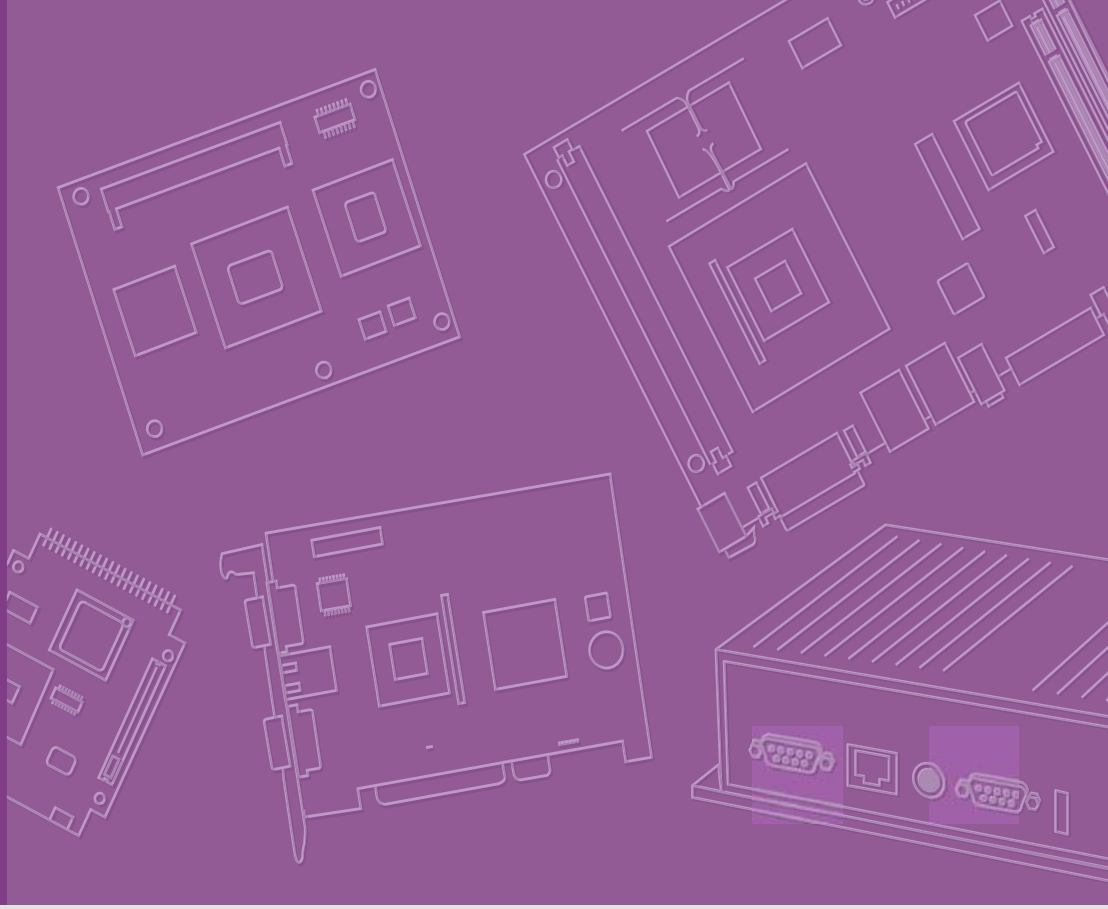


User Manual

iManager 2.0
Software API

ADVANTECH

Enabling an Intelligent Planet

Copyright

Part No. 200IMGR201 Edition 1

Printed in Taiwan November 2011

Contents

Chapter 1

Introduction

1.1 Intelligent Management for Embedded Platform

Advantech's new platforms come equipped with iManager, a micro controller that provides embedded features for system integrators. Embedded features have been moved from the OS/BIOS level to the board level, to increase reliability and simplify integration.

iManager runs whether the operating system is running or not; it can count the boot times and running hours of the device, monitor device health, and provide an advanced watchdog to handle errors just as they happen.

iManager also comes with a secure & encrypted EEPROM for storing important security key or other customer define information. All the embedded functions are configured through API or by a DEMO utility. Advantech is happy to provide its customers with the release of this suite of software APIs (Application Programming Interfaces). These offer not only the underlying drivers required but also a rich set of userfriendly, intelligent and integrated interfaces, which speeds development, enhances security and offers add-on value for Advantech platforms.

1.2 Benefits

- **Enhance System Reliability**
  - Protect system with multi-level watchdog
  - Auto adjust the fan speed, based on the temperature
  - Real-time monitoring of system status
- **Manage Onboard Devices**
  - Record boot information
  - Protect important information in encrypted data space
  - Multi-control interfaces for peripheral devices

1.2.1 Simplify Integration

Unique embedded functions are built-in to the iManager's uniform set of APIs, such as watchdog, monitoring, smart battery, and so on. This offers a multi-control interface for easy integration with all kind of peripherals; we have standard I^2C, SMBus and multi GPIO.

1.2.2 Enhance Reliability

Advanced watchdog, smart fan, hardware monitoring, and CPU throttling are provided by iManager, totally independent of the OS.

1.2.3 Secure the System

iManager provides an encryption space for storage of sensitive customer data such as user ID and password, secure keys for hard drive locks, and security IDs to protect your applications.

1.2.4 Easy System Upgrade

Uniform and OS-independent interface for cross-hardware platforms and uniform API across different OSs make it easy to migrate to other platforms or OSs.

1.2.5 Increased Performance

New "**Mail Box**" Technology : Collect all await data into a memory buffer and write out through iManager GPIO/I^2C/SMBus interfaces at once, It's the technology introduced to improve the accessing speed.

Chapter 2

System Requirements

2.1 iManager Utility & API System Requirements

2.1.1 Hardware

This Utility & API supports Advantech platforms only, with the iManager2.0 module; please see the release notes and check the support list before using it.

2.1.2 Operating Systems

- Windows® XP Professional SP3
- Windows® XP Embedded SP3
- Windows® Embedded Standard 2009
- Windows® 7 SP1 x86 / x64
- Windows® Embedded Standard 7 SP1 x86 / x64
- Ubuntu 10.04.1

2.1.3 Software Requirements

- NET Framework 2.0 (Windows OS)
- Basic Development utilities & libraries (Linux OS)

Chapter 3

API & Utility Installation

3.1 iManager Utility for Windows OS

Run the SETUP wizard to begin the installation of iManager 2.0. All files, including **Utility**, **Library**, **Header**, **User Manual** and **Sample Code**, will deploy to the location: "C:\Program Files\Advantech\EmbStore".

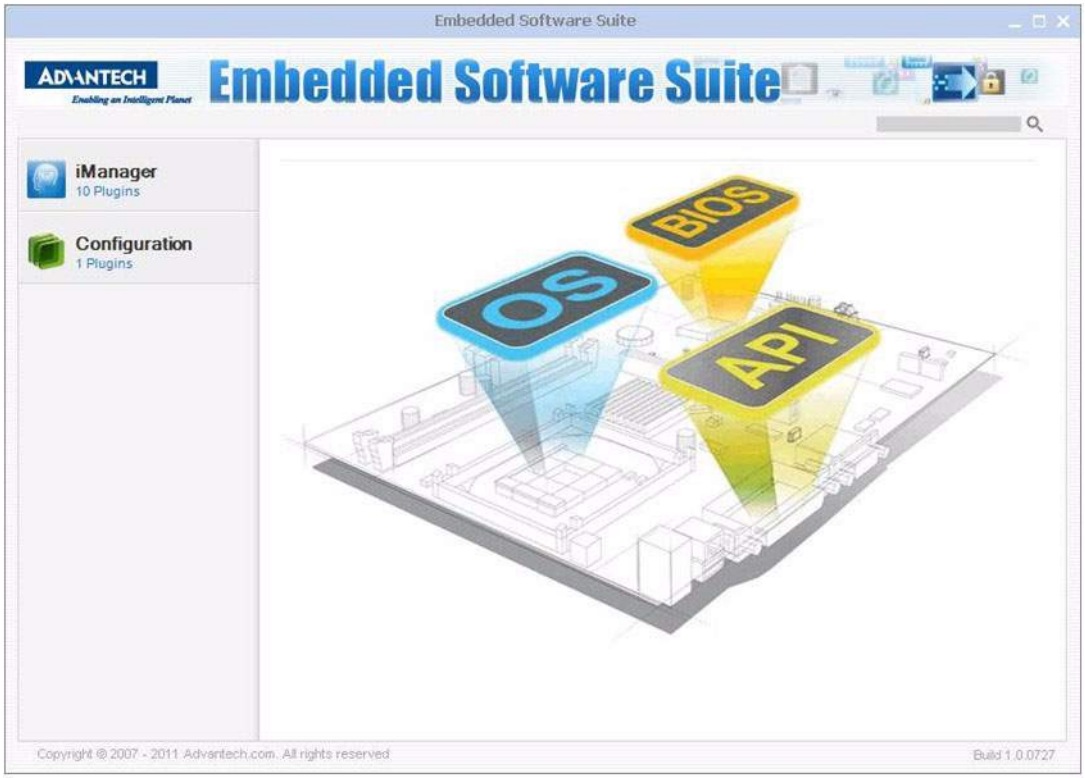
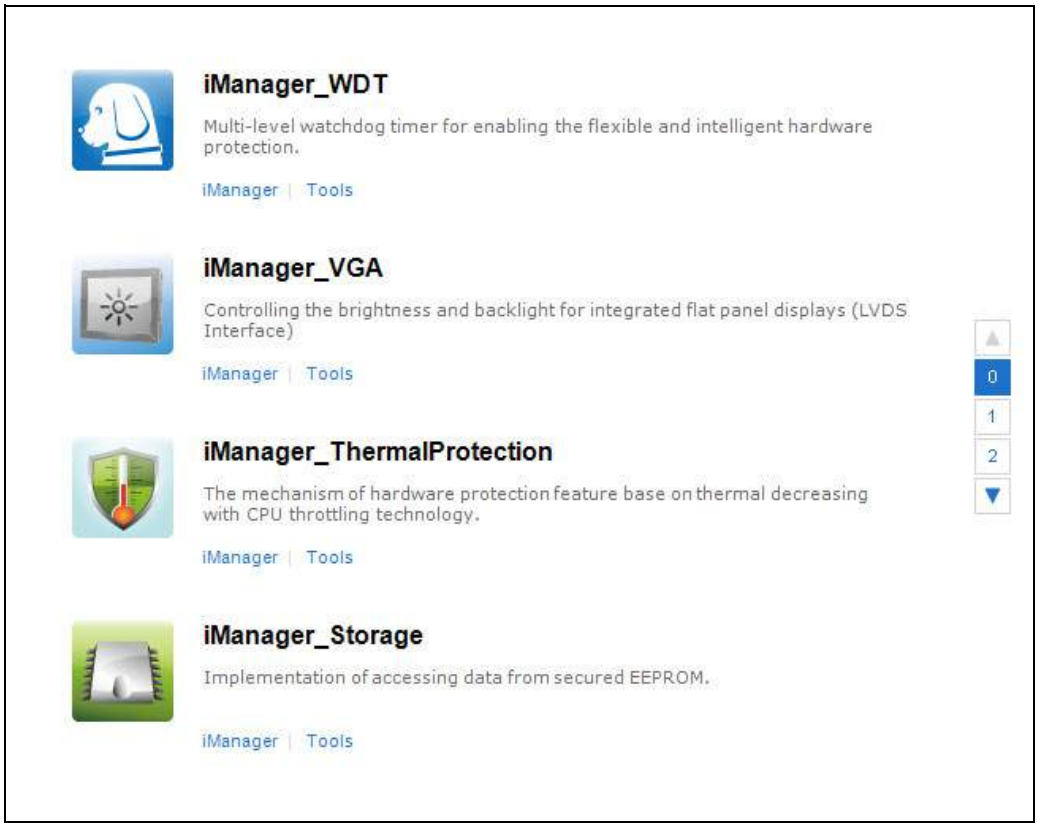
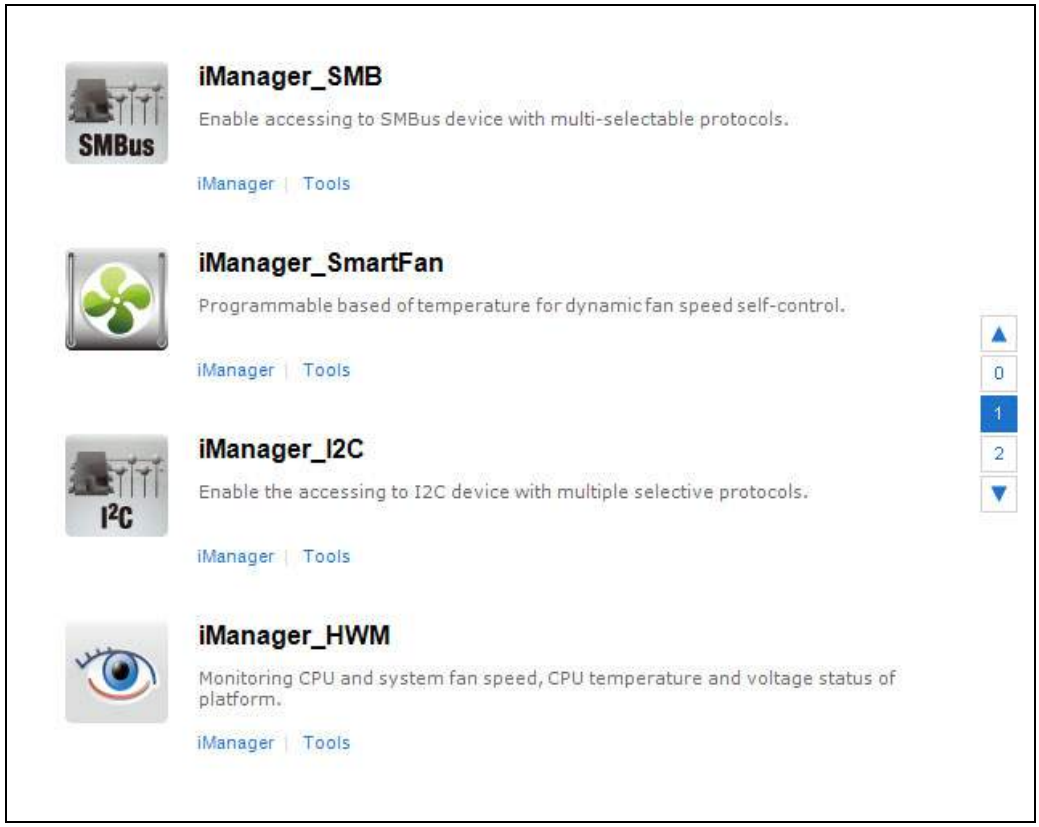


Figure 3.1 iManager 2.0 Utility Main Window





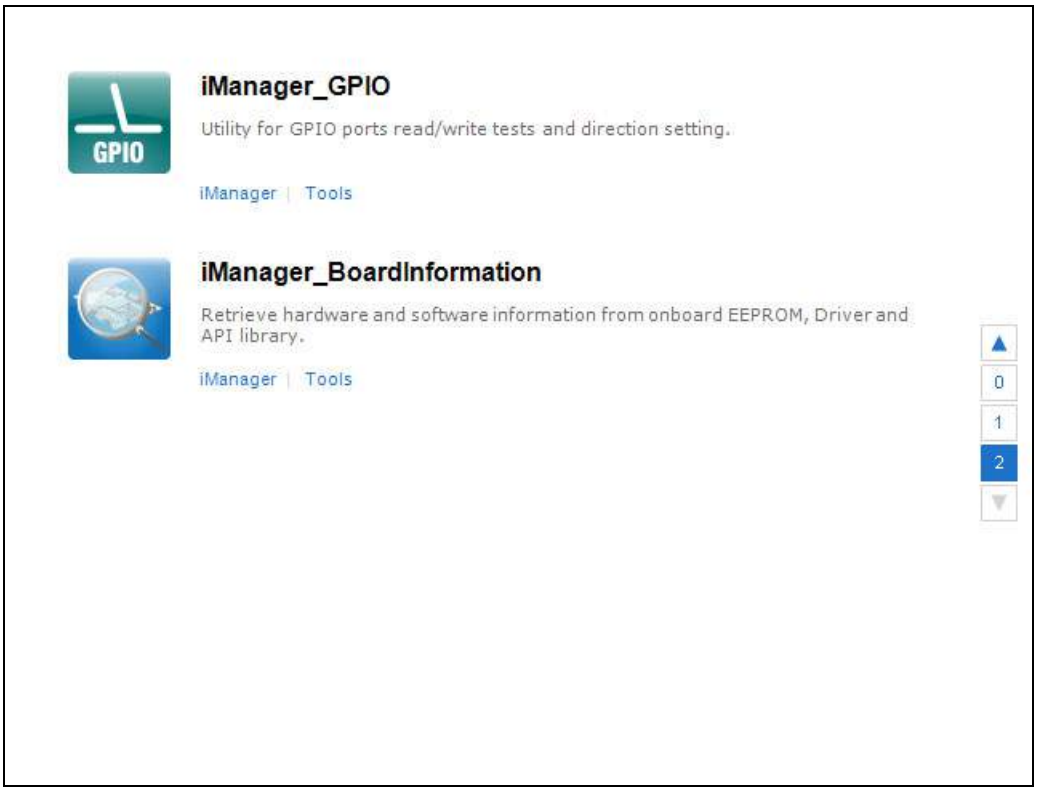


Figure 3.2 Function List ithin the Utility

[Microsoft Windows]

1.	ESS.exe	Utility
2.	Resource \ Doc \	User Manual

The device manager will show the “Advantech Embedded Software Suite Driver”and “Advantech EC Driver” after the installation is successful.

3.2 iManager API & Example Source Code

The iManager API is easy to install. To use the iManager API, just copy the following files to your application folder. There is no need to do any formal installation. To make your life easier, we provide the C# sample code of how to use the API library, which will facilitate development of your own programs.

The provided files are:

Windows:

1.	Resource \ API \ EAPI_1.dll	iManager dynamic link library
2.	Resource \ API \ EAPI_1.lib	iManager static library
3.	Resource \ API \ REL_EC_API.h	iManager dynamic link library header
4.	Resource \ API \ Define.h	Header file, definitions
5.	Resource \ SampleCode \ SampleCode.zip	iManager API example source code in C#

Linux:

1.	libEApi.so	iManager library file for Linux OS
2.	REL_EC_API.h	iManager dynamic link library header
3.	Define.h	iManager header and definitions

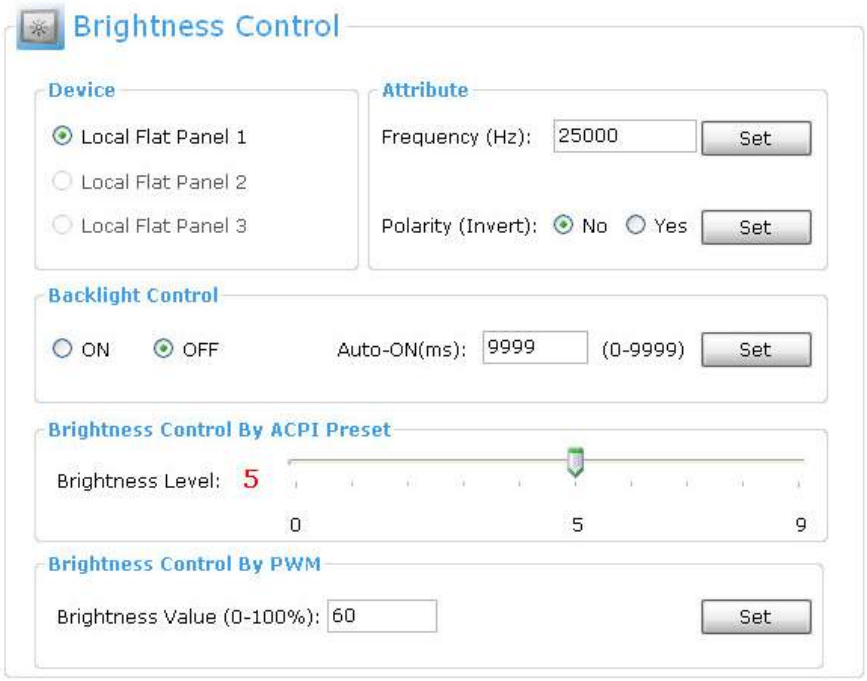
Chapter 4

iManager 2.0 Utilities

iManager 2.0 Utilities are plug-ins for Advantech Embedded Software Suite for Windows. They can be used to monitor / control the entire system and help the developer to test iManager's features. By default, the Embedded Software Suite will install with pre-installed iManager functions. Functions supported in the Utility are:

4.1 VGA

Enable the support of Low Level Backlight & Brightness control.



[Option intro]

Device: Select panel target (multiple outputs supported).

Attribute:

- Frequency (Hz): Set the output frequency of the Inverter.
- Polarity (Invert): Set whether or not to change polarity of Inverter PWM signal.

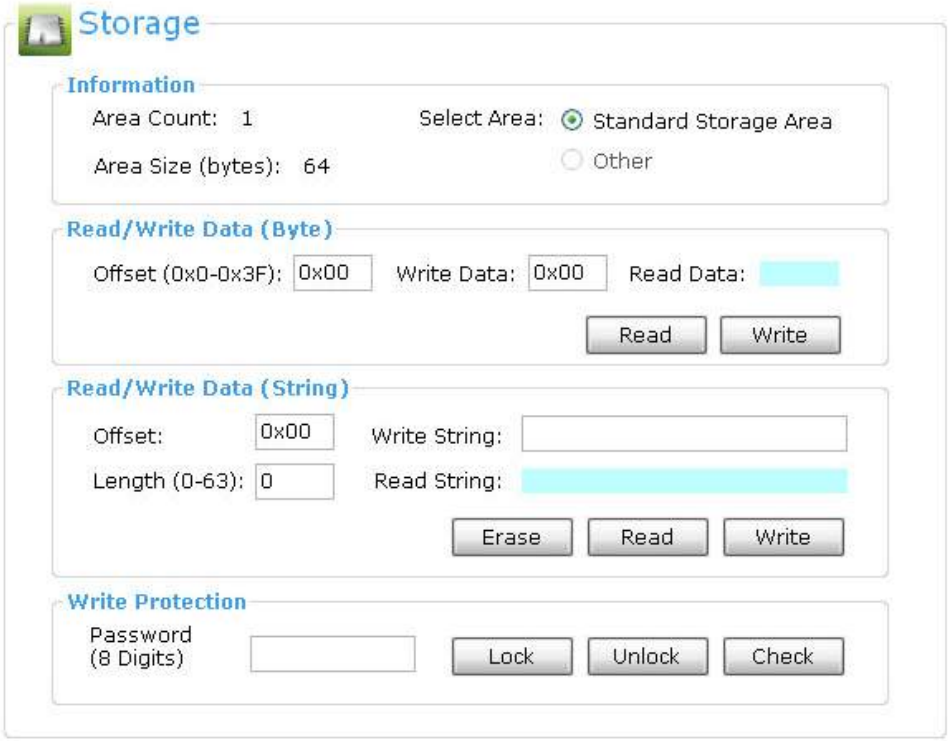
Backlight control: Set On or Off press set button to apply (auto-ON is a software method to prevent display loss for this demo utility only).

Brightness control by ACPI preset: Switch to specific level of ACPI brightness table.

Brightness control by PWM: Output brightness in PWM mode.

4.2 Storage

Access storage information and read / write data to the selected user data area, Lock Down EEPROM. The total size of OEM EEPROM area is 64 bytes. Developers can use this area to store their own data.



[Function DEMO]

DEMO 1.

Read/Write Data (Byte) into EEPROM:

1. Give the proper value to the "**Offset**" and **"Write Data**" text boxes.
2. Click the "**Read**" button for read operation. Click the "**Write**" button with value in "**Write Data**" field for write operation.
3. The value read or to be written is showed in the "**Read Data**" text box.

DEMO 2.

Read/Write Data (String) or Erase a block into EEPROM:

1. Give the proper value to the "**Offset**" and "**Length**" text boxes.
2. Click the "**Read**" button for read string operation. Click the "**Write**" button with-value in "**Write Data**" field for write string operation. Click the "**Erase**" button toerase block.
3. The value read or to be written is showed in the "**Read Data**" text box.

DEMO 3.

Write Protection Lock/Unlock OEM EEPROM AREA:

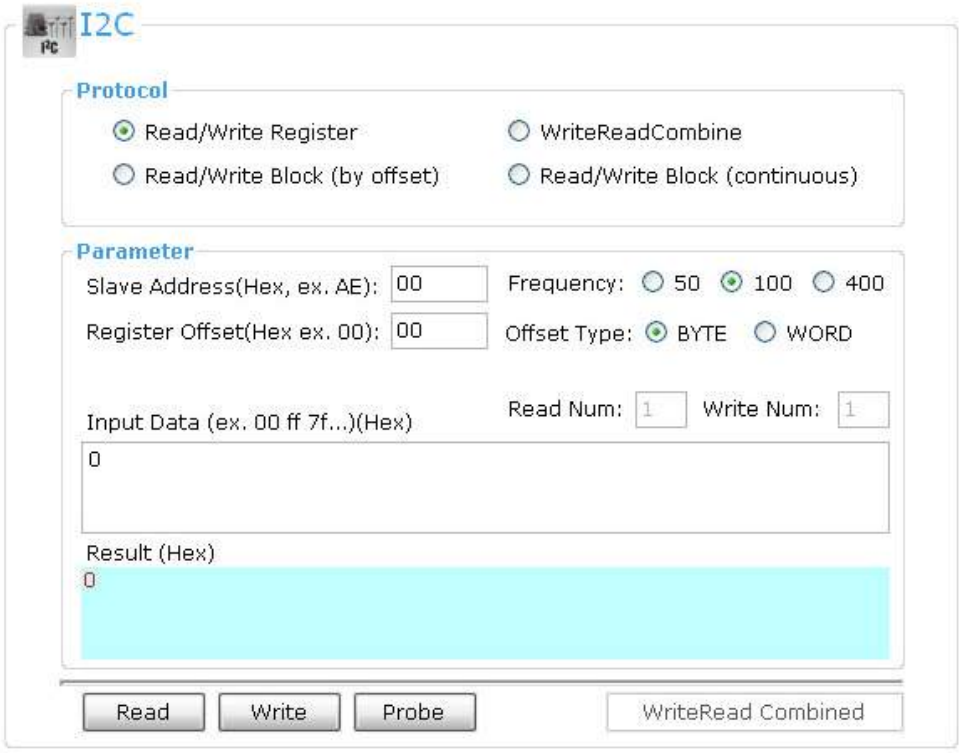
1. Enter the password in the "**Password**" text box.
2. Click "**Check**" button, the status will be displayed in an Advantech SUSI Message Box.
3. If the message shows Unlocked, you can click the "**Lock**" button, then the area will be locked by the password.

4. Otherwise, you can click the “**Unlock**” button to unlock this area if the secret key is correct.
5. You can always click the “**Check**” button to check the protection status.

Note! *For the default password of a new platform, please remain **empty** and unlock the storage at first time setup.*

4.3 I²C

Probe and access I²C bus to get the capabilities, and doing read / write tests to specific registers in selected I²C devices.



[Function DEMO]

Common Step :

Set I²C device - slave address

Demo 1.

Read/write register: Access specific register once a time through I²C interface:

- Set offset.
- Set offset type: Byte / WORD (2 bytes).
- Key in input data (only 1 byte/word).
- Click “**Write**”.
- Click “**Read**” to confirm the data has been written or not.

Demo 2.

Write read combine: Read specific range of data from I²C device after specific offset address. (ignore the register offset configuration):

- Set offset type.
- Set read & write number.
- Key in input offset & data. Note: The first entry of data is the offset address.
- Click "**WriteRead Combined**".

Demo 3.

Read/write block (by block) :

- Set offset. Block read/write to the data on I²C device after specific offset.
- Set "**Offset Type**".
- Set "**Read/Write Number**".
- Key in input data(maximum : 32 bytes).
- Click "**Write**" to write the data (refer to "**Write Num**").
- Click "**Read**" to read the data (refer to "**Read Num**").

Demo 4.

Read/write block (Continuous): Append the Block read/write to the data on I²C device after specific offset flag continuously:

- Set "**Offset Type**".
- Set "**Read/Write Number**".
- Key in input data.
- Click "**Write**" to write the data (refer to "**Write Num**").
- Click "**Read**" to read the data (refer to "**Read Num**").

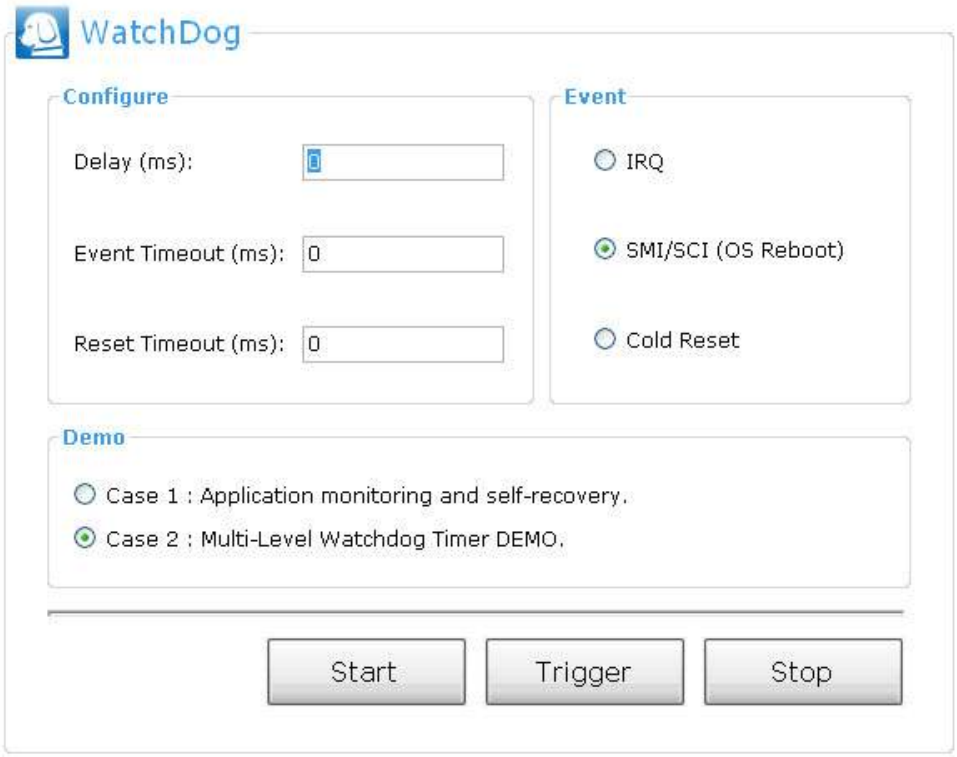
The Frequency refers to the Frequency of the I²C Bus, all values in kHz.

Note! *If you want to change the frequency of the I²C Bus make sure ALL your I2C devices support the faster speed, otherwise don't change it! Default is 100 kHz.*

4.4 Watchdog Timer

In general, a watchdog timer is a function that performs a specific operation after a certain period of time when something goes wrong with the system. A watchdog timer can be programmed to restart the system after a certain time period when a program or computer fails to respond.

Since many customers like to program different responses to different events, Advantech has designed an advanced watchdog which consists of both a **single stage** and a **multi-stage** timer.



[Option intro]

Delay Time: Time between error condition and Watchdog launch.

Event Timeout: If EC does not receive the trigger within this value of time out, the watchdog will enter the 1st stage event (Event: IRQ, OS reboot, Cold reset).

Reset Timeout: If EC does not receive the trigger within this value of time out, the watchdog will enter the 2nd stage event: forced reboot.

There are 2 Demo examples in the Watchdog utility; no matter which you select, you have to set timeouts for possible error scenarios:

- **Case 1:**
  DEMO: Application & OS hang off and self-recover.
  Scenario:
  Clicking the "**Start**" button launches an application that keeps sending the trigger to iManager EC, then click **App Crash** in the popup window to make it crash at once. The application will stop sending the trigger event to EC, after 1st stage Event Time-out, EC will send an IRQ event to terminate and recover itself, and the app will back to normal and keep sending the trigger to EC; if you click the **BSOD** button, the application & OS will crash immediately, the app cannot send the trigger, so the 1st stage event will be ignored, and after 2nd stage Reset Time-out it will enter into 2nd stage.

- **Case 2:**
  DEMO: Multi-stage watchdog timer level testing.
  Scenario :
  Click "**start**" button, a message box will appear, you may click the "**Trigger**" button manually to send the trigger and make your OS and App respond, if you stop clicking the "**Trigger**" button, then it will automatically enter the 2nd stage: reboot the OS.

Note! *The **SCI/SMI** event: depends on the BIOS setting, on SOM-5890, the event is OS reboot signal by default.*

Note! *Valid event types will change for different platforms due to hardware limitations. Please reference the hardware platform user manual to get detailed information.*

Note! *Make sure you uncheck the option "**automatically restart**" in "**Advanced System Properties**" of Windows OS.*

4.5 General Purpose IOs (GPIO)

GPIO Control: Set single GPIO Direction (In or Out) or Level Status (High or Low): DATA Read / Write Testing and Define the Direction and protocol of 8-Pin GPIO.

[Function DEMO]

Demo 1.

Set single GPIO:

1. In GPIO control frame.
2. Click radio button.
3. Click **icons** to setting **I/O direction** and **High/Low** status.
4. Click "**Set**" to apply.

Demo 2.

Read single GPIO:

1. In GPIO frame.
2. Click radio button.
3. Click "**Get**" to read data and show the result to icons.

Demo 3.

Bank protocol: write 8 pin GPIO data in one time:

1. In GPIO bank control frame.
2. Click radio button.
3. Click **icons** to set the **I/O direction** and **High/Low** status.
4. Click "**Set**" to apply.

Demo 3.

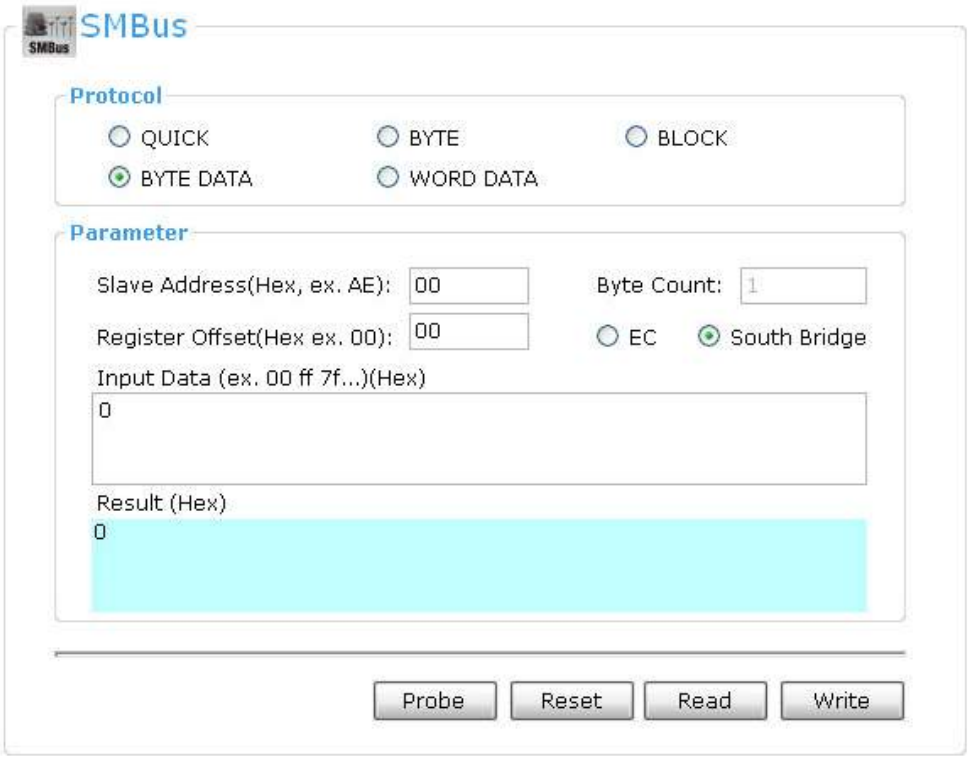
Bank protocol: read 8 pin GPIO data in one time:

1. In GPIO bank control frame.
2. Click radio button.
3. Click “**Get**“ to read data and shows the result to icons.

Note! *GPIO pin definition will change for different platforms due to hardware design. Please reference the hardware platform user manual to get detailed information.*

4.6 SMBus

Allows to interface an embedded system environment and transfer serial messages using protocol, allowing multiple simultaneous device control.



[Function DEMO]

DEMO 1.

Read data by SMBus:

1. Choose one of the protocol operations in the Protocol field: **QUICK**, **BYTE**, **BYTE DATA**, **WORD DATA** and **Block**.
2. Enter the proper values for "**Slave address**" and "**Register offset**" text boxes. Some protocol operations don't have register offsets.
3. Click the "**Read**" button to read/receive operations.
4. The value read from this address is showed in the "**Result (Hex)**" text box.

DEMO 2.

Write data by SMBus:

1. Choose one of the protocol operations in Protocol field: **QUICK**, **BYTE**, **BYTE DATA**, **WORD DATA** and **Block**.
2. Enter the proper values for "**Slave address**" and "**Register offset**" text boxes. Some protocol operations don't have register offsets.
3. Type a value in Input Data box.
4. Click the "**Write**" button to write/send operations.
5. The values to be written are showed in the "**Result (Hex)**" text box.

Probe SMBus

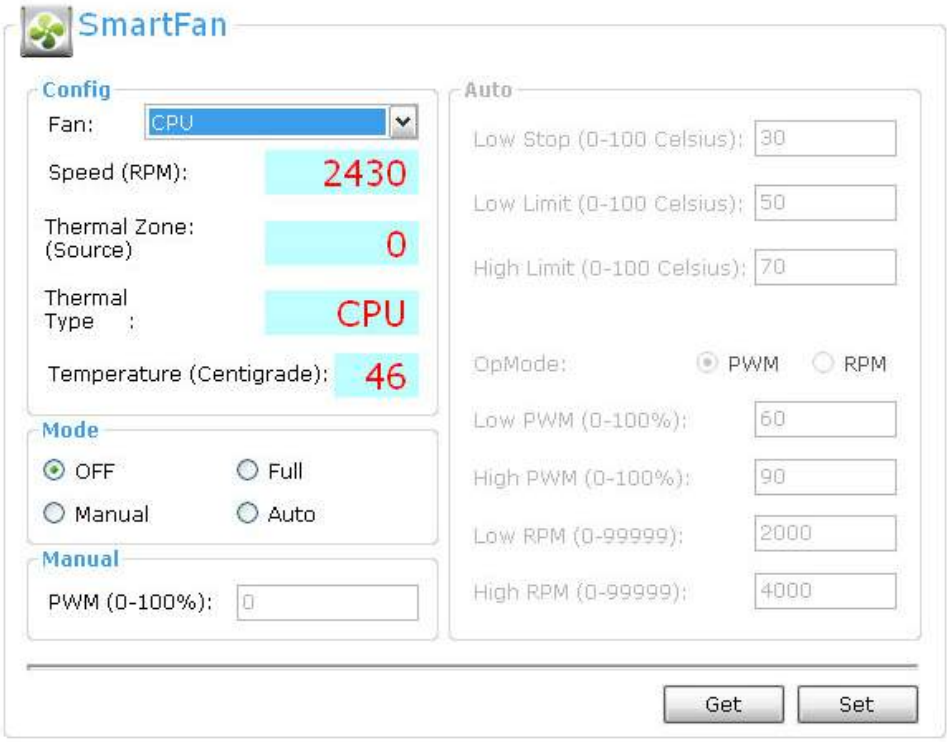
Probe initiates auto detect of all connected SMBus devices. It will show all occupied addresses in the Result Box. This function is specially provided for developers and engineers who need a fast overview of which addresses are free and which are occupied.

Note! *SMBus API supports dual channel interface, you can select the channel between I^2C and SMBus, the SMBus DEMO utility demonstrate the default scenario of the accessing to SMBus device.*

4.7 Smart Fan

iManager's design provides the smart fans for the user to monitor the fan speed and pre-define it based on the system temperature.

Monitoring the states of the device fan, and capable to self-control the fan speed actively depends on the thermal changes.



[Option intro]

There are four modes supported:

Mode:

- **Off**: Stop the fan.
- **Full**: The fan runs at full speed.
- **Manual**: Setting the fan speed manually in PWM format.
- **Auto**: Setting the policy to dynamic control the fan speed based on the system temperature.

[Function Demo]

Demo 1.

Read a fan speed value:

1. Select the reference fan target in the list. (Ex. CPU, System, and so on.)
2. The fan speed value will be shown.

Demo 2.

Fan speed control - Off:

1. From Mode list, select "**OFF**".
2. Click the "**Set**" button. The fan will be disabled immediately.

Demo 3.

Fan speed control - Full:

1. From Mode list, select “**Full**”.
2. Click “**Set**” button. The fan will operate at full speed.

Note! *The full speed depends on the high limit set in auto-mode (below).*

Demo 4.

Fan speed control - Manual:

1. From Mode list, select “**Manual**”.
2. Fill in the PWM value. (Ex. 50, in text box.)
3. Click “**Set**” button. The fan will operate at the designed speed.

Demo 5.

Fan speed control - Auto:

The fan speed can be set according to change of the CPU temperature.

1. From Mode list, select “**Auto**”.
2. Choose **PWM** or **RPM** options in **OpMode** list.
3. Fill in temperature value of Low Limit and High limit. (Ex: Low Limit: 65, High Limit: 80.)
4. Setting the fan speed in both PWM and RPM mode. (Ex: select the RPM Mode -> Low limit fan speed: 1800 RPM, High limit fan speed : 4000 RPM.)
5. Press “**Set**” button, then the device fan will dynamic & self-control automatically according to the state of CPU temperature.

Note! *The temperature for a zone is between the "Lower Temperature Stop" and "Lower Temperature Limit", the speed of the fan assigned is determined as follows:*

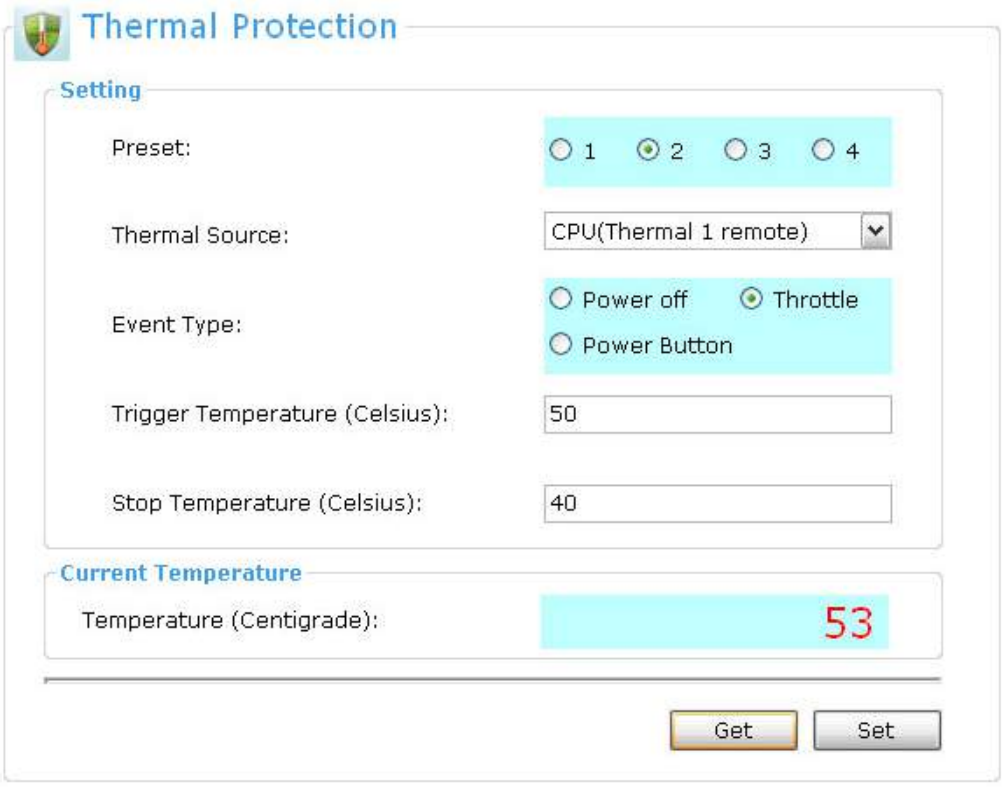
When the temperature reaches the Fan Temp "Low Limit" for a zone, the PWM output assigned to that zone will be Fan "Low PWM/RPM".

Between "Low Limit" and "High Limit", the PWM/RPM duty cycle will increase linearly according to the temperature. The PWM duty cycle will be 100% at "High Limit".

Note! *The fan speed range depends on the fan, and some fans don’t report the speed status. Please refer to the spec. of the selected fan and adjust the range to suit; check the details.*

4.8 Thermal Protection

Setup the controls of CPU throttling threshold to auto sense the state of temperature.



[Option intro]

Preset: iManager 2.0 provides four presets for thermal protection.

Thermal source: Sensor source selection.

Event type: Select protection trigger event: shutdown (sends power button signal to the OS to shut down the device);Throttle (forces CPU frequency to the lowest setting: for example: to the SOM-5890 platform: the lowest CPU clock is 800 MHz per core); Power Off (hard shutdown, power off).

Trigger temperature: Above this temperature, CPU throttling kicks in (do not set too low).

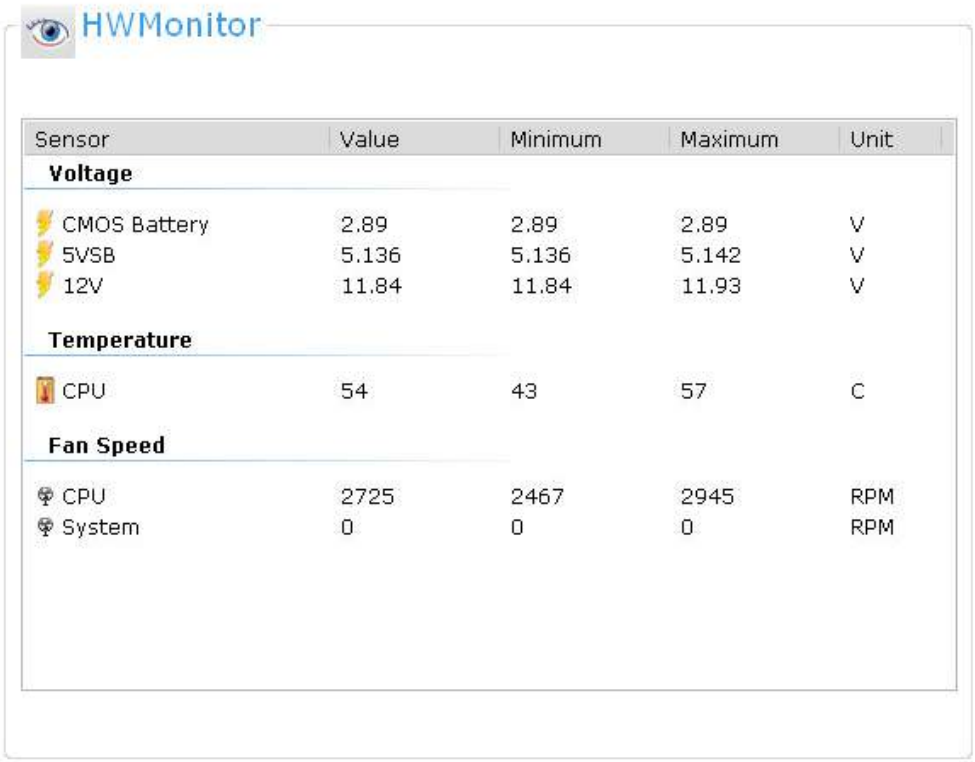
Stop temperature: Above this temperature, CPU throttling discontinues.

Note! *Do not set value of temp. too low in shutdown or power off mode, or the platform will not boot up correctly (**solution**: un-plug the power and wait for reset the settings)*

Note! *The event type "**Throttle**" supports specific Intel® processors only.*

4.9 Hardware Monitor

The Hardware Monitor shows you all system important features on one page.



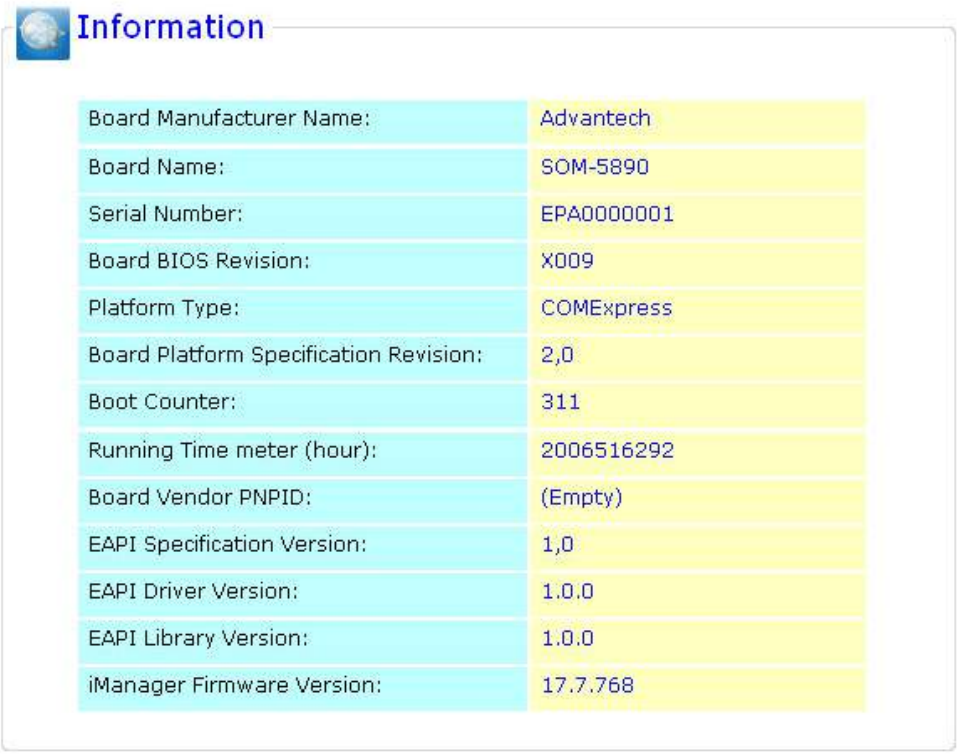
[Function intro]

Get various information in value or text format from the hardware platform. The hardware monitor contains three features: **voltage**, **temperature** and **fan speed**. These items are important to the operation of the system because when errors happen, they may cause permanent damage to the PC.

The values will be always up to date. Additional it will keep track of lower and upper limits of all the values. The limits are calculated by the software during run time, they do not include any other drops or peaks which occur when the system starts or the software is not running.

4.10 Board Information

iManager can gather and record system information for users to manage their devices.



Board Information:

Access Information of the hardware platform from EEPROM, all params are follow standard definition of EAPI.

- **Board Manufacturer Name**: The creator of this platform, usually ADVANTECH.
- **Board Name**: This is platform name.
- **Serial Number**: This number is input by the factory, used for sales tracking and service, *e.g. ABC000000020*.
- **Board BIOS Revision**: The version of BIOS file, *e.g. BIOS Version: 1.10.*
- **Platform Type**: Spec type of hardware.
- **Board Platform Specification Revision**: Board spec revision.
- **Boot Counter**: Boot up times.
- **Running Time Meter**: Running times in hours.
- **Board Vendor PNPID**: Microsoft Plug-and-Play ID
- **API Specification Version**: Version of the used EAPI specification.
- **EAPI Driver Version**: Version of the EAPI Driver.
- **EAPI Library Version**: Version of the EAPI Library.
- **iManager 2.0 Firmware Version**: Advantech firmware revision of iManager.

Chapter 5

Programming Overview & API Reference

The iManager API provides the functions to control ADVANTECH iManager platforms. API functions are based on a dynamic library. Our Advantech iManager API can be implemented in various other programming languages.

Header Files

- **REL_EC_API.H** includes the API declaration, constants and flags that are required for programming.
- Define.h include defines of variables

Library Files

- **EAPI_1.dll** is a dynamic link library that exports all the API functions.

Demo Program

- The iManager 2.0 utility, released with sample source code, demonstrates how to fully use iManager features. The Library is written in C++ and utility is written in C#.

Drivers

- **AdvEC.sys (x86)** or **AdvEC_Win7_AMD64.sys(x64)** is the driver that controls the hardware.

5.1 Status Codes

All EAPI* functions immediately return a status code from a common list of possible errors. Any function may return any of the defined status codes. See the Appendix for more detailed information.

5.1.1 Status Code Description

EAPI_STATUS_NOT_INITIALIZED
Description
The EAPI library is not yet or unsuccessfully initialized. EApiLibInitialize needs to be called prior to the first access of any other EAPI function.
Actions
Call EApiLibInitialize.

EAPI_STATUS_INITIALIZED
Description
Library is initialized.
Actions
None.

EAPI_STATUS_ALLOC_ERROR
Description
Memory Allocation Error.
Actions
Free memory and try again.

EAPI_STATUS_SW_TIMEOUT
Description
Software timeout. This is Normally caused by hardware/software semaphore timeout.
Actions
Retry.

EAPI_STATUS_INVALID_PARAMETER
Description
One or more of the EAPI function call parameters are out of the defined range.
Actions
Verify Function Parameters.

EAPI_STATUS_INVALID_BLOCK_LENGTH
Description
This means that the Block length is too long.
Actions
Use relevant Capabilities information to correct select block lengths.

EAPI_STATUS_INVALID_BLOCK_ALIGNMENT
Description
The Block Alignment is incorrect.
Actions
Use Alignment Capabilities information to correctly align write access.

EAPI_STATUS_INVALID_DIRECTION
Description
The current Direction Argument attempts to set GPIOs to a unsupported directions. I.E. Setting GPI to Output.
Actions
Use pInputs and pOutputs to correctly select input and outputs.

EAPI_STATUS_INVALID_BITMASK
Description
The Bitmask Selects bits/GPIOs which are not supported for the current ID.
Actions
Use pInputs and pOutputs to probe supported bits.

EAPI_STATUS_UNSUPPORTED
Description
This function or ID is not supported at the actual hardware environment.
Actions
None.

EAPI_STATUS_NOT_FOUND

Description

Selected device was not found.

Example

The I^2C device address is not Acknowledged, device is not present or inactive.

Actions

None.

EAPI_STATUS_BUSY_COLLISION

Description

The selected device or ID is busy or a data collision was detected.

Example

- The addressed I^2C bus is busy or there is a bus collision.
- The I^2C bus is in use. Either CLK or DAT are low.
- Arbitration loss or bus Collision, data remains low when writing a 1.

Actions

Retry.

EAPI_STATUS_RUNNING

Description

Watchdog timer already started.

Actions

Call EApiWDogStop, before retrying.

EAPI_STATUS_HW_TIMEOUT

Description

Function call timed out.

Example

I^2C operation lasted too long.

Actions

Retry.

EAPI_STATUS_READ_ERROR

Description

An error was detected during a read operation.

Example

I^2C Read function was not successful.

Actions

Retry.

EAPI_STATUS_WRITE_ERROR

Description

An error was detected during a write operation.

Example

- I^2C Write function was not successful.
- No Acknowledge was received after writing any byte after the first address byte.
- Can be caused by unsupported device command/index.
- 10Bit Address Device Not Present
- Storage Write Error

Actions

Retry.

EAPI_STATUS_MORE_DATA

Description

The amount of available data exceeds the buffer size.Storage buffer overflow was prevented. Read count was larger than the defined buffer length.

Actions

Either increase the buffer size or reduce the block length.

EAPI_STATUS_ERROR

Description

Generic error message. No further error details are available.

Actions

None.

EAPI_STATUS_SUCCESS

The value for this status code is defined as 0.

Description

The operation was successful.

Actions

None.

5.2 EAPI

5.2.1 Define

```
EC_API
EAPI_CALLTYPE WINAPI      //  stdcall
ULONG     unsigned     long
```

5.2.2 Initialization Functions

5.2.2.1 EApiLibInitialize

```
EC_API
ULONG
EAPI_CALLTYPE
EApiLibInitialize(void);
```

Description:

General initialization of the EAPI. Prior to calling any EAPI function the library needs to be initialized by calling this function. The status code for all EAPI function will be EAPI_STATUS_NOT_INITIALIZED unless this function is called.

Parameters:

None.

Return Status Code

Condition	Return Value
Library initialized	EAPI_STATUS_INITIALIZED
Library initial fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.2.2 EApiLibUnInitialize

```
EC_API
ULONG
EAPI_CALLTYPE
EApiLibUnInitialize(void);
```

Description:

General function to uninitialized the EAPI library. Should be called before program exit. In a dynamic library environment this function is not expected to replace the native uninitialize routines. It Is expected that in this environments this function has no functionality.

Parameters:

None.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Success	EAPI_STATUS_SUCCESS

5.2.3 EAPI Information Functions

5.2.3.1 EApiBoardGetStringA

```
EC_API
ULONG
EAPI_CALLTYPE
EApiBoardGetStringA(
                        ULONG       Id,          /* Name Id */
                        char        *pBuffer ,   /* Destination pBuffer */
                        ULONG       *pBufLen     /* pBuffer Length */
);
```

Description:

Text information about the hardware platform.

Parameters:

Id

Selects the Get String Sub function Id.

Id	Description	Example
EAPI_ID_BOARD_MANUFACTURER_STR	Board Manufacturer Name	Advantech
EAPI_ID_BOARD_NAME_STR	Board Name	SOM-5890
EAPI_ID_BOARD_SERIAL_STR	Serial Number	EPA0000001
EAPI_ID_BOARD_BIOS_REVISION_STR	Board BIOS Revision	X001
EAPI_ID_BOARD_PLATFORM_TYPE_STR	Platform ID See 'Platform Specification'	COMExpress

pBuffer

Pointer to a buffer that receives the value's data. This parameter can be NULL if the data is not required.

pBufLen

Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the pBuffer parameter. When the function returns, this variable contains the size of the data copied to pBuffer including the terminating null character.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBufLen==NULL	EAPI_STATUS_INVALID_PARAMETER
*pBufLen&&pBuffer==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
Success	EAPI_STATUS_SUCCESS

5.2.3.2 EApiBoardGetValue

```
EC_API
ULONG
EAPI_CALLTYPE
EApiBoardGetValue(
                    ULONG     Id,          /* Value Id */
                    ULONG     *pValue      /* Return Value */
);
```

Description:

Information about the hardware platform in value format.

Parameters:

Id

Selects the Get Value Sub function Id.

pValue

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

Id	Description	Units/Format
EAPI_ID_GET_EAPI_SPEC_VERSION	EAPI Specification Version used to implement API	Specification Version Number Format see Appendix E
EAPI_ID_BOARD_BOOT_COUNTER_V AL	Boot Counter	boots1
EAPI_ID_BOARD_RUNNING_TIME_MET ER_VAL	'Running Time Meter	hours1
EAPI_ID_BOARD_PNPID_VAL	Board Vendor PNPID	Not Support Yet
EAPI_ID_BOARD_PLATFORM_REV_VA L	Platform Specification Version used to create board. Number Format' see Appendix E	'Specification Version
EAPI_ID_BOARD_DRIVER_VERSION_V AL	Vendor Specific Driver Version	'General Version number Format' 1 see Appendix E
EAPI_ID_BOARD_LIB_VERSION_VAL	Vendor Specific Library Version	'General Version number Format' 1 see Appendix E
EAPI_ID_BOARD_FIRMWARE_VERSION_VAL	EC firmware Version	'General Version number Format' 1 see Appendix E
EAPI_ID_HWMON_CPU_TEMP	CPU Temperature	1 centigrade
EAPI_ID_HWMON_CHIPSET_TEMP	Chipset Temperature	1 centigrade
EAPI_ID_HWMON_SYSTEM_TEMP	System Temperature	1 centigrade
Id	**Description**	**Units/Format**
EAPI_ID_HWMON_VOLTAGE_VCORE	CPU Core Voltage	volts1
EAPI_ID_HWMON_VOLTAGE_2V5	2.5V Voltage	volts1

EAPI_ID_HWMON_VOLTAGE_3V3	3.3V Voltage	volts1
EAPI_ID_HWMON_VOLTAGE_VBAT	Battery Voltage	volts1
EAPI_ID_HWMON_VOLTAGE_5V	5V Voltage	volts1
EAPI_ID_HWMON_VOLTAGE_5VSB	5V Standby Voltage	volts1
EAPI_ID_HWMON_VOLTAGE_12V	12V Voltage	volts1
EAPI_ID_HWMON_FAN_CPU	CPU Fan	RPM1
EAPI_ID_HWMON_FAN_SYSTEM	System Fan	RPM1

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pValue==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
Success	EAPI_STATUS_SUCCESS

5.2.4 Backlight Functions

This function sub set facilitates backlight control for Integrated flat panel displays, typically LVDS.

5.2.4.1 Common Parameters

Backlight Ids

Selects the flat panel display.

Id	Description
EAPI_ID_BACKLIGHT_1	Backlight Local Flat Panel 1
EAPI_ID_BACKLIGHT_2	Backlight Local Flat Panel 2
EAPI_ID_BACKLIGHT_3	Backlight Local Flat Panel 3

Backlight Enable Values

Name	Description
EAPI_BACKLIGHT_SET_ON	Requests/Signifies that the Backlight be Enabled
EAPI_BACKLIGHT_SET_OFF	Requests/Signifies that the Backlight be Disabled

5.2.4.2 EApiVgaGetBacklightEnable

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaGetBacklightEnable(
                              ULONG     Id,          /* Backlight Id */
                              ULONG     *pEnable     /* Backlight Enable */
);
```

Description:

Returns current Backlight Enable state for specified Flat Panel.

Parameters:

Id

See 'Backlight Ids'.

pEnable

Pointer to a buffer that receives the current backlight enable state. See Backlight Enable Values.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pEnable==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.4.3 EApiVgaSetBacklightEnable

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaSetBacklightEnable(
                                ULONG   Id,         /* Backlight Id */
                                ULONG   Enable   /* Backlight Enable */
);
```

Description:

Enables the backlight of the selected flat panel display.

Parameters:

Id

See 'Backlight Ids'.

Enable

Backlight Enable options. See Backlight Enable Values.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Enable!=EAPI_BACKLIGHT_SET_ON && Enable!=EAPI_BACKLIGHT_SET_OFF	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.4.4 EApiVgaGetBacklightBrightness

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaGetBacklightBrightness(
                                ULONG     Id,         /* Backlight Id */
                                ULONG     *pBright  /* Backlight Brightness */
);
```

Description:

Reads the current brightness of the selected flat panel display.

Parameters:

Id

See 'Backlight Ids'.

pBright

Pointer to a buffer that receives the current backlight brightness level. See Backlight Brightness Value Range.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBright==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.4.5 EApiVgaSetBacklightBrightness

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaSetBacklightBrightness(
                                ULONG   Id,       /* Backlight Id */
                                ULONG   Bright    /* Backlight Brightness */
);
```

Description:

Sets the brightness of the selected flat panel display.

Parameters:

Id

See 'Backlight Ids'.

Bright

Backlight Brightness level. (The value is from 0 to 100 percentage.)

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Bright>EAPI_BACKLIGHT_SET_BRIGHTEST	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.4.6 EApiVgaSetFrequency

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaSetFrequency (
                        ULONG   Id,         /* Backlight Id */
                        ULONG   dwSetting   /* Frequency of Panel */
);
```

Description:

Sets the frequency of the selected flat panel display.

Parameters:

Id

See 'Backlight Ids'.

dwSetting

Frequency value. (The value is from 0 to 1000000 Hz)

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Bright>EAPI_BACKLIGHT_SET_BRIGHTEST	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.4.7 EApiVgaSetPolarity

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaSetPolarity (
                        ULONG   Id,             /* Backlight Id */
                        ULONG   dwSetting   /* Frequency of Panel */
);
```

Description:

Sets the polarity of the selected flat panel display.

Parameters:

Id

See 'Backlight Ids'.

dwSetting

Polarity state. (1 is invert, 0 is no-invert)

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
(dwSetting != 0) || (dwSetting != 1)	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.4.8 EApiVgaGetBacklightLevel

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaGetBacklightLevel(
                              ULONG    Id,          /* Backlight Id */
                              ULONG    *pLevel     /* Backlight Level */
);
```

Description:

Reads the current brightness level of the selected flat panel display.

Parameters:

Id

See 'Backlight Ids'.

pLevel

Pointer to a buffer that receives the current backlight brightness level. (From 0 to 9).

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBright==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.4.9 EApiVgaSetBacklightLevel

```
EC_API
ULONG
EAPI_CALLTYPE
EApiVgaSetBacklightLevel(
                              ULONG    Id,        /* Backlight Id */
                              ULONG    Level     /* Backlight Level */
);
```

Description:

Sets the brightness level of the selected flat panel display.

Parameters:

Id

See 'Backlight Ids'.

Level

Backlight Brightness level. (The value is from 0 to 9.)

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Level>EAPI_BACKLIGHT_SET_LEVELEST	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.5 Storage Functions

The EAPI defines one user storage area with a minimal size of 64 Byte.

5.2.5.1 Common Parameters

Storage Ids

The EAPI only defines one user storage area. Additional vendor specific IDs are possible

Id	Description
EAPI_ID_STORAGE_STD	Standard Storage Area

5.2.5.2 EApiStorageCap

```
EC_API
ULONG
EAPI_CALLTYPE
EApiStorageCap(
                    ULONG     Id,                 /* Storage Area Id */
                    ULONG     *pStorageSize ,     /* Total */
                    ULONG     *pBlockLength       /* Write Block Length*/
);
```

Description:

Get the capabilities of the selected storage area.

Parameters:

Id

See 'Storage Ids'.

pStorageSize

Pointer to a buffer that receives storage area size. This parameter can be NULL if the data is not required.

pBlockLength

Pointer to a buffer that receives the storage areas Block size.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
((pStorageSize==NULL)&& (pBlockLength==NULL))	EAPI_STATUS_INVALID_PARAMETER
Unsupported Id	EAPI_STATUS_UNSUPPORTED
Success	EAPI_STATUS_SUCCESS

5.2.5.3 EApiStorageAreaRead

```
EC_API
ULONG
EAPI_CALLTYPE
EApiStorageAreaRead(
                        ULONG   Id,          /* Storage Area Id */
                        ULONG   Offset,      /* Byte Offset */
                        void    *pBuffer ,   /* Pointer to Data pBuffer */
                        ULONG   BufLen,      /* Data pBuffer Size in bytes*/
                        ULONG   ByteCnt      /* Number of bytes to read */
);
```

Description:

Reads data from the selected user data area.

Parameters:

Id

See 'Storage Ids'.

Offset

Storage area start address offset in bytes.

pBuffer

Pointer to a buffer that receives the read data.

BufLen

Size, in bytes, of the buffer pointed to by the pBuffer parameter

ByteCnt

Size, in bytes, of the information read to the buffer pointed to by the pBuffer parameter.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBuffer==NULL	EAPI_STATUS_INVALID_PARAMETER
ByteCnt==0	EAPI_STATUS_INVALID_PARAMETER
BufLen==0	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
Offset+ByteCnt>pStorageSize	EAPI_STATUS_INVALID_BLOCK_LENGTH
Read Error	EAPI_STATUS_READ_ERROR
ByteCnt>BufLen	EAPI_STATUS_MORE_DATA
Success	EAPI_STATUS_SUCCESS

5.2.5.4 EApiStorageAreaWrite

```
EC_API
ULONG
EAPI_CALLTYPE
EApiStorageAreaWrite(
                        ULONG    Id,          /* Storage Area Id */
                        ULONG    Offset,      /* Byte Offset */
                        void     *pBuffer ,   /* Pointer to Data pBuffer */
                        ULONG    ByteCnt      /* Number of bytes to write*/
);
```

Description:

Writes data to the selected user data area.

Parameters:

Id

See 'Storage Ids'.

Offset

Storage area start address offset in bytes. This value must be a multiple of *pBlockLength.

pBuffer

Pointer to a buffer containing the data to be stored.

ByteCnt

Size, in bytes, of the information pointed to by the pBuffer parameter.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBuffer==NULL	EAPI_STATUS_INVALID_PARAMETER
ByteCnt==0	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
Offset+ByteCnt>pStorageSize	EAPI_STATUS_INVALID_BLOCK_LENGTH
Write Error	EAPI_STATUS_WRITE_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.5.5 SusiStorageAreaIsLocked

```
EC_API
ULONG
EAPI_CALLTYPE
SusiStorageAreaIsLocked(
                        ULONG   Id,         /* Storage Area Id */
                        ULONG   dwFlags
);
```

Description:

Check the storage area is locked.

Parameters:

Id

See 'Storage Ids'.

dwFlags

Reserved.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.5.6 SusiStorageAreaLock

```
EC_API
ULONG
EAPI_CALLTYPE
SusiStorageAreaLock(
                        ULONG   Id,             /* Storage Area Id*/
                        ULONG   dwFlags,        /* Reserved */
                        char    *pBytes,
                        ULONG   dwLen
);
```

Description:

Lock a storage area for write protect.

Parameters:

Id

See 'Storage Ids'.

dwFlags

Reserved for future use, set to 0.

pByte

Lock of key buffer.\

dwLen

Number of key buffer.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBytes==NULL	EAPI_STATUS_INVALID_PARAMETER
dwLen>EEPROM_SECURE_KEY_SIZE	EAPI_STATUS_INVALID_PARAMETER
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.5.7 SusiStorageAreaUnlock

```
EC_API
ULONG
EAPI_CALLTYPE
SusiStorageAreaUnlock(
                        ULONG   Id,          /* Storage Area Id*/
                        ULONG   dwFlags,     /* Reserved */
                        char    *pBytes,
                        ULONG   dwLen
);
```

Description:

Lock a storage area for write protect.

Parameters:

Id

See 'Storage Ids'.

dwFlags

Reserved.

pByte

Un-Lock of key buffer.

dwLen

Number of key buffer.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBytes==NULL	EAPI_STATUS_INVALID_PARAMETER
dwLen>EEPROM_SECURE_KEY_SIZE	EAPI_STATUS_INVALID_PARAMETER
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6 Functions for the I²C Bus

Set of function to access the I²C bus.

5.2.6.1 Common Parameters

I²C Bus Ids

The EAPI specification currently defines three I²C buses for COM Express.

Id	Description
EAPI_ID_I2C_EXTERNAL	Baseboard I²C Interface
EAPI_ID_I2C_LVDS_1	LVDS/EDP 1 Interface
EAPI_ID_I2C_LVDS_2	LVDS/EDP 2 Interface

5.2.6.2 EapiI2CGetBusCap

```
EC_API
ULONG
EAPI_CALLTYPE
EApiI2CGetBusCap(
                        ULONG  Id,              /* I2C Bus Id */
                        ULONG  *pMaxBlkLen /* Max Block Length Supported on this interface */
);
```

Description:

Returns the capabilities of the selected I²C bus.

Parameters:

Id

See 'I²C Bus Ids'.

pMaxBlkLen

Size in bytes, Pointer to a buffer that receives the maximum transfer block length for the given interface.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pMaxBlkLen==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
Success	EAPI_STATUS_SUCCESS

5.2.6.3 EapiI2CWriteReadRaw

```
EC_API
ULONG
EAPI_CALLTYPE
EApiI2CWriteReadRaw(
                        ULONG    Id,            /* I2C Bus Id */
                        ULONG    Addr,          /* Encoded 7Bit I2C Device Address */
                        void     *pWBuffer ,    /* Write Data pBuffer */
                        ULONG    WriteBCnt,     /* Number of Bytes to write*/
                        void     *pRBuffer ,    /* Read Data pBuffer */
                        ULONG    RBufLen,       /* Data pBuffer Length */
                        ULONG    ReadBCnt       /* Number of Bytes to Read*/
);
```

Description:

Universal function for read and write operations to the I^2C bus.

Parameters:

Id

See 'I^2C Bus Ids'.

Addr

First Byte of I^2C Device Address.

pWBuffer

Pointer to a buffer containing the data to be transferred. This parameter can be NULL if the data is not required.

WriteBCnt

Size, in bytes, of the information pointed to by the pWBuffer parameter plus 1 If pWBuffer is NULL this must be zero or one.

pRBuffer

Pointer to a buffer that receives the read data. This parameter can be NULL if the data is not required.

RBufLen

Size, in bytes, of the buffer pointed to by the pRBuffer parameter. If pRBuffer is NULL this must be zero.

ReadBCnt

Size, in bytes, to be read to pRBuffer plus 1 If pRBuffer is NULL this must be zero or one.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
(WriteBCnt>1)&&(pWBuffer==NULL)	EAPI_STATUS_INVALID_PARAMETER
(ReadBCnt>1)&&(pRBuffer==NULL)	EAPI_STATUS_INVALID_PARAMETER
(ReadBCnt>1)&&(RBufLen==0)	EAPI_STATUS_INVALID_PARAMETER
((WriteBCnt==0)&&(ReadBCnt==0))	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6.4 EapiI2CReadTransfer

```
EC_API
ULONG
EAPI_CALLTYPE
EApiI2CReadTransfer(
                        ULONG    Id,          /* I2C Bus Id */
                        ULONG    Addr,        /* Encoded 7/10Bit I2C Device Address*/
                        ULONG    Cmd,         /* I2C Command/Offset */
                        Void     *pBuffer ,   /* Transfer Data pBuffer */
                        ULONG    BufLen,      /* Data pBuffer Length */
                        ULONG    ByteCnt      /* Byte Count to read */
);
```

Description:

Reads from a specific register in the selected I^2C device. Reads from I^2C device at the I^2C address Addr the amount of ByteCnt bytes to the buffer pBuffer while using the device specific command Cmd. Depending on the addressed I^2C device Cmd can be a specific command or a byte offset.

Parameters:

Id

See 'I^2C Bus Ids'.

Addr

Encoded 7/10 Bit I^2C Device Address.

Cmd

Encoded I^2C Device Command / Index.

pBuffer

Pointer to a buffer that receives the read data. This parameter can be NULL if the data is not required.

BufLen

Size, in bytes, of the buffer pointed to by the pBuffer parameter.

ByteCnt

Size, in bytes, of data to be read.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBuffer==NULL	EAPI_STATUS_INVALID_PARAMETER
ByteCnt==0	EAPI_STATUS_INVALID_PARAMETER
BufLen==0	EAPI_STATUS_INVALID_PARAMETER
Unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6.5 EApiI2CWriteTransfer

```
EC_API
ULONG
EAPI_CALLTYPE
EApiI2CWriteTransfer(
                        ULONG    Id,          /* I2C Bus Id */
                        ULONG    Addr,        /* Encoded 7/10Bit I2C Device Address*/
                        ULONG    Cmd,         /* I2C Command/Offset */
                        void     *pBuffer ,   /* Transfer Data pBuffer */
                        ULONG    ByteCnt      /* Byte Count to write */
);
```

Description:

Write to a specific register in the selected I^2C device. Writes to an I^2C device at the I^2C address Addr the amount of ByteCnt bytes from the buffer *pBuffer while using the device specific command Cmd. Depending on the addressed I^2C device Cmd can be a specific command or a byte offset.

Parameters:

Id

See 'I^2C Bus Ids'.

Addr

Encoded 7/10 Bit I^2C Device Address.

Cmd

Encoded I^2C Device Command / Index.

pBuffer

Pointer to a buffer containing the data to be transferred.

ByteCnt

Size, in bytes, of the information pointed to by the pBuffer parameter.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBuffer==NULL	EAPI_STATUS_INVALID_PARAMETER
ByteCnt=0	EAPI_STATUS_INVALID_PARAMETER
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6.6 EApiI2CProbeDevice

```
EC_API
ULONG
EAPI_CALLTYPE
EApiI2CProbeDevice(
                        ULONG    Id,         /* I2C Bus Id */
                        ULONG    Addr        /* Encoded 7/10Bit I2C Device Address*/
                        Char     *pBytes
                        ULONG    dwLen
);
```

Description:

Probes I^2C address to test I^2C Device present.

Parameters:

Id

See 'I^2C Bus Ids'.

Addr

Encoded 7/10 Bit I^2C Device Address.

pBytes

The pointer to the destination buffer.

dwLen

The number of sequential bytes to read.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Probe Device and no response	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6.7 SusiI2CRead

```
EC_API
ULONG
EAPI_CALLTYPE
SusiI2CRead(
                ULONG   bAddr,
                UCHAR   *pBytes,
                ULONG   dwLen
);
```

Description:

Read a continuous data from I^2C device. (The offset will increase automatically)

Parameters:

bAddr

Encoded 7/10 Bit I^2C Device Address.

pByte

Pointer to a buffer that receives the read data.

dwLen

Size, in bytes, of data to be read.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBytes == NULL	EAPI_STATUS_INVALID_PARAMETER
EC command fail	EAPI_STATUS_READ_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6.8 SusiI2CWrite

```
EC_API
ULONG
EAPI_CALLTYPE
SusiI2CWrite(
                ULONG   bAddr,
                UCHAR   *pBytes,
                ULONG   dwLen
);
```

Description:

Write a continuous data to I^2C device. (The offset will increase automatically)

Parameters:

bAddr

Encoded 7/10 Bit I^2C Device Address.

pByte

Pointer to a buffer containing the data to be transferred.

dwLen

Size, in bytes, of data to be read.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pBytes == NULL	EAPI_STATUS_INVALID_PARAMETER
EC command fail	EAPI_STATUS_READ_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6.9 EApiSetI2CMode

```
EC_API
ULONG
EAPI_CALLTYPE
EApiSetI2CMode(ULONG   dwModeFlag);
```

Description:

Assign I^2C device to be byte or word offset. (Default value is byte offset)

Parameters:

dwModeFlag

The value ‘1’ is byte offset, ‘2’ is word offset.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
(dwModeFlag!=1)||(dwModeFlag!=2)	EAPI_STATUS_INVALID_PARAMETER
Success	EAPI_STATUS_SUCCESS

5.2.6.10 EApiGetI2CSMBFrequency

```
EC_API
ULONG
EAPI_CALLTYPE
EApiGetI2CSMBFrequency (
                                ULONG      Id,
                                PULONG     pFreq,
);
```

Description:

Get I^2C clock frequency.

Parameters:

Id

See ‘I^2C Bus Ids’.

pFreq

Get the current I^2C frequency. (The frequency range is from 0KHz to 400KHz)

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.6.11 EApiSetI2CSMBFrequency

```
EC_API
ULONG
EAPI_CALLTYPE
EApiSetI2CSMBFrequency (
                              ULONG     Id,
                              ULONG     Freq,
);
```

Description:

Get I^2C clock frequency.

Parameters:

Id

See 'I^2C Bus Ids'.

Freq

Update a new frequency of I^2C. (The frequency must be 0 to 100KHz, or 400KHz)

Return Status Code

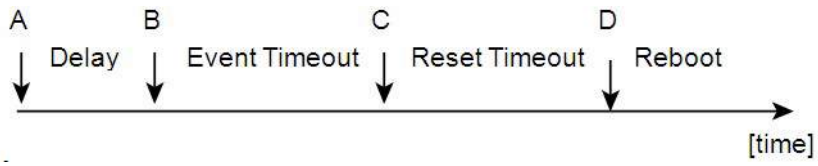
Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
EC command fail	EAPI_STATUS_ERROR
(Freq > 100) && (Freq != 400)	EAPI_STATUS_INVALID_PARAMETER
Success	EAPI_STATUS_SUCCESS

5.2.7 WATCHDOG

After the watchdog timer has been set by the EApiWDogStart function it must be triggered by EApiWDogTrigger within (delay+EventTimeout) milliseconds as set with the EApiWDogStart function, following the initial trigger every subsequent trigger must occur within (EventTimeout) milliseconds. Should EApiWDogTrigger not be called within the relevant time limit a system reset will occur.

The EAPI watchdog timer may support two stages. If the watchdog is not triggered within the event timeout, an NMI, IRQ, or hardware output will be generated. Then the reset timeout becomes active. If the watchdog timer is not triggered within the reset timeout a reset will be generated.

Initial Timing



Timing after EApiWDogTrigger

Stage A

Watchdog is started.

Stage B

Initial Delay Period is exhausted.

Stage C/F

Event is triggered, NMI, IRQ, or PIN is Triggered. To Allow for possible Soft ware Recovery.

Stage D/G

System is reset.

Stage E

- Watchdog is Triggered.
- EApiWDogTrigger / EApiWDogStop Must be called before Stage C/F to prevent event from being generated.
- EApiWDogTrigger / EApiWDogStop Must be called before Stage D/G to prevent The system from being reset.

5.2.7.1 EApiWDogGetCap

```
EC_API
ULONG
EAPI_CALLTYPE
EApiWDogGetCap(
                    ULONG   *pMaxDelay ,          /* Max. supported delay in msec */
                    ULONG   *pMaxEventTimeout, /* Max. supported event timeout
                                                  in msec, 0 == Unsupported*/
                    ULONG   *pMaxResetTimeout /* Max. supported reset timeout in msec*/
);
```

Description:

Get the capabilities of the watchdog timer. (Check the timer's boundary)

Parameters:

pMaxDelay

Pointer to a buffer that receives maximum supported initial delay time of the watchdog timer in milliseconds.

pMaxEventTimeout

Pointer to a buffer that receives maximum supported event timeout of the watchdog timer in milliseconds.

pMaxResetTimeout

Pointer to a buffer that receives maximum supported event timeout of the watchdog timer in milliseconds.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Unsupported	EAPI_STATUS_UNSUPPORTED
pMaxDelay==NULL&& pMaxResetTimeout== NULL&& pMaxEventTimeout==NULL	EAPI_STATUS_INVALID_PARAMETER
Success	EAPI_STATUS_SUCCESS

5.2.7.2 EApiWDogStart

```
EC_API
ULONG
EAPI_CALLTYPE
EApiWDogStart(
                    ULONG     delay,              /* Delay in msec */
                    ULONG     EventTimeout,    /* Timeout in msec*/
                    ULONG     ResetTimeout     /* Reset in msec*/
);
```

Description:

Start the watchdog timer and set the parameters. To adjust the parameters, the watchdog must be stopped via EApiWDogStop and then EApiWDogStart must be called again with the new values. If the hardware implementation of the watchdog timer does not allow a setting at the exact time selected, the EAPI selects the next possible longer timing.

Parameters:

delay

Initial delay for the watchdog timer in milliseconds. The first trigger must happen within (delay + EventTimeout) milliseconds, of calling EApiWDogStart.

EventTimeout

Watchdog timeout interval in milliseconds to trigger an event.

ResetTimeout

Watchdog timeout interval in milliseconds to trigger a reset.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Unsupported	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.7.3 EApiWDogTrigger

```
EC_API
ULONG
EAPI_CALLTYPE
EApiWDogTrigger(void);
```

Description:

Trigger the watchdog timer.

Parameters:

None.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Not Started	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.7.4 EApiWDogStop

```
EC_API
ULONG
EAPI_CALLTYPE
EApiWDogStop(void);
```

Description:

Stops the operation of the watchdog timer.

Parameters:

None.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.7.5 EApiWDogSetEventType

```
EC_API
ULONG
EAPI_CALLTYPE
EApiWDogSetEventType(ULONG   EventType);
```

Description:

To select one kind of event type and set it's timeout, the event type contains delay, IRQ, SCI, SMI, Shutdown and Reset.

Parameters:

EventType

Event Type	Value
SUSI_WDOG_EVENT_DLY	0
SUSI_WDOG_EVENT_IRQ	1
SUSI_WDOG_EVENT_SCI	2
SUSI_WDOG_EVENT_BTN (shutdown)	3
SUSI_WDOG_EVENT_RST	4
SUSI_WDOG_EVENT_NMI (Reserved)	5

Return Status Code

Condition	**Return Value**
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
EventType!=SUSI_WDOG_EVENT_DLY&&	EAPI_STATUS_INVALID_PARAMETER
EventType!=SUSI_WDOG_EVENT_IRQ&&	
EventType!=SUSI_WDOG_EVENT_SCI&&	
EventType!=SUSI_WDOG_EVENT_BTN&&	
EventType!=SUSI_WDOG_EVENT_RST&&	
EventType!=SUSI_WDOG_EVENT_NMI	
Success	EAPI_STATUS_SUCCESS

5.2.7.6 SetCallback

```
EC_API
VOID
EAPI_CALLTYPE
SetCallback(SUSI_WDOG_CALLBACK_EVENT_INT    *pfnCallback);
```

Description:

The call back function pointer can be transmit from Application.

Parameters:

pfnCallback

Call back function pointer, SUSI_WDOG_CALLBACK_EVENT_INT * is function pointer type. The type define just like show below,
typedef void (SUSI_SDOG_CALLBACK_EVENT_INT) ();

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.8 GPIO Functions

COM Express specifies pins for general purpose I/Os. The EAPI provides a set of functions to control these hardware GPIO pins.

5.2.8.1 Common Parameters

Single GPIO addressing

Each GPIO pin can be addressed individually.

Individual GPIO Ids	Description
EAPI_GPIO_ID0	'GPIO 0' Bit mapped to Bit 0
EAPI_GPIO_ID1	'GPIO 1' Bit mapped to Bit 0
EAPI_GPIO_ID2	'GPIO 2' Bit mapped to Bit 0
EAPI_GPIO_ID3	'GPIO 3' Bit mapped to Bit 0
EAPI_GPIO_ID4	'GPIO 4' Bit mapped to Bit 0
EAPI_GPIO_ID5	'GPIO 5' Bit mapped to Bit 0
EAPI_GPIO_ID6	'GPIO 6' Bit mapped to Bit 0
EAPI_GPIO_ID7	'GPIO 7' Bit mapped to Bit 0

Parallel GPIO addressing

A group of selected GPIO pins can be addressed simultaneously.

Multiple GPIO Ids	Description
EAPI_ID_GPIO_BANK00	GPIO 0-31 Bit mapped to Bit 0-31

5.2.8.2 EApiGPIOGetDirectionCaps

```
EC_API
ULONG
EAPI_CALLTYPE
EApiGPIOGetDirectionCaps(
                              ULONG   Id,              /* GPIO Id */
                              ULONG   *pInputs,        /* Supported GPIO Input Bit Mask */
                              ULONG   *pOutputs        /* Supported GPIO Output Bit Mask */
);
```

Description:

Reads the capabilities of the current GPIO implementation from the selected GPIO interface. The ports where both input and output bit masks are 1 are GPIOs. The direction of this ports can be configured by EApiGPIOSetDirection.

Parameters:

Id

See 'GPIO Ids'.

pInputs

Pointer to a buffer that receives the bit mask of the supported inputs.

pOutputs

Pointer to a buffer that receives the bit mask of the supported outputs.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
((pInputs==NULL)&&(pOutputs==NULL))	EAPI_STATUS_INVALID_PARAMETER
Unsupported ID	EAPI_STATUS_UNSUPPORTED
Not Started	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.8.3 EApiGPIOGetDirection

```
EC_API
ULONG
EAPI_CALLTYPE
EApiGPIOGetDirection(
                        ULONG   Id,             /* GPIO Id */
                        ULONG   Bitmask,        /* Bit mask of Affected Bits */
                        ULONG   *pDirection     /* Current Direction */
);
```

Description:

Reads the current configuration of the selected GPIO ports.

Parameters:

Id

See 'GPIO Ids'.

Bitmask

Bit mask.

pDirection

Pointer to a buffer that receives the direction of the selected GPIO ports.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Bitmask==0	EAPI_STATUS_INVALID_PARAMETER
pDirection==NULL	EAPI_STATUS_INVALID_PARAMETER
Unsupported ID	EAPI_STATUS_UNSUPPORTED
Unknown status	EAPI_STATUS_NOT_FOUND
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.8.4 EApiGPIOSetDirection

```
EC_API
ULONG
EAPI_CALLTYPE
EApiGPIOSetDirection(
                        ULONG   Id,         /* GPIO Id */
                        ULONG   Bitmask,    /* Bit mask of Affected Bits*/
                        ULONG   Direction   /* Direction */
);
```

Description:

Sets the configuration for the selected GPIO ports.

Parameters:

Id

See 'GPIO Ids'.

Bitmask

Bit mask.

Direction

Sets the direction of the selected GPIO ports.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Bitmask==0	EAPI_STATUS_INVALID_PARAMETER
(Direction!=0)&&(Direction!=1)	EAPI_STATUS_INVALID_PARAMETER
Unsupported ID	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.8.5 EApiGPIOGetLevel

```
EC_API
ULONG
EAPI_CALLTYPE
EApiGPIOGetLevel(
                    ULONG   Id,         /* GPIO Id */
                    ULONG   Bitmask,    /* Bit mask of Affected Bits*/
                    ULONG   *pLevel     /* Current Level */
);
```

Description:

Read the from GPIO ports.

Parameters:

Id

See 'GPIO Ids'.

Bitmask

Bit mask. Only selected bits are returned.

pLevel

Pointer to a buffer that receives the GPIO level. Results can be read on a bit level.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Bitmask==0	EAPI_STATUS_INVALID_PARAMETER
Unsupported ID	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.8.6 EApiGPIOSetLevel

```
EC_API
ULONG
EAPI_CALLTYPE
EApiGPIOSetLevel(
                        ULONG   Id,         /* GPIO Id */
                        ULONG   Bitmask,    /* Bit mask of Affected Bits
                        ULONG   Level       /* Level */
);
```

Description:

Write to GPIO ports. Depending on the hardware implementation writing multiple GPIO ports with the bit mask option does not guarantee a time synchronous change of the output levels.

Parameters:

Id

See 'GPIO Ids'.

Bitmask

Value for a bit mask. Only selected bits are changed. Unselected bits remain unchanged.

Level

Input level of the selected GPIO port. Output for single ports is on a bit level.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
Bitmask==0	EAPI_STATUS_INVALID_PARAMETER
Unsupported ID	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.9 SmartFan Functions

5.2.9.1 Fan Speed Control

The “SusiFanSetConfigStruct” function call is used to set fan speed configuration. You can use this function to easily control the fan speed. It takes a pointer to an instance of structure SUSIFANCONFIG, which is defined as follows:

```
typedef struct _SUSIAUTOFANCONFIG{
   ULONG dwZone;  // the fan control connects to which thermal source (from 0 to 3)
   ULONG dwOpMode;           // RPM=1 or PWM=0
   ULONG dwLowStopTemp;   /* when the temperature drops to this value, the fan
                             will stop.*/
   ULONG dwLowTemp;         /* when the temperature rises to this value, the fan
                             will work in dwLow* speed. */
   ULONG dwHighTemp;        /* when the temperature rises to this value, the fan
                             will work in dwHigh* speed. */
   ULONG dwLowPWM;   // fan speed in low status using the PWM module
   ULONG dwHighPWM;   // fan speed in high status using the PWM module
   ULONG dwLowRPM;   // fan speed in low status using the RPM module
   ULONG dwHighRPM;   // fan speed in high status using the RPM module
}
typedef struct _SUSIFANCONFIG{
   ULONG dwSize;              /* size of the structure itself, must be initialized with size
                                 of (SUSIFANCONFIG) */
   ULONG dwMode;              // TurnOFF=0, TurnFull=1, TurnManual=2, TurnAuto=3
   ULONG dwPWM;               // fan speed controlled by PWM. (0 to 100)
   SUSIAUTOFANCONFIG safConfig;
} SUSIFANCONFIG, *PSUSIFANCONFIG;
```

5.2.9.2 SusiFanSetConfigStruct

```
EC_API
ULONG
SusiFanSetConfigStruct(
                        ULONG               dwUnit,
                        SUSIFANCONFIG   *pConfig
);
```

Description:

Set auto fan function mode.

Parameters:

dwUnit

The unit number you want to control.

pConfig

Pointer to the auto fan function config.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
dwUnit>1	EAPI_STATUS_INVALID_PARAMETER
pConfig==NULL	EAPI_STATUS_INVALID_PARAMETER
pConfig->dwMode>SUSI_FAN_MODE_AUTO	EAPI_STATUS_INVALID_PARAMETER
pConfig->dwPWM>SUSI_FAN_PWM_MAX	EAPI_STATUS_INVALID_PARAMETER
pConfig->dwZone>SUSI_FAN_MODE_AUTO_ZONE3	EAPI_STATUS_INVALID_PARAMETER
dwHighTemp>SUSI_FAN_TEMP_MAX	EAPI_STATUS_INVALID_PARAMETER
dwLowTemp>SUSI_FAN_TEMP_MAX	EAPI_STATUS_INVALID_PARAMETER
dwLowStopTemp>SUSI_FAN_TEMP_MAX	EAPI_STATUS_INVALID_PARAMETER
dwLowStopTemp>dwLowTemp	EAPI_STATUS_INVALID_PARAMETER
dwHighRPM>SUSI_FAN_RPM_MAX	EAPI_STATUS_INVALID_PARAMETER
dwLowRPM>SUSI_FAN_RPM_MAX	EAPI_STATUS_INVALID_PARAMETER
dwHighPWM>SUSI_FAN_PWM_MAX	EAPI_STATUS_INVALID_PARAMETER
dwLowPWM>SUSI_FAN_PWM_MAX	EAPI_STATUS_INVALID_PARAMETER
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.9.3 SusiFanGetConfigStruct

```
EC_API
ULONG
SusiFanGetConfigStruct(
                        ULONG           dwUnit,
                        SUSIFANCONFIG   *pConfig
);
```

Description:

Get information about auto fan function mode.

Parameters:

dwUnit

The unit number you want to control.

pConfig

Pointer to the auto fan function config.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
dwUnit>1	EAPI_STATUS_INVALID_PARAMETER
pConfig==NULL	EAPI_STATUS_INVALID_PARAMETER
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.10 Thermal Protection Functions

5.2.10.1 Type define

```
typedef struct _THERMALPROTECTNO
{       ULONG     dw_source;
/* Setting thermal source code here can make iManager use this ACPI source to check temperature. */

        ULONG     dw_event_type;
/* This byte can set up a thermal protect event, 0x00 is Shutdown, 0x01 is Throttle, 0x02 is Power off, and 0x08 is No event */

        ULONG     dw_send_event_temperature;
/* 0~120. When thermal source goes over this value, iManager will send event according Event Type. There are also some special values to compatible original ACPI. These can make iManager use ACPI ram as Event temperature. */

        ULONG     dw_clear_event_temperature;
/* 0~120. When thermal source goes below this value and Event is sent, iManager will clear event according Event Type. */

} THERMALPROTECTNO, *PTHERMALPROTECTNO;

Typedef struct _SUSITHERMALCONFIG
{
        THERMALPROTECTNO protect_number[4];
// There are 4 sets of thermal protects.

} SUSITHERMALCONFIG, *PSUSITHERMALCONFIG;
```

5.2.10.2 SusiEC_ThermalProtectionGetConfigStruct

```
EC_API
ULONG
EAPI_CALLTYPE
SusiEC_ThermalProtectionGetConfigStruct(
                                        ULONG                   Id,
                                        SUSITHERMALCONFIG       *pConfig
);
```

Description:

Get Thermal Protection Setting.

Parameters:

Id

Id	Description
EAPI_ID_THERMAL_PROTECTION_0	The first thermal protect area.
EAPI_ID_THERMAL_PROTECTION_1	The second thermal protect area.
EAPI_ID_THERMAL_PROTECTION_2	The third thermal protect area.
EAPI_ID_THERMAL_PROTECTION_3	The fourth thermal protect area.

pConfig

A data package for thermal protect, see SUSITHERMALCONFIG structure.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.10.3 SusiEC_ThermalProtectionSetConfigStruct

```
EC_API
ULONG
EAPI_CALLTYPE
SusiEC_ThermalProtectionSetConfigStruct(
                                        ULONG                   Id,
                                        SUSITHERMALCONFIG       *pConfig
);
```

Description:

Set Thermal Protection configuration.

Parameters:

Id

Id	Description
EAPI_ID_THERMAL_PROTECTION_0	The first thermal protect area.
EAPI_ID_THERMAL_PROTECTION_1	The second thermal protect area.
EAPI_ID_THERMAL_PROTECTION_2	The third thermal protect area.
EAPI_ID_THERMAL_PROTECTION_3	The fourth thermal protect area.

pConfig

A data package for thermal protection; see SUSITHERMALCONFIG structure.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown Id	EAPI_STATUS_UNSUPPORTED
EC command fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11 SMBus Functions

iManager can communicate with 4SMBus channel. System write slave address, command, data, and protocol into corresponding RAM address, then iManager will translate it to SMBus protocol and communicate with SMBus device.

5.2.11.1 Select one Controller

The iManager provides two-way to access SMBus's device, one is access by iManager, another one is access by southbridge, The API's name will be defined as below.

Access by southbridge, don't need any change, Access by iManager, need inside two character "EC" in it. For Example:

Modify SusiSMBusReadByte to SusiECSMBusReadByte

5.2.11.2 SusiSMBusReset

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusReset(void);
```

Description:

SMBus slaves are expected to reset their interface whenever Clock is low for longer than the time out specified in the SMBus specification of 35ms.

Parameters:

None.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.3 SusiSMBusReadByte/SusiECSMBusReadByte

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusReadByte(
                        UCHAR    bAddr,
                        UCHAR    bReg,
                        UCHAR    *pDataByte
);
```

Description:

Read a byte of data from the target slave device in the SMBus.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

bReg

Specifies the offset of the device register to read data from.

pDataByte

Pointer to a variable in which the function reads the byte data.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pDataByte==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.4 SusiSMBusReadWord/SusiECSMBusReadWord

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusReadWord(
                        UCHAR    bAddr,
                        UCHAR    bReg,
                        USHORT   *pDataWord
);
```

Description:

Read a word (2 bytes) of data from the target slave device in the SMBus.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

bReg

Specifies the offset of the device register to word data from.

pDataWord

Pointer to a variable in which the function reads the word data.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
pDataByte==NULL	EAPI_STATUS_INVALID_PARAMETER
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.5 SusiSMBusWriteByte/SusiECSMBusWriteByte

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusWriteByte(
                        UCHAR    bAddr,
                        UCHAR    bReg,
                        UCHAR    bData
);
```

Description:

Write a byte of data to the target slave device in the SMBus.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

bReg

Specifies the offset of the device register to read data from.

bData

Specifies the byte data to be written .

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail by iManager or southbridge	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.6 SusiSMBusWriteWord/SusiECSMBusWriteWord

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusWriteWord(
                        UCHAR     bAddr,
                        UCHAR     bReg,
                        USHORT    wData
);
```

Description:

Write a word (2 bytes) of data to the target slave device in the SMBus.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

bReg

Specifies the offset of the device register to write data to.

wData

Specifies the word data to be written.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.7 SusiSMBusReceiveByte/SusiECSMBusReceiveByte

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusReceiveByte(
                        UCHAR    bAddr,
                        UCHAR    *pDataByte
);
```

Description:

Receive information in a byte from the target slave device in the SMBus

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

pDataByte

Pointer to a variable in which the function receives the byte information.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.8 SusiSMBusSendByte/SusiECSMBusSendByte

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusSendByte(
                        UCHAR    bAddr,
                        UCHAR    bData
);
```

Description:

Send information in bytes to the target slave device in the SMBus.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

bData

Specifies the byte information to be sent.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.9 SusiSMBusWriteQuick/SusiECSMBusWriteQuick

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusWriteQuick(UCHAR bAddr);
```

Description:

Turn SMBus device function off (on) or disable (enable) a specific device mode.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.10 SusiSMBusReadQuick/SusiECSMBusReadQuick

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusReadQuick(UCHAR bAddr);
```

Description:

Turn SMBus device function on (off) or enable (disable) a specific device mode.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.11 SusiSMBusScanDevice/SusiECSMBusScanDevice

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusScanDevice (UCHAR  bAddr_7);
```

Description:

Scan if the address is taken by one of the slave devices currently connected to the SMBus.

Parameters:

bAddr_7

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.12SusiSMBusWriteBlock/SusiECSMBusWriteBlock

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusWriteBlock(
                        UCHAR   bAddr,
                        UCHAR   bReg,
                        UCHAR   *Result,
                        UCHAR   ByteCount
);
```

Description:

Write multi-data to the target slave device in the SMBus.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

bReg

Specifies the offset of the device register to write data to.

Result

Pointer to a byte array in which the function writes the block data.

ByteCount

Specifies the number of bytes to be read.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

5.2.11.13 SusiSMBusReadBlock/SusiECSMBusReadBlock

```
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusReadBlock(
                        UCHAR   bAddr,
                        UCHAR   bReg,
                        UCHAR   *Result,
                        UCHAR   *ByteCount
);
```

Description:

Read multi-data from the target slave device in the SMBus.

Parameters:

bAddr

Specifies the 8-bit device address, ranging from 0x00 - 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

bReg

Specifies the offset of the device register to write data to.

Result

Pointer to a byte array in which the function reads the block data.

ByteCount

Pointer to a byte in which specifies the number of bytes to be read and also return succeed bytes.

Return Status Code

Condition	Return Value
Library Uninitialized	EAPI_STATUS_NOT_INITIALIZED
unknown access method (by iManager)	EAPI_STATUS_UNSUPPORTED
SMBus Controller fail	EAPI_STATUS_ERROR
Success	EAPI_STATUS_SUCCESS

Appendix A

Specification Version Number Format

A.1 Specification Version Number Format

Definition

Bits	Description
[31:24]	Version
[23:16]	Revision
[15:0]	0

Example

Hex	Interpreted
0x03040000	3,4
0x01100000	1,16
0x02010000	2,1

Appendix B

General Version Number Format

B.1 General Version Number Format

Definition

Bits	Description
[31:24]	Major Version
[23:16]	Minor Version
[15:0]	Build Number

Example

Hex	Interpreted
0x03040010	3.4.16
0x01100100	1.16.256
0x02010001	2.1.1

Appendix C

OS Specific Requirements

C.1 Windows

C.1.1 DLL Naming Convention

EAPI_X.dll

X represents the EAPI Specification Version Number

Example

EAPI_1.dll

C.1.2 Version Resource Information

Problem

Due to the nature of the EAPI DLLs in Microsoft Windows, it may not be possible to distinguish one manufacturer's DLL from another. Although it would possible to do this using a tool that uses the API, it may not be possible to load the DLL, due to missing dependencies.

Solution

The solution is to require that Version Resource Information be present for every EAPI DLL. It is then easy to check Manufacturer and versions in the Windows Explorer Properties window.

Appendix D

Linux/Unix Shared Library Naming Convention

D.1 Linux/Unix Shared Library Naming Convention

Problem

Due to the nature of ELF Shared Libraries in Linux/UNIX/... it may not be possible to distinguish one manufacturer's DLL from another. Although it would possible to do this using a tool that uses the API, it may not be possible to load the shared library, due to missing dependencies.

Solution

Filename Convention

libEApiYYY.so.W.Z

Part	Example	Description
YYY	**PMG**	**Vendor PNPID**
W	1	EAPI Specification Version number
X	0	EAPI Specification Revision number

Soname Convention

libEApi.so.W

Part	Example	Description
W	1	EAPI Specification Version number

Example

Shared Library

Filename = libEapiPMG.so.1.0

soname = libEApi.so.1

in file system.

/usr/lib/libEApi.so.1 → /usr/lib/libEapiPMG.so.1.0

/usr/lib/libEapiPMG.so.1.0

see http://EApiDK.sourceforge.net for sample implementation.

D.1.1 ELF/a.out Format Shared Libraries

D.1.1.1 Library Output Format

Problem

Due to the nature of ELF Dynamic Link Libraries in Linux/UNIX/... it may not be possible to distinguish one manufacturer's DLL from another. Although it would possible to do this using a tool that uses the API, It may not be possible to load the DLL, due to missing dependencies.

Solution

The solution is to require the shared libraries be executable. Upon Execution the library should then print out the following information.

```
The Output format is
<Variable name>=Value
and should be matched by this regular expression. m/^\s*(\w+)\s*=\s*(.+)\s*$/
where
$1=Variable name
$2=Value/Data.
```

Variable Name	Description
SVersion	EApi Standard Version used to create Library
LVersion	Vendor Specific Library Version
Manufacturer	Library Manufacturer
MPNPID	Manufacturer PNPID
OFilename	Original File name
Description	Library Description

Sample

```
+----------------------------------------------------------------------+

                    Copyright 2010 ADVANTECH

+----------------------------------------------------------------------+

SVersion=0.5

LVersion=0.5.721

Manufacturer=ADVANTECH

MPNPID=PMG OFilename=libEApiPMG.so.0.5

Description=Embedded Application Programming Interface
```

Appendix E

EAPI ID Definition

```
//////////////////////////////////////////////////////////////////////////
// EAPI ID
// Board information
#define EAPI_ID_BOARD_MANUFACTURER_STR          0x00000000
#define EAPI_ID_BOARD_NAME_STR                  0x00000001
#define EAPI_ID_BOARD_REVISION_STR              0x00000002
#define EAPI_ID_BOARD_SERIAL_STR                0x00000003
#define EAPI_ID_BOARD_BIOS_REVISION_STR         0x00000004
#define EAPI_ID_BOARD_HW_REVISION_STR           0x00000005
#define EAPI_ID_BOARD_PLATFORM_TYPE_STR         0x00000006

//
#define EAPI_ID_GET_EAPI_SPEC_VERSION           0x00000000
#define EAPI_ID_BOARD_BOOT_COUNTER_VAL          0x00000001
#define EAPI_ID_BOARD_RUNNING_TIME_METER_VAL    0x00000002
#define EAPI_ID_BOARD_PNPID_VAL                 0x00000003
#define EAPI_ID_BOARD_PLATFORM_REV_VAL          0x00000004
#define EAPI_ID_BOARD_DRIVER_VERSION_VAL        0x00010000
#define EAPI_ID_BOARD_LIB_VERSION_VAL           0x00010001
#define EAPI_ID_BOARD_FIRMWARE_VERSION_VAL      0x00010002

// Temperature
#define EAPI_ID_HWMON_CPU_TEMP                  0x00020000
#define EAPI_ID_HWMON_CHIPSET_TEMP              0x00020001
#define EAPI_ID_HWMON_SYSTEM_TEMP               0x00020002
#define EAPI_ID_HWMON_CPU_TEMP1                 0x00020010
#define EAPI_ID_HWMON_CHIPSET_TEMP1             0x00020011
#define EAPI_ID_HWMON_SYSTEM_TEMP1              0x00020012
#define EAPI_ID_HWMON_CPU_TEMP2                 0x00020020
#define EAPI_ID_HWMON_CHIPSET_TEMP2             0x00020021
#define EAPI_ID_HWMON_SYSTEM_TEMP2              0x00020022
#define EAPI_ID_HWMON_CPU_TEMP3                 0x00020030
#define EAPI_ID_HWMON_CHIPSET_TEMP3             0x00020031
#define EAPI_ID_HWMON_SYSTEM_TEMP3              0x00020032

// VOLTAGE
#define EAPI_ID_HWMON_VOLTAGE_VCORE             0x00021004
#define EAPI_ID_HWMON_VOLTAGE_2V5               0x00021008
#define EAPI_ID_HWMON_VOLTAGE_3V3               0x0002100C
#define EAPI_ID_HWMON_VOLTAGE_VBAT              0x00021010
#define EAPI_ID_HWMON_VOLTAGE_5V                0x00021014
#define EAPI_ID_HWMON_VOLTAGE_5VSB              0x00021018
```

```
#define EAPI_ID_HWMON_VOLTAGE_12V                0x0002101C

// Fan
#define EAPI_ID_HWMON_FAN_CPU                    0x00022000
#define EAPI_ID_HWMON_FAN_SYSTEM                 0x00022001
#define EAPI_ID_HWMON_FAN_THIRD                  0x00022002

// Backlight
#define EAPI_ID_BACKLIGHT_1                      0x00000000
#define EAPI_ID_BACKLIGHT_2                      0x00000001
#define EAPI_ID_BACKLIGHT_3                      0x00000002
#define EAPI_ID_BACKLIGHT_BY_PCH                 0x0000000F

// GPIO ID
#define EAPI_GPIO_ID0                            0x00000000
#define EAPI_GPIO_ID1                            0x00000001
#define EAPI_GPIO_ID2                            0x00000002
#define EAPI_GPIO_ID3                            0x00000003
#define EAPI_GPIO_ID4                            0x00000004
#define EAPI_GPIO_ID5                            0x00000005
#define EAPI_GPIO_ID6                            0x00000006
#define EAPI_GPIO_ID7                            0x00000007

#define EAPI_ID_GPIO_BANK00                      0x00010000
#define EAPI_ID_GPIO_BANK01                      0x00010001
#define EAPI_ID_GPIO_BANK02v 0x00010002

// I2C Bus
#define EAPI_ID_I2C_EXTERNAL                     0x00000000
#define EAPI_ID_I2C_LVDS_1                       0x00000001
#define EAPI_ID_I2C_LVDS_2                       0x00000002

// Storage
#define EAPI_ID_STORAGE_STD                      0x00000000

// Thermal Protection
#define EAPI_ID_THERMAL_PROTECTION_0             0x00000000
#define EAPI_ID_THERMAL_PROTECTION_1             0x00000001
#define EAPI_ID_THERMAL_PROTECTION_2             0x00000002
#define EAPI_ID_THERMAL_PROTECTION_3             0x00000003

// EAPI ID
////////////////////////////////////////////////////////////////////////////
```

ADVANTECH

Enabling an Intelligent Planet

www.advantech.com

Please verify specifications before quoting. This guide is intended for reference purposes only.

All product specifications are subject to change without notice.