USER'S MANUAL

Protech
API Package

Ver. M2   Date: 2012/04/06

Protech API Package
User's Manual

Preface

This manual explains how to operate and configure Protech API Package. No part of this publication may be reproduced or transmit in any form, or by any means, electronic, or mechanical, including photocopying and recording, without written permission of Protech Systems Co., Ltd. The information contained in this document is subject to change without prior notice. Protech does not warrant that the document or information is error-free. If you find any problems in the documentation, please report them to us in writing.

The software contains proprietary information of Protech Systems Co., Ltd.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Protech shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this software and accompanying documentation.

Protech reserves the right to make changes to any product or software to improve reliability, function or design. For version updates or technical support, please contact your local sales representative.

Protech Systems Co., Ltd.
No. 24, Lane 365, Yang Goang Street,
Nei Hu District, Taipei 114, Taiwan, R. O. C.
Tel: +886-2-8751-1111
Fax: +886-2-8751-1199

http://www.protech.com.tw

Introduction

Thank you for using Protech API Package.

The API solution provided by Protech Systems is a benefit to users to control the device with ease without having to analyze the hardware. It means that the time-wasting issues happened in general program development process, including trouble dealing with a diversity of hardware systems and catching on individual hardware specifications, control methods and communi- cation protocols in practical applications, and the like can be resolved with Protech API Package.

Feature

The API solution provided by Protech Systems is a benefit to users for the following reasons:

- **Speed up product release date:**
  The API package helps developers design programs without being familiar with the chipset specifications and driver architecture.
- **Reduce workload on programming development items:**
  Users can control the device by Protech API package directly – save time to write the hardware drivers from zero.

Environment

- Windows 32 bit OS + .NET Framework version 2.0 or above

Applicable Field

- Industrial CPU Board
- POS PC
- Applied Computer
- Panel PC

Supported Function

- Programmable GPIO
- Digital IO
- Watch Dog
- Cash Drawer
- Hardware Monitor
- i-Button
- UPS

Table of Contents

Chapter 1 Getting Started

1

In this Chapter, you will have a brief on the API Package functions and content, and be ready to use the API interface.

Sections included:

Section 1 API Package Content

Users can find the enclosed API Package files inside the Protech Manual / Driver CD. Depending on machine types, the API Package files may include the following:

Operation System	Windows 32 bit + .NET Framework 2.0 or above		
Directory	**Contents / File Name**		**Description**
Document	Protech API Package User Guide A01-0000-000-02-xxxxxx_en.pdf		User Manual in English
	Protech API Package User Guide A01-0000-000-02-xxxxxx_ch.pdf		User Manual in Chinese
	IO Description.pdf		---
	UPS Standard SBS Commands.pdf		---
Function DLL			
Directory	**Function**	**File Name**	**Description**
ProxAPI standard	*Cash Drawer*	Cash Drawer.dll	Driver to control Cash Drawer
	Digital	Digital.dll	Driver to control Digital IO
	GPIO	GPIO.dll WinIo.dll WinIo.lib WinIo.sys WINIO.VXD	Driver to control GPIO
	SMBUS	WinIo.dll WinIo.lib WinIo.sys WINIO.VXD SMBUS.dll	Driver to use SMBUS
	WDT	Watchdog.dll	Driver to control Watchdog
	i-Button	IButtonAPI.dll IBFS32.dll	Driver to get i-Button
	Hardware Monitor	Hardware Monitor.dll	Driver to read hardware data
	Battery	SBS_Battery.dll phymem.sys pmdll.dll	Driver to read and control battery data
	multilangXML.dll		Driver to open XML file
	Initial.xml		XML file to initiate the API Package
	ProxAP.exe		API program executable file
	XML Files\Model Name*\Initial.xml		XML file for each model
	Version.ini		Version information

🕮 Model Name is dependent on your machine type.

(continued)

Sample Program		
Directory	**Contents / File Name**	**Description**
DEMO PROJECT	DEMO PROJECT\GPIO Sample Code	C# VB6 VB.net Source Code
	DEMO PROJECT\Digital Sample Code	C# VB6 VB.net Source Code
	DEMO PROJECT\Watchdog Sample Code	C# VB6 VB.net MFC Source Code

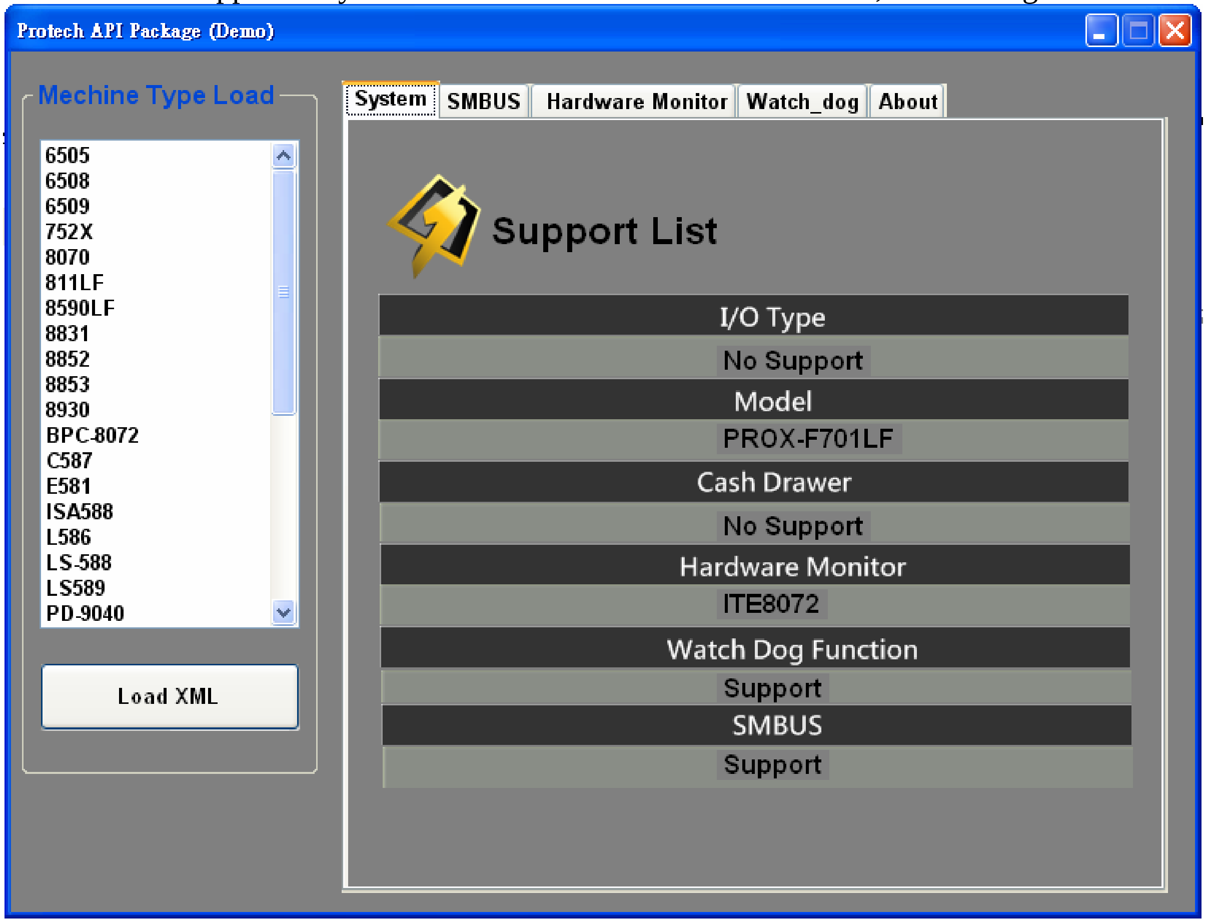
Section 2 Open API Package Program

An XML file must be included in the API Package for the API program to be executed normally. Take PS6509 for example, users will need the following files to run API Package:
- ProxAPI standard\Cash Drawer.dll
- ProxAPI standard\multilangXML.dll
- ProxAPI standard\Watch dog.dll
- ProxAPI standard\Hardware Monitor.dll
- ProxAPI standard\XML Files\6509\Initial.xml
- ProxAPI standard\ProxAP.exe

🕮 When developing the program, make sure all necessary files are present in your working directory, including the function DLLs, multilangXML.dll, and Initial.xml.

After executing the API program (ProxAP.exe), the program will display the related tabs based on the machine type selected. That is, on the System tab, select your product model name from the "Machine Type Load" list on the left pane, and then tap **[Load XML]** to get the supported functions displayed in the Support List as shown below:

API functions supported by PROX-F701LF are - Hardware Monitor, Watch Dog and SMBUS.



I/O Type
No Support
Model
PROX-F701LF
Cash Drawer
No Support
Hardware Monitor
ITE8072
Watch Dog Function
Support
SMBUS
Support

Chapter 2 Using API

2

In this Chapter, your will learn how to use the API procedure in several programming languages.

Sections included:

Section 1 API Procedure

Take **VB2005 .NET** for example, first you must declare a function. You may create a module in your project and fill in the function, cash drawer for example.

```
Declare Function GetCashDrawerStatus Lib CashDrawer.dll (ByVal num_drawer as short)
As Boolean

Declare Function CashDrawerOpen Lib CashDrawer.dll (ByVal num_drawer as short) As
Boolean
```

Next, create a button to call API Function

1. Call Cash drawer open event:

```
Private Sub cash_btn1_Click (ByVal Sender As System.Object, ByVal e As
System.EventArgs) Handles cash_btn1.Click
CashDrawerOpen(1), “1” specifies the cash drawer 1 port
CashDrawerOpen(2), “2” specifies the cash drawer 2 port
Timer1.start
```

2. Detect Cash drawer status:
   A timer event can be created.

```
Private Sub Timer1_Tick (ByVal Sender As System.Object,ByVal e As System.EventArgs)
Handles Timer1.Tick
    Dim Receive_Status1 as Boolean
    Dim Receive_Status2 as Boolean
    Receive_Status1 = CashDrawerOpen(&H1)
    If Receive_Status1 = true then
       Text1.text = “cash drawer1 open”    ‘enter text into textbox.
Else
       Text1.text = “cash drawer1 close”    ‘enter text into textbox.
End if
‘=========================================
    Receive_Status2 = CashDrawerOpen(&H2)
    If Receive_Status2 = true then
       Text2.text = “cash drawer2 open”    ‘enter text into textbox.
Else
       Text2.text = “cash drawer2 close”    ‘enter text into textbox.
End if
‘=========================================
End sub
```

Section 2 Sample Code

(1) VB Declaration

```
Declare Function GetCashDrawerStatus Lib CashDrawer.dll (ByVal num_drawer as short) As Boolean

Declare Function CashDrawerOpen Lib CashDrawer.dll (ByVal num_drawer as short) As Boolean
```

(2) Call Function

Open cash drawer:

```
CashDrawerOpen(1)
```

Open cash drawer1

```
CashDrawerOpen(2)
```

Open cash drawer2

Check cash drawer status:

```
Dim receive_status as Boolean
```

Check cash drawer1 status

```
Receive_Status = CashDrawerOpen(&H1)
```

Check cash drawer2 status

```
Receive_Status = CashDrawerOpen(&H2)
```

----

(1) C# Declaration Method

```
Public class PortAccess
{
[DllImport("CashDrawer.dll",EntryPoint = "Initial_CashDrawer")]
Public static extern void Initial_CashDrawer();
[DllImport("CashDrawer.dll",EntryPoint= "GetCashDrawerStatus")]
Public static extern bool GetCashDrawerStatus()
[DllImport("CashDrawer.dll",EntryPoint = "CashDrawerOpen")]
Public static extern bool CashDrawerOpen(short num_drawer);
}
```

(2) Call Function

Open cash drawer1

```
PortAccess.CashDrawerOpen(0x01);          //check cash drawer1 status
```

Open cash drawer2

```
PortAccess.CashDrawerOpen(0x02);          //check cash drawer2 status

Bool bstatus;
bstatus = PortAccess.GetCashDrawerStatus(0x01);
bstatus = PortAccess.GetCashDrawerStatus(0x02); //Before get cash drawer status, need to initial cash drawer first
```

VB.NET extern function:

```
Declare Function SetMinSec Lib “WatchDog.dll” (ByVal kind As Short,ByVal
delay_time As Short) As Boolean
Declare Function Stopwatchdog Lib “WatchDog.dll” ( ) As Short
Declare Function Setwatchdog Lib “WatchDog.dll” (ByVal value As Short) As Boolean
‘==========================================================
Declare Function Digital_Initial Lib “Digital.dll” ( ) As Long
Declare Function Digtial_Set Lib “Digital.dll”(ByVal hex_value As Short) As Long
Declare Function Digtial_Get Lib ”Digital.dll” ( ) As Short
‘==========================================================
Declare Function GPIO_Initial Lib “GPIO.dll” ( ) As Long
Declare Function GPIO_SetPort Lib “GPIO.dll”(ByVal direct As long)
Declare Function GPIO_Set Lib “GPIO.dll”(ByVal dout_value As long) As Boolean
Declare Function GPIO_Get Lib “GPIO.dll”( ) As Short
‘==========================================================
Declare Function GetCashDrawerStatus Lib CashDrawer.dll (ByVal num_drawer as short)
As Boolean
Declare Function CashDrawerOpen Lib CashDrawer.dll (ByVal num_drawer as short) As
Boolean
```

VB 6 extern function:

```
Declare Function CashDrawerOpen Lib "CashDrawer.dll" (ByVal num_drawer As
Integer) As Boolean
Declare Function GetCashDrawerStatus Lib "CashDrawer.dll" (ByVal num_drawer As
Integer) As Boolean
```

🕮 VB.net short = integer VB6

Chapter 3 API Package Program

3

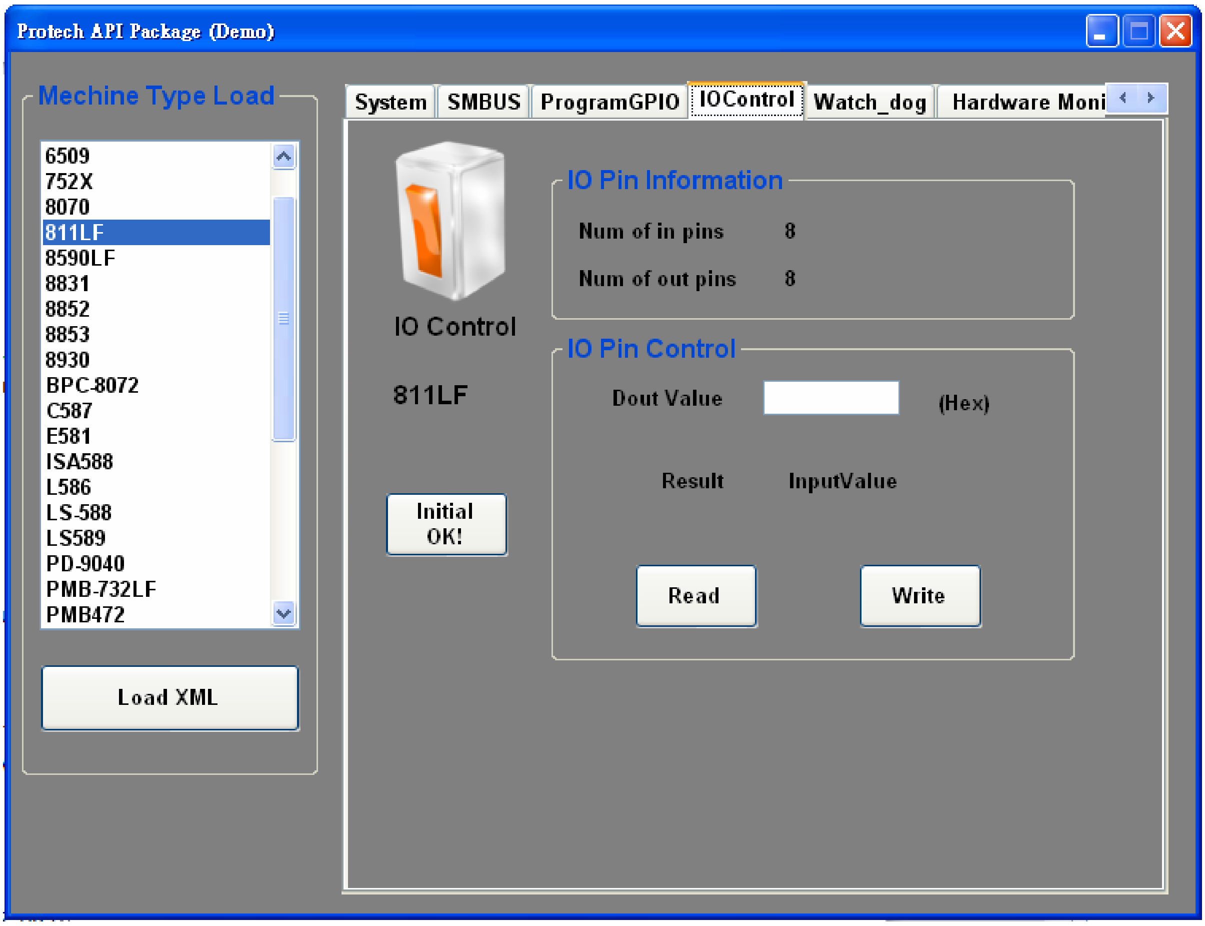
In this Chapter, you will learn to use the API Package program.

Sections included:

Section 1 IO Control

The API Package program demonstrates how to use the API Library in a user's application.

🕮 This program developed by VB.NET requires Microsoft .NET Framework version 2.0 or above.

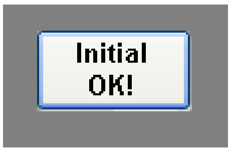


Num of in pins 8

Num of out pins 8

[Initial]

Initialize IO Function, and if successful the button will become **[Initial OK!]** as shown right.



🕮 If **[Initial OK!]** is not displayed, the execution continued may fail.

IO Pin Information

The input and output pin numbers on this machine type will be displayed.

IO Pin Control

▸ **Dout Value** Input the hex value to send to the IO Port.

Take 811LF for example, by default there are 8 output pins in total. If you want to set all the output pins as “High”, fill “0x00FF” in the **Dout Value** text field.

🕮 The “FF” indicates the 8-bit binary value (11111111) as shown below:

Bit7(IO7)	Bit6(IO6)	Bit5(IO5)	Bit4(IO4)	Bit3(IO3)	Bit2(IO2)	Bit1(IO1)	Bit0(IO0)
1	1	1	1	1	1	1	1

Likewise, if you want to set all the output pins as “Low”, fill “0x0000” in the **Dout Value** text field.

When working with a 4in/ 4out type, fill in “0F”.
(i.e. the later 4 bits indicate the IO pin positions to be controlled)

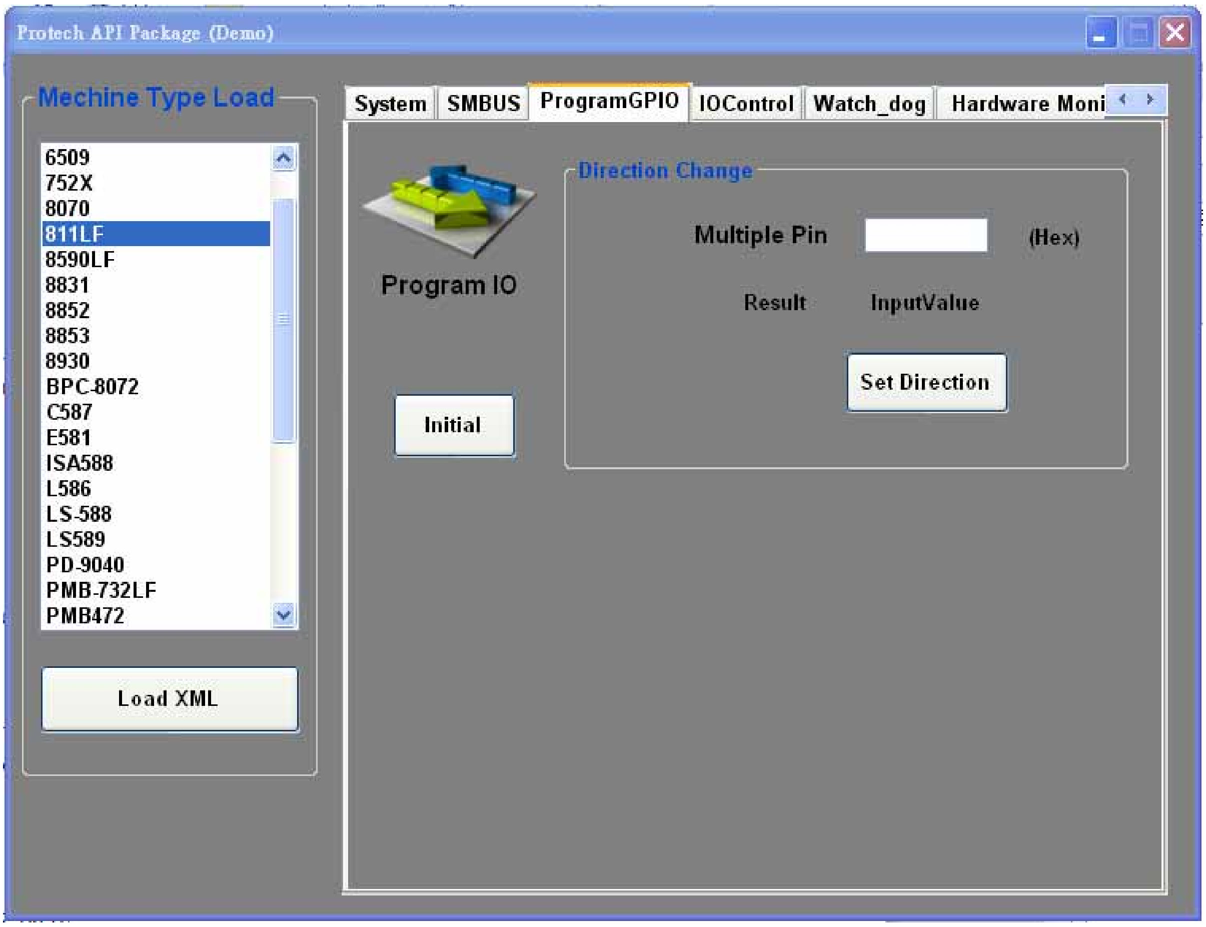
N/A	N/A	N/A	N/A	Bit3(IO3)	Bit2(IO2)	Bit1(IO1)	Bit0(IO0)
0	0	0	0	1	1	1	1

▸ **[Write]** Tap to output the value of **Dout Value** to the hardware.

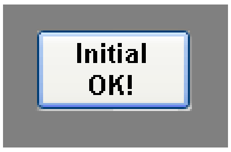
▸ **[Read]** Tap to read the input signal value and show the value to the **Result** field.

▸ **Result** The input signal value will be displayed in hex after **[Read]** is tapped.

Section 2 Program GPIO



[Initial]



Initialize IO Function, and if successful the button will become **[Initial OK!]** as shown right.
🕮 If **[Initial OK!]** is not displayed, the execution continued may fail.

Direction Change

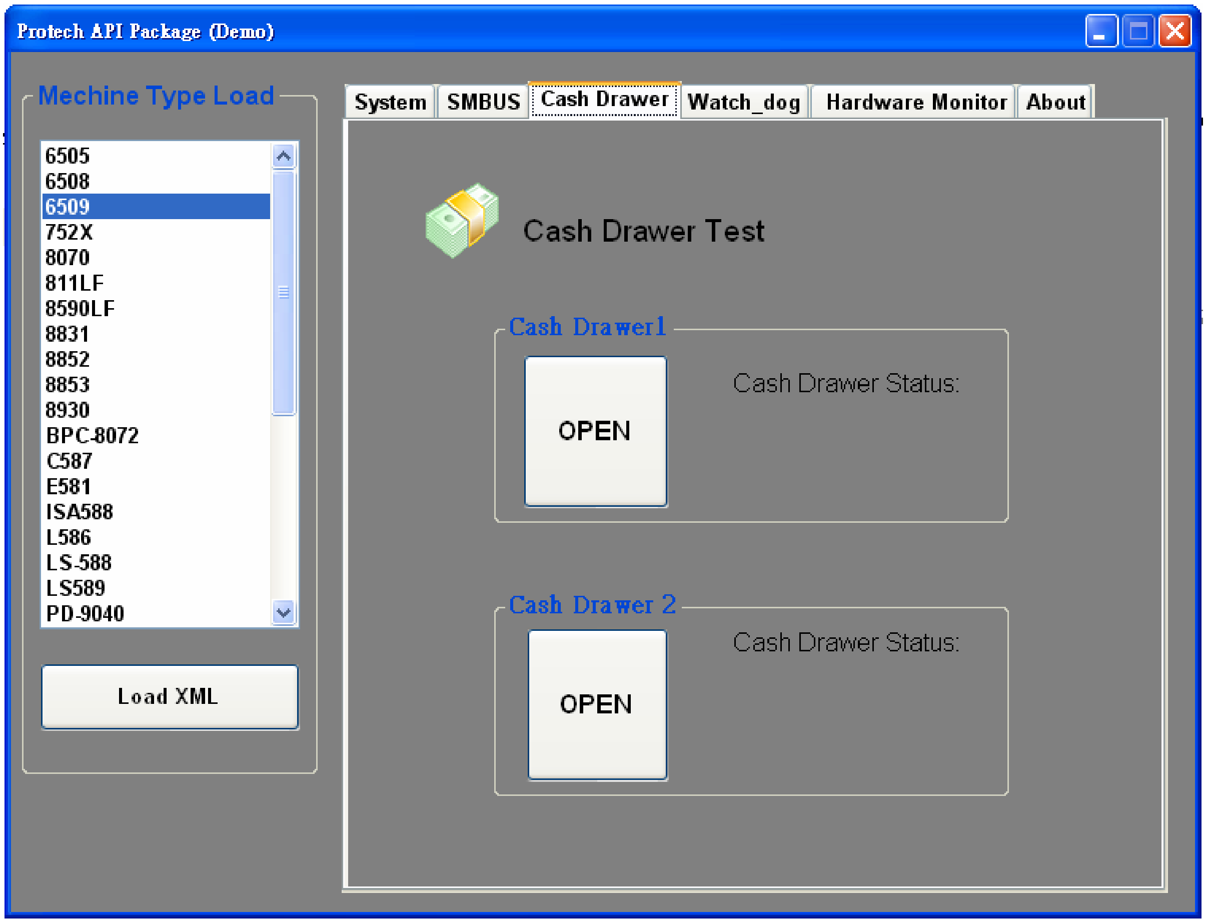
▸ **Multiple Pin** Input the hex value to control pin functions as input or output.
For Protech products, the defined output is binary 1, and the defined input is binary 0.

Take 811LF for example, by default it is 8in/ 8out type. Each pin can be configured as input or output. If you want to set all the 16 pins as output, fill “FFFF” in the **Multiple Pin** text field.
“FFFF” represents to bit16 ~ bit1 from left (MSB) to right (LSB).
🕮 To restore factory default, reset the power to the machine.

▸ **[Set Direction]** Tap to output the value of **Multiple Pin** to the system IO.

▸ **Result** The returned value, true on success or false on failure, will be displayed after **[Set Direction]** is tapped.

Section 3 Cash Drawer



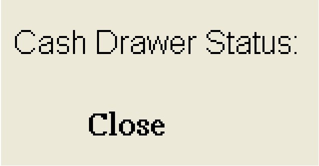
[OPEN]

Tap to open the cash drawer.

Cash Drawer Status

Cash drawer status will be displayed after **[OPEN]** is tapped.

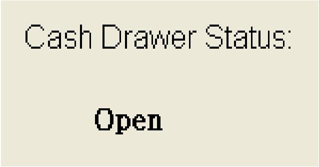
- Cash drawer is closed as shown.



Cash Drawer Status:

Close

- Cash drawer is open as shown.

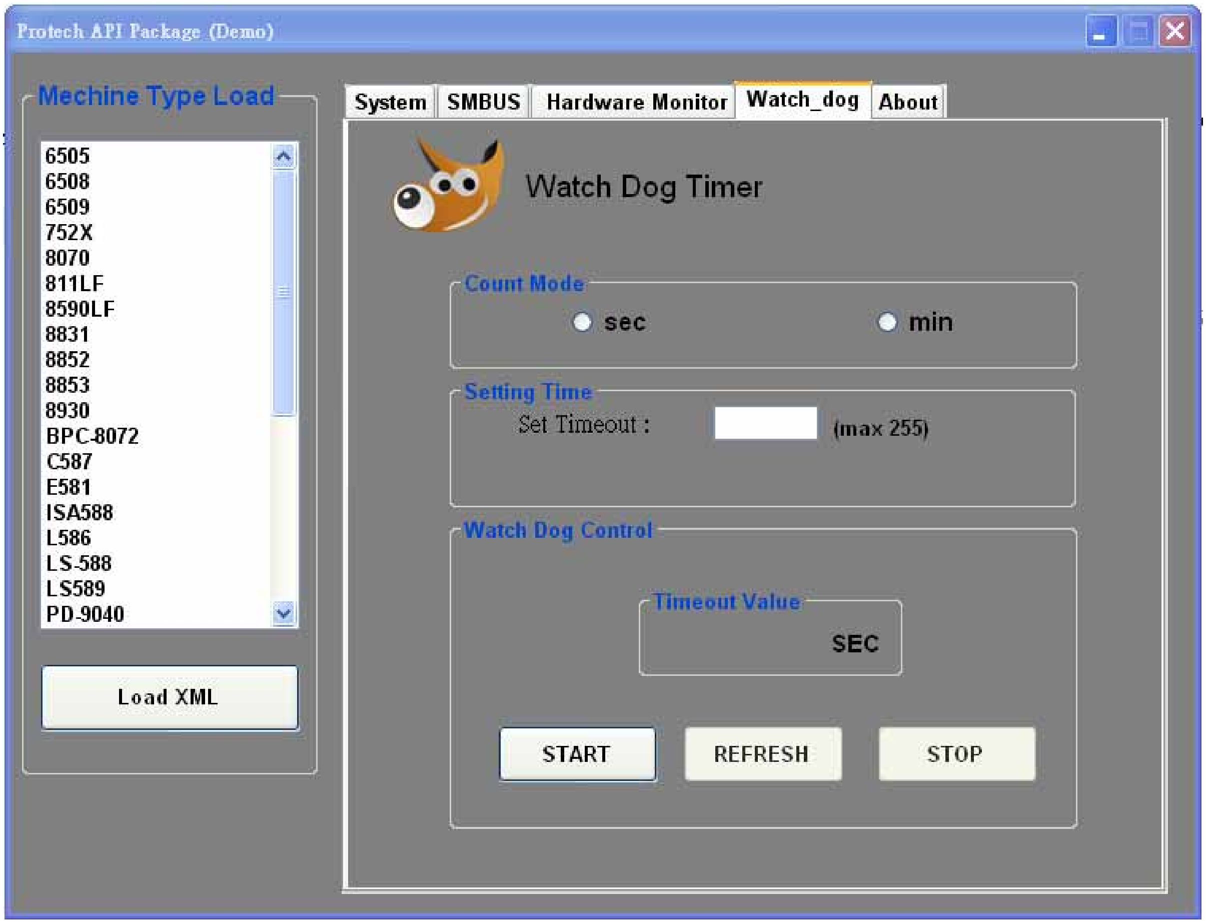


Cash Drawer Status:

Open

🕮 For example, PS6509 has two cash drawers, so the API program displays two buttons for each drawer. For a machine with single cash drawer, on the other hand, the API program displays one button, and so does to a machine that supports one cash drawer only.

Section 4 Watch Dog



Count Mode

Select the unit of time, second or minute, for the watchdog timer.

Setting Time

- **Set Timeout** Set the timeout for the watchdog. The maximum timeout value is 255 seconds or minutes.

Watch Dog Control

- **Timeout Value** Simulation timer of the API program, the running watchdog timeout will be displayed (in seconds). It is not as accurate as a hardware watchdog clock.

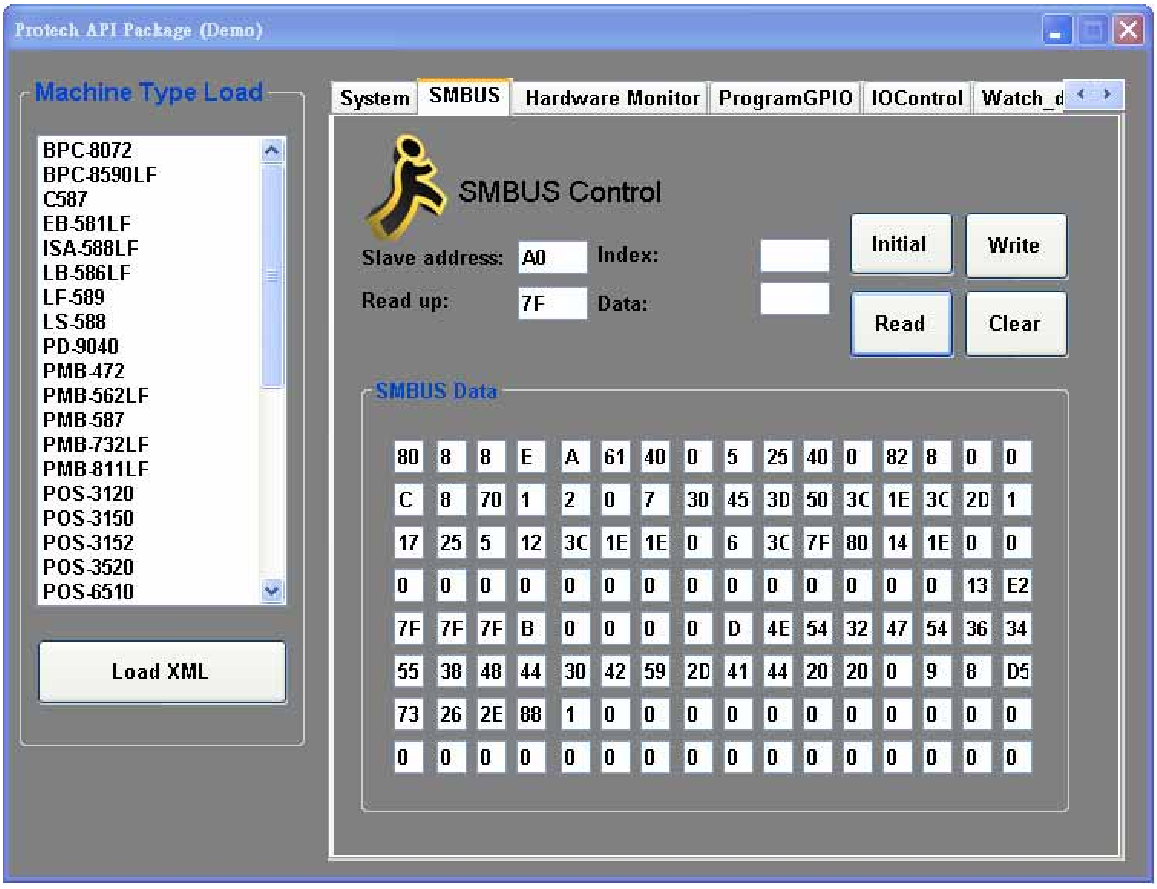
- **[START]** Tap to start the watchdog timer. Meanwhile the **[REFRESH]** and **[STOP]** buttons will be enabled.

- **[STOP]** Tap to stop the watchdog timer.

- **[REFRESH]** Tap to restart the watchdog timer.

Section 5 SMBUS

Users are able to test peripheral devices through the SMBus controller under this tab.



BPC-8072
BPC-8590LF
C587
EB-581LF
ISA-588LF
LB-586LF
LF-589
LS-588
PD-9040
PMB-472
PMB-562LF
PMB-587
PMB-732LF
PMB-811LF
POS-3120
POS-3150
POS-3152
POS-3520
POS-6510

Slave address: A0
Index:
Read up: 7F
Data:

80	8	8	E	A	61	40	0	5	25	40	0	82	8	0	0
C	8	70	1	2	0	7	30	45	3D	50	3C	1E	3C	2D	1
17	25	5	12	3C	1E	1E	0	6	3C	7F	80	14	1E	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	E2
7F	7F	7F	B	0	0	0	0	D	4E	54	32	47	54	36	34
55	38	48	44	30	42	59	2D	41	44	20	20	0	9	8	D5
73	26	2E	88	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[Initial]

Tap to initialize the SMBus API program.

Slave Address

Set the SMBus position (in hex) to be read or written.

▸ To read data:	▸ To write data:
Read up Set the maximum amount (in hex) of data to be read. **[Read]** Tap to read data to the text boxes below.	**Index** Set the index position (in hex) for writing data. **Data** Set data (in hex) to be written. **[Write]** Tap to write data to the text boxes below.

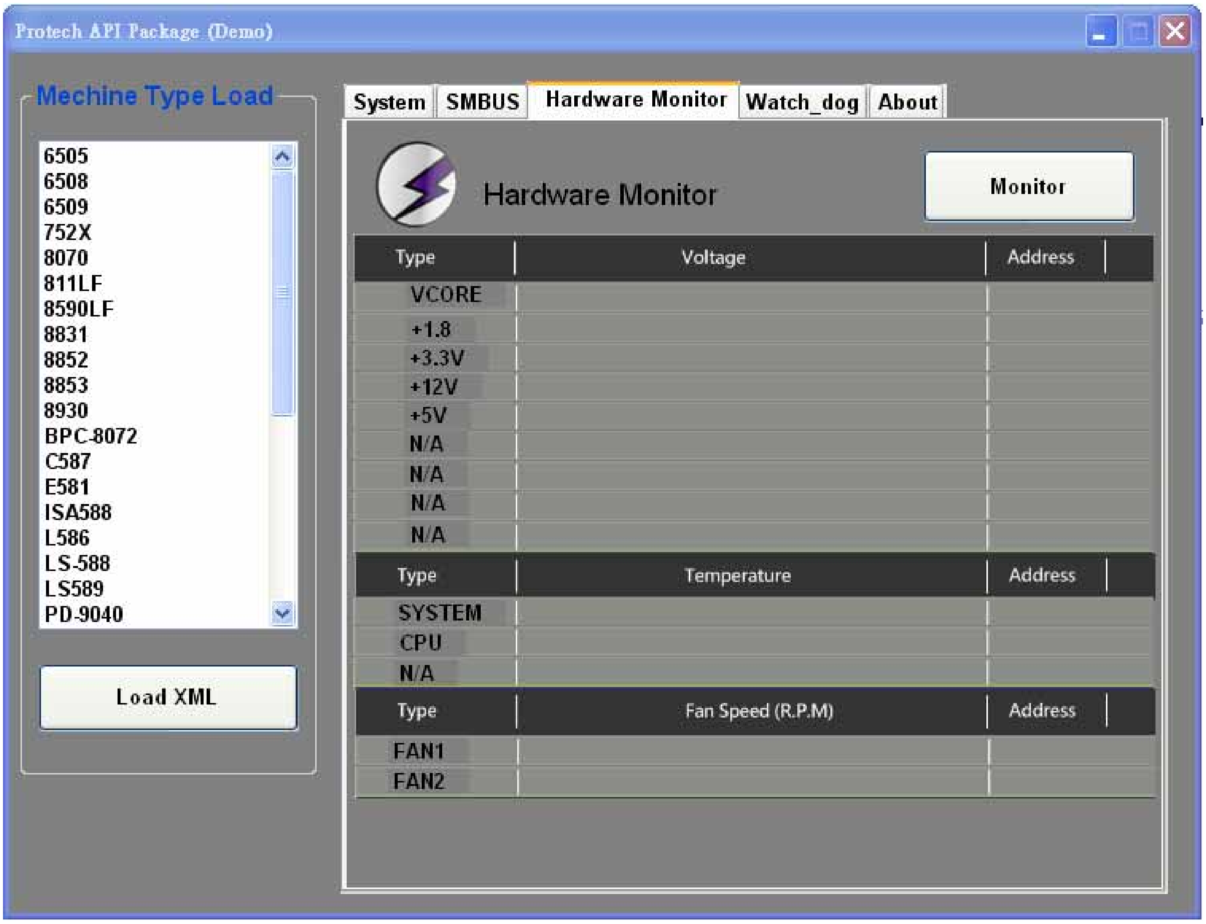
SMBUS Data

Data being read or written will be displayed in the text boxes below, after **[Read]** or **[Write]** is tapped.

[Clear]

Tap to clear all the text boxes under **SMBUS Data** ready for another entry.

Section 6 Hardware Monitor



Type	Voltage	Address
VCORE		
+1.8		
+3.3V		
+12V		
+5V		
N/A		
N/A		
N/A		
N/A		

Type	Temperature	Address
SYSTEM		
CPU		
N/A		

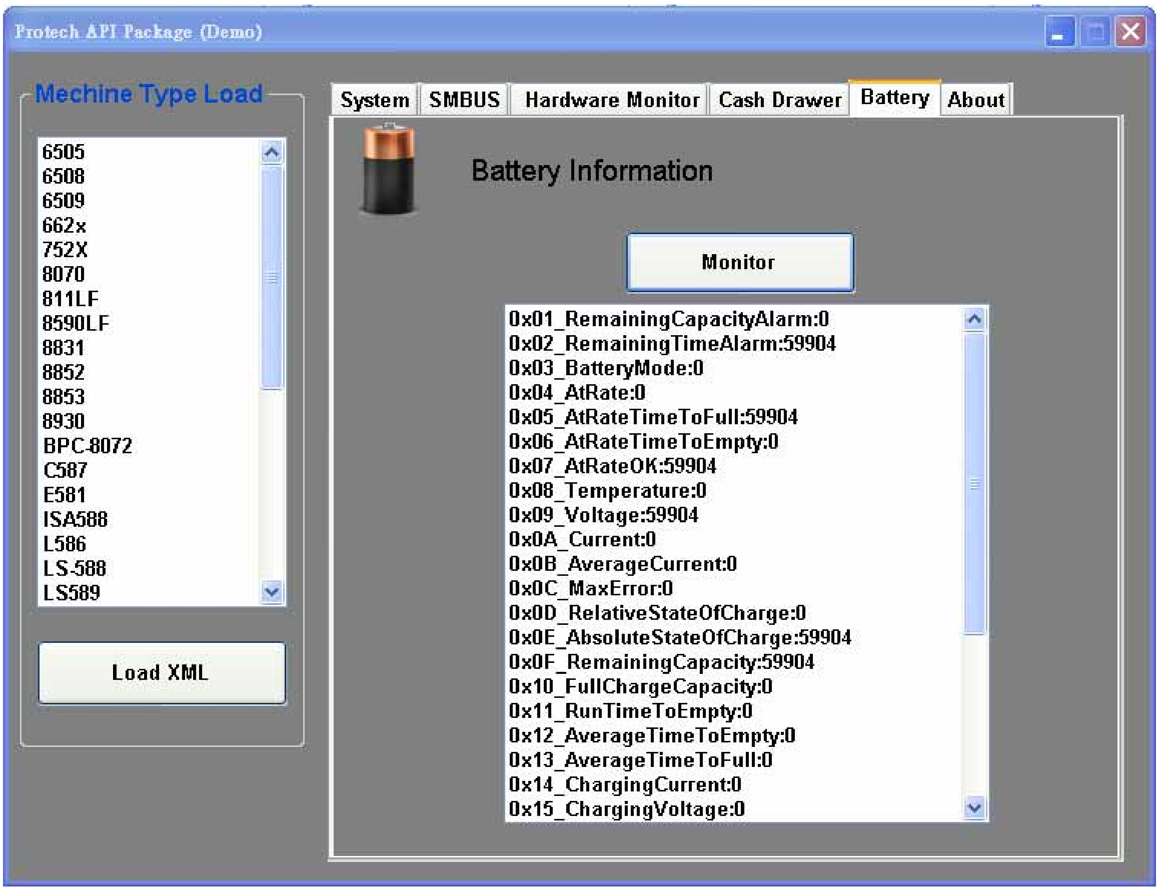
Type	Fan Speed (R.P.M)	Address
FAN1		
FAN2		

[Monitor]

Tap to get the hardware monitoring values, such as the voltages, temperatures, and fan speeds (rpm).

🕮 It is machine type dependent.

Section 7 Battery



```
6505
6508
6509
662x
752X
8070
811LF
8590LF
8831
8852
8853
8930
BPC-8072
C587
E581
ISA588
L586
LS-588
LS589
```

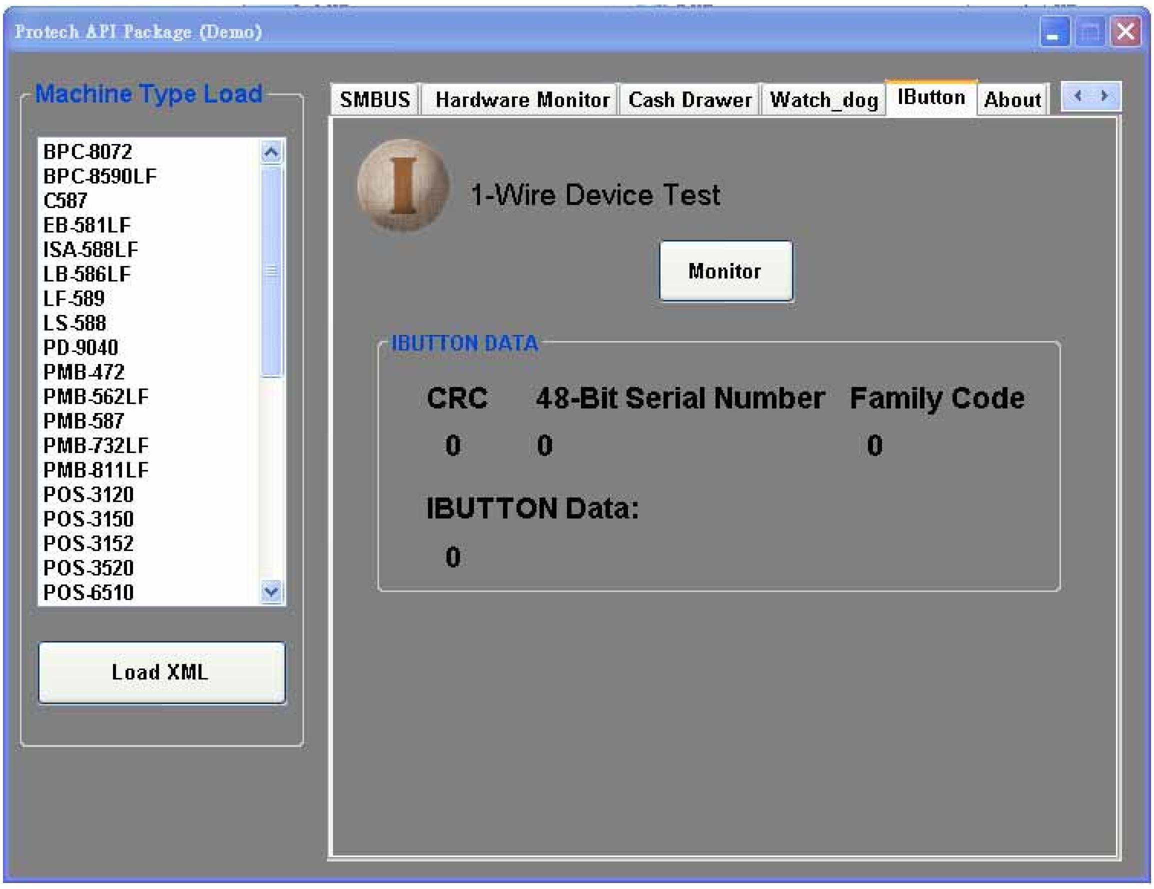
```
0x01_RemainingCapacityAlarm:0
0x02_RemainingTimeAlarm:59904
0x03_BatteryMode:0
0x04_AtRate:0
0x05_AtRateTimeToFull:59904
0x06_AtRateTimeToEmpty:0
0x07_AtRateOK:59904
0x08_Temperature:0
0x09_Voltage:59904
0x0A_Current:0
0x0B_AverageCurrent:0
0x0C_MaxError:0
0x0D_RelativeStateOfCharge:0
0x0E_AbsoluteStateOfCharge:59904
0x0F_RemainingCapacity:59904
0x10_FullChargeCapacity:0
0x11_RunTimeToEmpty:0
0x12_AverageTimeToEmpty:0
0x13_AverageTimeToFull:0
0x14_ChargingCurrent:0
0x15_ChargingVoltage:0
```

[Monitor]

Tap to get the UPS values.

📖 It is machine type dependent.

Section 8 I-Button



BPC-8072
BPC-8590LF
C587
EB-581LF
ISA-588LF
LB-586LF
LF-589
LS-588
PD-9040
PMB-472
PMB-562LF
PMB-587
PMB-732LF
PMB-811LF
POS-3120
POS-3150
POS-3152
POS-3520
POS-6510

CRC	48-Bit Serial Number	Family Code
0	0	0

IBUTTON Data:
0

[Monitor]

Tap to get the i-Button data that will be displayed below the **IBUTTON DATA** field.

Chapter 4 Program Developing

4

In this Chapter, you will learn essential functions when developing the program.

Sections included:

Section 1 API Function

The API program-related sample programs, developed in VB.Net and C#, are provided for easy use of the API Package. Refer to the main API functions listed as below.

API Function		DLL	
Digital IO	Digital_Initial Digital_Set Digital_Get	multilangXML.dll	Digital.dll
GPIO (IO)	GPIO_Initial GPIO_SetPort GPIO_Set GPIO_Get		GPIO.dll
Cash Drawer	CashDrawerOpen GetCashDrawerStatus		CashDrawer.dll
Watchdog (WD)	Watchdog_Set Watchdog_Stop Watchdog_SetMinSec Watchdog_Recount		WatchDog.dll
Hardware Monitor	HMWVoltage_Get HWMtTemperature_Get HWMFanSpeed_Get		Hardware Montior.dll
SMBUS	SMBUS_Initialization SMBUS_Write SMBUS_Read		SM_Control.dll

Section 2 Digital IO Function

Digital_Initial

bool Digital_Initial ( ) ;

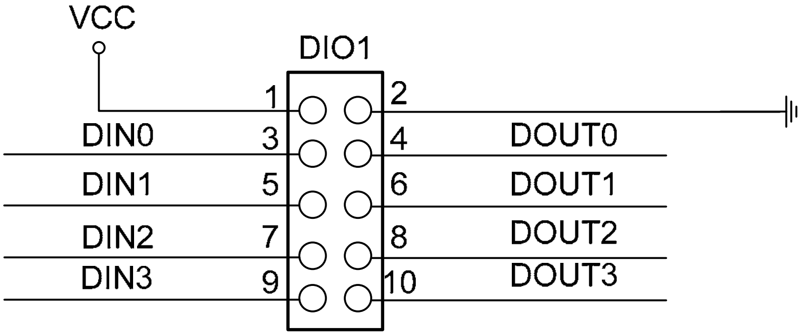
Purpose Initialize Digital API Package.
Value None
Return True (1) on success, False (0) on failure
🕮 Before using the API Package, this function should be called to pass XML variables to the DLL.

Digital_Set

bool Digital_Set (short hex_value);

Purpose Set the digital logic state.
Value hex_value
Return True (1) on success, False (0) on failure

For a 4in/ 4out type, as illustrated below:



The 4-bit (bit0 ~ bit3) binary value represents the digital output signal.
The binary variable is defined as High (1) and Low (0).

Example Digtial_Set(0x01); // Set DOUT0 as High

Digtial_Set(0x09); // 1001, DOUT3 and DOUT0 are High; DOUT2 and DOUT1 are low

Digital_Get

short   Digital_Get (void);

Purpose	Get the digital input signal.
Value	None
Return	Digital input pin logic state

Example

```
Short data;
data = Digital_Get( );        // DIN data, High/ Low input status
```

Section 3 GPIO Function

GPIO_Initial

bool GPIO_Initial (void);

Purpose Initialize the GPIO API Package.
Value None
Return True (1) on success, False (0) on failure
🕮 Before using the API Package, this function should be called.

GPIO_Set

bool GPIO_Set (long dout_value)

Purpose Set the GPIO logic state.
Value dout_value (in hex)
Return True (1) on success, False (0) on failure

GPIO_Get

long GPIO_Get ( )

Purpose Get the GPIO input signal.
Value None
Return GPIO input pin logic state
🕮 Make sure the GPIO pin is set as input.

GPIO_Setport

bool GPIO_SetPort (long Directvalue)

Purpose Set the GPIO pin as input/ output.
Value DirectValue (in hex)
Return True (1) on success, False (0) on failure

For an 8in/ 8out type of Protech products, the binary variable is defined as Output (1) and Input (0).
The 8-bit (bit0 ~ bit7) binary value represents each GPIO Pin.

Example GPIO_Set(0x11); // 00010001, GPIO4 and GPIO0 are set to Output; the others are Input

Section 4 Cash Drawer Function

CashDrawerOpen

bool CashDrawerOpen (short num_drawer);

Purpose Open the cash drawer API.
Value num_drawer = 1 (Open the Cash Drawer1)
2 (Open the Cash Drawer2)
Return True (1) on success, False (0) on failure

Example
```
CashDrawerOpen(0x01);    // Open the Cash Drawer1
```

GetCashDrawerStatus

bool GetCashDrawerStatus (short num_drawer);

Purpose Get the cash drawer status.
Value num_drawer = 1 (Get the Cash Drawer1 status)
2 (Get the Cash Drawer2 status)
Return True (1) on success, False (0) on failure

Example
```
Short data;
data= GetCashDrawerStatus(0x01);   // Get the Cash Drawer1 status
if (data)
MsgBox(“open1”);    // Cash Drawer1 status “Open”
Else
MsgBox(“close1”);    // Cash Drawer1 status “Close”
Endif
```

Section 5 Watch Dog Function

Watchdog_Set

bool Watchdog_Set (int value)

Purpose	Set the timeout for the watchdog timer.
Value	value = 0 ~ 255
Return	True (1) on success, False (0) on failure

Watchdog_SetMinSec

bool Watchdog_SetMinSec (int kind)

Purpose	Set the unit of time as second/ minute.
Value	kind = 1 (Measured in unit of second) 2 (Measured in unit of minute)
Return	True (1) on success, False (0) on failure

Watchdog_Stop

bool Watchdog_Stop (void)

Purpose	Stop the watchdog timer.
Value	None
Return	True (1) on success, False (0) on failure

Watchdog_Recount

bool Watchdog_Recount (void)

Purpose	Restart the watchdog timer.
Value	None
Return	True (1) on success, False (0) on failure

Section 6 Hardware Monitor Function

HMWVoltage_Get

float HMWVoltage_Get (short VoltType)

Purpose Get the hardware monitoring voltage value.

Value

VoltType	W83627HF	W83627EHF	SMSC3114	W83627UHG
0x01	VCoreA	CPU VCore	N/A	VCore
0x02	VCoreB	VIN0	+1.5V	VIN0
0x03	+3.3VIN	AVCC	N/A	AVCC
0x04	+5VIN	+3VCC	+5VIN	5VCC
0x05	+12VIN	VIN1	+12V	VIN1
0x06	-12VIN	VIN2	N/A	VIN2
0x07	-5VIN	VIN3	N/A	N/A

Return Float type data on voltage value

HMWTemperature_Get

float HMWTemperature_Get (short TempType)

Purpose Get the hardware monitoring temperature value.

Value

TempType	W83627HF	W83627EHF	SMSC3114	W83627UHG
0x01	CPU temperature	System temperature	CPU temperature	CPU temperature
0x02	N/A	CPU2 temperature	N/A	N/A
0x03	N/A	N/A	N/A	N/A

Return Float type data on temperature value

HMWFanSpeed_Get

float HMWFanSpeed_Get (short FanType)

Purpose Get the hardware monitoring fan speed value.

Value

FanType	W83627HF	W83627EHF	SMSC3114	W83627UHG
0x01	Fan1	SysFanIN	FAN1	FAN1
0x02	Fan2	CPUFANIN	FAN2	FAN2
0x03	N/A	AUXFANIN	N/A	N/A

Return Float type data on fan speed value (rpm)

Section 7 SMBUS Function

SMBUS_Initialization

bool SMBUS_Initialization (int Device)

Purpose	Initialize the SMBus API program and set the SMBus device address.	
Value	None	
Return	True (1) on success, False (0) on failure	

SMBUS_Read

int SMBUS_Read (int Index)

Purpose	Read the SMBus data.	
Value	Index	(SMBus address to be read)
Return	A byte Array representing the data	

SMBUS_Write

bool SMBUS_Write (int Index, int data)

Purpose	Write data into the SMBus.	
Value	Index	(SMBus address to be written)
	Data	(Data to be written)
Return	True (1) on success, False (0) on failure	

Section 8 UPS Function

Initialization

bool SMBUS_Initialization (int   Decive)

Value   Device   =   0x16   (The bq20z90/bq20z95 SBS Device Address)
Return   True (1) on success, False (0) on failure

RemainingCapacityAlarm

uint RemainingCapacityAlarm( )

Value   None
Return   Unsigned int value with a range of 0 to 65535

RemainingTimeAlarm

uint RemainingTimeAlarm( )

Value   None
Return   Unsigned int value with a range of 0 to 65535

BatteryMode

byte BatteryMode( )

Value   None
Return   Hex value with a range of 0 to 0xe383

AtRate

int AtRate( )

Value   None
Return   Signed int value with a range of -32768 to 32767

AtRateTimeToFull

uint AtRateTimeToFull( )

Value	None
Return	Unsigned int value with a range of 0 to 65534

AtRateTimeToEmpty

uint AtRateTimeToEmpty( )

Value	None
Return	Unsigned int value with a range of 0 to 65534

AtRateOK

uint AtRateOK( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

Temperature

uint Temperature( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

Voltage

uint Voltage( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

Current

int Current( )

Value	None
Return	Signed int value with a range of -32768 to 32767

AverageCurrent

int AverageCurrent( )

Value	None
Return	Signed int value with a range of -32768 to 32767

MaxError

uint MaxError( )

Value	None
Return	Unsigned int value with a range of 0 to 100

RelativeStateOfCharge

uint RelativeStateOfCharge( )

Value	None
Return	Unsigned int value with a range of 0 to 100

AbsoluteStateOfCharge

uint AbsoluteStateOfCharge( )

Value	None
Return	Unsigned int value with a range of 0 to 100

RemainingCapacity

uint RemainingCapacity( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

FullChargeCapacity

uint FullChargeCapacity( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

RunTimeToEmpty

uint RunTimeToEmpty( )

Value	None
Return	Unsigned int value with a range of 0 to 65534

AverageTimeToEmpty

uint AverageTimeToEmpty( )

Value	None
Return	Unsigned int value with a range of 0 to 65534

AverageTimeToFull

uint AverageTimeToFull( )

Value	None
Return	Unsigned int value with a range of 0 to 65534

ChargingCurrent

uint ChargingCurrent( )

Value	None
Return	Unsigned int value with a range of 0 to 65534

ChargingVoltage

uint ChargingVoltage( )

Value	None
Return	Unsigned int value with a range of 0 to 65534

BatteryStatus

uint BatteryStatus( )

Value	None
Return	Unsigned int value with a range of 0x0000 to 0xdbff

CycleCount

uint CycleCount( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

DesignCapacity

uint DesignCapacity( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

DesignVoltage

uint DesignVoltage( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

SpecificationInfo

byte SpecificationInfo( )

Value	None
Return	Hex value with a range of 0 to 0xFFFF

CellBoltage01

uint CellBoltage01( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

CellBoltage02

uint CellBoltage02( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

CellBoltage03

uint CellBoltage03( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

CellBoltage04

uint CellBoltage04( )

Value	None
Return	Unsigned int value with a range of 0 to 65535

SBS Command Values

Name	Format	Size in Bytes	Min Value	Max Value	Default Value	Unit
RemainingCapacityAlarm	unsigned int	2	0	65535	300	mAh or 10mWh
RemainingTimeAlarm	unsigned int	2	0	65535	10	min
BatteryMode	hex	2	0x0000	0xe383	—	
AtRate	signed int	2	-32768	32767	—	mA or 10mW
AtRateTimeToFull	unsigned int	2	0	65534	—	min
AtRateTimeToEmpty	unsigned int	2	0	65534	—	min
AtRateOK	unsigned int	2	0	65535	—	
Temperature	unsigned int	2	0	65535	—	0.1 K
Voltage	unsigned int	2	0	65535	—	mV
Current	signed int	2	-32768	32767	—	mA
AverageCurrent	signed int	2	-32768	32767	—	mA
MaxError	unsigned int	1	0	100	—	%
RelativeStateOfCharge	unsigned int	1	0	100	—	%
AbsoluteStateOfCharge	unsigned int	1	0	100+	—	%
RemainingCapacity	unsigned int	2	0	65535	—	mAh or 10mWh
FullChargeCapacity	unsigned int	2	0	65535	—	mAh or 10mWh

(continued)

Name	Format	Size in Bytes	Min Value	Max Value	Default Value	Unit
RunTimeToEmpty	unsigned int	2	0	65534	—	min
AverageTimeToEmpty	unsigned int	2	0	65534	—	min
AverageTimeToFull	unsigned int	2	0	65534	—	min
ChargingCurrent	unsigned int	2	0	65534	—	mA
ChargingVoltage	unsigned int	2	0	65534	—	mV
BatteryStatus	unsigned int	2	0x0000	0xdbff	—	
CycleCount	unsigned int	2	0	65535	—	
DesignCapacity	unsigned int	2	0	65535	4400	mAh or 10mWh
DesignVoltage	unsigned int	2	0	65535	14400	mV
SpecificationInfo	hex	2	0x0000	0xffff	0x0031	
CellVoltage4	unsigned int	2	0	65535	—	mV
CellVoltage3	unsigned int	2	0	65535	—	mV
CellVoltage2	unsigned int	2	0	65535	—	mV
CellVoltage1	unsigned int	2	0	65535	—	mV

Section 9 I-Button Function

Decode_Ibutton_Process

bool Decode_Ibutton_Process(short[] buffer)

Purpose Get the i-Button data.
Value Buffer = i-Button read will sent to this buffer
Return True (1) on success, False (0) on failure

A

Appendix A FAQ

In this Chapter, frequently asked questions accompanying the API Package will be clarified.

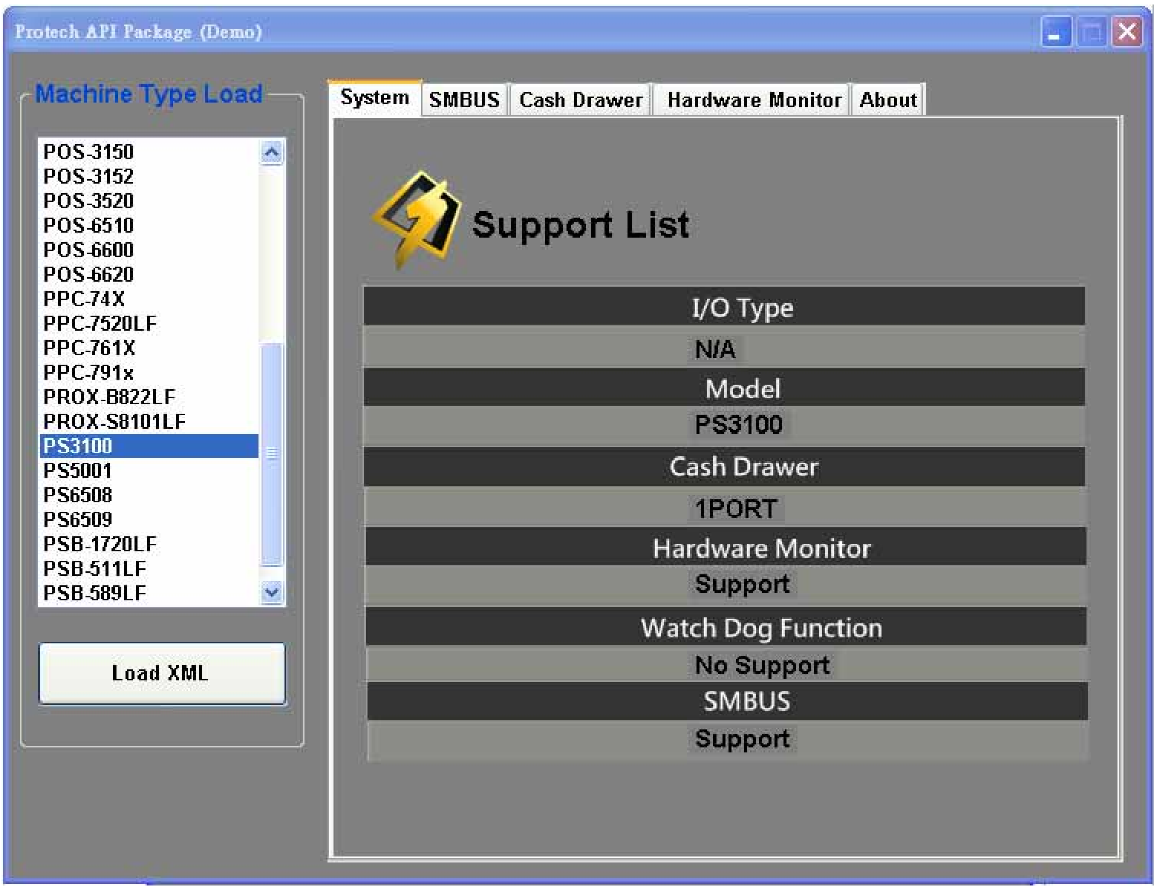
Sections included:

Section 1 Cannot Open API Program

Answer: There are two possible reasons:
(1) .Net framework 2.0 or above is not installed on the operating system yet.
(2) Lack of an XML file for the API Package.

Section 2 Cannot Make Sure XML File Correct or Not

Answer: After opening the API program, you can verify whether all functions for this model are presented in the Support List on the **System** tab.



Machine Type Load

- POS-3150
- POS-3152
- POS-3520
- POS-6510
- POS-6600
- POS-6620
- PPC-74X
- PPC-7520LF
- PPC-761X
- PPC-791x
- PROX-B822LF
- PROX-S8101LF
- PS3100
- PS5001
- PS6508
- PS6509
- PSB-1720LF
- PSB-511LF
- PSB-589LF

Support List

I/O Type
N/A
Model
PS3100
Cash Drawer
1PORT
Hardware Monitor
Support
Watch Dog Function
No Support
SMBUS
Support

🕮 The Initial.xml file in the ProxAPI standard folder is required to be replaced when using different machine type.

For example, if the PS3100 is desired, replace the XML file by one of the following:
1) Manually replace the XML file, by overwriting the old Initial.xml (ProxAPI standard\) with the new one (ProxAPI standard\XML Files\PS3100\Initial.xml). Then verify it in the API program.
2) In API program, select PS3100 from the “Machine Type Load” list on the left pane, and then tap [Load XML] to have the program replace the Initial.xml automatically.

Section 3 Cannot Find Functions in Support List

Answer: Functions displayed in the Support List are machine type dependent. Take PS3100 for example, the I/O Type field is marked with “N/A” in the Support List and you will be unable to find the **IO Control** tab as the PS3100 does not support Digital I/O.

Section 4 Cannot Run Self-developed Program

Answer: Make sure that all the API Package files are placed in your working directory and all links are already set. Meanwhile, the Initial.xml file has to be in place as well for the functions to work correctly.

Section 5 Cannot Use Demo Project

Answer: When using the Demo Project provided by Protech, you should make sure the Initial.xml file included in the API package corresponds to your developing machine type, to secure the link between files.

Section 6 Differences between Digital IO and GPIO

Answer: Each GPIO pin can be configured to be input or output, while Digital IO cannot. Therefore, you can change the GPIO pin direction from input to output, and vice versa.

By default, a 4in/ 4out type will be provided for developing applications. Note that these changes will be overwritten with default values after restarting the machine.

If the machine type supports GPIO, the additional **Program GPIO** tab will be displayed in the API program.