Y	Y	Y	Y	Y	Y
pac_SetWatchDogTime	Y	Y	Y	Y	Y	Y	Y

Function List

The following functions are used to retrieve or set watchdog functions.

Function	Description
pac_DisableWatchDog	This function turns off the watchdog operation.
pac_EnableWatchDog	This function turns on the watchdog operation.
pac_GetWatchDogState	This function retrieves the watchdog state.
pac_GetWatchdogTime	This function retrieves the watchdog time.
pac_RefreshWatchDog	This function refreshes the watchdog.
pac_SetWatchDogTime	This function specifies the watchdog time.

5.1. pac_DisableWatchDog

This function turns off the watchdog operation.

Syntax

```
void pac_DisableWatchDog(
    int wdt
);
```

Parameters

WDT

[in] Specifies the Watchdog type:

WINpac_WDT_HW 0

WINpac_WDT_OS 1

Return Values

None

Examples

[eVC]

```
pac_DisableWatchDog(WINpac_WDT_OS);
```

[C#]

```
WinPAC.pac_DisableWatchDog(1);
```

5.2. pac_EnableWatchDog

This function turns on the watchdog operation.

Syntax

```
bool pac_EnableWatchDog(
    int wdt,
    DWORD value
);
```

Parameters

WDT

[in] Specifies the Watchdog type:

WINpac_WDT_HW 0

WINpac_WDT_OS 1

millisecond

[in] Specifies the watchdog time.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
DWORD millisecond = 1000;
bool ret;
ret = pac_EnableWatchDog(WINpac_WDT_OS, millisecond);
```

[C#]

```
uint millisecond = 1000;
bool ret_err;
ret_err = WinPAC.pac_EnableWatchDog(1, millisecond);
```

Remarks

The hardware watchdog second Parameters is a value which is between 0~31 unit. A unit is about 200 milliseconds. 0 means the shortest timeout, otherwise 31 is longest.

5.3. pac_GetWatchDogState

This function retrieves the watchdog state.

Syntax

```
bool pac_GetWatchDogState(
    int wdt
);
```

Parameters

WDT

[in] Specifies the Watchdog type:

WINpac_WDT_HW 0

WINpac_WDT_OS 1

Return Values

The return value specifies the watchdog state.

If turning on, return true, otherwise false.

Examples

[eVC]

```
bool bState;
bState = pac_GetWatchDogState(WINpac_WDT_OS);
```

[C#]

```
bool bState;
bState = WinPAC.pac_GetWatchDogState(1);
```

5.4. pac_GetWatchdogTime

This function retrieves the watchdog time.

Syntax

```
DWORD pac_GetWatchDogTime(
    int wdt
);
```

Parameters

WDT

[in] Specifies the Watchdog type:

WINpac_WDT_HW 0

WINpac_WDT_OS 1

Return Values

The return value is the watchdog time which has been assigned by pac_EnableWatchDog or pac_SetWatchDogTime.

Examples

[eVC]

```
DWORD dwTime;
dwTime = pac_GetWatchDogTime(WINpac_WDT_OS);
```

[C#]

```
uint uTime;
uTime = WinPAC.pac_GetWatchDogTime(1);
```

Remarks

The same as the pac_EnableWatchDog function, the pac_GetWatchDogTime of hardware watchdog retrieves a value between 0~31 but millisecond.

5.5. pac_RefreshWatchDog

This function refreshes the watchdog.

Syntax

```
void pac_RefreshWatchDog(
    int wdt
);
```

Parameters

WDT

[in] Specifies the Watchdog type:

WINpac_WDT_HW 0

WINpac_WDT_OS 1

Return Values

None

Examples

[eVC]

```
pac_RefreshWatchDog(WINpac_WDT_OS);
```

[C#]

```
WinPAC.pac_RefreshWatchDog(1);
```

5.6. pac_SetWatchDogTime

This function specifies the watchdog time.

Syntax

```
void pac_SetWatchDogTime(
    int wdt,
    DWORD value
);
```

Parameters

WDT

[in] Specifies the Watchdog type:

WINpac_WDT_HW

WINpac_WDT_OS

millisecond

[in] Specifies the watchdog time.

Return Values

None

Examples

[eVC]

```
DWORD dwTime = 1000;
pac_SetWatchDogTime(WINpac_WDT_OS, dwTime);
```

[C#]

```
uint uTime = 1000;
WinPAC.pac_SetWatchDogTime(1, uTime);
```

Remarks

The same as the pac_EnableWatchDog function.

The hardware watchdog second Parameters is a value which is between 0~31 unit.

A unit is about 200 milliseconds.

0 means the shortest timeout, otherwise 31 is longest.

6. microSD Management API

MicroSD operations include basic management operations, such as mounting and unmounting. The following topics describe how you can mount, or unmount MicroSD programmatically using the memory functions.

Supported Modules

The following shows the overview of the memory access functions which are available with WinPAC.

Functions / Modules	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_SDExists	Y	Y	-	Y	Y	Y	Y
pac_SDMount	Y	Y	-	Y	Y	Y	Y
pac_SDOnside	Y	Y	-	Y	Y	Y	Y
pac_SDUnmount	Y	Y	-	Y	Y	Y	Y

Function List

The following functions are used to retrieve or set memory access functions.

Function	Description
pac_SDExists	This function is used to check that the Micro SD whether has been standby ready or not.
pac_SDMount	This function is used to mount Micro SD if the Micro SD has been inserting to the device without mounting.
pac_SDOnside	This function is used to check the Micro SD whether on-side or not.
pac_SDUnmount	This function is used to dismount Micro SD if the Micro SD has been mounted.

6.1. pac_SDExists

This function is used to check that the Micro SD whether has been standby ready or not.

Syntax

```
bool pac_SDExists();
```

Parameters

None

ReturnValues

Returns TRUE if successful, otherwise FALSE.

Examples

[eVC]

```
bool bExist;
bExist = pac_SDExists();
```

[C#]

```
bool bExist;
bExist = WinPAC.pac_SDExists();
```

6.2. pac_SDMount

This function is used to mount Micro SD if the Micro SD has been inserting to the device without mounting.

Syntax

```
bool pac_SDMount(
   LPTSTR szPartitionName
);
```

Parameters

szPartitionName

[in] Name of the partition. Example Part00.

ReturnValues

Returns TRUE if successful, otherwise FALSE.

Examples

[eVC]

```
bool ret;
ret = pac_SDMount(TEXT("Part00"));
```

[C#]

```
bool ret;
ret = WinPAC.pac_SDMount("Part00");
```

6.3. pac_SDOnside

This function is used to check the Micro SD whether on-side or not.

Syntax

```
bool pac_SDOnside();
```

Parameters

None

ReturnValues

Returns TRUE if successful, otherwise FALSE.

Examples

[eVC]

```
bool ret;
ret = pac_SDOnside();
```

[C#]

```
bool ret;
ret = WinPAC.pac_SDOnside();
```

6.4. pac_SDUnmount

This function is used to dismount Micro SD if the Micro SD has been mounted.

Syntax

```
bool pac_SDUnmount();
```

Parameters

None

ReturnValues

Returns TRUE if successful, otherwise FALSE.

Examples

[eVC]

```
bool ret;
ret = pac_SDUnmount();
```

[C#]

```
bool ret;
ret = WinPAC.pac_SDUnmount();
```

7. Registry API

Registry operations include basic management operations, such as reading from and writing to the registry. The following topics describe how you can create, delete, or modify registry keys programmatically using the registry functions.

Supported Modules

The following shows the overview of the registry functions which are available with WinPAC.

Functions\Models	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_RegCountKey	Y	Y	Y	Y	Y	Y	Y
pac_RegCountValue	Y	Y	Y	Y	Y	Y	Y
pac_RegCreateKey	Y	Y	Y	Y	Y	Y	Y
pac_RegDeleteKey	Y	Y	Y	Y	Y	Y	Y
pac_RegDeleteValue	Y	Y	Y	Y	Y	Y	Y
pac_RegGetDWORD	Y	Y	Y	Y	Y	Y	Y
pac_RegGetKeyByIndex	Y	Y	Y	Y	Y	Y	Y
pac_RegGetKeyInfo	Y	Y	Y	Y	Y	Y	Y
pac_RegGetString	Y	Y	Y	Y	Y	Y	Y
pac_RegGetValueByIndex	Y	Y	Y	Y	Y	Y	Y
pac_RegKeyExist	Y	Y	Y	Y	Y	Y	Y
pac_RegSave	Y	Y	Y	Y	Y	Y	Y
pac_RegSetString	Y	Y	Y	Y	Y	Y	Y
pac_RegSetDWORD	Y	Y	Y	Y	Y	Y	Y

Function List

The following functions are used to retrieve or set registry information.

Function	Description
pac_RegCountKey	This function retrieves the specified registry key which has how many sub keys.
pac_RegCountValue	This function retrieves the specified registry key which has how many values.
pac_RegCreateKey	This function creates the specified registry key.
pac_RegDeleteKey	This function deletes a named subkey from the specified registry key.
pac_RegDeleteValue	This function removes a named value from the specified registry key.
pac_RegGetDWORD	This function retrieves value of the specified registry key.
pac_RegGetKeyByIndex	This function retrieves the name of specified index of registry key.
pac_RegGetKeyInfo	This function retrieves the type of specified index of registry key.
pac_RegGetString	This function retrieves value of the specified registry key.
pac_RegGetValueByIndex	This function retrieves the value of specified index of registry key.
pac_RegKeyExist	This function determinants the specified registry key exist or not.
pac_RegSave	This function writes all the attributes of the specified open registry key into the registry.
pac_RegSetString	This function assigns the specified registry key data whose type is string.
pac_RegSetDWORD	This function assigns the specified registry key data whose type is DWORD.

7.1. pac_RegCountKey

This function retrieves the specified registry key which has how many sub keys.

Syntax

```
DWORD pac_RegCountKey(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Specific the path of registry key which you want to count.

Return Values

Return the number of subkeys contained by the specified key.

Examples

[eVC]

```
DWORD i;
i = pac_RegCountKey(TEXT("HKEY_USERS\\myKey\\"));
```

[C#]

```
uint i;
i = WinPAC.pac_RegCountKey("HKEY_USERS\\myKey\\");
```

7.2. pac_RegCountValue

This function retrieves the specified registry key which has how many values.

Syntax

```
DWORD pac_RegCountValue(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Specific the path of registry key which you want to count.

Return Values

Return the number of values associated with the key.

Examples

[eVC]

```
DWORD i;
i = pac_RegCountValue(TEXT("HKEY_USERS\\myKey\\"));
```

[C#]

```
uint i;
i = WinPAC.pac_RegCountValue("HKEY_USERS\\myKey\\");
```

7.3. pac_RegCreateKey

This function creates the specified registry key.

Syntax

```
bool pac_RegCreateKey(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Specific the path of registry key which you want to create.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
ret = pac_RegCreateKey(TEXT("HKEY_USERS\\myKey\\"));
```

[C#]

```
bool ret;
ret = WinPAC.pac_RegCreateKey("HKEY_USERS\\myKey\\");
```

7.4. pac_RegDeleteKey

This function deletes a named subkey from the specified registry key.

Syntax

```
bool pac_RegDeleteKey(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Specifics the path of registry key which you want to delete.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
ret = pac_RegDeleteKey(TEXT("HKEY_USERS\\myKey\\"));
```

[C#]

```
bool ret;
ret = WinPAC.pac_RegDeleteKey("HKEY_USERS\\myKey\\");
```

Remarks

If you delete a key which doesn't exist, nothing happen. If the function succeeds, the function will delete the specified key including all of its subkeys and values. An application cannot call RegDeleteKey for a key that an application currently has open.

7.5. pac_RegDeleteValue

This function removes a named value from the specified registry key.

Syntax

```
bool pac_RegDeleteValue(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Specifics the path of registry key which you want to delete.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
ret = pac_RegDeleteValue(TEXT("HKEY_USERS\\myKey\\value"));
```

[C#]

```
bool ret;
ret = WinPAC.pac_RegDeleteValue(TEXT("HKEY_USERS\\myKey\\value"));
```

Remarks

The function could be used only in leaf key. And the function, pac_RegDeleteKey, can be used in any registry key except leaf key. If you would delete a leaf key, you should call pac_RegDeleteValue.

7.6. pac_RegGetDWORD

This function retrieves value of the specified registry key.

Syntax

```
DWORD pac_RegGetDWORD(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Specific the path of registry key.

Return Values

Return the value of the specific key.

Examples

[eVC]

```
DWORD dwValue;
dwValue = pac_RegGetDWORD(TEXT("HKEY_USERS\\myKey\\value"));
```

[C#]

```
uint uValue;
uValue = WinPAC.pac_RegGetDWORD("HKEY_USERS\\myKey\\value");
```

7.7. pac_RegGetKeyByIndex

This function retrieves the name of specified index of registry key.

Syntax

```
bool pac_RegGetKeyByIndex(
    LPCTSTR KeyName,
    DWORD dwIndex,
    LPTSTR lpName
);
```

Parameters

KeyName

[in] Specific the path of registry key.

dwIndex

[in] Specific the index of registry key.

lpName

[out] Assign a buffer to retrieves the specific the key name.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
DWORD index=0;
TCHAR strName[10];
ret = pac_RegGetKeyByIndex(TEXT("HKEY_USERS\\myKey\\"), index, strName);
```

[C#]

```
bool ret;
uint index = 0;
string strName = new String('\0', 10);
ret = WinPAC.pac_RegGetKeyByIndex("HKEY_USERS\\myKey\\", index, strName);
```

7.8. pac_RegGetKeyInfo

This function retrieves the type of specified index of registry key.

Syntax

```
DWORD pac_RegGetKeyInfo(LPCTSTR KeyName);
```

Parameters

KeyName

[in] Specific the path of registry key.

Return Values

We define four types about the return value:

PKT_NONE 0

PKT_KEY 1

PKT_STRING 2

PKT_DWORD 3

Examples

[eVC]

```
DWORD dwType;
dwType = pac_RegGetKeyInfo(TEXT("HKEY_USERS\\myKey\\value"));
```

[C#]

```
uint uType;
uType=WinPAC.pac_RegGetKeyInfo("HKEY_USERS\\myKey\\value");
```

7.9. pac_RegGetString

This function retrieves value of the specified registry key.

Syntax

```
bool pac_RegGetString(
    LPCTSTR KeyName,
    LPTSTR lpData,
    DWORD dwLength
);
```

Parameters

KeyName

[in] Specific the path of registry key.

lpData

[out] Pointer to a buffer that receives the value's data.

dwLength

[in] Specific the size of data.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
TCHAR strName[10];
ret = pac_RegGetString(TEXT("HKEY_USERS\\myKey\\value"), strName,
sizeof(strName));
```

[C#]

```
bool ret;
string strName = new String('\0', 10);
ret = WinPAC.pac_RegGetString("HKEY_USERS\\myKey\\value", strName,
(uint)strName.Length);
```

7.10. pac_RegGetValueByIndex

This function retrieves the value of specified index of registry key.

Syntax

```
bool pac_RegGetValueByIndex(
    LPCTSTR KeyName,
    DWORD dwIndex,
    LPTSTR lpName
);
```

Parameters

KeyName

[in] Specific the path of registry key.

dwIndex

[in] Specific the index of value.

lpName

[out] Pointer to a buffer that receives the value's data.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
DWORD index=0;
TCHAR strName[10];
ret = pac_RegGetValueByIndex(TEXT("HKEY_USERS\\myKey\\"), index, strName);
```

[C#]

```
bool ret;
uint index = 0;
string strName = new String('\0', 10);
ret = WinPAC.pac_RegGetValueByIndex("HKEY_USERS\\myKey\\", index, strName);
```

7.11. pac_RegKeyExist

This function determinants the specified registry key exist or not.

Syntax

```
bool pac_RegKeyExist(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Specific the path of registry key which you want to check whether it exists or not.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool bExist;
bExist = pac_RegKeyExist(TEXT("HKEY_USERS\\myKey\\"));
```

[C#]

```
bool bExist;
bExist = WinPAC.pac_RegKeyExist("HKEY_USERS\\myKey\\");
```

7.12. pac_RegSave

This function writes all the attributes of the specified open registry key into the registry.

Syntax

```
bool pac_RegSave(
   LPCTSTR KeyName
);
```

Parameters

KeyName

[in] Handle to a currently open key or one of the following predefined reserved handle values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
ret = pac_RegSave(TEXT("HKEY_USERS "));
```

[C#]

```
bool ret;
ret = WinPAC.pac_RegSave("HKEY_USERS ");
```

7.13. pac_RegSetString

This function assigns the specified registry key data whose type is string.

Syntax

```
bool pac_RegSetString(
    LPCTSTR KeyName,
    LPCTSTR assignStr,
    DWORD dwLength
);
```

Parameters

KeyName

[in] Specific the path of registry key which you want to assign data.

assignStr

[in] Specific the data.

dwLength

[in] Specific the size of data.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
ret = pac_RegSetString(TEXT("HKEY_USERS\\myKey\\value"),
TEXT("Hello.exe"), 2*wcslen( TEXT("Hello.exe"))); //sizeof(TCHAR)=2
```

[C#]

```
bool ret;
ret = WinPAC.pac_RegSetString(("HKEY_USERS\\myKey\\value"), "Hello.exe", 2 *
(uint)"Hello.exe".Length);
```

7.14. pac_RegSetDWORD

This function assigns the specified registry key data whose type is DWORD.

Syntax

```
bool pac_RegSetDWORD(
   LPCTSTR KeyName,
   DWORD assignVal
);
```

Parameters

KeyName

[in] Specific the path of registry key which you want to assign data.

assignStr

[in] Specific the data.

Return Values

Return true if success, otherwise false. To get an error code, call pac_GetLastError. A nonzero error code defined in PACERROR.h indicates failure. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
bool ret;
ret = pac_RegSetDWORD(TEXT("HKEY_USERS\\myKey\\value"), 40);
```

[C#]

```
bool ret;
ret = WinPAC.pac_RegSetDWORD("HKEY_USERS\\myKey\\value", 40);
```

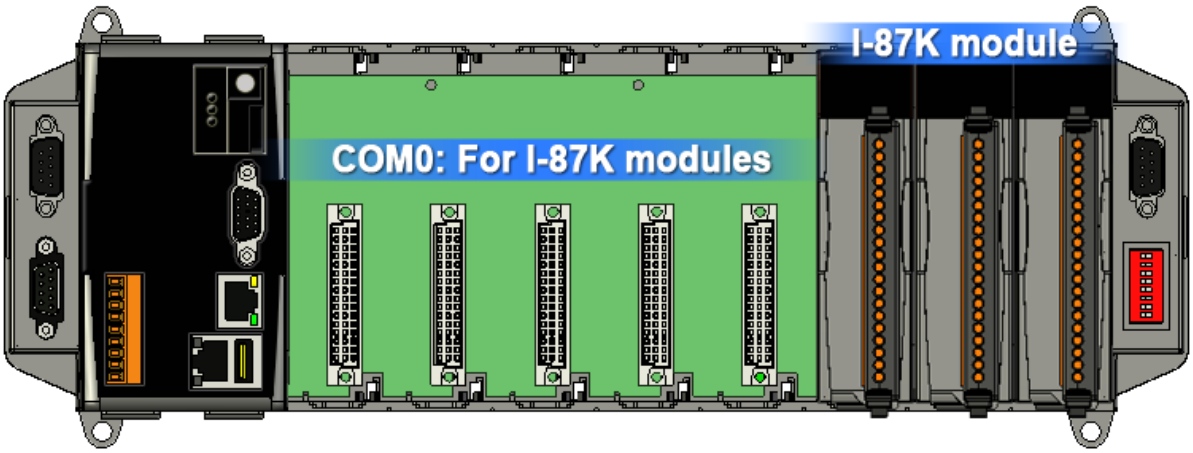
8. UART API

Uart operations include basic management operations, such as opening, sending, receiving, and closing. The following topics describe how you can operate uart programmatically using the uart functions.

We provide several COM port functions (uart_Send/uart_Recv…) to communicate with ICPDAS modules (High profile I-87K series, I-811xW/I-814xW series, I-7000 series). All the functions are based on standard COM port function in EVC++ (CreateFile/CloseHandle/WriteFile/ReadFile /GetCommModemStatus…..).

WP-8000 / ViewPAC (WinCE)

Use these functions of this section to communicate with I-87K

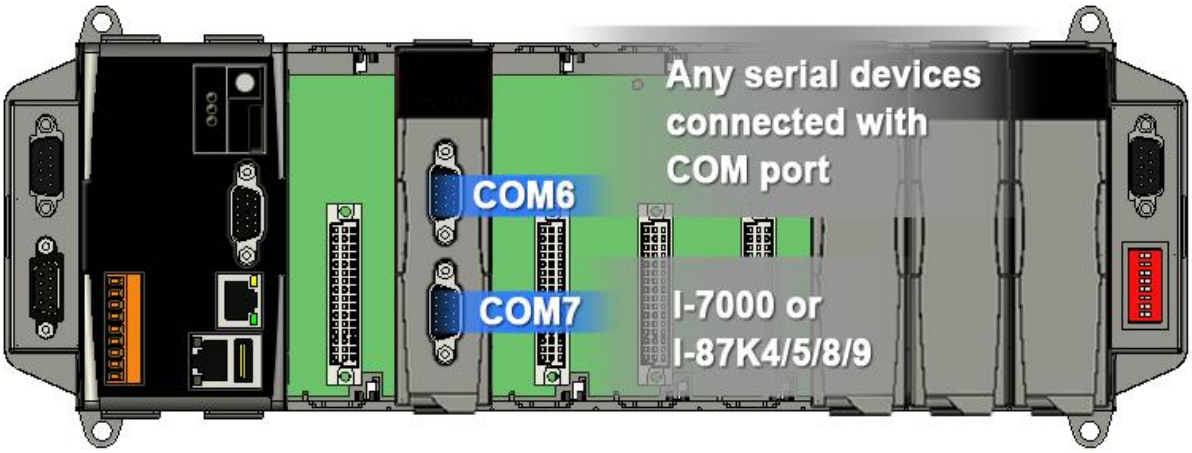


When a high profile I-87K is plugged in slot, please call the function, pac_ChangeSlot, to change to the specific slot before doing other operations. Please refer to demo “87k_Basic”.

About I-87K commands (DCON protocol), please refer http://ftp.icpdas.com/pub/cd/8000cd/napdos/dcon/io_module/87k_high_profile_modules.htm

Although user can use UART API to set and read values for high profile I-87K series modules, we provide a more convenient API to do it. Please refer Section 9 PAC_IO API

Use these functions of this section to communicate with external devices by I-811xW/I-814xW serises modules.



To see more information, please reference user manual - Chapter 5 API and Demo Reference.

WP-5000

The WP-5000 series has no slots for plugging the high profile I-8K and I-87K series, but the UART API on this section can also be used for COM1/COM2/COM3 of WP-5000 series.

In addition of COM1/COM2/COM3, all of the functions can be used to communicate with external device by XW5xxx expansion board.

To see more information, please reference user manual - Chapter 5 API and Demo Reference.

Supported Modules

The following shows the overview of the uart functions which are available with WinPAC.

Functions\Models	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
uart_BinRecv	Y	Y	Y	Y	Y	Y	Y
uart_BinSend	Y	Y	Y	Y	Y	Y	Y
uart_BinSendCmd	Y	Y	Y	Y	Y	Y	Y
uart_Close	Y	Y	Y	Y	Y	Y	Y
uart_EnableCheckSum	Y	Y	Y	Y	Y	Y	Y
uart_GetDataSize	Y	Y	Y	Y	Y	Y	Y
uart_GetLineStatus	Y	Y	Y	Y	Y	Y	Y
uart_Open	Y	Y	Y	Y	Y	Y	Y
uart_Recv	Y	Y	Y	Y	Y	Y	Y
uart_Send	Y	Y	Y	Y	Y	Y	Y
uart_SendCmd	Y	Y	Y	Y	Y	Y	Y
uart_SetTerminator	Y	Y	Y	Y	Y	Y	Y
uart_SetTimeOut	Y	Y	Y	Y	Y	Y	Y

Function List

The following functions are used to retrieve or set uart information.

Function	Description
uart_BinRecv	This function is applied to receive the fix length response.
uart_BinSend	This function can send out command string with or without null character under the consideration of the command length.
uart_BinSendCmd	This function is used to Send binary command and receive binary data with the fixed length.
uart_Close	This function closes the COM port which has been opened.
uart_EnableCheckSum	This function turns on the check sum or not.
uart_GetDataSize	This function retrieves the number of bytes received by the serial provider but not yet read by a uart_Recv operation, or of user data remaining to transmit for write operations.
uart_GetLineStatus	This function retrieves the modem control-register values.
uart_Open	This function opens the COM port and specifies the baud rate, parity bits, data bits, and stop bits.
uart_Recv	This function retrieves data through the COM port which have been opened.
uart_Send	This function sends data through the COM port which have been opened.
uart_SendCmd	This function sends commands through the COM port which have been opened.
uart_SetTerminator	This function sets the terminate characters.
uart_SetTimeOut	This function sets the time out timer.

8.1. uart_BinRecv

This function is applied to receive the fix length response. The length of the receiving response is controlled by the Parameters “in_Len”. The difference between this function and uart_Recv is that uart_BinRecv terminates the receiving process by the string length “in_Len” instead of the character "CR"(Carry return). Therefore, this function can be used to receive the response string data with or without null character under the consideration of receiving length. Besides, because of this function without any error checking mechanism (checksum, CRC, LRC... etc.), users have to remove the error checking information from the raw data by themselves if communication checking system is used.

Syntax

```
bool uart_BinRecv(
    HANDLE hPort,
    LPSTR buf,
    DWORD in_Len
);
```

Parameters

hPort

[in] Handle to the open COM port.

buf

[out] A pointer to a buffer that receives the data.

in_Len

[in] The length of result string.

Return Values

1 indicates success. 0 indicates failure.

Examples

[eVC]

```
bool ret;
    HANDLE hPort;
    char buf[2];
    hPort = uart_Open("COM3:,9600,N,8,1");
    ret = uart_BinSend(hPort, "AB", 2);
    ret = uart_BinRecv(hPort, buf, 2);
uart_Close(hPort);
```

[C#]

```
bool ret;
IntPtr hPort;
byte[] buf = new byte[100];
hPort = WinPAC.uart_Open("COM3:,9600,N,8,1");
ret = WinPAC.uart_BinSend(hPort, WinPAC.AnsiString("AB"), 2);
ret = WinPAC.uart_Recv(hPort, buf);
WinPAC.uart_Close(hPort);
```

Remarks

Note that this function is usually applied to communicate with the other device, but not for ICPDAS DCON (I-7000/8000/87K) series modules.

8.2. uart_BinSend

Send out the command string by fix length, which is controlled by the Parameters “in_Len”. The difference between this function and uart_Send is that uart_BinSend terminates the sending process by the string length “in_Len” instead of the character "CR"(Carry return). This function can send out command string with or without null character under the consideration of the command length. Besides, because of this function without any error checking mechanism (Checksum, CRC, LRC... etc.), users have to add the error checking information to the raw data by themselves if communication checking system is required.

Syntax

```
bool uart_BinSend(
    HANDLE hPort,
    LPCSTR buf,
    DWORD in_Len
);
```

Parameters

hPort

[in] Handle to the open COM port.

buf

[in] A pointer to a buffer that send the data.

in_Len

[in] The length of result string.

Return Values

1 indicates success. 0 indicates failure.

Examples

[eVC]

```
bool ret;
HANDLE hPort;
char buf[2];
buf[0] =0x41;
buf[1] =0x42;
hPort = uart_Open("COM3:,9600,N,8,1");
ret = uart_BinSend(hPort, buf, 2);
uart_Close(hPort);
```

[C#]

```
bool ret;
IntPtr hPort;
string buf = "AB";
hPort = WinPAC.uart_Open("COM3:,9600,N,8,1");
ret = WinPAC.uart_BinSend(hPort, WinPAC.AnsiString(buf), 2);
WinPAC.uart_Close(hPort);
```

Remarks

Note that this function is usually applied to communicate with the other device, but not for ICPDAS DCON (I-7000/8000/87K) series modules.

This function will call PurgeComm() to clear serial COM port output buffer.

8.3. uart_BinSendCmd

This function is used to Send binary command and receive binary data with the fixed length.

This function is a combination of uart_BinSend and uart_BinRecv.

The operation for sending a command is the same as uart_BinSend.

The operation for receiving a response is the same as uart_BinRecv.

Syntax

```
bool uart_BinSendCmd(
    HANDLE hPort,
    LPCSTR ByteCmd,
    DWORD in_Len,
    LPSTR ByteResult,
    DWORD out_Len
);
```

Parameters

hPort

[in] Handle to the open COM.

ByteCmd

[in] A pointer to a command.

in_Len

[in] The length of command string.

ByteResult

[out] A pointer to a buffer that receives the data.

out_Len

[in] The length of result string.

Return Values

1 indicates success. 0 indicates failure.

Examples

[eVC]

```
bool ret;
HANDLE hPort;
char buf[2];
char cmd[2];
hPort = uart_Open("COM3,9600,N,8,1");
cmd[0] =0x41;
cmd[1] =0x42;
ret = uart_BinSendCmd( hPort, cmd, 2, buf, 2);
uart_Close(hPort);
```

[C#]

```
bool ret;
byte[] cmd = new byte[2];
IntPtr hPort;
byte[] buf = new byte[2];
cmd[0] =0x41;
cmd[1] =0x42;
hPort = WinPAC.uart_Open("COM3:,9600,N,8,1");
ret = WinPAC.uart_BinSendCmd(hPort, cmd, 2, buf, 2);
WinPAC.uart_Close(hPort);
```

Remarks

This function will call PurgeComm() to clear serial COM port output and input buffer.

8.4. uart_Close

This function closes the COM port which has been opened.

Syntax

```
bool uart_Close(
   HANDLE hPort
);
```

Parameters

hPort

[in] Handle to the open COM port to close.

Return Values

1 indicates success. 0 indicates failure.

Examples

[eVC]

```
bool ret;
HANDLE hPort;
hPort = uart_Open("COM2:,9600,N,8,1");
ret = uart_Close(hPort);
```

[C#]

```
bool ret;
IntPtr hPort;
hPort = WinPAC.uart_Open("COM2:,9600,N,8,1");
ret = WinPAC.uart_Close(hPort);
```

Remarks

The function for a specified COM port should not be used after it has been closed.

8.5. uart_EnableCheckSum

This function turns on the check sum or not.

Add two checksum bytes to the end of data which is used to produce checksum.

Syntax

```
void uart_EnableCheckSum(
    HANDLE hPort,
    bool bEnable
);
```

Parameters

hPort

[in] Handle to the open COM port.

bEnable

[in] Decide the check sum turning on or not.

Default is disabling.

Return Values

None

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM3,9600,N,8,1");
uart_EnableCheckSum(hPort, true);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM3,9600,N,8,1");
WinPAC.uart_EnableCheckSum(hPort, true);
WinPAC.uart_Close(hPort);
```

8.6. uart_GetDataSize

This function retrieves the number of bytes received by the serial provider but not yet read by a uart_Recv operation, or of user data remaining to transmit for write operations.

Syntax

```
BOOL uart_GetDataSize(
   HANDLE hPort,
   int data_type
);
```

Parameters

hPort

[in] Handle to the open COM port.

data_type

[in] A value specifies to retrieve in or out buffer.

This Parameters can be following values:

#define IN_DATA 0

#define OUT_DATA 1

Return Values

The number of bytes in/out buffer but not yet read/write.

Examples

[eVC]

```
char buf[1024];
DWORD in_Len;
HANDLE hPort = uart_Open("COM1,9600,N,8,1");
in_Len = uart_GetDataSize(hPort, IN_DATA);
BOOL ret = uart_BinRecv( hPort, buf, in_Len);
uart_Close(hPort);
```

[C#]

```
string buf;
uint in_Len;
IntPtr hPort = WinPAC.uart_Open("COM1,9600,N,8,1");
in_Len = WinPAC.uart_GetDataSize(hPort, 0);      // 0: IN_DATA; 1: OUT_DATA
bool ret = WinPAC.uart_BinRecv( hPort, WinPAC.AnsiString(buf), in_Len);
WinPAC.uart_Close(hPort);
```

8.7. uart_GetLineStatus

This function retrieves the modem control-register values.

Syntax

```
BOOL uart_GetLineStatus(
    HANDLE hPort,
    int pin
);
```

Parameters

hPort

[in] Handle to the open COM port.

pin

[in] A variable specifies state of a pin of the COM port.

This Parameters can be following values:

```
#define   CTS   0
#define   DSR   1
#define   RI    2
#define   CD    3
```

Return Values

TRUE indicates the pin's state is ON. 0 indicates OFF.

Examples

[eVC]

```
HANDLE hPort = uart_Open("COM4,115200,N,8,1");
BOOL ret = uart_GetLineStatus(hPort, DSR);      //the pin, DSR, for example
if(ret)
printf("The status of DSR is ON\n");
else
printf("The status of DSR is OFF\n");
uart_Close(hPort);
```

[C#]

```
IntPtr hPort = WinPAC.uart_Open("COM4,115200,N,8,1");
//the pin, DSR, for example
bool ret = WinPAC.uart_GetLineStatus(hPort, DSR);
if(ret)
Console.WriteLine("The status of DSR is ON");
else
Console.WriteLine ("The status of DSR is OFF");
WinPAC.uart_Close(hPort);
```

8.8. uart_Open

This function opens the COM port and specifies the baud rate, parity bits, data bits, and stop bits.

Syntax

```
HANDLE uart_Open(
   LPCSTR ConnectionString
);
```

Parameters

ConnectionString

[in] Specifies the COM port, baud rate, parity bits, data bits, and stop bits.

The default setting is COM0:,115200,N,8,1.

The format of ConnectionString is as follow:

“com_port, baud_rate, parity_bits, data_bits, stop_bits”

Warning: there is no blank space between each Parameters.

Com_port: COM0, COM1……

baud_rate : 9600/19200/38400/57600/115200

parity_bits: 'N' = NOPARITY
'O' = ODDPARITY
'E' = EVENPARITY
'M' = MARKPARITY
'S' = SPACEPARITY

Data_bits: 5/6/7/8

Stop_bits: "1" = ONESTOPBIT
"2" = TWOSTOPBITS
"1.5" = ONE5STOPBITS;

Return Values

A handle to the open COM port.

Nonzero indicates success.

If the function fails, the return value is INVALID_HANDLE_Value.

(INVALID_HANDLE_Value should be0xffffffff in eVC.

INVALID_HANDLE_Value should be -1 in .NET.)

To get extended error information, call GetLastError. To get a generic description of the error, call pac_GetErrorMessage. The message resource is optional; therefore, if you call pac_GetErrorMessage it could fail.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM2:,9600,N,8,1");
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM2:,9600,N,8,1");
```

Remarks

The uart_Open function does not open the specified COM port if the COM port has been opened.

[Use I-811xW/I-814xW series modules]

The COM port name is COM6/COM7/MSA1/MSB1…., MSAx/MSBx is an earlier old usage. The new usage is COMx.

For example:

uart_Open("COM6:,9600,N,8,1");

uart_Open("MSA1:,9600,N,8,1");

About how to set I-811xW/I-814xW series modules, Please refer to the manual below:
w1-007-1_how_to_set_up_a_communication_module(I-8112_I-8114_I-8142_I-8144)_use_COM_english.pdf

w1-007-2_how_to_set_up_a_communication_module(I-8112_I-8114_I-8142_I-8144)_use_MSA(B..)_english.pdf

[Use I-87K series modules]

Only use COM0 to communicate with I-87K series modules. Please refer to Sec.8 UART API.

8.9. uart_Recv

This function retrieves data through the COM port which have been opened.

This function will receive a string+0x0D. Wait a character [0x0D] to mean the termination of a string. And then if the checksum is enabled by the uart_EnableCheckSum function, this function automatically check the two checksum bytes to the string. This function will provides a string without the last byte[0x0D].

Syntax

```
bool uart_Recv(
    HANDLE hPort,
    LPCSTR buf
);
```

Parameters

hPort

[in] Handle to the open COM port.

buf

[out] A pointer to a buffer that receives the data.

Return Values

1 indicates success. 0 indicates failure.

If this function doesn't receive a character0x0D, the other data still store to "buf" but the return value is 0. Calling GetLastError function will get an error code (pac_ERR_uart_READ_TIMEOUT).

Examples

[eVC]

```
bool ret;
HANDLE hPort;
char buf[10];
hPort = uart_Open("COM2,9600,N,8,1");
ret = uart_Recv(hPort, buf);
uart_Close(hPort);
```

[C#]

```
bool ret;
IntPtr hPort;
string buf;
hPort = WinPAC.uart_Open("COM2:,9600,N,8,1");
ret = WinPAC.uart_Recv(hPort, WinPAC.AnsiString(buf));
WinPAC.uart_Close(hPort);
```

Remarks

The terminate characters is0x0D. (Refer to uart_SetTerminator function to change.)

For example:

a. Check sum is disabled. This function receives five bytes (ABCD+0x0D). The “buf” will be five bytes (ABCD+0x0).

b. Check sum is disabled. This function receives four bytes (ABCD). The “buf” will be four bytes (ABCD). But the reurn value is 0.

8.10. uart_Send

This function sends data through the COM port which have been opened.

This function will send a string. If the checksum is enabled by the uart_EnableCheckSum function, this function automatically adds the two checksum bytes to the string. And then the end of sending string is further added [0x0D] to mean the termination of the string(buf).

Syntax

```
bool uart_Send(HANDLE hPort, LPCSTR buf);
```

Parameters

hPort

[in] Handle to the open COM port.

buf

[in] A pointer to a buffer that send the data, 2048 bytes maximum.

Return Values

1 indicates success. 0 indicates failure.

Examples

[eVC]

```
bool ret;
HANDLE hPort;
char buf[4];
sprintf(buf,"abcd");
hPort = uart_Open("COM2:,9600,N,8,1");
ret = uart_Send(hPort, buf);
uart_Close(hPort);
```

[C#]

```
bool ret;
IntPtr hPort;
string buf;
buf = "abcd";
hPort = WinPAC.uart_Open("COM2:,9600,N,8,1");
ret = WinPAC.uart_Send(hPort, WinPAC.AnsiString(buf));
WinPAC.uart_Close(hPort);
```

Remarks

A string for “buf” cannot include space character within the string. Otherwise, the string will be stoped by space character. For example: “$01M 02 03” of the user defined string. Howerver, the actual string sent out is “$01M”+0x0D.

The terminate characters is0x0D. (Refer to uart_SetTerminator function to change.)

This function will call PurgeComm() to clear serial COM port output buffer.

This function sends data with a terminate character0x0D. For example:

1. Check sum is disabled. The “buf” are five bytes (ABCD+0x0). This function will send five bytes(ABCD+0x0D).

8.11. uart_SendCmd

This function sends commands through the COM port which have been opened.

This function is a combination of uart_Send and uart_Recv.

The operation for sending a command is the same as uart_Send.

The operation for receiving a response is the same as uart_Recv.

Syntax

```
bool uart_SendCmd(
    HANDLE hPort,
    LPCSTR cmd,
    LPSTR szResult
);
```

Parameters

hPort

[in] Handle to the open COM.

cmd

[in] A pointer to a command.

szResult

[out] A pointer to a buffer that receives the data.

Return Values

1 indicates success. 0 indicates failure.

Examples

[eVC]

```
//Put an I-87K module in slot0 like I-87017W.
bool ret;
HANDLE hPort;
char buf[10];
hPort = uart_Open("COM0,115200,N,8,1");
pac_ChangeSlot(0);
ret = uart_SendCmd(hPort,"$00M", buf);      // $00M: ask the device name,DCON proctol for ICPDAS modules.
uart_Close(hPort);
```

[C#]

```
bool ret;
IntPtr hPort;
byte[] buf = new byte[100];
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
// $00M: ask the device name
WinPAC.pac_ChangeSlot(0);
ret = WinPAC.uart_SendCmd(hPort, WinPAC.AnsiString("$00M"), buf);
// $00M: ask the device name,DCON proctol for ICPDAS modules.
string result = WinPAC.WideString(buf).Replace("\0", "");
WinPAC.uart_Close(hPort);
```

Remarks

This function will call PurgeComm() to clear serial COM port output and input buffer.

8.12. uart_SetTerminator

This function sets the terminate characters.

Syntax

```
void uart_SetTerminator(
    HANDLE hPort,
    LPCSTR szTerm
);
```

Parameters

hPort

[in] Handle to the open COM port.

szTerm

[in] Pointer the terminate characters.

Default is CR (0x0D).

Return Values

None

Examples

[eVC]

```
HANDLE hPort;
char result[32];
hPort = uart_Open("");// Open COM0, data format: 115200,N,8,1
uart_SetTerminator(hPort, "\r");
pac_ChangeSlot(0); //A I-87K module is in slot0.
uart_SendCmd(hPort, "$00M", result); // $00M: ask the device name,DCON
uart_Close(hPort);
```

[C#]

```
byte[] result = new byte[32];
IntPtr hPort = WinPAC.uart_Open("");// Open COM0, data format: 115200,N,8,1
WinPAC.uart_SetTerminator(hPort, WinPAC.AnsiString("\r"));
WinPAC.pac_ChangeSlot(0); //A I-87K module is in slot0.
WinPAC.uart_SendCmd(hPort, WinPAC.AnsiString("$00M"), result); // $00M: ask the device name,DCON
string str = WinPAC.WideString(result);
WinPAC.uart_Close(hPort);
```

Remarks

This function relates to uart_Send/uart_Recv/uart_SendCmd.

8.13. uart_SetTimeOut

This function sets the time out timer.

Syntax

```
void uart_SetTimeOut(
    HANDLE hPort,
    DWORD msec,
    int ctoType
);
```

Parameters

hPort

[in] Handle to the open COM port.

msec

[in] millisecond to the timer.

ctoType

[in] Specifies the time out timer type as following:

CTO_TIMEOUT_ALL 0

CTO_READ_RETRY_TIMEOUT 1

CTO_READ_TOTAL_TIMEOUT 2

CTO_WRITE_TOTAL_TIMEOUT 3

Return Values

None

Examples

[eVC]

```
HANDLE hPort;
DWORD mes;
hPort = uart_Open("COM3,9600,N,8,1");
mes = 300;
uart_SetTimeOut(hPort, mes, CTO_TIMEOUT_ALL);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
uint mes;
hPort = WinPAC.uart_Open("COM3:,9600,N,8,1");
mes = 300;
WinPAC.uart_SetTimeOut(hPort, mes, 0);
WinPAC.uart_Close(hPort);
```

Remarks

CTO_READ_TOTAL_TIMEOUT:

A constant used to calculate the total time-out period for read operations, in milliseconds.

A value of zero for the CTO_READ_TOTAL_TIMEOUT indicates that total time-outs are not used for read operations.

CTO_WRITE_TOTAL_TIMEOUT:

A constant used to calculate the total time-out period for write operations, in milliseconds.

A value of zero for the CTO_WRITE_TOTAL_TIMEOUT indicates that total time-outs are not used for write operations.

CTO_READ_RETRY_TIMEOUT:

A constant used to calculate the time-out period for read operations, in

system tick count.

CTO_TIMEOUT_ALL:

A constant used to calculate the total time-out period for write and read operations, in milliseconds.

A value of zero for the CTO_TIMEOUT_ALL indicates that total time-outs are not used for write and read operations.

9. PAC_IO API

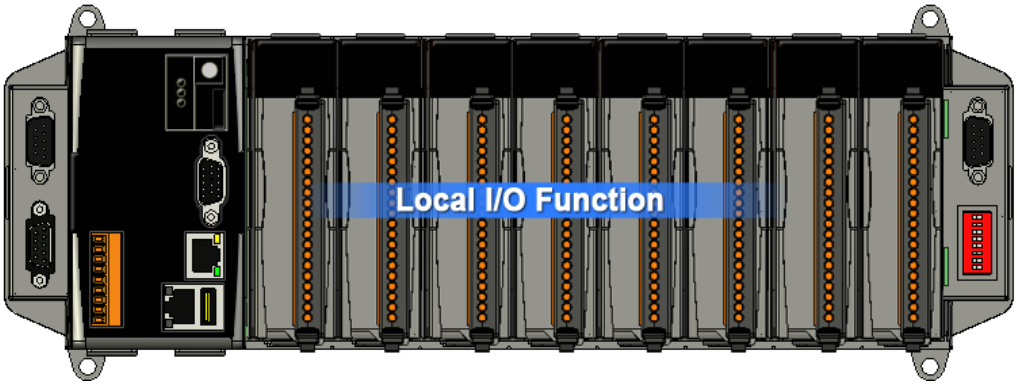
Additionally, some of those properties offer access to the environment around the device, such as ambient brightness or atmospheric pressure.

PAC_IO API supports to operate IO modules not only in slot but in Remote mode.

Besides, these API support for 8K or 87K modules.

Following, let's show how we could use these API for local and remote.

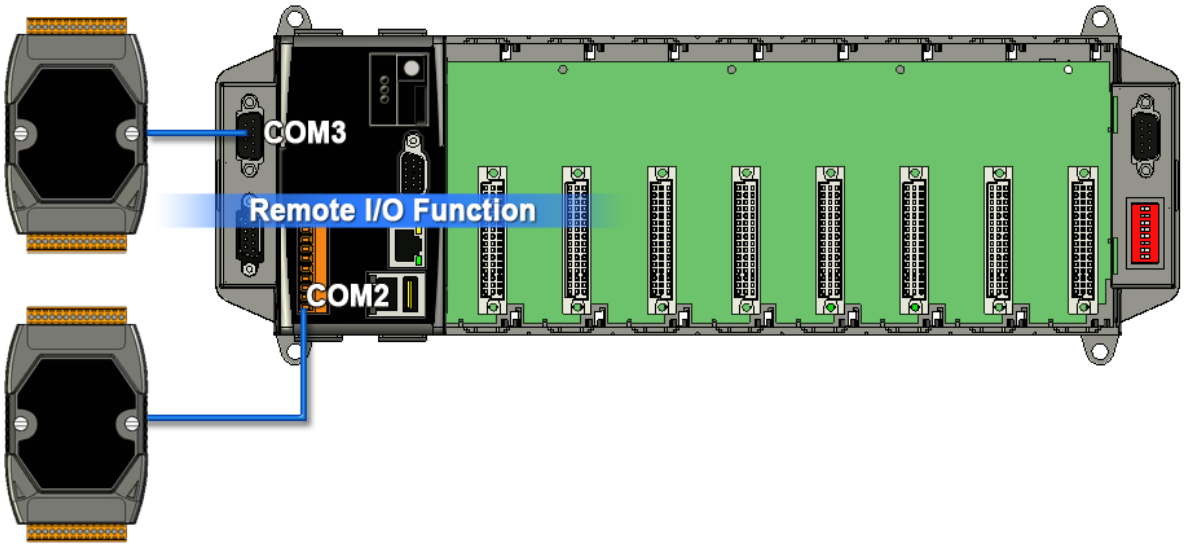
Local (IO in slot)



In the local mode, the slot range is from 0 to 7, the same the iSlot as follow.

```
hPort = WinPAC.uart_Open("");
//clear DO
WinPAC.pac_WriteDO(hPort, iSlot, iDo_Totalch, 0);
```

Remote



If remote mode, the address need call a macro, PAC_REMOTE_IO. And its range is from 0 to 255. For example as follow:

```
// Write DO value to the remote module
 Handle hPort = WinPAC.uart_Open(ConnectionString);
 if (!hPort) AfxMessageBox(_T("Open Com Error"));

 WinPAC.pac_WriteDO (hPort,WinPAC.PAC_REMOTE_IO(iAddr),m_iDOCHs,lDO_Value);
```

To see more information, please reference user manual - Chapter 5 API and Demo Reference

Supported Modules

The following shows the overview of the PAC_IO functions which are available with WinPAC.

Functions\Models	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_ClearCNT	Y	Y	Y	Y	Y	Y	Y
pac_ClearDICNT	Y	Y	Y	Y	Y	Y	Y
pac_ClearDILatch	Y	Y	Y	Y	Y	Y	Y
pac_ClearDIOLatch	Y	Y	Y	Y	Y	Y	Y
pac_GetBit	Y	Y	Y	Y	Y	Y	Y
pac_ReadAI	Y	Y	Y	Y	Y	Y	Y
pac_ReadAIAll	Y	Y	Y	Y	Y	Y	Y
pac_ReadAIAllHex	Y	Y	Y	Y	Y	Y	Y
pac_ReadAIHex	Y	Y	Y	Y	Y	Y	Y
pac_ReadAO	Y	Y	Y	Y	Y	Y	Y
pac_ReadCNT	Y	Y	Y	Y	Y	Y	Y
pac_ReadCNTOverflow	Y	Y	Y	Y	Y	Y	Y
pac_ReadDI	Y	Y	Y	Y	Y	Y	Y
pac_ReadDICNT	Y	Y	Y	Y	Y	Y	Y
pac_ReadDILatch	Y	Y	Y	Y	Y	Y	Y
pac_ReadDIO	Y	Y	Y	Y	Y	Y	Y
pac_ReadDIOLatch	Y	Y	Y	Y	Y	Y	Y
pac_ReadDO	Y	Y	Y	Y	Y	Y	Y
pac_WriteAO	Y	Y	Y	Y	Y	Y	Y
pac_WriteDO	Y	Y	Y	Y	Y	Y	Y
pac_WriteDOBit	Y	Y	Y	Y	Y	Y	Y

Function List

The following functions are used to retrieve or set PAC_IO information.

Function	Description
pac_ClearCNT	This function clears the counter values of the counter/frequency modules.
pac_ClearDICNT	This function clears the counter value of the DI channel of the DI module.
pac_ClearDILatch	This function clears the latch value of the DI module.
pac_ClearDIOLatch	This function clears the latch values of DI and DO channels of the DIO module.
pac_GetBit	The function can retrieve the value which in specific bit.
pac_ReadAI	This function reads the engineering-mode AI value of the AI module.
pac_ReadAIAll	This function reads all the AI values of all channels in engineering-mode of the AI module.
pac_ReadAIAllHex	This function reads all the AI values of all channels in 2's complement-mode of the AI module.
pac_ReadAIHex	This function reads the 2's complement-mode AI value of the AI module.
pac_ReadAO	This function reads the AO value of the AO module.
pac_ReadCNT	This function reads the counter values of the counter/frequency modules.
pac_ReadCNTOverflow	This function clears the counter overflow value of the counter/frequency modules.
pac_ReadDI	This function reads the DI value of the DI module.
pac_ReadDICNT	This function reads the counts of the DI channels of the DI module.
pac_ReadDILatch	This function reads the DI latch value of the DI module.

pac_ReadDIO	This function reads the DI and the DO values of the DIO module.
pac_ReadDIOLatch	This function reads the latch values of the DI and DO channels of the DIO module.
pac_ReadDO	This function reads the DO value of the DO module.
pac_WriteAO	This function writes the AO value to the AO modules.
pac_WriteDO	This function writes the DO values to DO modules.
pac_WriteDOBit	This function writes a single bit of value to the DO module, that is, only the channel corresponding to the bit is changed.

9.1. pac_ClearCNT

This function clears the counter values of the counter/frequency modules.

Syntax

```
bool pac_ClearCNT(
    HANDLE hPort,
    int slot,
    int iChannel
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] The channel that clears the counter value back from the counter/frequency module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=0;
bool ret = pac_ClearCNT(hPort, iSlot, iChannel);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel=0;
bool ret = WinPAC.pac_ClearCNT(hPort, iSlot, iChannel);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.2. pac_ClearDICNT

This function clears the counter value of the DI channel of the DI module.

Syntax

```
bool pac_ClearDICNT(
    HANDLE hPort,
    int slot,
    int iChannel,
    int iDI_TotalCh
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] The channel that the counter value belongs.

iDI_TotalCh

[in] Total number of the DI channels of the DI module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=2;
int iDI_TotalCh=8;
bool ret = pac_ClearDICNT(hPort, iSlot,iChannel,iDI_TotalCh);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel=2;
int iDI_TotalCh=8;
bool ret = WinPAC.pac_ClearDICNT(hPort, iSlot,iChannel,iDI_TotalCh);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.3. pac_ClearDILatch

This function clears the latch value of the DI module.

Syntax

```
bool pac_ClearDILatch(
    HANDLE hPort,
    int slot
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
bool ret = pac_ClearDILatch(hPort, iSlot);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
bool ret = pac_ClearDILatch(hPort, iSlot);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.4. pac_ClearDIOLatch

This function clears the latch values of DI and DO channels of the DIO module.

Syntax

```
bool pac_ClearDIOLatch(
    HANDLE hPort,
    int slot
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
bool ret = pac_ClearDIOLatch(hPort, iSlot);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
bool ret = WinPAC.pac_ClearDIOLatch(hPort, iSlot);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.5. pac_GetBit

The function can retrieve the value which in specific bit.

Syntax

```
pac_GetBit(
    v,
    ndx
);
```

Parameters

v

Which IO result wants to get bit.

ndx

Specific bit to retrieve.

Return Values

The value of specific index.

Examples

[C#]

```
bool bit;
bit = WinPAC.pac_GetBit(0x5A, 0);
```

Remarks

The function is used the same as v & (1<<index).

If you use WinPacNet.dll to develop, the return value is on or off for the specifics bit.

9.6. pac_ReadAI

This function reads the engineering-mode AI value of the AI module.

Syntax

```
bool pac_ReadAI(
    HANDLE hPort,
    int slot,
    int iChannel,
    int iAI_TotalCh,
    float *fValue
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] Read the AI value from the channel.

iAI_TotalCh

[in] The total number of the AI channels of the AI module.

fValue

[in] The pointer to the AI value that is read back from the AI module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=2;
int iAI_TotalCh=8;
float fValue;
bool ret = pac_ReadAI(hPort, iSlot,iChannel,iAI_TotalCh, &fValue);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel=2;
int iAI_TotalCh=8;
float fValue = new float();
bool ret = WinPAC.pac_ReadAI(hPort, iSlot,iChannel,iAI_TotalCh, ref fValue);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.7. pac_ReadAIAll

This function reads all the AI values of all channels in engineering-mode of the AI module.

Syntax

```
bool pac_ReadAIAll(
    HANDLE hPort,
    int slot,
    float fValue[]
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

fValue[]

[out] The array contains the AI values that read back from the AI module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
float fValue[8];
bool ret = pac_ReadAIAll(hPort, iSlot, fValue);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=1;
float[] fValue = new float[8];
bool ret =WinPAC.pac_ReadAIAll(hPort, iSlot, fValue);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.8. pac_ReadAIAllHex

This function reads all the AI values of all channels in 2's complement-mode of the AI module.

Syntax

```
bool pac_ReadAIAllHex(
    HANDLE hPort,
    int slot,
    int iValue[]
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iValue[]

[out] The array contains the AI values that read back from the AI module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iValue[8];
bool ret = pac_ReadAIAllHex(hPort, iSlot, iValue);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int[] iValue = new int[8];
bool ret = WinPAC.pac_ReadAIAllHex(hPort, iSlot, iValue);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.9. pac_ReadAIHex

This function reads the 2's complement-mode AI value of the AI module.

Syntax

```
bool pac_ReadAIHex(
    HANDLE hPort,
    int slot,
    int iChannel,
    int iAI_TotalCh,
    int *iValue
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] Read the AI value from the channel.

iAI_TotalCh

[in] The total number of the AI channels of the AI module.

iValue

[in] The pointer to the AI value that is read back from the AI module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=2;
int iAI_TotalCh=8;
int iValue;
bool ret = pac_ReadAIHex(hPort, iSlot,iChannel,iAI_TotalCh, &iValue);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel=2;
int iAI_TotalCh=8;
int iValue = new int();
bool ret = WinPAC.pac_ReadAIHex(hPort, iSlot,iChannel,iAI_TotalCh, ref iValue);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.10. pac_ReadAO

This function reads the AO value of the AO module.

Syntax

```
bool pac_ReadAO(
    HANDLE hPort,
    int slot,
    int iChannel,
    int iAO_TotalCh,
    float *fValue
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] Read the AO value from the channel.

iAO_TotalCh

[in] The total number of the AO channels of the AO module.

float fValue

[in] The pointer to the AO value that is read back from the AO module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=2;
int iAO_TotalCh=4;
float fValue;
bool ret = pac_ReadAO(hPort, iSlot,iChannel,iAO_TotalCh, &fValue);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel=2;
int iAO_TotalCh=4;
float fValue = new float();
bool ret = WinPAC.pac_ReadAO(hPort, iSlot,iChannel,iAO_TotalCh,ref fValue);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.11. pac_ReadCNT

This function reads the counter values of the counter/frequency modules.

Syntax

```
bool pac_ReadCNT(
    HANDLE hPort,
    int slot,
    int iChannel,
    DWORD *lCounter_Value
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] The channel that reads the counter value back from the counter/frequency module.

lCounter_Value

[out] The pointer to the counter value that reads back from the counter/frequency module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=0;
DWORD lCounter_Value;
bool ret = pac_ReadCNT(hPort, iSlot,iChannel,&lCounter_Value);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel=0;
uint lCounter_Value = new uint();
bool ret = WinPAC.pac_ReadCNT(hPort, iSlot,iChannel,ref lCounter_Value);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.12. pac_ReadCNTOverflow

This function clears the counter overflow value of the counter/frequency modules.

Syntax

```
bool pac_ReadCNTOverflow(
    HANDLE hPort,
    int slot,
    int iChannel,
    int *iOverflow
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] The channel that reads the counter overflows value back from the counter/frequency module.

iOverflow

[out] The pointer to the counter overflow that is read back from the counter/frequency module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=0;
int iOverflow;
bool ret = pac_ReadCNT_Overflow(hPort, iSlot,iChannel,&iOverflow);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot = 0;
int iChannel = 0;
int iOverflow = new int();
bool ret = WinPAC.pac_ReadCNTOverflow(hPort, iSlot, iChannel, ref iOverflow);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.13. pac_ReadDI

This function reads the DI value of the DI module.

Syntax

```
bool pac_ReadDI(
    HANDLE hPort,
    int slot,
    int iDI_TotalCh,
    DWORD *lDI_Value
);
```

Parameters

hPort

[in] The serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iDI_TotalCh

[in] The total channels of the DI module.

lDI_Value

[out] The pointer to DI value to read back.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot = 0;
int iDI_TotalCh = 8;
DWORD lDI_Value;
bool ret = pac_ReadDI(hPort, iSlot,iDI_TotalCh, &lDI_Value);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot = 0;
int iDI_TotalCh = 8;
uint lDI_Value = new uint();
bool ret = WinPAC.pac_ReadDI(hPort, iSlot,iDI_TotalCh, ref lDI_Value);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.14. pac_ReadDICNT

This function reads the counts of the DI channels of the DI module.

Syntax

```
bool pac_ReadDICNT(
    HANDLE hPort,
    int slot,
    int iChannel,
    int iDI_TotalCh,
    DWORD *lCounter_Value
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] The channel that the counter value belongs.

iDI_TotalCh

[in] Total number of the DI channels of the DI module.

lCounter_Value

[out] The pointer to the counter value.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel =2;
int iDI_TotalCh=8;
DWORD lCounter_Value;
bool ret = pac_ReadDICNT(hPort, iSlot,iChannel,iDI_TotalCh, &lCounter_Value);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel =2;
int iDI_TotalCh=8;
uint lCounter_Value = new uint();
bool ret = WinPAC.pac_ReadDICNT(hPort, iSlot,iChannel,iDI_TotalCh, ref
lCounter_Value);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.15. pac_ReadDILatch

This function reads the DI latch value of the DI module.

Syntax

```
bool pac_ReadDILatch(
    HANDLE hPort,
    int slot,
    int iDI_TotalCh,
    int iLatchType,
    DWORD *lDI_Latch_Value
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iDI_TotalCh

[in] The total number of the DI channels of the DI module.

iLatchType

[in] The latch type specified to read latch value back.

1 = latched high status

0 = latched low status

lDI_Latch_Value

[out] The pointer to the latch value read back from the DI module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iDI_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;
bool ret = pac_ReadDILatch(hPort, iSlot, iDI_TotalCh, iLatchType, &lDI_Latch_Value);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iDI_TotalCh=8;
int iLatchType=0;
uint lDI_Latch_Value = new uint();
bool ret = WinPAC.pac_ReadDILatch(hPort, iSlot, iDI_TotalCh, iLatchType, ref
lDI_Latch_Value);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.16. pac_ReadDIO

This function reads the DI and the DO values of the DIO module.

Syntax

```
bool pac_ReadDIO(
    HANDLE hPort, int slot,
    int iDI_TotalCh,
    int iDO_TotalCh,
    DWORD* lDI_Value,
    DWORD* lDO_Value
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iDI_TotalCh

[in] The total number of DI channels of the DIO module.

iDO_TotalCh

[in] The total number of DO channels of the DIO module.

lDI_Value

[out] The pointer to the value of DI read back.

lDO_Value

[out] The pointers to the value of DO read back.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
DWORD lDI_Value;
DWORD lDO_Value;
bool ret = pac_ReadDIO(hPort, iSlot,iDI_TotalCh, iDO_TotalCh, &lDI_Value,
&lDO_Value);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
uint lDI_Value = new uint();
uint lDO_Value = new uint();
bool ret = WinPAC.pac_ReadDIO(hPort, iSlot,iDI_TotalCh, iDO_TotalCh, ref lDI_Value,
ref lDO_Value);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for local or remote. When the module is local, the second parameters's range is from 0 to 7. If remote, the second parameters need use the macro, PAC_REMOTE_IO (Address), which address if from 0 to 255.

9.17. pac_ReadDIOLatch

This function reads the latch values of the DI and DO channels of the DIO module.

Syntax

```
bool pac_ReadDIOLatch(
    HANDLE hPort,
    int slot,
    int iDI_TotalCh,
    int iDO_TotalCh,
    int iLatchType,
    DWORD *lDI_Latch_Value,
    DWORD *lDO_Latch_Value
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iDI_TotalCh

[in] The total number of the DI channels of the DIO module.

iDO_TotalCh

[in] The total number of the DO channels of the DIO module.

iLatchType

[in] The type of the latch value read back.

lDI_Latch_Value

[out] The pointer to the DI latch value read back.

1 = latched high status

0 = latched low status

lDO_Latch_Value

[out] The pointer to the DO latch value read back.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

HANDLE hPort;

```
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;
DWORD lDO_Latch_Value;
bool ret = pac_ReadDIOLatch(hPort, iSlot,iDI_TotalCh,iDO_TotalCh,iLatchType,
&lDI_Latch_Value,&lDO_Latch_Value);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot = 0;
int iDI_TotalCh = 8;
int iDO_TotalCh = 8;
int iLatchType = 0;
uint lDI_Latch_Value = new uint();
uint lDO_Latch_Value = new uint();
bool ret = WinPAC.pac_ReadDIOLatch(hPort, iSlot, iDI_TotalCh, iDO_TotalCh,
iLatchType, ref lDI_Latch_Value, ref lDO_Latch_Value);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.18. pac_ReadDO

This function reads the DO value of the DO module.

Syntax

```
bool pac_ReadDO(
    HANDLE hPort,
    int slot,
    int iDO_TotalCh,
    DWORD *lDO_Value
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iDO_TotalCh

[in] The total number of DO channels of the DO modules.

lDO_Value

[in] The pointer of the DO value to read from the DO module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE slot = 0;
int total_channel = 8;
DWORD do_value;
bool ret = pac_ReadDO(hPort, slot , total_channel , &do_value );
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte slot = 0;
int total_channel = 8;
uint do_value = new uint();
bool ret = WinPAC.pac_ReadDO(hPort, slot , total_channel , ref do_value );
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.19. pac_WriteAO

This function writes the AO value to the AO modules.

Syntax

```
bool pac_WriteAO(
    HANDLE hPort,
    int slot,
    int iChannel,
    int iAO_TotalCh,
    float fValue
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] The channel that is written thee AO value to.

iAO_TotalCh

[in] The total number of the AO channels of the AO module.

float fValue

[in] The AO value to write to the AO module.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot=0;
int iChannel=2;
int iAO_TotalCh=4;
float fValue=5;
bool ret = pac_WriteAO(hPort, iSlot,iChannel,iAO_TotalCh,fValue);
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot=0;
int iChannel=2;
int iAO_TotalCh=4;
float fValue=5;
bool ret = WinPAC.pac_WriteAO(hPort, iSlot,iChannel,iAO_TotalCh,fValue);
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.20. pac_WriteDO

This function writes the DO values to DO modules.

Syntax

```
bool pac_WriteDO(
    HANDLE hPort, int slot,
    int iDO_TotalCh,
    DWORD lDO_Value
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO (Address).

iDO_TotalCh

[in] The total number of DO channels of the DO modules.

lDO_Value

[in] 1 is to turn on the DO channel; 0 is off.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
// If the module is remote
HANDLE hPort;
hPort = uart_Open("COM2:,9600,N,8,1");
int total_channel = 8;
DWORD do_value = 3;      // turn on the channel 0,1
bool ret = pac_WriteDO(hPort, pac_REMOTE_IO(1) , total_channel , do_value );
uart_Close(hPort);
```

[C#]

```
// If the module is remote
IntPtr hPort;
hPort = WinPAC.uart_Open("COM2:,9600,N,8,1");
int total_channel = 8;
uint do_value = 3;      // turn on the channel 0,1
bool ret = WinPAC.pac_WriteDO(hPort, WinPAC.pac_REMOTE_IO(1) , total_channel , do_value );
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

9.21. pac_WriteDOBit

This function writes a single bit of value to the DO module, that is, only the channel corresponding to the bit is changed.

Syntax

```
bool pac_WriteDOBit(
    HANDLE hPort, int slot,
    int iDO_TotalCh,
    int iChannel,
    int iBitValue
);
```

Parameters

hPort

[in] the serial port HANDLE opened by uart_Open().

iSlot

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

If the IO module is remote, please use the macro, pac_REMOTE_IO(0...255).

iChannel

[in] The DO channel to change.

iDO_TotalCh

[in] The total number of DO channels of the DO modules.

iBitValue

[in] 1 is to turn on the DO channel; 0 is off.

Return Values

Return true if success, otherwise false.

Examples

[eVC]

```
HANDLE hPort;
hPort = uart_Open("COM0:,115200,N,8,1");
BYTE iSlot = 0;
int iChannel = 2;
int iDO_TotalCh = 8;
int iBitValue = 1;
bool ret = pac_WriteDOBit(hPort, iSlot , iChannel , iDO_TotalCh , iBitValue );
uart_Close(hPort);
```

[C#]

```
IntPtr hPort;
hPort = WinPAC.uart_Open("COM0:,115200,N,8,1");
byte iSlot = 0;
int iChannel = 2;
int iDO_TotalCh = 8;
int iBitValue = 1;
bool ret = WinPAC.pac_WriteDOBit(hPort, iSlot , iChannel , miDO_TotalCh ,
iBitValue );
WinPAC.uart_Close(hPort);
```

Remarks

The function can support for Local or Remote. When the module is local, the second Parameters's range is from 0 to 7. If remote, the second Parameters need use the macro, pac_REMOTE_IO (Address), which Address is from 0 to 255.

10. Backplane timer API

Backplane timer API supports to hardware timer including timerout/timer1/timer2.

Supported Modules

The following shows the overview of the backplane timer functions which are available with WinPAC.

Functions / Modules	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_GetBPTimerTimeTick_ms	Y	Y	Y	-	Y	Y	-
pac_GetBPTimerTimeTick_us	Y	Y	Y	-	Y	Y	-
pac_KillBPTimer	Y	Y	Y	-	Y	Y	-
pac_SetBPTimer	Y	Y	Y	-	Y	Y	-
pac_SetBPTimerOut	Y	Y	Y	-	Y	Y	-

Function List

The following functions are used to retrieve or set backplane timer functions.

Function	Description
pac_GetBPTimerTimeTick_ms	This function returns the number of milliseconds since the device booted, excluding any time that the system was suspended.
pac_GetBPTimerTimeTick_us	This function returns the number of microsecond since the device booted, excluding any time that the system was suspended.
pac_KillBPTimer	This function kills the timer event identified by type, set by an earlier call to pac_SetBPTimer.
pac_SetBPTimer	This function installs a hardware timer. A time-out value is specified, and every time a time-out occurs, the system posts an interrupt signal to the system and the pass the message to an application-defined callback function.
pac_SetBPTimerOut	The timerout pin on each slot will be triggered while a timerout signal has been outputted. The timeourput pin can be used to acquire the synchronized data on each slot.

10.1. pac_GetBPTimerTimeTick_ms

This function returns the number of milliseconds since the device booted, excluding any time that the system was suspended.

Syntax

```
DWORD pac_GetBPTimerTimeTick_ms(void);
```

Parameters

None

Return Values

The number of milliseconds indicates success.

Examples

None

10.2. pac_GetBPTimerTimeTick_us

This function returns the number of microsecond since the device booted, excluding any time that the system was suspended.

Syntax

```
DWORD pac_GetBPTimerTimeTick_us (void);
```

Parameters

None

Return Values

The number of microseconds indicates success.

Examples

None

10.3. pac_KillBPTimer

This function kills the timer event identified by type, set by an earlier call to pac_SetBPTimer.

Syntax

```
void pac_KillBPTimer(int type);
```

Parameters

type

[in] Two type timer, 1 microsecond timer and 10 microsecond timer

Return Values

None

Examples

None

10.4. pac_SetBPTimer

This function installs a hardware timer. A time-out value is specified, and every time a time-out occurs, the system posts an interrupt signal to the system and the pass the message to an application-defined callback function.

Syntax

```
Bool pac_SetBPTimer(int type,unsigned int uElapse,
pac_TIMEROUT_CALLBACK_FUNC f);
```

Parameters

type

[in] Two type timer, 1 microsecond timer and 10 microseconds timer

uElapse

[in]

Timer 1: Specifies the elapses time value for a timerout signal 0~65535, in microseconds.

Timer 2: Specifies the elapses time value for a timerout signal 0~65535, in 10 microseconds.

f

Specifies the address of the application-supplied f callback function.

Return Values

The value is true if the function was installed successfully. It is zero and the function cannot be installed.

Examples

None

10.5. pac_SetBPTimerOut

The timerout pin on each slot will be triggered while a timerout signal has been outputted. The timeourput pin can be used to acquire the synchronized data on each slot.

Syntax

```
bool pac_SetBPTimerOut(unsigned int uHighElapse,unsigned int uLOwElapse,
pac_TIMEROUT_CALLBACK_FUNC f);
```

Parameters

uHighElapse

[in] Specifies the elapses time value for a high wave of the timerout signal 0~65535, in microseconds.

uLOwElapse

[in] Specifies the elapses time value for a low wave of the timerout signal 0~65535, in microseconds.

f

Specifies the address of the application-supplied f callback function.

Return Values

The value is true if the function was installed successfully. It is zero and the function cannot be installed.

Examples

None

11. Error Handling API

Error handling operations include basic management operations, such as setting and the writing to the registry. The following topics describe how you can create, delete, or modify registry keys programmatically using the registry functions.

Supported Modules

The following shows the overview of the error handling functions which are available with WinPAC.

Functions / Modules	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
pac_GetErrorMessage	Y	Y	Y	-	Y	Y	-
pac_GetLastError	Y	Y	Y	-	Y	Y	-
pac_SetLastError	Y	Y	Y	-	Y	Y	-

Function List

The following functions are used to retrieve or set error handling functions.

Function	Description
pac_GetErrorMessage	This function retrieves a message string.
pac_GetLastError	This function returns the last-error code value.
pac_SetLastError	This function sets the last-error code.

Remarks:

To see more information, please reference user manual - Chapter 5 API and Demo Reference

11.1. pac_GetErrorMessage

This function retrieves a message string.

Syntax

```
void pac_GetErrorMessage(
   DWORD dwMessageID,
   LPTSTR lpBuffer
);
```

Parameters

dwMessageID

[in] Specifies the 32-bit message identifier for the requested message.

lpBuffer

[out] Pointer to a buffer for the error message.

Return Values

None

Examples

None

Examples

None

Remarks

The pac_GetErrorMessage function can be used to obtain error message strings for the WinPac error codes returned by pac_GetLastError, as shown in the following sample code.

```
TCHAR Buffer[32];

pac_GetErrorMessage(pac_GetLastError(), Buffer);

MessageBox( NULL, Buffer, L"Error", MB_OK | MB_ICONINFORMATION );
```

11.2. pac_GetLastError

This function returns the last-error code value.

Syntax

```
DWORD pac_GetLastError();
```

Parameters

None

Return Values

The Return Value section of each reference page notes the conditions under which the function sets the last-error code.

Examples

None

Remarks

You should call the pac_GetLastError function immediately when a function's return value indicates that such a call will return useful data. That is because some functions call pac_SetLastError(0) when they succeed, wiping out the error code set by the most recently failed function.

To obtain an error string for WinPac error codes, use the pac_GetErrorMessage function. For a complete list of error codes, see Error Values or the SDK header file PACERROR.H.

11.3. pac_SetLastError

This function sets the last-error code.

Syntax

```
void pac_SetLastError
   DWORD errno
);
```

Parameters

errno

[in] Specifies the last-error code.

Return Values

None

Examples

None

Remarks

Applications can retrieve the value saved by this function by using the pac_SetLastError function. The use of pac_SetLastError is optional; an application can call it to find out the specific reason for a function failure.

12. MISC API

Supported Modules

The following shows the overview of the misc functions which are available with WinPAC.

Functions / Modules	WP-8x4x	WP-8x3x	WP-8x5x	WP-5xxx	VP-25Wx	VP-23Wx	VH-25Wx
AnsiString							
pac_AnsiToWideString	Y	Y	Y	Y	Y	Y	Y
pac_DoEvents	Y	Y	Y	Y	Y	Y	Y
pac_WideStringToAnsi	Y	Y	Y	Y	Y	Y	Y
WideString							

Function List

The following functions are used to retrieve or set misc functions.

Function	Description
AnsiString	This function can convert unicode string to ansi byte array.
pac_AnsiToWideString	This function can convert ansi string to unicode string.
pac_DoEvents	This function marshals the case to handle the other events.
pac_WideStringToAnsi	This function can convert ansi byte array to unicode string.
WideString	This function can convert unicode string to ansi string.

12.1. AnsiString

This function can convert unicode string to ansi byte array.

Syntax

```
byte[] AnsiString(
    [in] string str
);
```

Parameters

str

[in] A pointer to a buffer that stores unicode string.

Return Values

Return ansi byte array.

Examples

[C#]

```
byte[] result = new byte[32];
IntPtr hPort = WinPAC.uart_Open("COM1:");
WinPAC.pac_ChangeSlot(Convert.ToByte(1));
WinPAC.uart_SendCmd(hPort, WinPAC.AnsiString("$00M"), result);
string str = WinPAC.WideString(result);
```

Remarks

In .NET, if we want to convert a Unicode string to ANSI or vice versa, we should convert through byte array.

12.2. pac_AnsiToWideString

This function can convert ansi string to unicode string.

Syntax

```
void pac_AnsiToWideString(
    const char *astr,
    LPTSTR wstr
);
```

Parameters

astr

[in] A pointer to a buffer that stores ansi string.

wstr

[in] A pointer to a buffer that receives unicode string.

Return Values

None

Examples

[eVC]

```
char ansiString[128] = "This is an ansi string";
TCHAR uniString[128];
pac_AnsiToWideString(ansiString, uniString);
MessageBox(NULL, uniString, NULL, MB_OK);      // The string "This is an ansi string" will show in the messagebox correctly
```

Remarks

Maximum string buffer size is 2 Kbytes.

12.3. pac_DoEvents

This function marshals the case to handle the other events.

When you run a Windows Form, it creates the new form, which then waits for events to handle. Each time the form handles an event, it processes all the code associated with that event. All other events wait in the queue. While your code handles the event, your application does not respond. If you call pac_DoEvents in your code, your application can handle the other events.

Syntax

```
void pac_DoEvents();
```

Parameters

None

Return Values

None

Examples

[eVC]

```
int counter = 0;
TCHAR buf[10];
bFlag = true;
while(bFlag)
{
pac_DoEvents();
swprintf(buf, TEXT("%d"), counter);
SetDlgItemText(IDC_EDIT1, buf);
counter++;
}
```

12.4. pac_WideStringToAnsi

This function can convert unicode string to ansi string.

Syntax

```
void pac_WideStringToAnsi(
    const TCHAR *wstr,
    LPSTR astr
);
```

Parameters

wstr

[in] A pointer to a buffer that stores unicode string.

astr

[in] A pointer to a buffer that receives ansi string.

Return Values

None

Examples

[eVC]

```
TCHAR uniString[128] = TEXT("This is a unicode string");
char ansiString[128];
pac_WideStringToAnsi(uniString, ansiString);
printf("%s", ansiString);
// The string "This is a unicode string" will show the console mode correctly
```

Remarks

Maximum string buffer size is 2 Kbytes.

12.5. WideString

This function can convert ansi byte array to unicode string.

Syntax

```
string WideString(
    byte[] CharStr
);
```

Parameters

CharStr

[in] A pointer to a buffer that stores ansi byte array.

Return Values

Return unicode string.

Examples

[C#]

```
byte[] result = new byte[32];
IntPtr hPort = WinPAC.uart_Open("COM1:");
WinPAC.pac_ChangeSlot(Convert.ToByte(1));
WinPAC.uart_SendCmd(hPort, WinPAC.AnsiString("$00M"), result);
string str = WinPAC.WideString(result);
```

Remarks

In .NET, if we want to convert a Unicode string to ANSI or vice versa, we should convert through byte array.

Appendix A. System Error Codes

This following table provides a list of system error code. There are turned by the pac_GetLastError function when many functions fail. To retrieve the description text for the error in your application, use the pac_GetErrorMessage function.

Error Code	Error Message
0x00001	Unknow Error
0x10001	Slot registered error
0x10002	Slot not registered error
0x10003	Unknown Module
0x10004	Module doesn't exist
0x10005	Invalid COM port number
0x10006	Function not supported
0x11001	EEPROM accesses invalid address
0x11002	SRAM accesses invalid address
0x11003	SRAM accesses invalid type
0x11004	NVRAM accesses invalid address
0x11005	EEPROM write protection
0x11006	EEPROM write fail
0x11007	EEPROM read fail
0x12001	The input value is invalid
0x12002	The wdt doesn't exist
0x12003	The wdt init error
0x13001	Create interrupt's event failure
0x14001	Uart check sum error

0x14002	Uart read timeout
0x14003	Uart response error
0x14004	Uart under input range
0x14005	Uart exceed input range
0x14006	Uart open filed
0x14007	Uart get Comm Modem status error
0x14008	Uart get wrong line status
0x15001	IO card does not support this API function
0x15002	API unsupport this IO card
0x15003	Slot's value exceeds its range
0x15004	Channel's value exceeds its range
0x15005	Gain's value exceeds its range
0x15006	Unsupported interrupt mode
0x15007	I/O value is out of the range
0x15008	I/O channel is out of the range