# XPacSDK/XPacSDK_CE
# Standard API Manual

**Version 2.0.5, June 2011**

**Service and usage information for**

**XP-8041/XP-8041-CE**

**XP-8341/XP-8341-CE**

**XP-8741/XP-8741-CE**

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

# Preface

This manual is targeted to application developers who are developing XPAC applications with the SDK of XPAC series controller.

# XPacSDK for XPAC with Windows Embedded Standard and Windows CE

This reference supports SDK of XPAC serial controller, which has two kinds of OS. One is Windows Embedded Standard and the other is Windows Compact Edition.

※ Warning: All C# sample codes for XP-8000 series only.

## XPacSDK for XP-8000 series with Windows Embedded Standard (WES)

The OS of XP-8000 series is Windows Embedded Standard. It's a XP like OS.

Support more programming language: C/C++/MFC, C#, VB, VB.NET, Delphi, and BCB.

**Supported programming languages include:**

- For C/C++/MFC: XPacSDK.h/XPacSDK.lib/XPacSDK.dll
- For C#/VB.NET: XPacNET.dll
- For VB: XPacSDK.bas
- For Delphi: XPacSDK.pas
- For BCB: XPacSDK_BCB.lib

Each program no matter builds by which language, its execution needs XPacSDK.dll to excute.

The XPacSDK is applied for XP-8000 and XP-8000-Atom must be version 2.0.2.0 and later (The version earlier than V2.0.2.0 is only applied for XP-8000).

V: Work, X: Don't Work

| API | XP-8000 | PC Windows |
|---|---|---|
| System Information | V | X |
| Backplane Access | V | X |
| Interrupt | V | X |
| Memory Access | V | X |
| Uart | V | V |
| PAC_IO | V | V |

**Note 1:** For the PC windows solution, the Microsoft .NET Framework 2.0 (or above) must be installed on PC.

**Note 2:** The PAC_IO API can be used to connect the I-7000 module and other modules using the DCON protocol.

## XPacSDK_CE for XP-8000-CE series with Windows Compact Edition (WinCE)

The OS of XPAC_CE series is Windows Compact Edition. Different from WES OS of XP-8000 series, XPAC_CE only support Visual Studio 2005/2008, and the name of SDK is difference, either.

**Supported programming languages include:**

- For C/C++/MFC: XPacSDK_CE.h/XPacSDK_CE.lib/XPacSDK_CE.dll
- XPacSDK_CE.msi: This is used to install in a desktop, and then the Visual Studio can build programs which can run on the Windows CE.

The version of XPACSDK_CE library applied for both of XP-8000-CE6 and XP-8000-Atom-CE6 must be V2.0.0.5 and later (The version earlier than V2.0.0.5 is only applied for XP-8000-CE6).

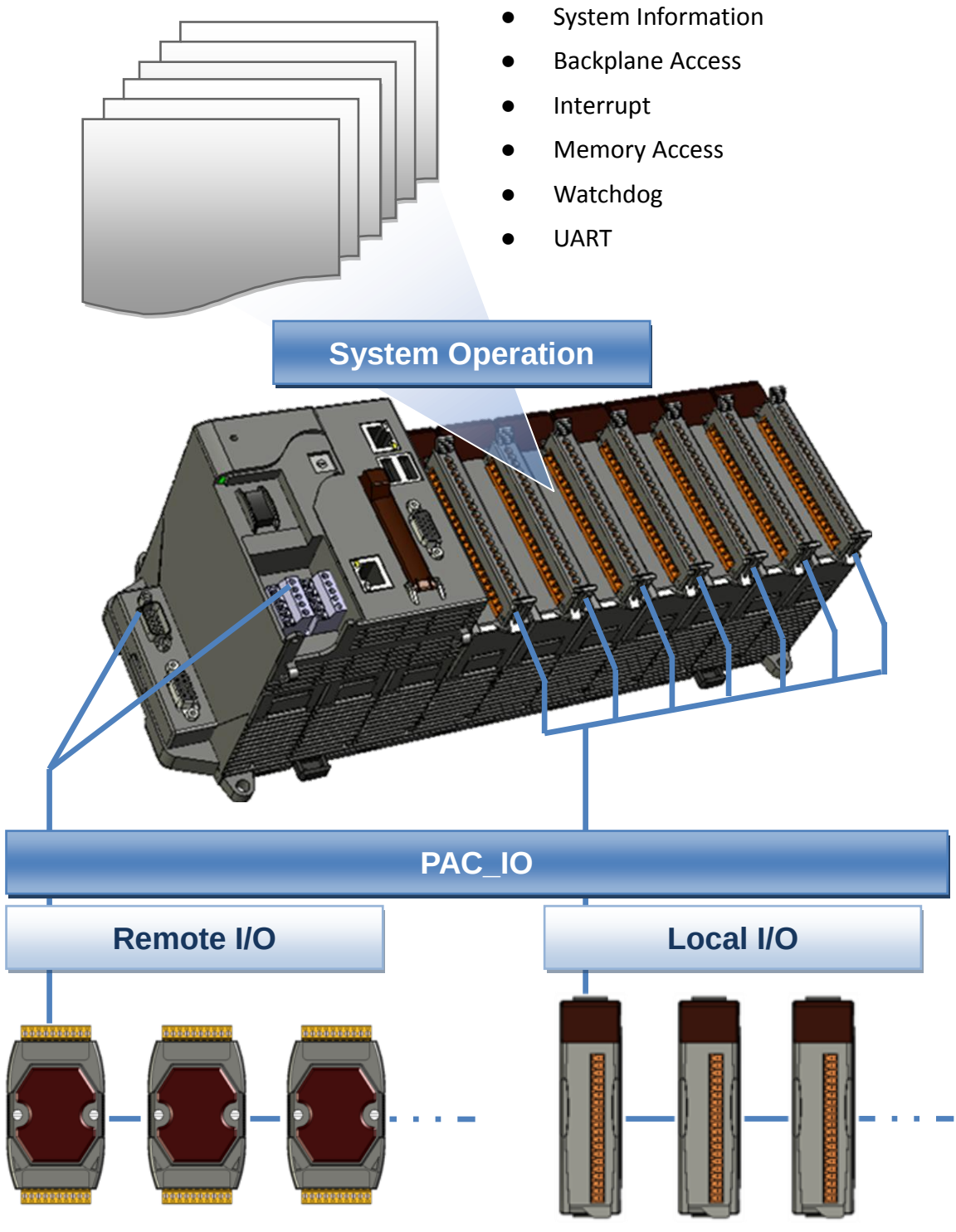
The available API functions used for XP-8000-CE6 and XP-8000-Atom-CE6

V: Work, X: Don't Work

| API Functions | XP-8000-CE6 | XP-8000-Atom_CE6 |
|---|---|---|
| Pac_EnableLED | X | V |
| Others | V | V |

# XPacSDK Overview

This API reference section contains descriptions of system operation and IO programming elements.



- **System Information Reference**

Provide reference information for the system status.

- **Backplane Access Reference**

Provide reference information for the backplane access APIs.

- **Interrupt Reference**

Provide reference information for the Interrupt APIs

- **Memory Access Reference**

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

- **Watchdog Reference**

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

- **Uart Reference**

Provide reference information for the Uart APIs.

- **PAC_IO Reference**

Provide reference information for IO APIs, including local and remote.

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

- **Error Handling API Reference**

Provide reference information for error handling.

# XPacSDK Plug-in for Development Tools for XP-8000 Series

This SDK for XP-8000 series supports several languages including C/C++/MFC/.NET/VB/Delphi. For different languages, they have to be called by different methods. Therefore, in this section, we provide a brief turtorial on how to render XPacSDK from these programming languages as you need.

# C/C++/MFC (for XP-8000 Series only)

## Required Libraries and Header files

To develop your C/C++/MFC program, you must use the appropriate libraries and header files for targeting your program.

1. XPacSDK.h
2. XPacSDK.lib
3. XPacSDK.dll (your execution file should be put in the same directory as the XPacSDK.dll)

## How to

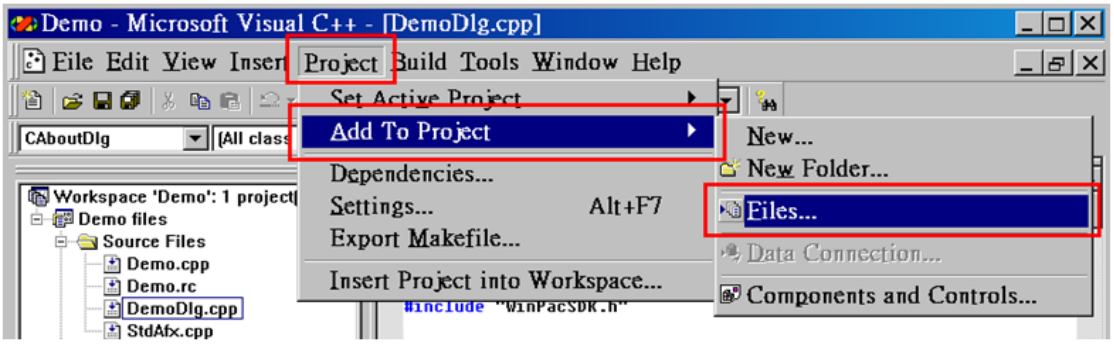
### Step 1: Include XPacSDK

```
// DemoDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Demo.h"
#include "DemoDlg.h"

#include "XPacSDK.h"
```

### Step 2: Add XPacSDK.lib into your project



# C# (for XP-8000 Series only)

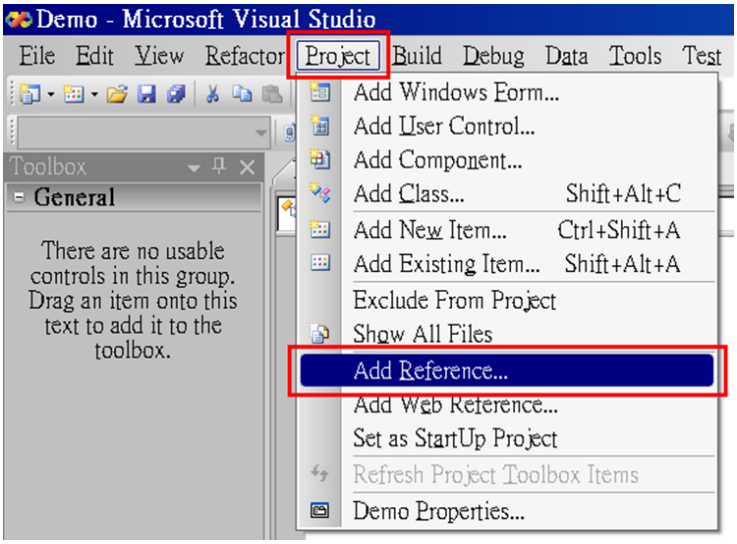
## Required Libraries and Header files

To develop your C# program, you must use the appropriate libraries and header files for targeting your program.

1. XPacNet.dll
2. XPacSDK.dll (your execution file should be put in the same directory as the XPacNet.dll and XPacSDK.dll)

## How to

### Step 1: Add reference, XPacNET.dll, into your project




### Step 2: Using XPacNET

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using XPacNET;
```

# VB.NET (for XP-8000 Series only)

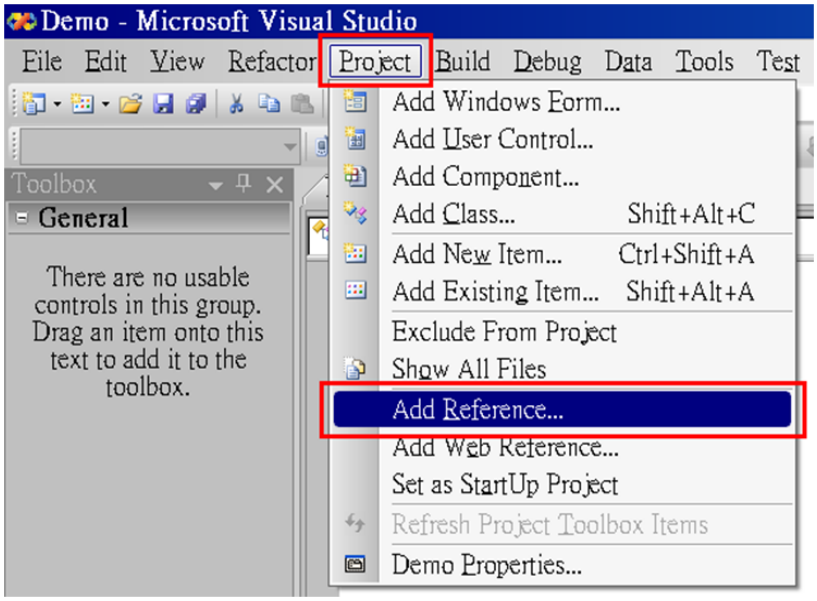
## Required Libraries and Header files

To develop your VB.NET program, you must use the appropriate libraries and header files for targeting your program.

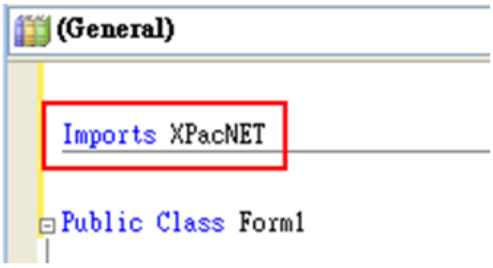
1. XPacNet.dll
2. XPacSDK.dll (your execution file should be put in the same directory as the XPacNet.dll and XPacSDK.dll)

## How to

### Step 1: Add reference, XPacNET.dll, into your project




### Step 2: Imports XPacNET



# VB (for XP-8000 Series only)

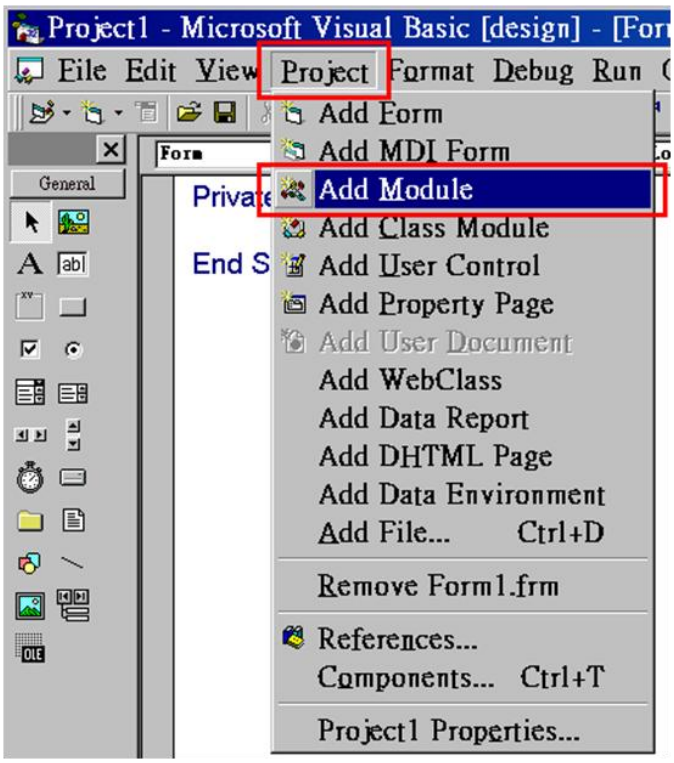
## Required Libraries and Header files

To develop your VB program, you must use the appropriate libraries and header files for targeting your program.

1. XPacSDK.bas
2. XPacSDK.dll (your execution file should be put in the same directory as the XPacSDK.dll)

## How to

### Step 1: Add module, XPacSDK.bas, into your project




### Step 2: Then you can use the APIs of XPacSDK in your project.

# Delphi (for XP-8000 Series only)

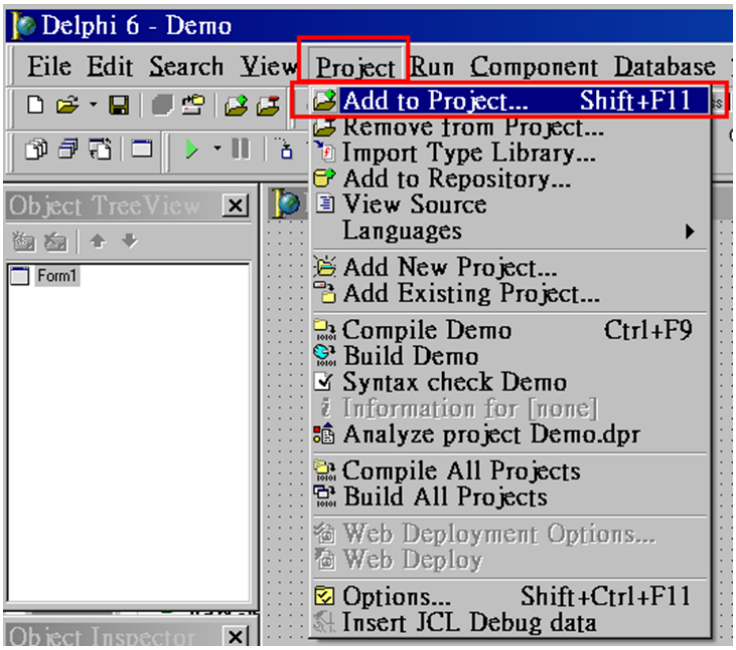
## Required Libraries and Header files

To develop your Delphi program, you must use the appropriate libraries and header files for targeting your program.

1. XPacSDK.pas
2. XPacSDK.dll (your execution file should be put in the same directory as the XPacSDK.dll)

## How to

### Step 1: Add unit, XPacSDK.pas, into your project




### Step 2: Uses XPacSDK.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Va
  Dialogs, XPacSDK;

type
  TForm1 = class(TForm)
```

# Borland C++ Builder (BCB) (for XP-8000 Series only)

## Required Libraries and Header files

To develop your Borland C++ Builder (BCB) program, you must use the appropriate libraries and header files for targeting your program.

1. XPacSDK_BCB.lib
2. XPacSDK.h
3. XPacSDK.dll (your execution file should be put in the same directory as the XPacSDK.dll)

## How to

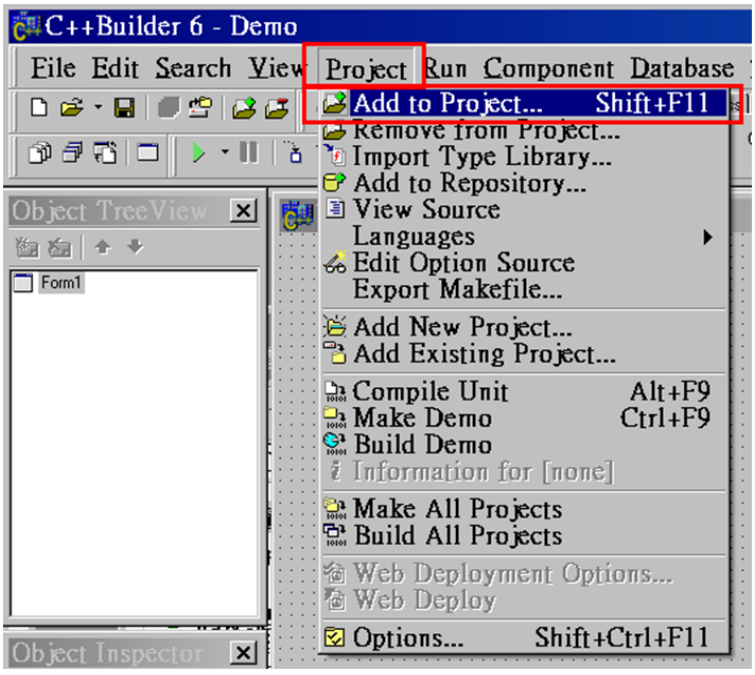
### Step 1: Include XPacSDK.h

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"

#include "XPacSDK.h"
```

### Step 2: Add XPacSDK_BCB.lib into your project



# XPacSDK Plug-in for Development Tools for XPAC_CE Series

Before developing your program by using this SDK on XPAC_CE series, you have to install XPacSDK_CE.msi in your desktop. The XPacSDK_CE is necessary for application developers targeting XPAC_CE. The corresponding XPacSDK.nsi must be installed on the Host PC.

## To install the XPacSDK.msi, please perform the following steps:

### Step 1: Run the “XPacSDK_CE.msi

You can download and unzip from:

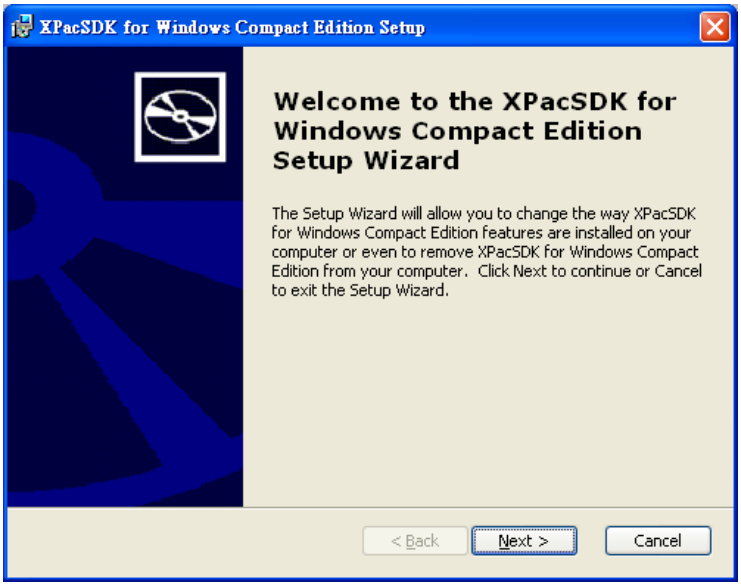
**XP-8000-CE6**

ftp://ftp.icpdas.com/pub/cd/xp-8000-ce6/sdk/platformsdk/xpacsdk_ce_xxx.msi

**XP-8000-Atom-CE6**

ftp://ftp.icpdas.com/pub/cd/xpac-atom-ce6/sdk/platformsdk/xpacsdk_ce_xxx.msi

### Step 2: Follow the prompts until the XPacSDK.msi installation process is complete.



# C/C++/MFC (for XPAC_CE Series only)

## Required Libraries and Header files

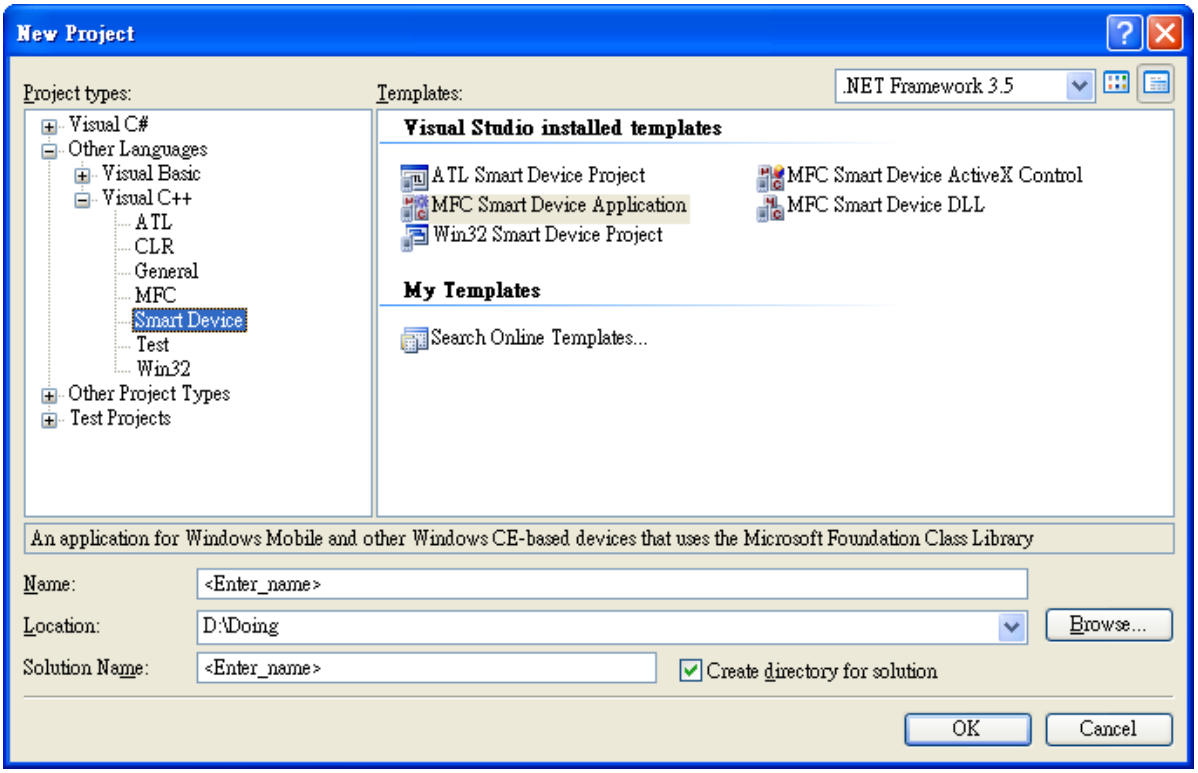
To develop your C/C++/MFC program, you must use the appropriate libraries and header files for targeting your program.

1. XPacSDK_CE.h
2. XPacSDK_CE.lib
3. XPacSDK_CE.dll (your execution file should be put in the same directory as the XPacSDK_CE.dll)

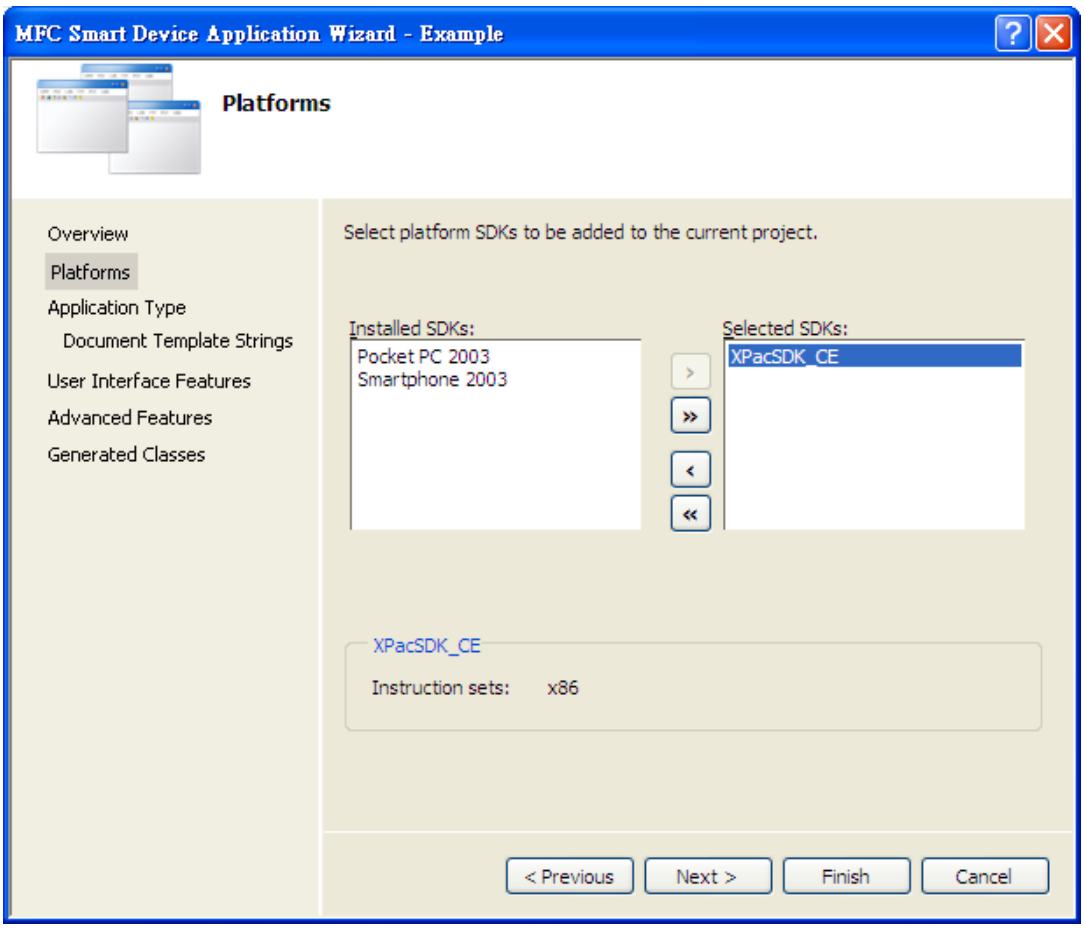
## How to

### Step 1: Create a new project by using Visual Studio 2005/2008

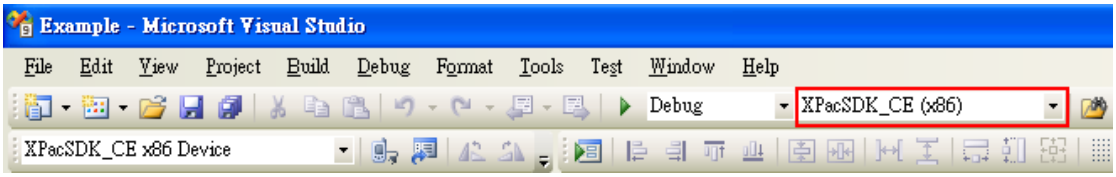
### Step 2: Select Smart Device



## Step 3: Select platform, XPacSDK_CE, to be added to the current project




## Step 4: On the configuration toolbar, select the XPacSDK_CE(x86)




## Step 5: Include XPacSDK_CE.h

## Step 6: Include XPacSDK_CE.lib

# C# (for XPAC_CE Series only)

## Required Libraries

To develop your C# program with XPacSDK_CE, you have to use the appropriate libraries and use DllImport to make calls to XPacSDK_CE.dll which is native code from a managed application.

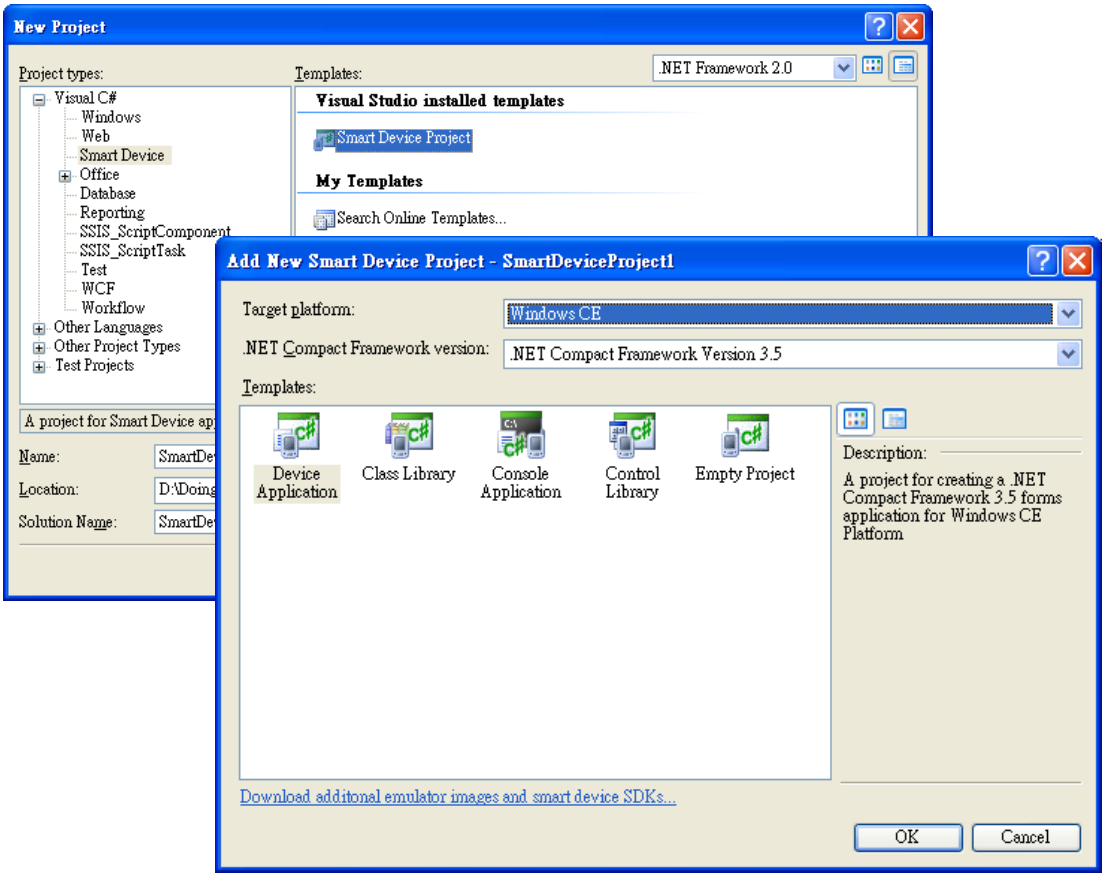
- XPacSDK_CE.dll (your execution file should be put in the same directory as the XPacSDK_CE.dll)

## How to

### 1. Usign DllImport:

#### Step 1: Create a new project by using Visual Studio 2005/2008

#### Step 2: Select Smart Device



### Step 3: In order to use “DllImport”, you should using System.Runtime.InteropServices, and then implement the function which you want to call in the XPacSDK_CE

**For example, you want use pac_WriteDO in .NET project directly.**

**[In XPacSDK_CE.h file]**

XPAC_API BOOL pac_WriteDO(HANDLE hPort, int slot, int iDO_TotalCh, DWORD lDO_Value);

**[In your .NET project]**

- Added this line in your project:

```
using System.Runtime.InteropServices;
```

- Declare this function as following:

```
[DllImport("XPacSDK_CE.dll", EntryPoint = "pac_WriteDO")]

public extern static bool pac_WriteDO(IntPtr hPort, int slot, int
iDO_TotalCh, uint lDO_Value);
```

- Then you can use this function, pac_WriteDO, in your .NET project.

**[Code Snippet]**

```
using System.Windows.Forms;
using System.Runtime.InteropServices;
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        [DllImport("XPacSDK_CE.dll", EntryPoint = "pac_WriteDO")]
        public extern static bool pac_WriteDO(IntPtr hPort, int slot, int
iDO_TotalCh, uint lDO_Value);

        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            pac_WriteDO((IntPtr)0, 1, 16, 0xff);
        }
    }
}
```

## 2. XPacNET.cs:

We support a C# file, XPacNET.cs, which have declared all the functions of XPacSDK_CE. You could just add it in your .NET project, and then you can use these functions easily.

- XPacNET.cs (which is located at

  CD root\XP-8000-CE6\SDK\ (in the companion CD)

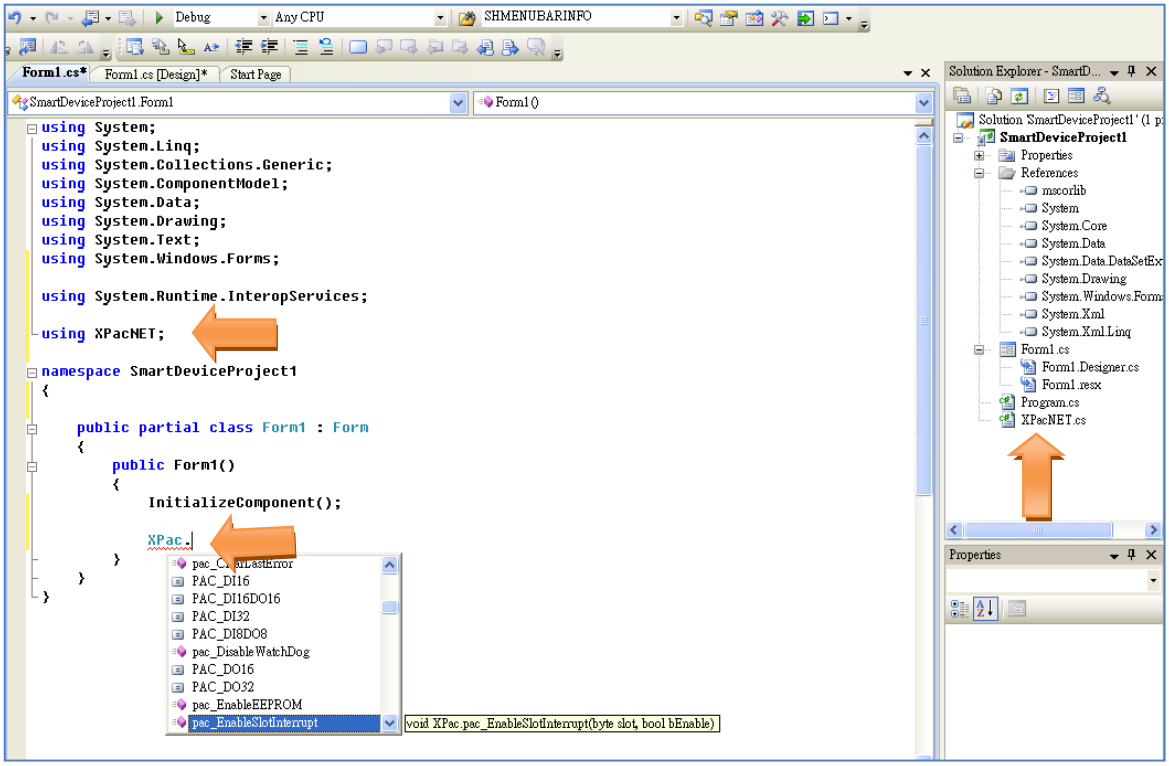
  CF Card root\SDK\

  ftp://ftp.icpdas.com/pub/cd/xpac_ce/xp-8000-ce6/sdk

### Step 1: Create a new project by using Visual Studio 2005/2008

### Step 2: Select Smart Device

### Step 3: Add the XPacNET.cs into your project, and using XPacNET



## 3. Using XPacNet.dll to replace the XpacNet.cs

XPacNet.dll is a .net Compact framework SDK and it is built from Xpacnet.cs and the other .cs file. XpacNet.dll isn't used for #C program but also used for VB.net program.

- XPacNET.dll (which is located at

  CD root\XP-8000-CE6\SDK\XPacNET (in the companion CD)

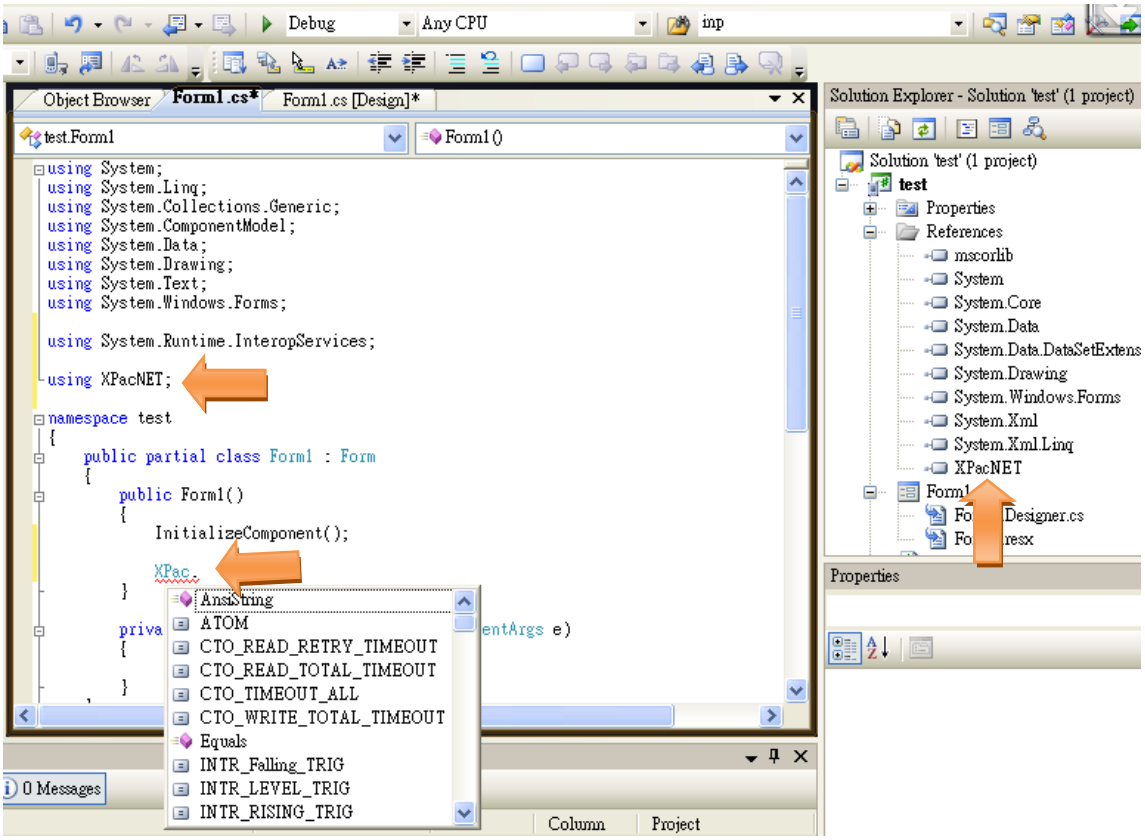
  CF Card root\SDK\XPacNET

  ftp://ftp.icpdas.com/pub/cd/xp-8000-ce6/sdk/xpacnet

### Step 1: Create a new project by using Visual Studio 2005/2008

### Step 2: Select Smart Device

### Step 3: Add the XPacNET.dll into the references of the project, and using XPacNET



**XP-8000-CE6**

Refer to the chapter 4 on xp-8000-ce6 user manual for more details

ftp://ftp.icpdas.com/pub/cd/xp-8000-ce6/document/user_manual/

**XP-8000-Atom-CE6**

Refer to the chapter 4 on xp-8000-Atom-ce6 user manual for more details

ftp://ftp.icpdas.com/pub/cd/xpac-atom-ce6/document/user_manual/

# 1. System Information API

## System Information Reference

System operations include basic operation, such as reboot and changing slot and version display, including OS, 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 XPAC.

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetModuleName | Y | Y | Y | Y |
| pac_GetRotaryID | Y | Y | Y | Y |
| pac_GetSerialNumber | Y | Y | Y | Y |
| pac_GetSDKVersion | Y | Y | Y | Y |
| pac_ChangeSlot | Y | Y | Y | Y |
| pac_CheckSDKVersion | Y | Y | Y | Y |
| pac_ModuleExists | Y | Y | Y | Y |
| pac_GetOSVersion | X | Y | X | Y |
| pac_GetMacAddress | X | Y | X | Y |
| pac_ReBoot | X | Y | X | Y |
| Pac_GetCPUVersion | Y | Y | Y | Y |
| Pac_EnableLED | X | X | Y | Y |

## Function List

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

| Function | Description |
|---|---|
| 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 devices. This function supports the collection of system hardware configurations. |
| pac_GetRotaryID | This function retrieves the rotary switch ID. |
| pac_GetSerialNumber | This function retrieves the serial number. |
| pac_GetSDKVersion | This function retrieves the SDK version. |
| pac_ChangeSlot | This function specifies the slot from one to another. |
| pac_CheckSDKVersion | This function specifies the application not access the XPacSDK.dll which is wrong version. |
| pac_ModuleExists | This function specifies the IO module whether exist or not, no matter the module is local or remote. |
| pac_GetOSVersion | This function retrives OS version. |
| pac_GetMacAddress | This function retrives the Mac address. |
| pac_ReBoot | This function reboots the device. |
| Pac_GetCPUVersion | This function retrieves the CPU version. |
| Pac_EnableLED | This function decides the LED turning on or not. |

# 1.1. 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 devices. This function supports the collection of system hardware configurations.

## Syntax

**C++**

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

## Parameter

*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 87k modules, otherwise 8k or undefined.

## Examples

### [VC/VS]

```
byte slot = 1;
char strName[MAX_NAME_SIZE];
pac_GetModuleType(slot, strName);
```

### [C#] for XP-8000 series only

```
byte slot = 1;
string strName;
int ModuleType = 0;
ModuleType = XPac.pac_GetModuleName(slot, ref strName);
// because of calling by reference, you should add key word, ref.
```

```
// 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 = XPac.pac_GetModuleName(slot);
```

# 1.2. pac_GetRotaryID

This function retrieves the rotary switch ID.

## Syntax

### C++

```
int pac_GetRotaryID();
```

## Parameter

None

## Return Values

Return the rotary ID.

## Examples

### [VC/VS]

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

### [C#] for XP-8000 series only

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

# 1.3. pac_GetSerialNumber

This function retrieves the serial number.

## Syntax

### C++

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

## Parameter

*SerialNumber*

[out] Pointer to a variable that specifies the serial number.

## Return Values

None

## Examples

### [VC/VS]

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

### [C#] for XP-8000 series only

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

## Remark

If the retrieved value is empty, means the function exectres failure or the device is not valid product.

# 1.4. pac_GetSDKVersion

This function retrieves the SDK version.

## Syntax

**C++**

```
void pac_GetSDKVersion(
        LPSTR sdk_version
);
```

## Parameter

*sdk_version*

[out] Pointer to a variable that specifies the SDK version.

## Return Values

None

## Examples

### [VC/VS]

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

### [C#] for XP-8000 series only

```
string XPacSDK;
string XPacNET;
XPacSDK = XPac.pac_GetXPacSDKVersion(); //retrieving XPacSDK version
XPacNET = XPac.pac_GetXPacNetVersion(); //retrieving XPacNET version
//In .net, ths API is different with VC. And there are two API,
//pac_GetXPacSDKVersion and pac_GetXPacNetVersion.
```

# 1.5. pac_ChangeSlot

This function specifies the slot from one to another.

## Syntax

**C++**

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

## Parameter

*slotNo*

[in] Specifies which slot.

## Return Values

None

## Examples

### [VC/VS]

```
BYTE slot;
HANDLE hPort;
BOOL ret;
char buf[Length];
hPort = uart_Open("");
pac_ChangeSlot(slot);
// Change to the slot which the 87k modules plug in
ret = uart_SendCmd(hPort,"$00M", buf);
// $00M: ask the device name
```

### [C#] for XP-8000 series only

```
BYTE slot;
IntPtr hPort;
bool ret;
string buf;
hPort = XPac.uart_Open(XPac.AnsiString(""));
XPac.pac_ChangeSlot(slot);
// Change to the slot which the 87k modules plug in
ret = XPac.uart_SendCmd(hPort, XPac.AnsiString("$00M"), buf);
```

## Remark

When you use uart API and the IO modules located as slots.

You have to call pac_ChangeSlot to change the slot.

Besides, other low level operations may use pac_ChangeSlot to change the slot.

If you just use PAC_IO APIs, you needn’t care about this.

# 1.6. pac_CheckSDKVersion

This function specifies the application not access the XPacSDK.dll which is wrong version.

## Syntax

**C++**

```
BOOL pac_CheckSDKVersion(
        DWORD version
);
```

## Parameter

*version*

[in] Specifies SDK version.

If your application should use the version, 1.0.0.1, or newer, your parameter should be 0x01000001.

## Return Values

Return TRUE, if the version is correctly.

Otherwise, return false.

## Examples

### [VC/VS]

```
//……
//Added this API in the begin of your application
BOOL bVersion;
bVersion = pac_CheckSDKVersion( 0x01000001);
//if your application should use newer than version 1.0.0.1
if(!bVersion)
{
   MessageBox(“The XPacSDK.dll version is wrong”);
   // display some warning and close the application
}
```

### [C#] for XP-8000 series only

```
//……
// Added this API in the begin of your application
bool bVersion;
bVersion = XPac.pac_CheckSDKVersion( 0x01000001);

// if your application should use newer than version 1.0.0.1
if(!bVersion)
{
   MessageBox.show(“The XPacSDK.dll version is wrong”);
// display some warning and close the application
}
```

# 1.7. pac_ModuleExists

This function specifies the IO module whether exist or not, no matter the module is local or remote.

## Syntax

**C++**

```
BOOL pac_ModuleExists(
        HANDLE hPort
        BYTE slot
);
```

## Parameter

*hPort*

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

It your IO module is located at local, this Parameter should be 0.

*Slot*

[in] The slot in which module is to check exists or not.

If the IO module is remote, please input the module's ID.

## Return Values

Return TRUE, if the module exists.

Otherwise, return false.

## Examples

### [VC/VS]

```
//……
//if you want to check a module which is in the slot 5
BOOL bExist;
bExist = pac_ModuleExists(0, 5);
if(bExist)
{
   MessageBox(“The module exist !”);
}
   else
{
   MessageBox(“The module unexist !”);
}
```

### [C#] for XP-8000 series only

```
//……
//if you want to check a module which is in the slot 5
bool bExist;
bExist = XPac.pac_ModuleExists(0, 5);
if(bExist)
{
   MessageBox.show(“The module exist !”);
}
   else
{
   MessageBox.show(“The module unexist !”);
}
```

# 1.8. pac_GetOSVersion (for XPAC_CE series only)

This function retrives OS version.

## Syntax

**C++**

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

## Parameter

*LPSTR*

[out] Pointer to a variable that specifies the OS version.

## Return Values

None.

## Examples

**[VS]**

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

# 1.9. pac_GetMacAddress (for XPAC_CE series only)

This function retrives the Mac address.

## Syntax

**C++**

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

## Parameter

*LAN*

[in] Specifies which LAN.

*LPSTR*

[out] Pointer to a variable that specifies the MAC address.

## Return Values

None.

## Examples

**[VS]**

```
char MAC[32];
BYTE LAN = 1;
pac_GetMacAddress(LAN, MAC);
```

# 1.10. pac_ReBoot (for XPAC_CE series only)

This function reboots the device.

## Syntax

**C++**

```
void pac_ReBoot();
```

## Parameter

None.

## Return Values

None.

## Examples

**[VS]**

```
pac_ReBoot();
```

# 1.11. Pac_GetCPUVersion

This function retrieves the CPU version.

## Syntax

**C++**

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

## Parameter

*cpu_version*

[out] The pointer to a buffer to receive the cpu version.

## Return Values

None

## Examples

**[VC]**

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

**[C#]**

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

# 1.12. Pac_EnableLED (for XPAC_Atom series only)

This function decides the LED turning on or not.

## Syntax

**C++**

```
void pac_EnableLED(
        int pin,
        bool bFlag
);
```

## Parameter

*pin*

0: L1 LED

1: L2 LED

*bFlag*

True: Turn on the LED

False: Turn off the LED

## Return Values

None

## Examples

### [VC]

```
pac_EnableLED(0,TRUE);
```

### [C#]

```
XPac.pac_EnableLED(0,true);
```

# 2. Backplane Access API

## Backplane Access Reference

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, and access the backplane by using these functions.

## Supported Modules

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

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetDIPSwitch | Y | Y | Y | Y |
| pac_GetSlotCount | Y | Y | Y | Y |
| pac_EnableRetrigger | Y | Y | Y | Y |
| pac_GetBackplaneID | Y | Y | Y | Y |
| pac_GetBatteryLevel | Y | Y | Y | Y |

## Function List

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

| Function | Description |
|---|---|
| pac_GetDIPSwitch | This function retrieves the dip switch. |
| pac_GetSlotCount | This function retrieves the number of slot. |
| 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. |

# 2.1. pac_GetDIPSwitch

This function retrieves the dip switch.

## Syntax

### C++

```
int pac_GetDIPSwitch();
```

## Parameter

None

## Return Values

The return value specifies the dip switch.

## Examples

### [VC/VS]

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

### [C#] for XP-8000 series only

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

# 2.2. pac_GetSlotCount

This function retrieves the number of slot.

## Syntax

### C++

```
int pac_GetSlotCount();
```

## Parameter

None

## Return Values

The return value is the number of the slot of this device.

## Examples

### [VC/VS]

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

### [C#] for XP-8000 series only

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

# 2.3. pac_EnableRetrigger

This function decides the retrigger turning on or not.

## Syntax

**C++**

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

## Parameter

*iValues*

[in] Specify the retrigger value, 0~255, unit= 10 microsecond. (0 means disable retrigger function)

## Return Values

None

## Examples

None

## Remark

The retrigger mechanism is used when the below situation occurred.

If an interrupt is sent but not be serviced, the retrigger function will send an interrupt again. This operation will continue until the interrupt has been serviced.

# 2.4. pac_GetBackplaneID

This function retrieves the backplane ID.

## Syntax

### C++

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

## Parameter

*backplane_version*

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

## Return Values

None

## Examples

### [VC/VS]

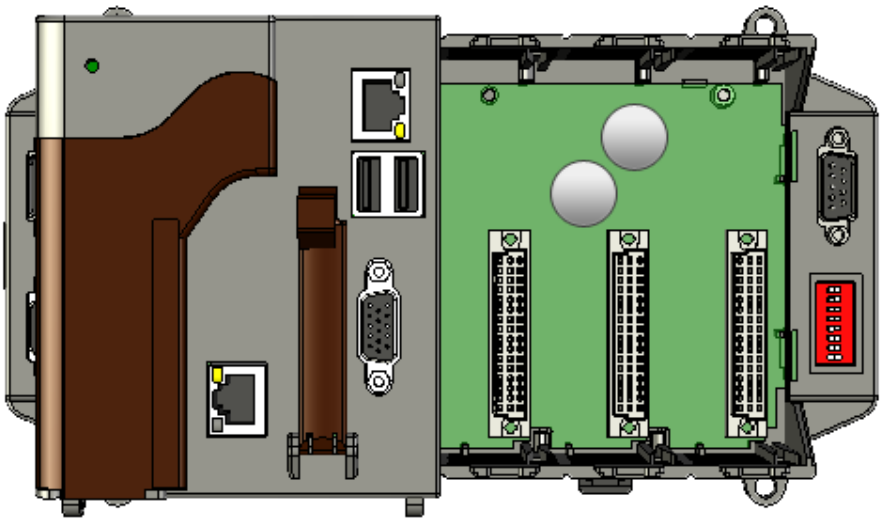
```
char Backplane[32];
pac_GetBackplaneID(Backplane);
```

### [C#] for XP-8000 series only

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

# 2.5. pac_GetBatteryLevel

This function retrieves the backplane battery status.



## Syntax

### C++

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

## Parameter

*nBattery*

[in] Specifies the index of battery.

1 means first battery.

2 means second battery.

3 means RTC battery. (For XPAC_Atom series only)

## Return Values

1 means high voltage.

0 means low voltage.

## Examples

### [VC/VS]

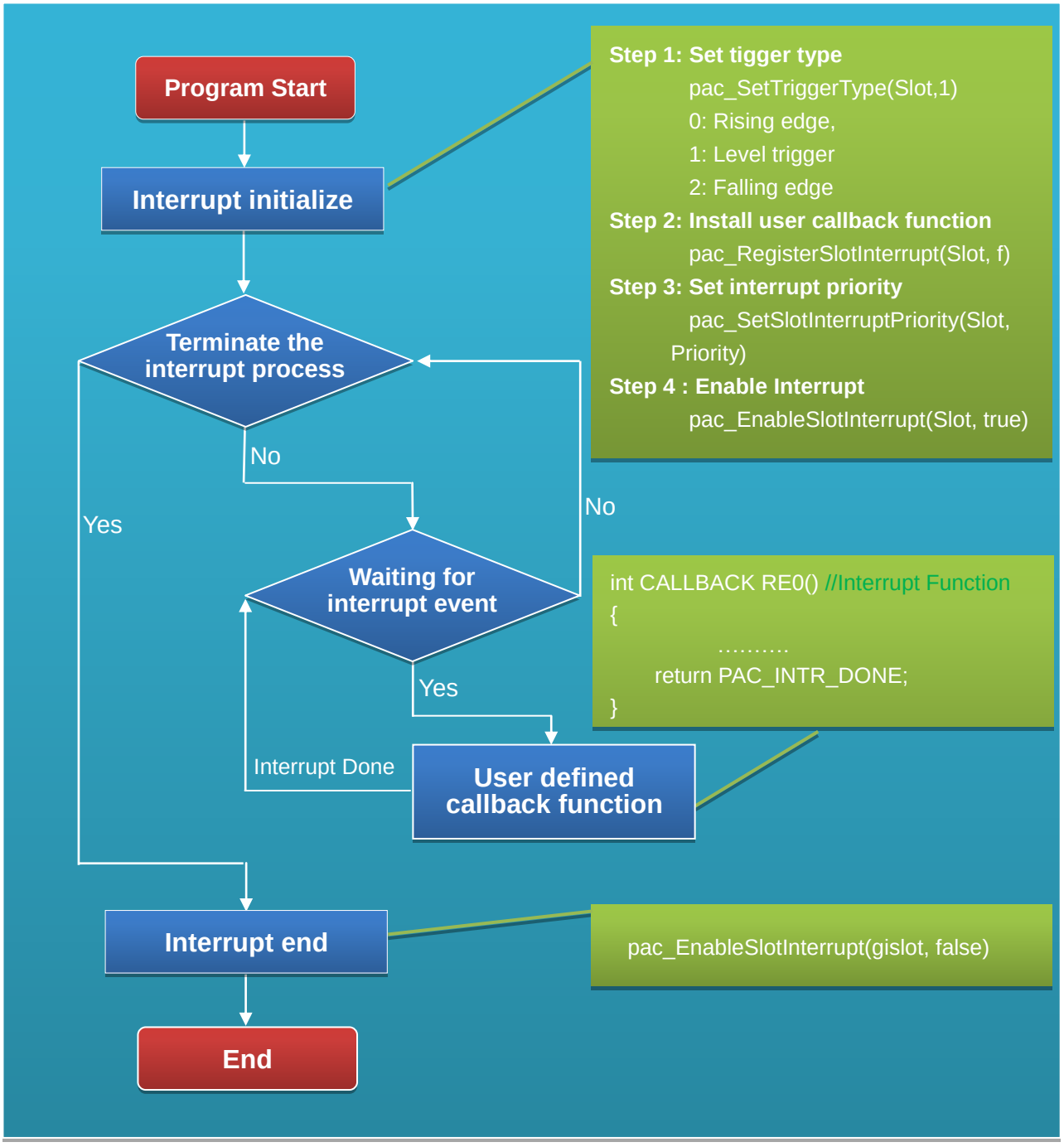
```
int nBattery;
int index = 1;
nBattery = pac_GetBatteryLevel(index);
```

# 3. Interrupt API

## Interrupt Reference

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.

## Interrupt Flow



## Supported Modules

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

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_RegisterSlotInterrupt | Y | Y | Y | Y |
| pac_UnregisterSlotInterrupt | Y | Y | Y | Y |
| pac_EnableSlotInterrupt | Y | Y | Y | Y |
| pac_SetSlotInterruptPriority | Y | Y | Y | Y |
| pac_InterruptInitialize | Y | Y | Y | Y |
| pac_GetSlotInterruptEvent | Y | Y | Y | Y |
| pac_SetSlotInterruptEvent | Y | Y | Y | Y |
| pac_SetTriggerType | Y | Y | Y | Y |
| pac_GetSlotInterruptID | Y | Y | Y | Y |
| pac_InterruptDone | Y | Y | Y | Y |

## Function List

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

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

# 3.1. pac_RegisterSlotInterrupt

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

## Syntax

**C++**

```
BOOL pac_RegisterSlotInterrupt(
        BYTE slot,
        PAC_CALLBACK_FUNC f
);
```

## Parameter

*slot*

[in] Specify the index of slot.

*f*

a call back function.

## Return Values

Return TRUE if success, otherwise return FALSE.

## Examples

### [VC/VS]

```
int slot = 3; // if slot is 3
int CALLBACK slot_callback_proc()
{
// do something
   return true;
// if return true, SDK will do pac_InterruptDone automatically
// else, users should do pac_InterruptDone by themselves if needed.
// if interrupt type is level trigger, no matter return true or flase,
// needn't add pac_InterruptDone and it will work correctly.
}
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
}
```

## Remark

Default trigger type is level trigger.

For XP-8000 series, only support level trigger type.

# 3.2. pac_UnregisterSlotInterrupt

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

## Syntax

**C++**

```
BOOL pac_UnregisterSlotInterrupt(
        BYTE slot
);
```

## Parameter

*slot*

[in] Specify the index of slot.

## Return Values

Return TRUE if success, otherwise return FALSE.

## Examples

### [VC/VS]

```
int slot = 3; // if slot is 3
int CALLBACK slot_callback_proc()
{
// do something
pac_InterruptDone(slot);
return false;
// if return true, SDK will do pac_InterruptDone automatically
//else, users should do pac_InterruptDone by themselves if needed
// if interrupt type is level trigger, no matter return true or flase,
// needn't add pac_InterruptDone and it will work correctly.
}
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
}
```

# 3.3. pac_EnableSlotInterrupt

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

## Syntax

**C++**

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

## Parameter

*slot*

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

*bEnable*

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

## Return Values

None

## Examples

### [VC/VS]

```
int slot = 3; // if slot is 3
int CALLBACK slot_callback_proc()
{
// do something
pac_InterruptDone(slot);
return true;
// if return true, SDK will do pac_InterruptDone automatically
//else, users should do pac_InterruptDone by themselves if needed
// if interrupt type is level trigger, no matter return true or flase,
// needn't add pac_InterruptDone and it will work correctly.
}
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
}
```

## Remark

Default trigger type is level trigger.

For XP-8000 series, only support level trigger type.

# 3.4. pac_SetSlotInterruptPriority

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

## Syntax

**C++**

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

## Parameter

*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

TRUE indicates success.

FALSE indicates failure.

## Examples

None

# 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

**C++**

```
BOOL pac_InterruptInitialize(
        BYTE slot
);
```

## Parameter

*slot*

[in] Specify the index of slot to initialize.

## Return Values

TRUE indicates success; FALSE indicates failure.

## Examples

None

## Remark

Default trigger type is level trigger.

For XP-8000 series, only support level trigger type.

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

# 3.6. pac_GetSlotInterruptEvent

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

## Syntax

**C++**

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

## Parameter

*slot*

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

## Return Values

A handle to the event object indicates success. NULL indicates failure.

## Examples

None

# 3.7. pac_SetSlotInterruptEvent

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

## Syntax

**C++**

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

## Parameter

*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_SetTriggerType

This function can assign the pulse trigger type for separate slot.

## Syntax

**C++**

```
void pac_SetTriggerType(
        BYTE slot,
        int iType
);
```

## Parameter

*iType*

[in] Specify the pulse trigger type.

0: Rising edge trigger(default)

1: Level trigger

2: Falling edge trigger

## Return Values

None

## Examples

None

## Remark

For XP-8000 series, only support level trigger type.

# 3.9. pac_GetSlotInterruptID

This function retrieves the slot interrupt ID.

## Syntax

**C++**

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

## Parameter

*slot*

[in] Specify the slot.

## Return Values

Return the slot interrupt ID.

## Examples

None

# 3.10. pac_InterruptDone

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

## Syntax

**C++**

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

## Parameter

*slot*

[in] Specify the slot to clear trigger.

## Return Values

None

## Examples

### [VC/VS]

```
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);
}
```

# 4. Memory Access API

## Memory Access Reference

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 XPAC.

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetMemorySize | Y | Y | Y | Y |
| pac_ReadMemory | Y | Y | Y | Y |
| pac_WriteMemory | Y | Y | Y | Y |
| pac_EnableEEPROM | Y | Y | Y | Y |

## 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 for writing EEPROM. |

# 4.1. pac_GetMemorySize

This function retrieves the size of the specified memory.

## Syntax

**C++**

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

## Parameter

*mem_type*

[in] Handle to a currently type memory.

XPAC_MEM_SRAM 0

XPAC_MEM_EEPROM 1

## Return Values

The return value specifies the memory size.

## Examples

### [VC/VS]

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

### [C#] for XP-8000 series only

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

# 4.2. pac_ReadMemory

This function retrieves data from the specified memory.

## Syntax

**C++**

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

## Parameter

*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.

XPAC_MEM_SRAM 0

XPAC_MEM_EEPROM 1

## 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

### [VC/VS]

```
#define LENGTH 1
BOOL ret;
DWORD address = 0;
BYTE Buffer[LENGTH] = 10;
ret = pac_ReadMemory(address, Buffer, LENGTH, XPAC_MEM_SRAM);
```

### [C#] for XP-8000 series only

```
bool ret;
uint address ;
byte[] Buffer = new byte [1];
Buffer[0] = 10;
ret = XPac.pac_ReadMemory(address, Buffer, LENGTH, 0);
```

# 4.3. pac_WriteMemory

This function stores data in the specified memory.

## Syntax

**C++**

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

## Parameter

*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.

XPAC_MEM_SRAM 0

XPAC_MEM_EEPROM 1

## 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

### [VC/VS]

```
#define LENGTH 1
BOOL ret;
DWORD address = 0;
BYTE Buffer[LENGTH] = 10;
ret = pac_WriteMemory(address, Buffer, LENGTH, XPAC_MEM_SRAM);
```

### [C#] for XP-8000 series only

```
bool ret;
uint address;
byte[] Buffer = new byte [1];
Buffer[0] = 10;
ret = XPac.pac_WriteMemory(address, Buffer, LENGTH, 0);
```

## Remark

Before writing EEPROM, the EEPROM should be enabled using pac_EnableEEPROM.

# 4.4. pac_EnableEEPROM

This function turns on/off for writing EEPROM.

## Syntax

**C++**

```
void pac_EnableEEPROM(
        BOOL bEnable
);
```

## Parameter

*bEnable*

[in] To turn on or protect EEPROM.

True: To turn on EEPROM

False: To protect EEPROM

## Return Values

None

## Examples

### [VC/VS]

```
int ret;
DWORD address;
BYTE Buffer[LENGTH];
pac_EnableEEPROM)(true);
ret = pac_WriteMemory(address, Buffer, LENGTH, XPAC_MEM_EEPROM);
pac_EnableEEPROM)(false) ;
```

### [C#] for XP-8000 series only

```
int ret;
uint address;
byte[] Buffer = new byte [LENGTH];
XPac.pac_EnableEEPROM)(true);
ret = XPac.pac_WriteMemory(address, Buffer, Buffer.Length, 0);
XPac.pac_EnableEEPROM)(false);
```

## Remark

Before writing EEPROM, need turn on the EEPROM; after written EEPROM, need turn off the EEPROM.

# 5. Watchdog API

## Watchdog Reference

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 interrupt functions which are available with XPAC.

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_EnableWatchDog | Y | Y | Y | Y |
| pac_DisableWatchDog | Y | Y | Y | Y |
| pac_RefreshWatchDog | Y | Y | Y | Y |
| pac_GetWatchDogState | Y | Y | Y | Y |
| pac_GetWatchDogTime | Y | Y | Y | Y |
| pac_SetWatchDogTime | Y | Y | Y | Y |

## Function List

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

| Function | Description |
|---|---|
| pac_EnableWatchDog | This function turns on the watchdog operation. |
| pac_DisableWatchDog | This function turns off the watchdog operation. |
| pac_RefreshWatchDog | This function refreshes the watchdog. |
| pac_GetWatchDogState | This function retrieves the watchdog state. |
| pac_GetWatchDogTime | This function retrieves the watchdog time. |
| pac_SetWatchDogTime | The same as the pac_EnableWatchDog function. |

# 5.1. pac_EnableWatchDog

This function turns on the watchdog operation.

## Syntax

**C++**

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

## Parameter

*wdt*

[in] Specifies the Watchdog type:

XPAC_WDT_HW 0

XPAC_WDT_OS 1

*value*

[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

### [VC/VS]

```
DWORD second;
BOOL ret;
ret = pac_EnableWatchDog(XPAC_WDT_OS, second);
```

### [C#] for XP-8000 series only

```
uint second;
bool ret;
ret = XPac.pac_EnableWatchDog(1, second);
```

## Remark

The unit of the second paremeter of the OS watchdog is second. In addition, the unit can not be zero.

The hardware watchdog second Parameter is a value which is between 0~63 unit.

A unit is about 0.5 seconds. 0 means the shortest timeout, otherwise 63 is longest which is about 30 seconds.

# 5.2. pac_DisableWatchDog

This function turns off the watchdog operation.

## Syntax

**C++**

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

## Parameter

*wdt*

[in] Specifies the Watchdog type:

XPAC_WDT_HW 0

XPAC_WDT_OS 1

## Return Values

None

## Examples

### [VC/VS]

```
pac_DisableWatchDog(XPAC_WDT_OS);
```

### [C#] for XP-8000 series only

```
XPac.pac_DisableWatchDog(1);
```

# 5.3. pac_RefreshWatchDog

This function refreshes the watchdog.

## Syntax

**C++**

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

## Parameter

*wdt*

[in] Specifies the Watchdog type:

XPAC_WDT_HW 0

XPAC_WDT_OS 1

## Return Values

None

## Examples

### [VC/VS]

```
pac_RefreshWatchDog(XPAC_WDT_OS);
```

### [C#] for XP-8000 series only

```
XPac.pac_RefreshWatchDog(1);
```

# 5.4. pac_GetWatchDogState

This function retrieves the watchdog state.

## Syntax

**C++**

```
BOOL pac_GetWatchDogState(
        int wdt
);
```

## Parameter

*wdt*

[in] Specifies the Watchdog type:

XPAC_WDT_HW 0

XPAC_WDT_OS 1

## Return Values

The return value specifies the watchdog state.

If turning on, return TRUE, otherwise FALSE.

## Examples

### [VC/VS]

```
BOOL bState;
bState = pac_GetWatchDogState(XPAC_WDT_OS);
```

### [C#] for XP-8000 series only

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

# 5.5. pac_GetWatchDogTime

This function retrieves the watchdog time.

## Syntax

**C++**

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

## Parameter

*wdt*

[in] Specifies the Watchdog type:

XPAC_WDT_HW 0

XPAC_WDT_OS 1

## Return Values

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

The unit of return value is second for OS watchdog value which is between 0~63 for hardware watchdog.

## Examples

### [VC/VS]

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

### [C#] for XP-8000 series only

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

# 5.6. pac_SetWatchDogTime

The same as the pac_EnableWatchDog function.

The hardware watchdog second Parameter is a value which is between 0~63 unit.

A unit is about 0.5 seconds. 0 means the shortest timeout, otherwise 63 is longest which is about 30 seconds.

## Syntax

### C++

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

## Parameter

*wdt*

[in] Specifies the Watchdog type:

XPAC_WDT_HW

XPAC_WDT_OS

*value*

[in] Specifies the watchdog time.

## Return Values

None

## Examples

### [VC/VS]

```
DWORD dwTime;
pac_SetWatchDogTime(XPAC_WDT_OS, dwTime);
```

### [C#] for XP-8000 series only

```
uint uTime;
XPac.pac_SetWatchDogTime(1, uTime);
```

## Remark

The same as the pac_EnableWatchDog function.

The unit of the second paremeter of the OS watchdog is second. In addition, the unit can not be zero.

The hardware watchdog second Parameter is a value which is between 0~63 unit.

A unit is about 0.5 seconds. 0 means the shortest timeout, otherwise 63 is longest which is about 30 seconds.

# 6. Uart API

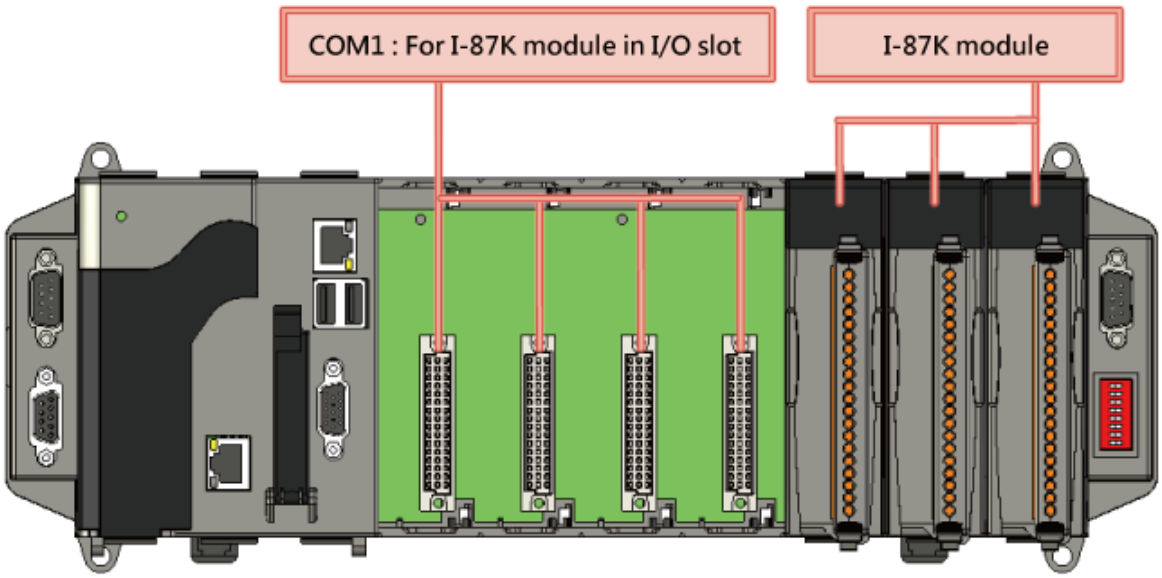
## Uart Reference

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.

## Remark

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 API functions in C++ (CreateFile/CloseHandle/WriteFile/ReadFile /GetCommModemStatus…..).

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

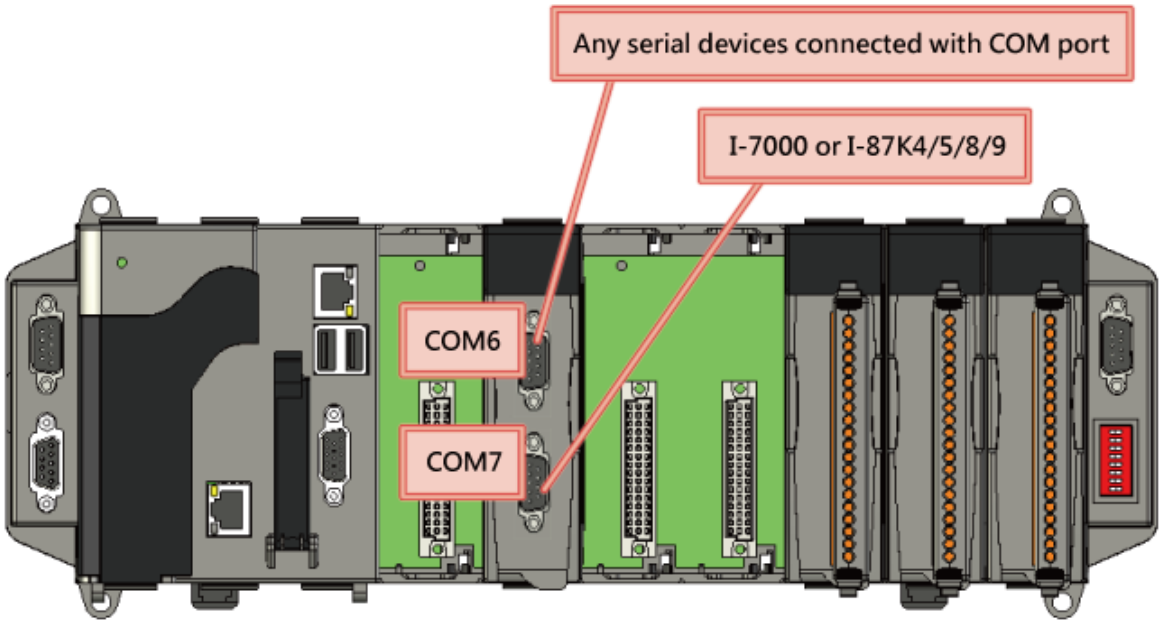


When a high profile I-87K is plugged in slot, please call the function, pac_ChangeSlot, to change to the specific slot before doing other operations. Please refer to demo “87k_Basic”.

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 7 PAC_IO API

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



## Supported Modules

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

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| uart_Open | Y | Y | Y | Y |
| uart_Close | Y | Y | Y | Y |
| uart_Send | Y | Y | Y | Y |
| uart_Recv | Y | Y | Y | Y |
| uart_SendCmd | Y | Y | Y | Y |
| uart_SetTimeOut | Y | Y | Y | Y |
| uart_EnableCheckSum | Y | Y | Y | Y |
| uart_SetTerminator | Y | Y | Y | Y |
| Uart_BinSend | Y | Y | Y | Y |
| Uart_BinRecv | Y | Y | Y | Y |
| Uart_BinSendCmd | Y | Y | Y | Y |
| Uart_GetLineStatus | Y | Y | Y | Y |
| Uart_GetDataSize | Y | Y | Y | Y |

## Function List

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

| Function | Description |
|---|---|
| uart_Open | This function opens the COM port and specifies the baud rate, parity bits, data bits, and stop bits. |
| uart_Close | This function closes the COM port which has been opened. |
| uart_Send | This function sends data through the COM port which have been opened. |
| uart_Recv | This function retrieves data through the COM port which have been opened. |
| uart_SendCmd | This function sends commands through the COM port which have been opened. |
| uart_SetTimeOut | This function sets the time out timer. |
| uart_EnableCheckSum | This function turns on the check sum or not. |
| uart_SetTerminator | This function sets the terminate characters. |
| Uart_BinSend | this function can send out command string with or without null character under the consideration of the command length. |
| Uart_BinRecv | This function is applied to receive the fix length response. |
| Uart_BinSendCmd | This function is used to Send binary command and receive binary data with the fixed length. |
| Uart_GetLineStatus | This function retrieves the modem control-register values. |
| 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 transmitted for write operations. |

# 6.1. uart_Open

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

## Syntax

**C++**

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

## Parameter

*connectionString*

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

The default setting is COM1, 115200, N, 8, 1.

The format of ConnectionString is as follows:

"com_port, baud_rate, parity_bits, data_bits, stop_bits"

Warning: there is no blank space between each parameter.

Com_port:

COM1, COM2……

baud_rate:

1200/2400/4800/9600/19200/38400/57600/115200

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 be 0xffffffff in C/C++/MFC. 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

### [VC/VS]

```
HANDLE hOpen;
hOpen = uart_Open("COM1,9600,N,8,1");
```

### [C#] for XP-8000 series only

```
IntPtr hOpen;
hOpen = XPac.uart_Open("COM1,9600,N,8,1");
```

Remark

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

# 6.2. uart_Close

This function closes the COM port which has been opened.

## Syntax

**C++**

```
BOOL uart_Close(
        HANDLE hPort
);
```

## Parameter

*hPort*

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

## Return Values

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

```
BOOL ret;
HANDLE hOpen;
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_Close(hOpen);
```

### [C#] for XP-8000 series only

```
bool ret;
IntPtr hOpen;
hOpen = XPac.uart_Open("COM1,9600,N,8,1");
ret = XPac.uart_Close(hOpen);
```

## Remark

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

# 6.3. 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

**C++**

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

## Parameter

*hPort*

[in] Handle to the open COM port.

*buf*

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

## Return Values

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_Send(hOpen, buf);
```

### [C#] for XP-8000 series only

```
bool ret;
IntPtr hOpen;
string buf;
hOpen = XPac.uart_Open("COM1,9600,N,8,1");
ret = XPac.uart_Send(XPac.AnsiString(buf));
```

## Remark

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 is 0x0D. (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 character 0x0D. For example:

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

# 6.4. 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 provide a string without the last byte [0x0D].

## Syntax

**C++**

```
BOOL uart_Recv(
        HANDLE hPort,
        LPSTR buf
);
```

## Parameter

*hPort*

[in] Handle to the open COM port.

*buf*

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

## Return Values

TRUE indicates success. FALSE indicates failure.

If this function doesn't receive a character 0x0D, 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

### [VC/VS]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_Recv(hOpen, buf);
```

### [C#] for XP-8000 series only

```
bool ret;
IntPtr hOpen;
string buf;
hOpen = XPac.uart_Open("COM1,9600,N,8,1");
ret = XPac.uart_Recv(XPac.AnsiString(buf));
```

## Remark

The terminate characters is 0x0D. (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.

# 6.5. 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

### C++

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

## Parameter

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_SendCmd(hOpen,"$00M", buf);      // $00M: ask the device name
uart_Close(hPort);
```

### [C#] for XP-8000 series only

```
bool ret;
IntPtr hOpen;
string buf;
hOpen = XPac.uart_Open("COM1,9600,N,8,1");
ret = XPac.uart_SendCmd(hOpen, XPac.AnsiString("$00M"), buf);
// $00M: ask the device name
XPac.uart_Close(hPort);
```

## Remark

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

# 6.6. uart_SetTimeOut

This function sets the time out timer.

## Syntax

**C++**

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

## Parameter

*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

### [VC/VS]

```
HANDLE hOpen;
DWORD mes;
hOpen = uart_Open("COM1,9600,N,8,1");
mes = 300;
uart_SetTimeOut(hOpen, mes, CTO_TIMEOUT_ALL);
uart_Close(hOpen);
```

### [C#] for XP-8000 series only

```
IntPtr hOpen;
uint msc;
hOpen = XPac.uart_Open("COM1,9600,N,8,1");
mes = 300;
XPac.uart_SetTimeOut(hOpen, msc, 0);
XPac.uart_Close(hOpen);
```

## Remark

**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.

# 6.7. 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

**C++**

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

## Parameter

*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

### [VC/VS]

```
HANDLE hUart;
char result[32];
hUart = uart_Open("");
uart_EnableCheckSum(hUart , true);
pac_ChangeSlot(1);
uart_SendCmd(hUart, "$00M", result);
```

### [C#] for XP-8000 series only

```
byte[] result = new byte[32];
IntPtr hPort = XPac.uart_Open("");
XPac.uart_EnableCheckSum(hPort, true);
XPac.pac_ChangeSlot(1);
XPac.uart_SendCmd(hPort, XPac.AnsiString("$00M"), result);
string str = XPac.WideString(result);
```

# 6.8. uart_SetTerminator

This function sets the terminate characters.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] Handle to the open COM port.

*szTerm*

[in] Pointer the terminate characters.

Default is CR.

## Return Values

None

## Examples

### [VC/VS]

```
HANDLE hUart;
char result[32];
hUart = uart_Open("");
uart_SetTerminator(hUart, "\015");
pac_ChangeSlot(1);
uart_SendCmd(hUart, "$00M", result);
```

### [C#] for XP-8000 series only

```
byte[] result = new byte[32];
IntPtr hPort = XPac.uart_Open("");
XPac.uart_SetTerminator(hPort, XPac.AnsiString("\015"));
XPac.pac_ChangeSlot(1);
XPac.uart_SendCmd(hPort, XPac.AnsiString("$00M"), result);
string str = XPac.WideString(result);
```

## Remark

This function relates to uart_Send/uart_Recv/uart_SendCmd.

# 6.9. Uart_BinSend

Send out the command string by fix length, which is controlled by the Parameter “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). Therefore, 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

### C++

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

## Parameter

*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

### [VC/VS]

```
bool ret;
HANDLE hPort;
char buf[2];
buf[0] = 0x41;
buf[1] = 0x42;
hPort = uart_Open("COM4,9600,N,8,1");
ret = uart_BinSend(hPort, buf, 2);
uart_Close(hPort);
```

### [C#]

```
bool ret;
IntPtr hPort;
string buf = "AB";
hPort = XPac.uart_Open("COM4,9600,N,8,1");
ret = XPac.uart_BinSend(hPort, XPac.AnsiString(buf), 2);
XPac.uart_Close(hPort);
```

## Remark

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.

# 6.10. Uart_BinRecv

This function is applied to receive the fix length response. The length of the receiving response is controlled by the Parameter “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

### C++

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

## Parameter

*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

### [VC/VS]

```
bool ret;
   HANDLE hPort;
   char buf[2];
   hPort = uart_Open("COM4,9600,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 = XPac.uart_Open("COM4,9600,N,8,1");
ret = XPac.uart_BinSend(hPort, XPac.AnsiString("AB"), 2);
ret = XPac.uart_Recv(hPort, buf);
XPac.uart_Close(hPort);
```

## Remark

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

# 6.11. 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

### C++

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

## Parameter

*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

### [VC/VS]

```
bool ret;
HANDLE hPort;
char buf[2];
char cmd[2];
hPort = uart_Open("COM4,9600,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 = XPac.uart_Open("COM4,9600,N,8,1");
ret = XPac.uart_BinSendCmd(hPort, cmd, 2, buf, 2);
XPac.uart_Close(hPort);
```

## Remark

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

# 6.12. Uart_GetLineStatus

This function retrieves the modem control-register values.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] Handle to the open COM port.

*pin*

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

This parameter 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

### [VC/VS]

```
HANDLE hPort = uart_Open("COM5,115200,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 = XPac.uart_Open("COM5,115200,N,8,1");
//the pin, DSR, for example
bool ret = XPac.uart_GetLineStatus(hPort, DSR);
if(ret)
Console.WriteLine("The status of DSR is ON");
else
Console.WriteLine ("The status of DSR is OFF");
XPac.uart_Close(hPort);
```

# 6.13. 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 transmitted for write operations.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] Handle to the open COM port.

*data_type*

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

This parameter 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

### [VC/VS]

```
char buf[1024];
DWORD in_Len;
HANDLE hPort = uart_Open("COM3,9600,N,8,1");
in_Len = uart_GetDataSize(hPort, IN_DATA);
BOOL ret = uart_BinRecv( hPort, buf, in_Len);
uart_Close(hPort);
```

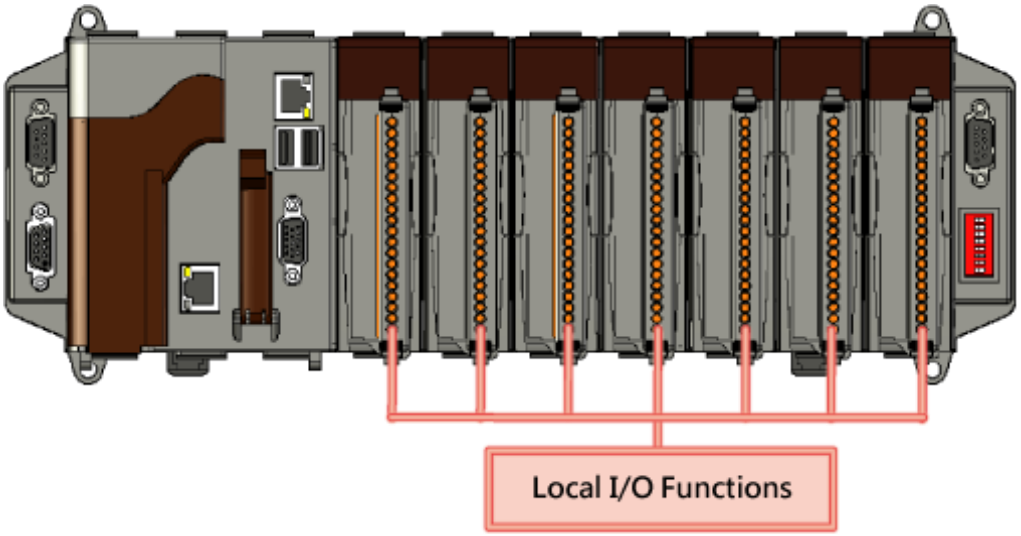
### [C#]

```
string buf;
uint in_Len;
IntPtr hPort = XPac.uart_Open("COM3,9600,N,8,1");
in_Len = XPac.uart_GetDataSize(hPort, 0);     // 0: IN_DATA; 1: OUT_DATA
bool ret = XPac.uart_BinRecv( hPort, XPac.AnsiString(buf), in_Len);
XPac.uart_Close(hPort);
```

# 7. PAC_IO API

## PAC_IO Reference

PAC_IO API supports to operate IO modules, which can be divided into the following parts:

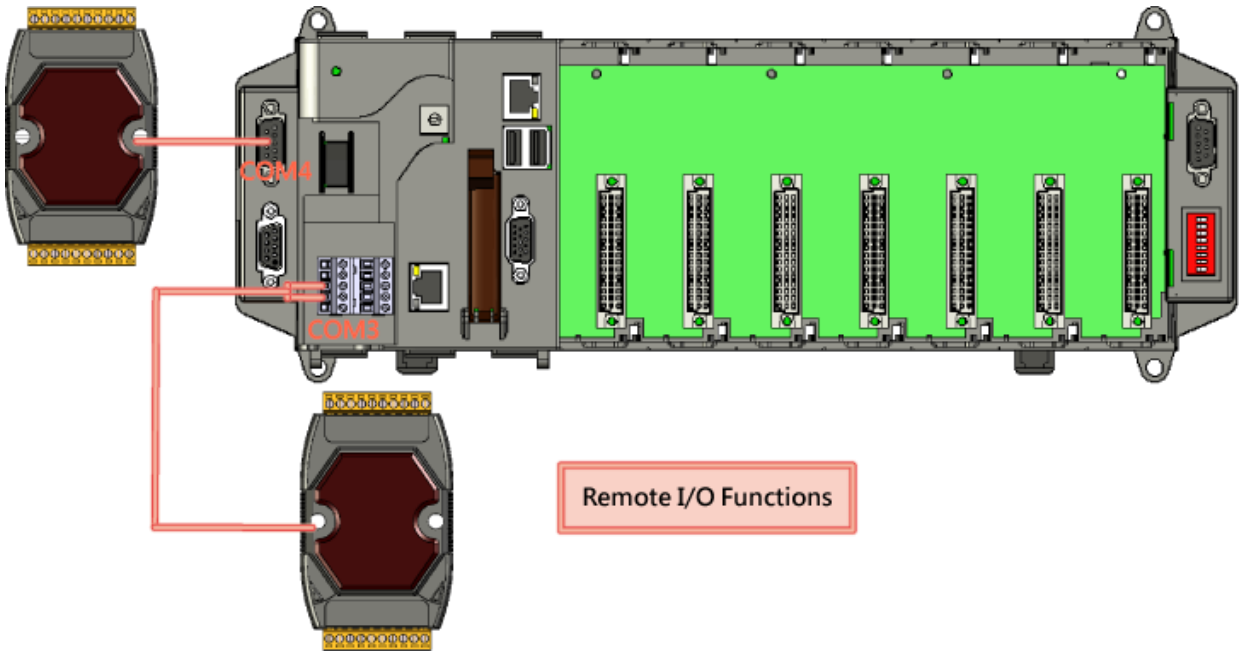



## Local (IO in slot)

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

```
hPort = uart_Open("");
//Clear DO
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 remote module
HANDLE hPort = uart_Open(ConnectionString);
if( !hPort ) AfxMessageBox( _T("Open Com Error"));

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

## Supported Modules

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

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetBit | Y | Y | Y | Y |
| pac_WriteDO | Y | Y | Y | Y |
| pac_WriteDOBit | Y | Y | Y | Y |
| pac_ReadDO | Y | Y | Y | Y |
| pac_ReadDI | Y | Y | Y | Y |
| pac_ReadDIO | Y | Y | Y | Y |
| pac_ReadDILatch | Y | Y | Y | Y |
| pac_ClearDILatch | Y | Y | Y | Y |
| pac_ReadDIOLatch | Y | Y | Y | Y |
| pac_ClearDIOLatch | Y | Y | Y | Y |
| pac_ReadDICNT | Y | Y | Y | Y |
| pac_ClearDICNT | Y | Y | Y | Y |
| pac_WriteAO | Y | Y | Y | Y |
| pac_ReadAO | Y | Y | Y | Y |
| pac_ReadAI | Y | Y | Y | Y |
| pac_ReadAIHex | Y | Y | Y | Y |
| pac_ReadAIAll | Y | Y | Y | Y |
| pac_ReadAIAllHex | Y | Y | Y | Y |
| pac_ReadCNT | Y | Y | Y | Y |
| pac_ClearCNT | Y | Y | Y | Y |
| pac_ReadCNTOverflow | Y | Y | Y | Y |

## Function List

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

| Function | Description |
|---|---|
| pac_GetBit | The function can retrieve the value which in specific bit. |
| 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. |
| pac_ReadDO | This function reads the DO value of the DO module. |
| pac_ReadDI | This function reads the DI value of the DI module. |
| pac_ReadDIO | This function reads the DI and the DO values of the DIO module. |
| pac_ReadDILatch | This function reads the DI latch value of the DI module. |
| pac_ClearDILatch | This function clears the latch value of the DI module. |
| pac_ReadDIOLatch | This function reads the latch values of the DI and DO channels of the DIO module. |
| pac_ClearDIOLatch | This function clears the latch values of DI and DO channels of the DIO module. |
| pac_ReadDICNT | This function reads the counts of the DI channels of the DI module. |
| pac_ClearDICNT | This function clears the counter value of the DI channel of the DI module. |
| pac_WriteAO | This function writes the AO value to the AO modules. |
| pac_ReadAO | This function reads the AO value of the AO module. |
| pac_ReadAI | This function reads the engineering-mode AI value of the AI module. |
| pac_ReadAIHex | This function reads the 2's complement-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_ReadCNT | This function reads the counter values of the counter/frequency modules. |

| | |
|---|---|
| pac_ClearCNT | This function clears the counter values of the counter/frequency modules. |
| pac_ReadCNTOverflow | This function clears the counter overflow value of the counter/frequency modules. |

# 7.1. pac_GetBit

The function can retrieve the value which in specific bit.

## Syntax

**C++**

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

## Parameter

*v*

Which IO result wants to get bit.

*ndx*

Specific bit to retrieve.

## Return Values

The value of specific index.

## Examples

### [VC/VS]

```
BYTE bit3;
BYTE iSlot = 2;
int iDI_TotalCh = 8;
DWORD lDI_Value;
HANDLE hPort;
hPort = uart_Open("");
BOOL iRet = pac_ReadDI(hPort, iSlot,iDI_TotalCh, &lDI_Value);
bit3 = pac_GetBit(lDI_Value, 3);
uart_Close(hPort);
```

### [C#] for XP-8000 series only

```
bool bit;
int index = 3;
byte iSlot = 2;
byte iSlot = 2;
int iDI_TotalCh = 8;
uint lDI_Value = 0;
IntPtr hPort;
hPort = XPac.uart_Open("");
bool iRet = XPac.pac_ReadDI(hPort, iSlot,iDI_TotalCh, ref lDI_Value);
bit = XPac.pac_GetBit(iDI_TotalCh, index);
XPac.uart_Close(hPort);
```

## Remark

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

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

# 7.2. pac_WriteDO

This function writes the DO values to DO modules.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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.

*iDO_Value*

[in] A 8-digit hexadecimal value, where bit 0 corresponds to DO0, bit 31 corresponds to DO31, etc. When the bit is 1, it denotes that the digital output channel is on, and 0 denotes that the digital output channel is off.

## Return Values

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is remote
HANDLE hPort;
hPort = uart_Open("COM1,9600,N,8,1");
int total_channel = 8;
DWORD do_value = 4; // turn on the channel two
BOOL ret = pac_WriteDO(hPort, PAC_REMOTE_IO(1) , total_channel , do_value );
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
int total_channel = 8;
DWORD do_value = 4;     // turn on the channel two
BOOL ret = pac_WriteDO(hPort, 1 , total_channel , do_value );
uart_Close(hPort);
```

**Example 3:**

```
// If the module is 8k remote
int total_channel = 8;
DWORD do_value = 4;     // turn on the channel two
BOOL ret = pac_WriteDO(0, 1 , total_channel , do_value );
```

### [C#] for XP-8000 series only

**Example 1:**

```
// If the module is remote
IntPtr hPort; hPort = XPac.uart_Open("COM1,9600,N,8,1");
int total_channel = 8;
uint do_value = 4;      // turn on the channel two
bool ret = XPac.pac_WriteDO(hPort, XPac.PAC_REMOTE_IO(1) , total_channel , do_value );
XPac.uart_Close(hPort);
```

**Example 2:**

```
// If the module is 87k local
IntPtr hPort; hPort = XPac.uart_Open("COM1,115200,N,8,1");
int total_channel = 8;
uint do_value = 4;      // turn on the channel two
bool ret = XPac.pac_WriteDO(hPort, 1 , total_channel , do_value );
XPac.uart_Close(hPort);
```

**Example 3:**

```
// If the module is 8k local
int total_channel = 8;
uint do_value = 4;      // turn on the channel two
bool ret = XPac.pac_WriteDO(0, 1 , total_channel , do_value );
```

## Remark

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

# 7.3. 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

### C++

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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 be 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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local

HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot = 1;
int iChannel = 2;
int iDO_TotalCh = 8;
int iBitValue = 1;
BOOL ret = pac_WriteDOBit(hPort, iSlot , iChannel , miDO_TotalCh , iBitValue );
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot = 1;
int iChannel = 2;
int iDO_TotalCh = 8;
int iBitValue = 1;
BOOL ret = pac_WriteDOBit(0, iSlot , iChannel , miDO_TotalCh , iBitValue );
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot = 1;
int iChannel = 2;
int iDO_TotalCh = 8;
int iBitValue = 1;
bool ret = XPac.pac_WriteDOBit(hPort, iSlot , iChannel , miDO_TotalCh , iBitValue );
XPac.uart_Close(hPort);
```

## Remark

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

# 7.4. pac_ReadDO

This function reads the DO value of the DO module.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE slot = 1;
int total_channel = 8;
DWORD do_value;
BOOL ret = pac_ReadDO(hPort, slot , total_channel , &do_value );
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE slot = 1;
int total_channel = 8;
DWORD do_value;
BOOL ret = pac_ReadDO(0, slot , total_channel , &do_value );
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte slot = 1;
int total_channel = 8;
uint do_value;
bool ret = XPac.pac_ReadDO(hPort, slot , total_channel , ref do_value );
XPac.uart_Close(hPort);
```

## Remark

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

# 7.5. pac_ReadDI

This function reads the DI value of the DI module.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot = 2;
int iDI_TotalCh = 8;
DWORD lDI_Value;
BOOL iRet = pac_ReadDI(hPort, iSlot,iDI_TotalCh, &lDI_Value);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot = 2;
int iDI_TotalCh = 8;
DWORD lDI_Value;
BOOL iRet = pac_ReadDI(0, iSlot,iDI_TotalCh, &lDI_Value);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot = 2;
int iDI_TotalCh = 8;
uint lDI_Value;
bool iRet = XPac.pac_ReadDI(hPort, iSlot,iDI_TotalCh, ref lDI_Value);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.6. pac_ReadDIO

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
DWORD lDI_Value;
DWORD lDO_Value;
BOOL iRet = pac_ReadDIO(hPort, iSlot,iDI_TotalCh, iDO_TotalCh, &lDI_Value, &lDO_Value);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
DWORD lDI_Value;
DWORD lDO_Value;
BOOL iRet = pac_ReadDIO(0, iSlot,iDI_TotalCh, iDO_TotalCh, &lDI_Value, &lDO_Value);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
uint lDI_Value;
uint lDO_Value;
bool iRet = XPac.pac_ReadDIO(hPort, iSlot,iDI_TotalCh, iDO_TotalCh, ref lDI_Value, ref lDO_Value);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.7. pac_ReadDILatch

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iDI_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;
BOOL iRet = pac_ReadDILatch(hPort, iSlot, iDI_TotalCh, iLatchType,
&lDI_Latch_Value);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iDI_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;
BOOL iRet = pac_ReadDILatch(0, iSlot, iDI_TotalCh, iLatchType,
&lDI_Latch_Value);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot=1;
int iDI_TotalCh=8;
int iLatchType=0;
uint lDI_Latch_Value;
bool iRet = XPac.pac_ReadDILatch(hPort, iSlot, iDI_TotalCh, iLatchType, ref lDI_Latch_Value);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.8. pac_ClearDILatch

This function clears the latch value of the DI module.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open(“COM1,115200,N,8,1”);
BYTE iSlot=1;
BOOL iRet = pac_ClearDILatch(hPort, iSlot);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
BOOL iRet = pac_ClearDILatch(0, iSlot);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
bool iRet = pac_ClearDILatch(hPort, iSlot);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.9. pac_ReadDIOLatch

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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.

1 = latched high status

0 = latched low status

*lDI_Latch_Value*

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

*lDO_Latch_Value*

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

## Return Values

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;
DWORD lDO_Latch_Value;
BYTE cDI_Latch_BitValue;
BYTE cDO_Latch_BitValue;
BOOL iRet = pac_ReadDIOLatch(hPort, iSlot,iDI_TotalCh,iDO_TotalCh,iLatchType,
&lDI_Latch_Value,&lDO_Latch_Value);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;
DWORD lDO_Latch_Value;
BYTE cDI_Latch_BitValue;
BYTE cDO_Latch_BitValue;
BOOL iRet = pac_ReadDIOLatch(0, iSlot,iDI_TotalCh,iDO_TotalCh,iLatchType,
&lDI_Latch_Value,&lDO_Latch_Value);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
int iLatchType=0;
uint lDI_Latch_Value;
uint lDO_Latch_Value;
byte cDI_Latch_BitValue;
byte cDO_Latch_BitValue;
bool iRet = XPac.pac_ReadDIOLatch(hPort,
iSlot,iDI_TotalCh,iDO_TotalCh,iLatchType, ref lDI_Latch_Value, ref
lDO_Latch_Value);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.10. pac_ClearDIOLatch

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
BOOL iRet = pac_ClearDIOLatch(hPort, iSlot);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
BOOL iRet = pac_ClearDIOLatch(0, iSlot);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot=1;
bool iRet = XPac.pac_ClearDIOLatch(hPort, iSlot);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.11. pac_ReadDICNT

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel =2;
int iDI_TotalCh=8;
DWORD lCounter_Value;
BOOL iRet = pac_ReadDICNT(hPort, iSlot,iChannel,iDI_TotalCh,
&lCounter_Value);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel =2;
int iDI_TotalCh=8;
DWORD lCounter_Value;
BOOL iRet = pac_ReadDICNT(0, iSlot,iChannel,iDI_TotalCh, &lCounter_Value);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot=1;
int iChannel =2;
int iDI_TotalCh=8;
uint lCounter_Value;
bool iRet = XPac.pac_ReadDICNT(hPort, iSlot,iChannel,iDI_TotalCh, ref lCounter_Value);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.12. pac_ClearDICNT

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=2;
int iDI_TotalCh=8;
BOOL iRet = pac_ClearDICNT(hPort, iSlot,iChannel,iDI_TotalCh);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=2;
int iDI_TotalCh=8;
BOOL iRet = pac_ClearDICNT(0, iSlot,iChannel,iDI_TotalCh);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iChannel=2;
int iDI_TotalCh=8;
bool iRet = XPac.pac_ClearDICNT(hPort, iSlot,iChannel,iDI_TotalCh);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.13. pac_WriteAO

This function writes the AO value to the AO modules.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue=5;
BOOL iRet = pac_WriteAO(hPort, iSlot,iChannel,iAO_TotalCh,fValue);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue=5;
BOOL iRet = pac_WriteAO(0, iSlot,iChannel,iAO_TotalCh,fValue);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue=5;
bool iRet = XPac.pac_WriteAO(hPort, iSlot,iChannel,iAO_TotalCh,fValue);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.14. pac_ReadAO

This function reads the AO value of the AO module.

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue;
BOOL iRet = pac_ReadAO(hPort, iSlot,iChannel,iAO_TotalCh, &fValue);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue;
BOOL iRet = pac_ReadAO(0, iSlot,iChannel,iAO_TotalCh, &fValue);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue;
bool iRet = XPac.pac_ReadAO(hPort, iSlot,iChannel,iAO_TotalCh,ref fValue);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.15. pac_ReadAI

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=2;
int iAI_TotalCh=8;
float fValue;
BOOL iRet = pac_ReadAI(hPort, iSlot,iChannel,iAI_TotalCh, &fValue);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=2;
int iAI_TotalCh=8;
float fValue;
BOOL iRet = pac_ReadAI(0, iSlot,iChannel,iAI_TotalCh, &fValue);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iChannel=2;
int iAI_TotalCh=8;
float fValue;
bool iRet = XPac.pac_ReadAI(hPort, iSlot,iChannel,iAI_TotalCh, ref fValue);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.16. pac_ReadAIHex

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=2;
int iAI_TotalCh=8;
int iValue;
BOOL iRet = pac_ReadAIHex(hPort, iSlot,iChannel,iAI_TotalCh, &iValue);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=2;
int iAI_TotalCh=8;
int iValue;
BOOL iRet = pac_ReadAIHex(0, iSlot,iChannel,iAI_TotalCh, &iValue);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iChannel=2;
int iAI_TotalCh=8;
int iValue;
bool iRet = XPac.pac_ReadAIHex(hPort, iSlot,iChannel,iAI_TotalCh, ref iValue);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.17. pac_ReadAIAll

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
float fValue[8];
BOOL iRet = pac_ReadAIAll(hPort, iSlot, fValue);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
float fValue[8];
BOOL iRet = pac_ReadAIAll(0, iSlot, fValue);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot=1;
float fValue[8];
bool iRet = pac_ReadAIAll(hPort, iSlot, fValue);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.18. pac_ReadAIAllHex

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open(“COM1,115200,N,8,1”);
BYTE iSlot=1;
int iValue[8];
BOOL iRet = pac_ReadAIAllHex(hPort, iSlot, iValue);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iValue[8];
BOOL iRet = pac_ReadAIAllHex(0, iSlot, iValue);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iValue[8];
bool iRet = XPac.pac_ReadAIAllHex(hPort, iSlot, iValue);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.19. pac_ReadCNT

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=0;
DWORD lCounter_Value;
BOOL iRet = pac_ReadCNT(hPort, iSlot,iChannel,&lCounter_Value);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=0;
DWORD lCounter_Value;
BOOL iRet = pac_ReadCNT(0, iSlot,iChannel,&lCounter_Value);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iChannel=0;
uint lCounter_Value;
bool iRet = XPac.pac_ReadCNT(hPort, iSlot,iChannel,ref lCounter_Value);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.20. pac_ClearCNT

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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 modules.

## Return Values

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=0;
BOOL iRet = pac_ClearCNT(hPort, iSlot, iChannel);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=0;
BOOL iRet = pac_ClearCNT(0, iSlot, iChannel);
```

### [C#] for XP-8000 series only

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open("COM1,115200,N,8,1");
byte iSlot=1;
int iChannel=0;
bool iRet = XPac.pac_ClearCNT(hPort, iSlot, iChannel);
XPac.uart_Close(hPort);
```

## Remark

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

# 7.21. pac_ReadCNTOverflow

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

## Syntax

**C++**

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

## Parameter

*hPort*

[in] The serial port HANDLE opened by uart_Open(), if the module is 87k modules in local.

0, if the module is 8k modules in local.

*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

TRUE indicates success. FALSE indicates failure.

## Examples

### [VC/VS]

**Example 1:**

```
// If the module is 87k local
HANDLE hPort;
hPort = uart_Open("COM1,115200,N,8,1");
BYTE iSlot=1;
int iChannel=0;
int iOverflow;
BOOL iRet = pac_ReadCNT_Overflow(hPort, iSlot,iChannel,&iOverflow);
uart_Close(hPort);
```

**Example 2:**

```
// If the module is 8k local
BYTE iSlot=1;
int iChannel=0;
int iOverflow;
BOOL iRet = pac_ReadCNT_Overflow(0, iSlot,iChannel,&iOverflow);
```

**[C#] for XP-8000 series only**

```
// If the module is 87k local
IntPtr hPort;
hPort = XPac.uart_Open(“COM1,115200,N,8,1”);
byte iSlot=1;
int iChannel=0;
int iOverflow;
bool iRet = XPac.pac_ReadCNT_Overflow(hPort, iSlot,iChannel,ref iOverflow);
XPac.uart_Close(hPort);
```

## Remark

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

# 8. Error Handling API

## Supported Modules

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

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetLastError | Y | Y | Y | Y |
| pac_SetLastError | Y | Y | Y | Y |
| pac_GetErrorMessage | Y | Y | Y | Y |
| pac_ClearLastError | Y | Y | Y | Y |

## Function List

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

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

# 8.1. pac_GetLastError

This function returns the last-error code value.

## Syntax

### C++

```
DWORD pac_GetLastError();
```

## Parameter

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

## Remark

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.

For an example, please refer to pac_GetErrorMerrage in this document.

To obtain an error string for XPac error codes, use the pac_GetErrorMessage function. For a complete list of error codes, see Error Values or the SDK header file XPacSDK.h for XP-8000 series or XPacSDK_CE.h for XPAC_CE

| Error Type | Explanation | Reange |
|---|---|---|
| PAC_ERR_SUCCESS | No error, success | 0x00000 |
| PAC_ERR_UNKNOWN | Error which is undeifiend | 0x00001 |
| Basic | Defined comman error conditions | 0x10000 ~ |
| Memory | About memory access | 0x11000 ~ |
| WatchDog | About watchdog | 0x12000 ~ |
| Interrupt | About interrupt | 0x13000 ~ |
| UART | About uart protocol | 0x14000 ~ |
| IO | About IO modules | 0x15000 ~ |
| Users | For user | 0x20000 ~ |

# 8.2. pac_SetLastError

This function sets the last-error code.

## Syntax

**C++**

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

## Parameter

*errno*

[in] Specifies the last-error code.

## Return Values

None

## Examples

None

## Remark

Applications can optionally retrieve the value set by this function by using the pac_GetLastError function.

Error codes are DWORD values. If you are defiening an error code for your application, ensure that your error code does not conflict with any XPacSDK-defined error codes.

We recommend that your error code should be greater than 0x20000.

For more information about the definition of error codes, please refer to pac_GetLastError in this document.

# 8.3. pac_GetErrorMessage

This function retrieves a message string.

## Syntax

**C++**

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

## Parameter

*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

## Example

### [VC/VS]

```
int main(int argc, char* argv[])
{
   if(argc < 3)
   {
         printf("usage: ReadMemory [ address ] [ dwLength ] [ mem_type ]\n\n");
         printf("where\n");
         printf("     address:\n");
         printf("          - the memory address where read from.\n");
         printf("     dwLength:\n");
         printf("          - number of characters to be read.\n");
         printf("     mem_type:\n");
         printf("          - 0     SRAM\n");
         printf("          - 1     EEPROM\n");
   }
   else
   {
         BYTE buffer[4096];
         BOOL err;
         char strErr[32];
         memset(buffer, 0, 4096);
         if(atoi(argv[3]) == 0)
         {
               printf("The size of SRAM is %d\n", pac_GetMemorySize(atoi(argv[3])));
               err = pac_ReadMemory(atoi(argv[1]), buffer, atoi(argv[2]), atoi(argv[3]));
               if(err == FALSE)
               {
                     pac_GetErrorMessage(pac_GetLastError(), strErr);
                     printf("Read SRAM failure!. The error code is %x\n",
pac_GetLastError());
                     printf("%s", strErr);
                     return 0;
               }
```

```
            printf("%s\n", buffer);
        }
        else
        {
            printf("The size of EEPROM is %d\n",
pac_GetMemorySize(atoi(argv[3])));
            err = pac_ReadMemory(atoi(argv[1]), buffer, atoi(argv[2]), atoi(argv[3]));
            if(err == FALSE)
            {
                pac_GetErrorMessage(pac_GetLastError(), strErr);
                printf("Read EEPROM failure!. The error code is %x\n",
pac_GetLastError());
                printf("%s", strErr);
                return 0;
            }
            printf("%s\n", buffer);
        }
    }
    return 0;
}
```

## [C#] for XP-8000 series only

```
class Program
{
static void Main(string[] args)
    {
        if (args.Length < 3)
        {
            Console.WriteLine("pac_WriteDO for 8000 modules\n\n");
                Console.WriteLine("usage: pac_WriteDO [ Slot ] [ total channel ]
[ DO's value ]\n\n");
            Console.WriteLine("where\n");
            Console.WriteLine("Slot:\n");
            Console.WriteLine(" - number of slot for local modules\n");
            Console.WriteLine("total channel:\n");
            Console.WriteLine(" - number of DO's channel\n");
            Console.WriteLine("DO's value:\n");
            Console.WriteLine(" - 1 is to turn on the DO channel; 0 is off.\n");
        }
        else
        {
            bool err;
            err = XPac.pac_WriteDO(IntPtr.Zero,Convert.ToInt32(args[1]),
                            Convert.ToInt32(args[2]), Convert.ToUInt32(args[3]));
            if (err == false)
            {
                Cnsole.WriteLine("Write DO's Error: " +
XPac.pac_GetErrorMessage(XPac.pac_GetLastError()) + ". The error code is " +
   XPac.pac_GetLastError().ToString() + "\n");
                return;
            }
        }
   }
}
```

## Remark

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

# 8.4. pac_ClearLastError

This function clears the last-error code.

## Syntax

**C++**

```
void pac_ClearLastError();
```

## Parameter

None

## Return Values

None

## Examples

None

## Remark

The pac_ClearLastError function clears the last error, that is, the application is treated as success.

# Appendix A. MISC API

## Supported Modules

The following shows the overview of the Misc functions which are available with XPAC.

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| AnsiString | Y | X | Y | X |
| WideString | Y | X | Y | X |
| pac_AnsiToWideString | X | Y | X | Y |
| pac_WideToAnsiString | X | Y | X | Y |
| pac_DoEvents | Y | Y | Y | Y |

## Function List

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

| Function | Description |
|---|---|
| AnsiString | AnsiString |
| WideString | WideString |
| pac_AnsiToWideString | This function can convert conver ansi string to Unicode string. |
| pac_WideToAnsiString | This function can convert unicode string to ansi string. |
| pac_DoEvents | This function returns control to your system. |

# A1. AnsiString (for XP-8000 series only)

This function can convert unicode string to ansi byte array.

## Syntax

**C#**

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

## Parameter

*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 = XPac.uart_Open("COM1,115200,N,8,1");
XPac.pac_ChangeSlot(Convert.ToByte(1));
XPac.uart_SendCmd(hPort, XPac.AnsiString("$00M"), result);
string str = XPac.WideString(result);
```

## Remark

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

# A2. WideString (for XP-8000 series only)

This function can convert ansi byte array to unicode string.

## Syntax

**C#**

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

## Parameter

*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 = XPac.uart_Open("COM1,115200,N,8,1");
XPac.pac_ChangeSlot(Convert.ToByte(1));
XPac.uart_SendCmd(hPort, XPac.AnsiString("$00M"), result);
string str = XPac.WideString(result);
```

## Remark

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

# A3. pac_AnsiToWideString (for XPAC_CE series only)

This function can convert conver ansi string to Unicode string.

## Syntax

**C++**

```
void pac_AnsiToWideString(
        LPCSTR astr,
        LPTSTR wstr
);
```

## Parameter

*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

**[VS]**

```
char astr[128] = “This is an ansi string”
TCHAR wstr[128];
pac_AnsiToWideString(astr, wstr);
```

# A4. pac_WideToAnsiString (for XPAC_CE series only)

This function can convert unicode string to ansi string.

## Syntax

### C++

```
void pac_WideToAnsiString(
        LPCTSTR    wstr,
        LPSTR astr
);
```

## Parameter

*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

### [VS]

```
TCHAR wstr[128] = TEXT("This is a Unicode string");
char astr[128];
pac_WideToAnsiString(wstr, astr);
```

# A5. pac_DoEvents

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.

This function returns control to your system.

## Syntax

### C++

```
void pac_DoEvents();
```

## Parameter

None

## Return Values

None

## Examples

### [VC/VS]

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

# Appendix B. 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 |
| 0x11001 | EEPROM accesses invalid address |
| 0x11002 | SRAM accesses invalid address |
| 0x11003 | SRAM accesses invalid type |
| 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 |

# Appendix C. API Comparison

The following tables give a brief summary of the capabilities of each API function, where Y means supported and X means unsupported

## System Information API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetModuleName | Y | Y | Y | Y |
| pac_GetRotaryID | Y | Y | Y | Y |
| pac_GetSerialNumber | Y | Y | Y | Y |
| pac_GetSDKVersion | Y | Y | Y | Y |
| pac_ChangeSlot | Y | Y | Y | Y |
| pac_CheckSDKVersion | Y | Y | Y | Y |
| pac_ModuleExists | Y | Y | Y | Y |
| pac_GetOSVersion | X | Y | X | Y |
| pac_GetMacAddress | X | Y | X | Y |
| pac_ReBoot | X | Y | X | Y |
| Pac_GetCPUVersion | Y | Y | Y | Y |
| Pac_EnableLED | X | X | Y | Y |

## Backplane Access API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetDIPSwitch | Y | Y | Y | Y |
| pac_GetSlotCount | Y | Y | Y | Y |
| pac_EnableRetrigger | Y | Y | Y | Y |
| pac_GetBackplaneID | Y | Y | Y | Y |
| pac_GetBatteryLevel | Y | Y | Y | Y |

## Interrupt API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_RegisterSlotInterrupt | Y | Y | Y | Y |
| pac_UnregisterSlotInterrupt | Y | Y | Y | Y |
| pac_EnableSlotInterrupt | Y | Y | Y | Y |
| pac_SetSlotInterruptPriority | Y | Y | Y | Y |
| pac_InterruptInitialize | Y | Y | Y | Y |
| pac_GetSlotInterruptEvent | Y | Y | Y | Y |
| pac_SetSlotInterruptEvent | Y | Y | Y | Y |
| pac_SetTriggerType | Y | Y | Y | Y |
| pac_GetSlotInterruptID | Y | Y | Y | Y |
| pac_InterruptDone | Y | Y | Y | Y |

## Memory Access API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetMemorySize | Y | Y | Y | Y |
| pac_ReadMemory | Y | Y | Y | Y |
| pac_WriteMemory | Y | Y | Y | Y |
| pac_EnableEEPROM | Y | Y | Y | Y |

## Watchdog API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_EnableWatchDog | Y | Y | Y | Y |
| pac_DisableWatchDog | Y | Y | Y | Y |
| pac_RefreshWatchDog | Y | Y | Y | Y |
| pac_GetWatchDogState | Y | Y | Y | Y |
| pac_GetWatchDogTime | Y | Y | Y | Y |
| pac_SetWatchDogTime | Y | Y | Y | Y |

## Uart API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| uart_Open | Y | Y | Y | Y |
| uart_Close | Y | Y | Y | Y |
| uart_Send | Y | Y | Y | Y |
| uart_Recv | Y | Y | Y | Y |
| uart_SendCmd | Y | Y | Y | Y |
| uart_SetTimeOut | Y | Y | Y | Y |
| uart_EnableCheckSum | Y | Y | Y | Y |
| uart_SetTerminator | Y | Y | Y | Y |
| Uart_BinSend | Y | Y | Y | Y |
| Uart_BinRecv | Y | Y | Y | Y |
| Uart_BinSendCmd | Y | Y | Y | Y |
| Uart_GetLineStatus | Y | Y | Y | Y |
| Uart_GetDataSize | Y | Y | Y | Y |

## PAC_IO API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetBit | Y | Y | Y | Y |
| pac_WriteDO | Y | Y | Y | Y |
| pac_WriteDOBit | Y | Y | Y | Y |
| pac_ReadDO | Y | Y | Y | Y |
| pac_ReadDI | Y | Y | Y | Y |
| pac_ReadDIO | Y | Y | Y | Y |
| pac_ReadDILatch | Y | Y | Y | Y |
| pac_ClearDILatch | Y | Y | Y | Y |
| pac_ReadDIOLatch | Y | Y | Y | Y |
| pac_ClearDIOLatch | Y | Y | Y | Y |
| pac_ReadDICNT | Y | Y | Y | Y |
| pac_ClearDICNT | Y | Y | Y | Y |
| pac_WriteAO | Y | Y | Y | Y |
| pac_ReadAO | Y | Y | Y | Y |
| pac_ReadAI | Y | Y | Y | Y |
| pac_ReadAIHex | Y | Y | Y | Y |
| pac_ReadAIAll | Y | Y | Y | Y |
| pac_ReadAIAllHex | Y | Y | Y | Y |
| pac_ReadCNT | Y | Y | Y | Y |
| pac_ClearCNT | Y | Y | Y | Y |
| pac_ReadCNTOverflow | Y | Y | Y | Y |

## Error Handling API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| pac_GetLastError | Y | Y | Y | Y |
| pac_SetLastError | Y | Y | Y | Y |
| pac_GetErrorMessage | Y | Y | Y | Y |
| pac_ClearLastError | Y | Y | Y | Y |

## MISC API

| Functions\Models | XP-8000 (WES) | XP-8000-CE6 | XP-8000-Atom (WES) | XP-8000-Atom -CE6 |
|---|---|---|---|---|
| AnsiString | Y | X | Y | X |
| WideString | Y | X | Y | X |
| pac_AnsiToWideString | X | Y | X | Y |
| pac_WideToAnsiString | X | Y | X | Y |
| pac_DoEvents | Y | Y | Y | Y |