User Manual

iManager 2.0
Software API
Copyright

This document is copyright 2011, by Advantech Co., Ltd. and PICMG. All rights reserved. Advantech Co., Ltd. Reserves the right to make improvements to the products described in this manual at any time. Specifications are subject to change without notice.

No part of this manual may be reproduced, copied, translated, or transmitted in any form or by any means without prior written permission of Advantech Co., Ltd. Information provided in this manual is intended to be accurate and reliable. However, Advantech Co., Ltd., assumes no responsibility for its use, or for any infringements upon the rights of third parties which may result from its use.

All the trademarks of products and companies mentioned in this document belong to their respective owners.

Copyright 2011 Advantech Co., Ltd. All Rights Reserved.
Copyright 2009 PICMG. All rights reserved.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.9 SmartFan Functions</td>
<td>71</td>
</tr>
<tr>
<td>5.2.10 Thermal Protection Functions</td>
<td>74</td>
</tr>
<tr>
<td>5.2.11 SMBus Functions</td>
<td>77</td>
</tr>
<tr>
<td><strong>Appendix A</strong></td>
<td></td>
</tr>
<tr>
<td>A.1 Specification Version Number Format</td>
<td>89</td>
</tr>
<tr>
<td><strong>Appendix B</strong></td>
<td></td>
</tr>
<tr>
<td>B.1 General Version Number Format</td>
<td>91</td>
</tr>
<tr>
<td><strong>Appendix C</strong></td>
<td></td>
</tr>
<tr>
<td>C.1 OS Specific Requirements</td>
<td>93</td>
</tr>
<tr>
<td>C.1.1 Windows</td>
<td>94</td>
</tr>
<tr>
<td>C.1.2 Version Resource Information</td>
<td>94</td>
</tr>
<tr>
<td><strong>Appendix D</strong></td>
<td></td>
</tr>
<tr>
<td>D.1 Linux/Unix Shared Library Naming Convention</td>
<td>95</td>
</tr>
<tr>
<td>D.1.1 ELF/a.out Format Shared Libraries</td>
<td>96</td>
</tr>
<tr>
<td><strong>Appendix E</strong></td>
<td></td>
</tr>
<tr>
<td>EAPI ID Definition</td>
<td>99</td>
</tr>
</tbody>
</table>
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

---

**iManager 2.0**

*(Intelligent Management)*
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 I²C, 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/I²C/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 IManger2.0 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](image)

---

**Figure 3.1 iManager 2.0 Utility Main Window**

- **iManager_WDT**
  - Multi-level watchdog timer for enabling the flexible and intelligent hardware protection.

- **iManager_VGA**
  - Controlling the brightness and backlight for integrated flat panel displays (LVDS Interface)

- **iManager_ThermalProtection**
  - The mechanism of hardware protection feature base on thermal decreasing with CPU throttling technology.

- **iManager_Storage**
  - Implementation of accessing data from secured EEPROM.
Figure 3.2 Function List within the Utility
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.

![Brightness Control](image)

[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.

**Note!** Frequency option will be in effect after change in some specific situations; testing with oscilloscope to determine actual frequency status is strongly recommended.
### 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 to erase 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.

![I²C Protocol](image)

**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: Idle time after platform power on and before watchdog timer is enabled.

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, warm reset, 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 warm reset.
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, both 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 trigger a warm reset to restart the platform.

- **Case 2:**
  **DEMO: Multi-stage watchdog timer level testing.**
  **Scenario:**
  Click the “Start” button; the watchdog delay timeout will start counting down to zero, then enter into the 2nd stage “event timeout,” and start counting down; when the counter stops at zero, the IRQ event we’ve designed will be triggered and a message box will appear. At this moment you may click the “Trigger” button manually to send the trigger for resetting the timer. If you stop clicking the “Trigger” button, after the counter has counted down to zero, then it will automatically enter the last stage: a warm reboot of the platform.

**Note!** The SCI/SMI event: depends on the BIOS setting, the event is warm 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 for testing CASE 1 BSOD DEMO.

**Note!**
- Warm reset: Sends a hardware reset signal without turning off power to the platform.
- Cold reset: Sends a power down signal to platform, and then sends a power up signal to boot up the machine.
- Trigger has no effect on delay timeout
- When trigger is submitted while counting in either event timeout or reset timeout stage, the timeout flag will be reset to beginning of event timeout.
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 4.
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.

**Note!** Please make sure the device’s slave address is correct and writable before actually writing data! Otherwise, permanent hardware damage may result.

### 4.6 SMBus

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

![SMBus Interface](image)

**[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 by using Quick Protocol. 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 I²C and SMBus, the SMBus DEMO utility demonstrate the default scenario of the accessing to SMBus device.

Note! Please make sure the device's slave address is correct and writable before actually writing data! Otherwise, permanent hardware damage may result.
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.

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.
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 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.*

**Note!** *Thermal zone (or Thermal Source) represents the "thermal sensor presets" for smart fan speed reference; there are 4 thermal zones available for configuration through the API. By factory default zone 0 and zone 1 are configured and fixed to refer CPU sensor for both CPU fan and system fan.*
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: Power Button (sends power button signal to the OS to shut down the device); Throttle (forces CPU frequency to the lowest setting: for example: 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: unplug 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.

![HWMonitor]

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Voltage</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMOS Battery</td>
<td>2.89</td>
<td>2.89</td>
<td>2.89</td>
<td>V</td>
</tr>
<tr>
<td>5VSB</td>
<td>5.136</td>
<td>5.136</td>
<td>5.142</td>
<td>V</td>
</tr>
<tr>
<td>12V</td>
<td>11.94</td>
<td>11.94</td>
<td>11.93</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Temperature</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>54</td>
<td>43</td>
<td>57</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Fan Speed</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2725</td>
<td>2467</td>
<td>3045</td>
<td>RPM</td>
</tr>
<tr>
<td>System</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>RPM</td>
</tr>
</tbody>
</table>

[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. Additionally, 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.

### Information

<table>
<thead>
<tr>
<th>Information</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board Manufacturer Name</td>
<td>Advantech</td>
</tr>
<tr>
<td>Board Name</td>
<td>SCM-5820</td>
</tr>
<tr>
<td>Serial Number</td>
<td>EPA00000001</td>
</tr>
<tr>
<td>Board BIOS Revision</td>
<td>X009</td>
</tr>
<tr>
<td>Platform Type</td>
<td>COMExpress</td>
</tr>
<tr>
<td>Board Platform Specification Revision</td>
<td>2.0</td>
</tr>
<tr>
<td>Boot Counter</td>
<td>31.1</td>
</tr>
<tr>
<td>Running Time Meter (hour)</td>
<td>200616292</td>
</tr>
<tr>
<td>Board Vendor PNPID</td>
<td>(Empty)</td>
</tr>
<tr>
<td>EAPI Specification Version</td>
<td>1.0</td>
</tr>
<tr>
<td>EAPI Driver Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>EAPI Library Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>iManager Firmware Version</td>
<td>17.7.768</td>
</tr>
</tbody>
</table>

**Board Information:**

Access Information of the hardware platform from EEPROM, SPI ROM, iManager Driver and Library. All params 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

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
<td>The EAPI library is not yet or unsuccessfully initialized. EApiLibInitialize needs to be called prior to the first access of any other EAPI function.</td>
<td>Call EApiLibInitialize.</td>
</tr>
<tr>
<td>EAPI_STATUS_INITIALIZED</td>
<td>Library is initialized.</td>
<td>None.</td>
</tr>
<tr>
<td>EAPI_STATUS_ALLOC_ERROR</td>
<td>Memory Allocation Error.</td>
<td>Free memory and try again.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Actions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>EAPI_STATUS_SW_TIMEOUT</td>
<td>Software timeout. This is Normally caused by hardware/software semaphore timeout.</td>
<td>Retry.</td>
</tr>
<tr>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
<td>One or more of the EAPI function call parameters are out of the defined range.</td>
<td>Verify Function Parameters.</td>
</tr>
<tr>
<td>EAPI_STATUS_INVALID_BLOCK_LENGTH</td>
<td>This means that the Block length is too long.</td>
<td>Use relevant Capabilities information to correct select block lengths.</td>
</tr>
<tr>
<td>EAPI_STATUS_INVALID_BLOCK_ALIGNMENT</td>
<td>The Block Alignment is incorrect.</td>
<td>Use Alignment Capabilities information to correctly align write access.</td>
</tr>
<tr>
<td>EAPI_STATUS_INVALID_DIRECTION</td>
<td>The current Direction Argument attempts to set GPIOs to a unsupported directions. I.E. Setting GPI to Output.</td>
<td>Use pinputs and pOutputs to correctly select input and outputs.</td>
</tr>
<tr>
<td>EAPI_STATUS_INVALID_BITMASK</td>
<td>The Bitmask Selects bits/GPIOs which are not supported for the current ID.</td>
<td>Use pinputs and pOutputs to probe supported bits.</td>
</tr>
<tr>
<td>EAPI_STATUS_UNSUPPORTED</td>
<td>This function or ID is not supported at the actual hardware environment.</td>
<td>None.</td>
</tr>
</tbody>
</table>
**EAPI_STATUS_NOT_FOUND**

**Description**
Selected device was not found.

**Example**
The I²C 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 I²C bus is busy or there is a bus collision.
- The I²C 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**
I²C operation lasted too long.

**Actions**
Retry.

---

**EAPI_STATUS_READ_ERROR**

**Description**
An error was detected during a read operation.

**Example**
I²C Read function was not successful.

**Actions**
Retry.
<table>
<thead>
<tr>
<th><strong>EAPI_STATUS_WRITE_ERROR</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An error was detected during a write operation.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
</tr>
<tr>
<td>I2C Write function was not successful.</td>
<td></td>
</tr>
<tr>
<td>No Acknowledge was received after writing any byte after the first address byte.</td>
<td></td>
</tr>
<tr>
<td>Can be caused by unsupported device command/index.</td>
<td></td>
</tr>
<tr>
<td>10Bit Address Device Not Present</td>
<td></td>
</tr>
<tr>
<td>Storage Write Error</td>
<td></td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>Retry.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>EAPI_STATUS_MORE_DATA</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The amount of available data exceeds the buffer size. Storage buffer overflow was prevented. Read count was larger than the defined buffer length.</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>Either increase the buffer size or reduce the block length.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>EAPI_STATUS_ERROR</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generic error message. No further error details are available.</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>EAPI_STATUS_SUCCESS</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The value for this status code is defined as 0.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The operation was successful.</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>None.</td>
</tr>
</tbody>
</table>
5.2 EAPI

5.2.1 Define

5.2.2 Initialization Functions

5.2.2.1 EApiLibInitialize

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library initialized</td>
<td>EAPI_STATUS_INITIALIZED</td>
</tr>
<tr>
<td>Library initial fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
### 5.2.2.2 EApiLibUnInitialize

```c
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 uninitialized routines. It is expected that in this environments this function has no functionality.

**Parameters:**
None.

**Return Status Code**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.3 EAPI Information Functions

5.2.3.1 EApiBoardGetStringA

```c
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.

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBuffer==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>*pBufLen==NULL &amp; pBuffer==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.3.2  EApiBoardGetValue

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.

<table>
<thead>
<tr>
<th>Id</th>
<th>Description</th>
<th>Units/Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_ID_GET_EAPI_SPEC-version</td>
<td>EAPI Specification Version used to implement API</td>
<td>Spec. Version</td>
</tr>
<tr>
<td>EAPI_ID_BOARD_BOOT_COUNTER_VA</td>
<td>Boot Counter</td>
<td>boots1</td>
</tr>
<tr>
<td>EAPI_ID_BOARD_RUNNING_TIME_Metadata</td>
<td>'Running Time Meter'</td>
<td>hours1</td>
</tr>
<tr>
<td>EAPI_ID_BOARD_PNPID_VAL</td>
<td>Board Vendor PNPID</td>
<td>Not Support Yet</td>
</tr>
<tr>
<td>EAPI_ID_BOARD_PLATFORM_REV_VA</td>
<td>Platform Specification Version used to create board.</td>
<td>Spec. Version</td>
</tr>
<tr>
<td>EAPI_ID_BOARD_DRIVER_VERSION_VA</td>
<td>Vendor Specific Driver Version</td>
<td>Spec. Version</td>
</tr>
<tr>
<td>EAPI_ID_BOARD_LIB_VERSION_VAL</td>
<td>Vendor Specific Library Version</td>
<td>Spec. Version</td>
</tr>
<tr>
<td>EAPI_ID_BOARD_FIRMWARE_VERSION_VAL</td>
<td>EC firmware Version</td>
<td>Spec. Version</td>
</tr>
<tr>
<td>EAPI_ID_HWMON_CPU_TEMP</td>
<td>CPU Temperature</td>
<td>1 centigrade</td>
</tr>
<tr>
<td>EAPI_ID_HWMON_CHIPSET_TEMP</td>
<td>Chipset Temperature</td>
<td>1 centigrade</td>
</tr>
<tr>
<td>EAPI_ID_HWMON_SYSTEM_TEMP</td>
<td>System Temperature</td>
<td>1 centigrade</td>
</tr>
<tr>
<td>EAPI_ID_HWMON_VOLTAGE_VCORE</td>
<td>CPU Core Voltage</td>
<td>volts1</td>
</tr>
<tr>
<td>EAPI_ID_HWMON_VOLTAGE_2V5</td>
<td>2.5V Voltage</td>
<td>volts1</td>
</tr>
<tr>
<td>Condition</td>
<td>Return Value</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------</td>
<td></td>
</tr>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
<td></td>
</tr>
<tr>
<td>pValue==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
<td></td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
<td></td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
<td></td>
</tr>
</tbody>
</table>

Return Status Code

| 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       |
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.

<table>
<thead>
<tr>
<th>Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_ID_BACKLIGHT_1</td>
<td>Backlight Local Flat Panel 1</td>
</tr>
<tr>
<td>EAPI_ID_BACKLIGHT_2</td>
<td>Backlight Local Flat Panel 2</td>
</tr>
<tr>
<td>EAPI_ID_BACKLIGHT_3</td>
<td>Backlight Local Flat Panel 3</td>
</tr>
</tbody>
</table>

**Backlight Enable Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_BACKLIGHT_SET_ON</td>
<td>Requests/Signifies that the Backlight be Enabled</td>
</tr>
<tr>
<td>EAPI_BACKLIGHT_SET_OFF</td>
<td>Requests/Signifies that the Backlight be Disabled</td>
</tr>
</tbody>
</table>

5.2.4.2 **EApiVgaGetBacklightEnable**

```c
EC_API
ULONG
EAPICALLTYPE
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pEnable==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.4.3  **EApiVgaSetBacklightEnable**

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Enable!=EAPI_BACKLIGHT_SET_ON &amp;&amp;</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Enable!=EAPI_BACKLIGHT_SET_OFF</td>
<td></td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>


5.2.4.4 **EApiVgaGetBacklightBrightness**

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBright==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Bright&gt;EAPI_BACKLIGHT_SET_BRIGHTEST</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Bright&gt;EAPI_BACKLIGHT_SET_BRIGHTEST</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.4.7  EApiVgaSetPolarity

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>(dwSetting != 0)</td>
<td></td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBright==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.4.9 EApiVgaSetBacklightLevel

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Level&gt;EAPI_BACKLIGHT_SET_LEVELEST</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

C code:

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

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_ID_STORAGE_STD</td>
<td>Standard Storage Area</td>
</tr>
</tbody>
</table>

5.2.5.2 **EApiStorageCap**

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>((pStorageSize==NULL)&amp;&amp;(pBlockLength==NULL))</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Unsupported Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.5.3  EApiStorageAreaRead

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBuffer==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>ByteCnt==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>BufLen==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>Offset+ByteCnt&gt;pStorageSize</td>
<td>EAPI_STATUS_INVALID_BLOCK_LENGTH</td>
</tr>
<tr>
<td>Read Error</td>
<td>EAPI_STATUS_READ_ERROR</td>
</tr>
<tr>
<td>ByteCnt&gt;BufLen</td>
<td>EAPI_STATUS_MORE_DATA</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.5.4 **EApiStorageAreaWrite**

```c
EC_API
ULONGLONG
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBuffer==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>ByteCnt==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>Offset+ByteCnt&gt;pStorageSize</td>
<td>EAPI_STATUS_INVALID_BLOCK_LENGTH</td>
</tr>
<tr>
<td>Write Error</td>
<td>EAPI_STATUS_WRITE_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.5.5  

**SusiStorageAreaIsLocked**

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.5.6 SusiStorageAreaLock

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBytes==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwLen&gt;EEPROM_SECURE_KEY_SIZE</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.5.7 SusiStorageAreaUnlock

**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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBytes==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwLen&gt;EEPROM_SECURE_KEY_SIZE</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_ID_I2C_EXTERNAL</td>
<td>Baseboard I²C Interface</td>
</tr>
<tr>
<td>EAPI_ID_I2C_LVDS_1</td>
<td>LVDS/EDP 1 Interface</td>
</tr>
<tr>
<td>EAPI_ID_I2C_LVDS_2</td>
<td>LVDS/EDP 2 Interface</td>
</tr>
</tbody>
</table>

5.2.6.2 EapiI2CGetBusCap

```c
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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pMaxBlkLen==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.6.3 EapiI2CWriteReadRaw

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>(WriteBCnt&gt;1)&amp;&amp;(pWBuffer==NULL)</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>(ReadBCnt&gt;1)&amp;&amp;(pRBuffer==NULL)</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>(ReadBCnt&gt;1)&amp;&amp;(RBufLen==0)</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>((WriteBCnt==0)&amp;&amp;(ReadBCnt==0))</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.6.4   Eapi2CReadTransfer

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

**Parameters:**
- **Id**: See ‘\( \text{I}^2\text{C} \) Bus Ids’.
- **Addr**: Encoded 7/10 Bit \( \text{I}^2\text{C} \) Device Address.
- **Cmd**: Encoded \( \text{I}^2\text{C} \) 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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBuffer==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>ByteCnt==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>BufLen==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.6.5 EApiI2CWriteTransfer

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

**Parameters:**

**Id**
See 'I2C Bus Ids'.

**Addr**
Encoded 7/10 Bit I2C Device Address.

**Cmd**
Encoded $i^2$C 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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBuffer==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>ByteCnt==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.6.6 **EApiI2CProbeDevice**

```c
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 I2C address to test I2C Device present.

**Parameters:**
- **Id**
  - See ‘I2C Bus Ids’.
- **Addr**
  - Encoded 7/10 Bit I2C Device Address.
- **pBytes**
  - The pointer to the destination buffer.
- **dwLen**
  - The number of sequential bytes to read.

**Return Status Code**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Probe Device and no response</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.6.7 Susii2CRead

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBytes == NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_READ_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

```c
EC_API
ULONG
EAPI_CALLTYPE
Susii2CRead(
    ULONG     bAddr,
    UCHAR     *pBytes,
    ULONG     dwLen
);
```
5.2.6.8 SusiI2CWrite

**Description:**
Write a continuous data to \( \text{i}^2\text{C} \) device. (The offset will increase automatically)

**Parameters:**
- **bAddr**
  - Encoded 7/10 Bit \( \text{i}^2\text{C} \) 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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pBytes == NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_READ_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

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

```c
SusiI2CWrite(ulong bAddr, uchar *pBytes, ulong dwLen);
```
5.2.6.9 EApiSetI2CMode

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

**Description:**
Assign i²C 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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>(dwModeFlag!=1)</td>
<td></td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

5.2.6.10 EApiGetI2CSMBFrequency

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

**Description:**
Get i²C clock frequency.

**Parameters:**
- `Id`
  See ‘i²C Bus Ids’.
- `pFreq`
  Get the current i²C frequency. (The frequency range is from 0KHz to 400KHz)

**Return Status Code**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.6.11 EApiSetI2CSMBFrequency

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

Description:
Get I²C clock frequency.

Parameters:
Id
See "I²C Bus Ids".
Freq
Update a new frequency of I²C. (The frequency must be 0 to 100KHz, or 400KHz)

Return Status Code

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>(Freq &gt; 100) &amp;&amp; (Freq != 400)</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
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**

![Initial Timing Diagram](image)

**Timing after EApiWDogTrigger**

![Timing Diagram](image)

**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 Software 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

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Unsupported</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>pMaxDelay==NULL&amp;&amp; pMaxResetTimeout==NULL&amp;&amp; pMaxEventTimeout==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
### 5.2.7.2 EApiWDogStart

**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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Unsupported</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

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

**Description:**
Trigger the watchdog timer.

**Parameters:**
None.

**Return Status Code**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Not Started</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

5.2.7.4 EApiWDogStop

**Description:**
Stops the operation of the watchdog timer.

**Parameters:**
None.

**Return Status Code**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.7.5 EApiWDogSetEventType

**Description:**
To select one kind of event type and set its timeout, the event type contains delay, IRQ, SCI, SMI, Shutdown and Reset.

**Parameters:**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUSI_WDOG_EVENT_DLY</td>
<td>0</td>
</tr>
<tr>
<td>SUSI_WDOG_EVENT_IRQ</td>
<td>1</td>
</tr>
<tr>
<td>SUSI_WDOG_EVENT_SCI</td>
<td>2</td>
</tr>
<tr>
<td>SUSI_WDOG_EVENT_BTN (shutdown)</td>
<td>3</td>
</tr>
<tr>
<td>SUSI_WDOG_EVENT_RST</td>
<td>4</td>
</tr>
<tr>
<td>SUSI_WDOG_EVENT_NMI (Reserved)</td>
<td>5</td>
</tr>
</tbody>
</table>

**Return Status Code**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>EventType!=SUSI_WDOG_EVENT_DLY&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>EventType!=SUSI_WDOG_EVENT_IRQ&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>EventType!=SUSI_WDOG_EVENT_SCI&amp;&amp;</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>EventType!=SUSI_WDOG_EVENT_BTN&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>EventType!=SUSI_WDOG_EVENT_RST&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>EventType!=SUSI_WDOG_EVENT_NMI</td>
<td></td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

```c
EC_API
ULONG
EAPI_CALLTYPE
EApiWDogSetEventType(ULONG EventType);
```
5.2.7.6 SetCallback

```c
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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Individual GPIO Ids</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_GPIO_ID0</td>
<td>‘GPIO 0’ Bit mapped to Bit 0</td>
</tr>
<tr>
<td>EAPI_GPIO_ID1</td>
<td>‘GPIO 1’ Bit mapped to Bit 0</td>
</tr>
<tr>
<td>EAPI_GPIO_ID2</td>
<td>‘GPIO 2’ Bit mapped to Bit 0</td>
</tr>
<tr>
<td>EAPI_GPIO_ID3</td>
<td>‘GPIO 3’ Bit mapped to Bit 0</td>
</tr>
<tr>
<td>EAPI_GPIO_ID4</td>
<td>‘GPIO 4’ Bit mapped to Bit 0</td>
</tr>
<tr>
<td>EAPI_GPIO_ID5</td>
<td>‘GPIO 5’ Bit mapped to Bit 0</td>
</tr>
<tr>
<td>EAPI_GPIO_ID6</td>
<td>‘GPIO 6’ Bit mapped to Bit 0</td>
</tr>
<tr>
<td>EAPI_GPIO_ID7</td>
<td>‘GPIO 7’ Bit mapped to Bit 0</td>
</tr>
</tbody>
</table>

**Parallel GPIO addressing**
A group of selected GPIO pins can be addressed simultaneously.

<table>
<thead>
<tr>
<th>Multiple GPIO Ids</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_ID_GPIO_BANK00</td>
<td>GPIO 0-31 Bit mapped to Bit 0-31</td>
</tr>
</tbody>
</table>
5.2.8.2 EApiGPIOGetDirectionCaps

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>((pInputs==NULL)&amp;&amp;(pOutputs==NULL))</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Unsupported ID</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>Not Started</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.8.3 EApiGPIOGetDirection

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Bitmask==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>pDirection==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Unsupported ID</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>Unknown status</td>
<td>EAPI_STATUS_NOT_FOUND</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Bitmask==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>(Direction!=0)&amp;&amp;(Direction!=1)</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Unsupported ID</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Bitmask==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Unsupported ID</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.8.6 **EApiGPIOSetLevel**

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>Bitmask==0</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>Unsupported ID</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
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:

```c
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
} SUSIAUTOFANCONFIG;

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

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>dwUnit&gt;1</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>pConfig==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>pConfig-&gt;dwMode&gt;SUSI_FAN_MODE_AUTO</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>pConfig-&gt;dwPWM&gt;SUSI_FAN_PWM_MAX</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>pConfig-&gt;dwZone&gt;SUSI_FAN_MODE_AUTO_ZONE3</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwHighTemp&gt;SUSI_FAN_TEMP_MAX</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwLowTemp&gt;SUSI_FAN_TEMP_MAX</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwLowStopTemp&gt;dwLowTemp</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwHighRPM&gt;SUSI_FAN_RPM_MAX</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwLowRPM&gt;SUSI_FAN_RPM_MAX</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwHighPWM&gt;SUSI_FAN_PWM_MAX</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>dwLowPWM&gt;SUSI_FAN_PWM_MAX</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.9.3 SusiFanGetConfigStruct

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>dwUnit&gt;1</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>pConfig==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.10.3 SusiEC_ThermalProtectionSetConfigStruct

**Description:**
Set Thermal Protection configuration.

**Parameters:**

<table>
<thead>
<tr>
<th>Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAPI_ID_THERMAL_PROTECTION_0</td>
<td>The first thermal protect area.</td>
</tr>
<tr>
<td>EAPI_ID_THERMAL_PROTECTION_1</td>
<td>The second thermal protect area.</td>
</tr>
<tr>
<td>EAPI_ID_THERMAL_PROTECTION_2</td>
<td>The third thermal protect area.</td>
</tr>
<tr>
<td>EAPI_ID_THERMAL_PROTECTION_3</td>
<td>The fourth thermal protect area.</td>
</tr>
</tbody>
</table>

**pConfig**
A data package for thermal protection; see SUSITHERMALCONFIG structure.

**Return Status Code**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown Id</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>EC command fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11 SMBus Functions

iManager can communicate with 4 SMBus 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

```c
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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.3 SusiSMBusReadByte/SusiECSMBusReadByte

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pDataByte==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.4 SusiSMBusReadWord/SusiECSMBusReadWord

**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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>pDataByte==NULL</td>
<td>EAPI_STATUS_INVALID_PARAMETER</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

```c
EC_API
ULONG
EAPI_CALLTYPE
SusiSMBusReadWord(
    UCHAR     bAddr,
    UCHAR     bReg,
    USHORT   *pDataWord
);
```
5.2.11.5 SusiSMBusWriteByte/SusiECSMBusWriteByte

**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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail by iManager or southbridge</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.6 SusiSMBusWriteWord/SusiECSMBusWriteWord

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.7 SusiSMBusReceiveByte/SusiECSMBusReceiveByte

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
### 5.2.11.8 SusiSMBusSendByte/SusiECSMBusSendByte

**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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.9 SusiSMBusWriteQuick/SusiECSMBusWriteQuick

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>

5.2.11.10 SusiSMBusReadQuick/SusiECSMBusReadQuick

```c
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**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.11 SusiSMBusScanDevice/SusiECSMBusScanDevice

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.12 SusiSMBusWriteBlock/SusiECSMBusWriteBlock

```c
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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
5.2.11.13 SusiSMBusReadBlock/SusiECSMBusReadBlock

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library Uninitialized</td>
<td>EAPI_STATUS_NOT_INITIALIZED</td>
</tr>
<tr>
<td>unknown access method (by iManager)</td>
<td>EAPI_STATUS_UNSUPPORTED</td>
</tr>
<tr>
<td>SMBus Controller fail</td>
<td>EAPI_STATUS_ERROR</td>
</tr>
<tr>
<td>Success</td>
<td>EAPI_STATUS_SUCCESS</td>
</tr>
</tbody>
</table>
Appendix A

Specification Version
Number Format
## A.1 Specification Version Number Format

### Definition

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:24]</td>
<td>Version</td>
</tr>
<tr>
<td>[23:16]</td>
<td>Revision</td>
</tr>
<tr>
<td>[15:0]</td>
<td>0</td>
</tr>
</tbody>
</table>

### Example

<table>
<thead>
<tr>
<th>Hex</th>
<th>Interpreted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x03040000</td>
<td>3,4</td>
</tr>
<tr>
<td>0x01100000</td>
<td>1,16</td>
</tr>
<tr>
<td>0x02010000</td>
<td>2,1</td>
</tr>
</tbody>
</table>
B.1 General Version Number Format

**Definition**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:24]</td>
<td>Major Version</td>
</tr>
<tr>
<td>[15:0]</td>
<td>Build Number</td>
</tr>
</tbody>
</table>

**Example**

<table>
<thead>
<tr>
<th>Hex</th>
<th>Interpreted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x03040010</td>
<td>3.4.16</td>
</tr>
<tr>
<td>0x01100100</td>
<td>1.16.256</td>
</tr>
<tr>
<td>0x02010001</td>
<td>2.1.1</td>
</tr>
</tbody>
</table>
C.1 Windows

C.1.1 DLL Naming Convention

<table>
<thead>
<tr>
<th>EAPI_X.dll</th>
</tr>
</thead>
<tbody>
<tr>
<td>X represents the EAPI Specification Version Number</td>
</tr>
</tbody>
</table>

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**

\[ \text{libEApiYYY.so.W.Z} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYY</td>
<td>PMG</td>
<td>Vendor PNPID</td>
</tr>
<tr>
<td>W</td>
<td>1</td>
<td>EAPI Specification Version number</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>EAPI Specification Revision number</td>
</tr>
</tbody>
</table>

**Soname Convention**

\[ \text{libEApi.so.W} \]

<table>
<thead>
<tr>
<th>Part</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>1</td>
<td>EAPI Specification Version number</td>
</tr>
</tbody>
</table>

**Example**

Shared Library

- Filename = libEapiPMG.so.1.0
- soname = libEApi.so.1

in file system.

\[ /usr/lib/libEApi.so.1 \rightarrow /usr/lib/libEapiPMG.so.1.0 \]

\[ /usr/lib/libEapiPMG.so.1.0 \]

see [http://EApiDK.sourceforge.net](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
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OutputFormat>
  <Variable name>="Value"
and should be matched by this regular expression. m/\s*(\w+)\s*="\s*(.+)\s*"/ where
  %1=Variable name
  %2=Value/Data.
```

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVersion</td>
<td>EApi Standard Version used to create Library</td>
</tr>
<tr>
<td>LVersion</td>
<td>Vendor Specific Library Version</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Library Manufacturer</td>
</tr>
<tr>
<td>MPNPID</td>
<td>Manufacturer PN PID</td>
</tr>
<tr>
<td>OFilename</td>
<td>Original File name</td>
</tr>
<tr>
<td>Description</td>
<td>Library Description</td>
</tr>
</tbody>
</table>

**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_BANK02 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
//////////////////////////////////////////////////////////////////////////