KT-API-V2 User Manual

SW utility designed for the following Motherboard families:

886LCD-M
986LCD-M
KT690
KTUS15
KT965
KTGM45
KTQ45
KTG41
KT780

Supported by
DOS and Linux* (32B/64B)

* Tested on openSUSE 11.1

Document revision history.

Revision	Date	By	Comment
E	Nov.23rd 2010	OLA/MLA	Revision of table page 4. Updated IHH.
D	Dec. 10th 2009	OLA/MLA	Added info on ReadHWMonitorItems values and other minor details.
C	Nov. 19th 2009	OLA/MLA	Updated support table.
B	Feb. 5th 2009	OLA/MLA	Updated support table, API overview, API function detailed descriptions and KT-API-V2 package content. Other minor changes
A	Oct. 30th 2008	OLA/MLA	Info regarding available examples. Layout and minor changes.
0	Oct. 27th 2008	OLA/MLA	Preliminary version.

Life Support Policy

KONTRON Technology's PRODUCTS ARE NOT FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT EXPRESS WRITTEN APPROVAL OF THE GENERAL MANAGER OF KONTRON Technology A/S.

As used herein:
1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into body, or (b) support or sustain life and whose failure to perform, when properly used in accordance with instructions for use provided in the labelling, can be reasonably expected to result in significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

Table of contents

1. Introduction

The KT-API-V2 is a software API utility designed for Kontron Motherboards. The final application software based on these API’s will only run correctly on Kontron Motherboards supporting all the used API’s, see table below.

Using the API’s makes it possible for OEM customers to design software application accessing onboard features in order to monitor and control different functionalities like Fan speed, CPU temperature, GPIO’s, Watchdog, Monitoring voltages, Backlight Intensity, SMBus etc.

The Utility can be used to implement applications in a DOS/Windows/Linux environment. Please notice that the API’s will soon be available in a dll-file for Windows and maybe something similar for Linux.

Not all functions are available for all boards as shown in the following table.

API functions	**886LCD-M (EOL)**	**986LCD-M**	**KT690**	**KTUS15**	**KT965**	**KTGM45**	**KTQ45**	**KTG41**	**KT780**
ReadMonitor (see note)		x		x					
ReadHWMonitorItems			x	x	x	x	x		
GetHWMonitorItem			x	x	x	x	x		
SetClrGPIO	x	x	x	x	x	x	x		
ReadGPIO	x	x	x	x	x	x	x		
SetGPIODir	x	x	x	x	x	x	x		
SetFanSpeed		x	x	x	x	x	x		
EnableWD		x	x	x	x	x	x		
DisableWD		x	x	x	x	x	x		
SetWDTimer		x	x	x	x	x	x		
SetCPUThrottle		x		x					
SetBKLControl		x		x					
ReadBoardHeader	x	x	x	x	x	x	x		
GetBoardName -A -W	x	x	x	x	x	x	x		
GetSerialNumber -A -W	x	x	x	x	x	x	x		
GetPartNumber -A -W	x	x	x	x	x	x	x		
SelectFanTempTacChannel		x	x	x	x	x	x		
GetIntruderStatus		x	x	x	x	x	x		
ClrIntruderStatus		x	x	x	x	x	x		
StartThermalCruise		x	x	x	x	x	x		
GetMac		x		x			x		
SetFanTarget		x	x	x	x	x	x		
SetFanMode		x	x	x	x	x	x		
SmBus_RecvByte		x	x	x	x	x	x		
SmBus_ReadByte		x	x	x	x	x	x		
SmBus_ReadWord		x	x	x	x	x	x		
SmBus_ReadBlock		x	x	x	x	x	x		
SmBus_ReadBytes		x	x	x	x	x	x		
SmBus_SendByte		x	x	x	x	x	x		
SmBus_WriteByte		x	x	x	x	x	x		
SmBus_WriteWord		x	x	x	x	x	x		
SmBus_WriteBlock		x	x	x	x	x	x		
SmBus_PorcessCall		x	x	x	x	x	x		

Notes: The SmBus API’s are supporting the SmBus available on the Feature Connector only.
The greyed API function is EOL and will be removed in future KT-API-V2 package.

KT-API-V2 package content

The KT-API-V2 package contains the following file structure

```
├bin ──────────┐
│              ├linux ─────────┐
│              │               ktapi.bin
│              │               installdrv
│              │               uninstalldrv
│              │
│              └DOS────────────┐
│                              dosapi.exe
│                              dosapiex.exe
│                              dosrtc.exe
│                              ktapi.bin
│                              installdrv
│                              uninstalldrv
│                              ktapu.ko
│
├core ─────────┐
│              ├os ────────────┐
│              │               ├dos ───────────┐
│              │               │               dos.c
│              │               │
│              │               ├linux ─────────┐
│              │               │               ├ldrv ──────────┐
│              │               │               │               build
│              │               │               │               ktio.c
│              │               │               │               Makefile
│              │               │               │
│              │               │               linux.c
│              │               │
│              │               ├windows ───────┐
│              │               │               win.c
│              │               │               win.h
│              │               │
│              │               os.h
│              │
│              api.c
│              api.h
│
├examples ─────┐
│              ├apirtc ────────┐
│              │               apirtc.c
│              │
│              ├apitest ───────┐
│              │               apitest.c
│              │
│              └apitestex ─────┐
│                              apitestex.c
│
├obj
│
build
MAKEFILE
```

2. Installation

Depending on the OS environment the following must be noticed:

DOS	No installation needed.
Linux	The ktapi.ko driver needs to be install in root mode be fore the application can run. (installdrv)

3. Compiling

DOS

DOS examples are compiled with openwatcom, tested with version 1.8
To compile run wmake in root directory. Then exe will be placed in bin/dos

Linux

To compile driver go to “./core/os/linux/ldrv” and run ./build, the driver will be copied to bin/linux

To compile examples run ./build in root directory. The executables will be placed in bin/linux

4. API overview

API Function	Short form description
KT_API_Open	This function opens the device driver ktapi.bin for hardware communication and must be called in order to use any other functions within this API.
KT_API_Close	This function closes the device driver
KT_API_GetFunction	This function resolves the addressee for the named function.
ReadMonitor (see note)	This function takes a HWMON structure and fills the structure with valid data.
ReadHWMonitorItems	This function read the HW monitor, and return a pointer to a struct
GetHWMonitorItem	This function return a single item from the struct used in ReadHWMonitorItems, Call ReadHWMonitorItems to update the items
SetClrGPIO	This function set or clears a GPIO pin, located on the feature port.
ReadGPIO	This function reads a GPIO pin, located on the feature port.
SetGPIODir	This function set the direction of the GPIO pins, located on the feature port.
SetFanSpeed	This function sets the fan speed.
SelectFanTempTacChannel	This function selects the hardware Fan/Temp and Tachometer channel.
StartThermalCruise	This function enables Thermal Cruise Control the hardware monitor will automatically control the speed of CPU and System Fan.
SetFanTarget	This function sets the Temperature/Speed depending on the Mode selected.
SetFanMode	This function sets the mode: Thermal_Cruise or Fan_Speed_Cruise
GetIntruderStatus	This function returns the intruder status pin.
ClrIntruderStatus	This function clears the intruder status bit.
EnableWD	This function enables the watchdog timer.
DisableWD	This function disables the watchdog timer.
SetWDTimer	This function sets the watchdog timer.
SetBKLControl	Set Backlight intensity via PWM by setting the PWM frequency and duty cycle.
GetMac	This function receives the MAC address of a specific NIC
SetCPUThrottle	This function set CPU throttle. This function can be used to save power by slowing down the CPU speed.
ReadBoardHeader	This function read the Inside Header Info from the Memory Area.
GetBoardName	This function returns a pointer to a text containing the board name.
SmBus_RecvByte	This reads a single byte from a device, without specifying a device register.
SmBus_ReadByte	This reads a single byte from a device, from a designated register.
SmBus_ReadWord	This reads a word (16 bits) from a device, from a designated register.
SmBus_ReadBlock	This command reads a block of up to 32 bytes from a device
SmBus_ReadBytes	This reads multi bytes from a device, starting from a designated register.
SmBus_SendByte	This sends a single byte to a device, without specifying a device register.
SmBus_WriteByte	This writes a single byte to a device, to a designated register.
SmBus_WriteWord	This writes a word (16 bits) to a device, from a designated register.
SmBus_WriteBlock	This command writes a block of up to 32 bytes to a device
SmBus_PorcessCall	This command selects a device register (through the Command code), sends 16 bits of data to it and reads 16 bits of data in return.

Notes: API's return _API_se_OK (logic 1) if the call succeeds otherwise it returns _API_se_Error (logic 0).
The greyed API function is EOL and will be removed in future KT-API-V2 package.

5. API function detailed descriptions

Function	**int KT_API_Open (int sf)**
Description	This function opens the device driver ktapi.bin for hardware communication and must be called in order to use any other functions within this API.
Arguments	**sf** Where to find ktapi.bin sfNewest Check BIOS,disk and internal for the newest version and load it. sfBIOS Only load driver from BIOS. sfDISK Only load driver from DISK. sfInternal Use a internal/embedded version of ktapi.bin
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int KT_API_Close (void)**
Description	This function closes the device driver. After closing the driver no attempt to communicate with the driver will be accepted.
Arguments	None
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int KT_API_GetFunction(char * Name,void *pFunction)**
Description	This function resolves the addressee for the named function.
Arguments	**Name** Name of API function **pFunction** where to put function addressee
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int ReadMonitor(HWMON *Mon)** (EOL and will be removed in future KT-API-V2 package)
Description	This function takes a HWMON structure and fills the structure with valid data. For return structure see api.h for the individual data types.

Arguments	**HWMON *Mon**: pointer **HWMON** structure defined in api.h.
	
	typedef struct _HWMON
	{
	float VCORE;
	float VCOREb;
	float VCC2_5;
	float VCC3;
	float VCC5;
	float V12;
	float SB3;
	float V_12;
	float SB5;
	float VBATT;
	float CPUTemp;
	float FAN;
	float BoardTemp;
	float Reserved1;
	float Reserved2;
	float Reserved3;
	float Reserved4;
	float Reserved5;
	float Reserved6;
	
	} HWMON;

Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int ReadHWMonitorItems(_pHWM_Item * Items);**
Description	This function read the HW monitor and return a structure pointer

Arguments	**Items** return pointer to the structure
	typedef struct
	{
	_HWM_ID ID;
	float Value;
	char * Name;
	}_HWM_Item,*_pHWM_Item;
	
	typedef enum
	{ (See chapter "ReadHWMonitorItems - description of values" for more info)
	
	_HWM_Last=0x0000,
	_HWM_VCORE=0x0001,
	_HWM_VCOREb=0x0002,
	_HWM_VCC2_5=0x0003,
	_HWM_VCC3=0x0004,
	_HWM_VCC5=0x0005,
	_HWM_V12=0x0006,
	_HWM_SB3=0x0007,
	_HWM_V_12=0x0008,
	_HWM_SB5=0x0009,
	_HWM_VBATT=0x000a,
	_HWM_VCC1_2=0x000b,
	_HWM_VCC1_8=0x000c,
	_HWM_VIN=0x000d,
	_HWM_VCC1_5=0x000e,
	
	_HWM_CPU_Temp=0x1001,
	_HWM_System_Temp=0x1002,
	_HWM_FC_Temp=0x1003,
	
	_HWM_CPU_Fan=0x2001,
	_HWM_System_Fan=0x2002,
	_HWM_FC_Fan=0x2003,
	
	_HWM_Type_Mask=0xf000,
	_HWM_Type_Voltage=0x0000,
	_HWM_Type_Temperature=0x1000,
	_HWM_Type_Fan=0x2000,
	
	}_HWM_ID;

Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int GetHWMonitorItem(_HWM_ID ID,float * Value)**
Description	This function look up the _HWM_ID and return the value.
Arguments	**ID** look at ReadHWMonitorItems _HWM_ID **Value** pointer to the returned value
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int SetClrGPIO(unsigned char GPIO,int SetClr)**
Description	This function set or clears a GPIO pin, located on the feature port. Make sure to set pin direction before calling this function.
Arguments	**GPIO** Mask of GPIO to set or clear. **SetClr** 0 Clears, 1 Sets
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int ReadGPIO(unsigned char GPIO,unsigned char * Data)**	
Description	This function reads a GPIO pin, located on the feature port. Make sure to set pin direction before calling this function.	
Arguments	**GPIO**	Mask of GPIO to read
	Data	Pointer to an unsigned char valued read from the GPIO.
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int SetGPIODir(unsigned char GPIO)**	
Description	This function set the direction of the GPIO pins, located on the feature port. Make sure to call this function before calling ReadGPIO or SetClrGPIO.	
Arguments	**GPIO**	Mask of GPIO to set to output, none set bits will be input.
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int SetFanSpeed(unsigned char Speed)**	
Description	This function sets the fan speed in the interval between 0-127 where max. speed is 127. Any attempts to write values above 127 will be ignored. For some boards only 16 steps are possible, so that any value 120 - 127 generates maximum speed, 112 – 119 generates second most highest speed and so on. Please notice that the values in the range 0 – 47 might generate a voltage with is to low to start the Fan.	
Arguments	**Speed**	Fan speed value between 0 and 127.
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int SelectFanTempTacChannel(unsigned char Channel)**	
Description	This function selects the hardware Fan/Temp and Tachometer channel default is channel 0.	
Arguments	**Channel**	0 (CPU Fan/Temperature), 1 (System Fan/Temperature), 2 (Feature Connector Fan/Temperature)
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int StartThermalCruise (VOID)**
Description	This function enables Thermal Cruise Control the hardware monitor will automatically control the speed of CPU and System Fan. The target temperature of the CPU is set by using function SetFanTarget.
Arguments	None
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".

Function	**int SetFanTarget(unsigned int uiStt)**	
Description	This function sets the Target Temperature/Speed depending on the Mode selected. Mode selection can be set by calling function SetFanMode. **Note:** StartThermalCruise must be called prior to this call.	
Arguments	**uiStt**	This is the temperature or speed target
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int SetFanMode(UCHAR ucSfm)**
Description	This function set the Target Mode see modes available below. Thermal_Cruise_Mode 01h Fan_Speed_Cruise_Mode 02h **Note:** StartThermalCruise must be called prior to this call.
Arguments	**ucSfm** Fan mode
Return	If the function succeeds the return value is “_API_se_OK” otherwise it’s “_API_se_Error”.

Function	**int GetIntruderStatus (unsigned int * Status)**
Description	This function returns the intruder status pin. The status result is returned in **Status**. **Status** = 0x00000001 Intruder/Open case detected. **Status** = 0x00000000 Intruder/Open case not detected.
Arguments	**Status** Pointer to receive status;
Return	If the function succeeds the return value is “_API_se_OK” otherwise it’s “_API_se_Error”.

Function	**int ClrIntruderStatus (void)**
Description	This function clears the intruder status bit.
Arguments	None
Return	If the function succeeds the return value is “_API_se_OK” otherwise it’s “_API_se_Error”.

Function	**int EnableWD(void)**
Description	This function enables the watchdog timer. The user must call SetWDTimer and SetWDTimerInterval before calling this function to prevent immediately reboot.
Arguments	None
Return	If the function succeeds the return value is “_API_se_OK” otherwise it’s “_API_se_Error”.

Function	**int DisableWD(void)**
Description	This function disables the watchdog timer. Any attempts to modify watchdog timers after calling this function will have no effect.
Arguments	None
Return	If the function succeeds the return value is “_API_se_OK” otherwise it’s “_API_se_Error”.

Function	**int SetWDTimerInterval(unsigned char VAL)**
Description	This function set the watchdog timer interval. The interval is multiplied with the WDTimer value and represents the time-out period. There are to selectable intervals listed in the ktapi.h file. **_1SEC _1MIN**
Arguments	**VAL** Timer interval
Return	If the function succeeds the return value is “_API_se_OK” otherwise it’s “_API_se_Error”.

Function	**int SetWDTimer(unsigned char Time)**
Description	This function sets the watchdog timer. An application must service this function and reload the timer to prevent reboot; the number of units is between 0-255.
Arguments	**Time** Value used for the next timeout watchdog period
Return	If the function succeeds the return value is “_API_se_OK” otherwise it’s “_API_se_Error”.

Function	int SetBKLControl(unsigned int Freq, unsigned int Duty)	
Description	The duty cycle in %, value must be between 0-100. The Pwm frequency in Khz (1Khz-48Khz) value must be in the range 1-48.	
Arguments	**Freq**	Frequency
	Duty	Duty cycle
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	Int GetMac(unsigned char nMac,void * Buffer)	
Description	This function receives the MAC address of a specified controller passed in nMac (1=1st, 2=2nd, 3=3th). The input buffer must be at least 6 bytes long.	
Arguments	**nMac**	Ethernet controller number
	Buffer	Buffer to receive the MAC address in
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	int SetCPUThrottle(unsigned char DUTY)	
Description	This function set CPU throttle an application can call this function to slow down the CPU speed and save power. The selectable duty cycle intervals are listed in api.h	
Arguments	**DUTY**	Duty cycle
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	int ReadBoardHeader(void * Buffer)	
Description	This function read the Integrated Info Header from the Memory Area. The argument passed to the function must be a pointer to a structure of minimum 19 Bytes. The more information on the returned data see appendix A for structure info.	
Arguments	**Buffer**	Pointer to buffer to receive IIH
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	Int GetBoardName(char * * Name)	
Description	This function returns a pointer to a text containing the board name.	
Arguments	**Name**	Pointer to a char pointer that will receive the BoardName
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	Int GetPartNumber(char * * Number)	
Description	This function returns a pointer to a text containing the part number.	
Arguments	**Name**	Pointer to a char pointer that will receive the part number.
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	Int GetSerialNumber (char * * Number)	
Description	This function returns a pointer to a text containing the serial number.	
Arguments	**Name**	Pointer to a char pointer that will receive the serial number.
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	int SmBus_RecvByte(unsigned char ucDevAddr,unsigned char *ucpData)	
Description	This reads a single byte from a device, without specifying a device register. Some devices are so simple that this interface is enough; for others, it is a shorthand if you want to read the same register as in the previous SMBus command.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucpData	Pointer to received data
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**Int SmBus_ReadByte)(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned char * ucpData)**	
Description	This reads a single byte from a device, from a designated register. The register is specified through the Command Code.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	ucpData	Pointer to received data
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int SmBus_ReadWord(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned short *uspData)**	
Description	This reads a word (16 bits) from a device, from a designated register. The register is specified through the Command Code.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	uspData	Pointer to received data
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int SmBus_ReadBlock(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned char * ucpByteCnt,unsigned char *ucpDataBuf)**	
Description	This command reads a block of upto 32 bytes from a device, from a designated register that is specified through the Command code. The amount of data is specified by the device in the ucpByteCnt. The actual amount of data in device is returned in ucpByteCnt.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	ucpByteCnt	**In :** size of data buffer, **Out :** numbers bytes received
	ucpDataBuf	Pointer to data buffer
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**Int SmBus_ReadBytes(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned int uiByteCnt,unsigned char *ucpDataBuf)**	
Description	This reads multi bytes from a device, starting from a designated register. The register is specified through the Command Code.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	uiByteCnt	Numbers bytes received
	ucpDataBuf	Pointer to data buffer
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**Int SmBus_SendByte(unsigned char ucDevAddr,unsigned char ucCommandCode)**	
Description	This is the reverse of RecvByte: it sends a single byte to a device. See Read Byte for more information.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**Int SmBus_WriteByte(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned char ucData)**	
Description	This writes a single byte to a device, to a designated register. The register is specified through the Command code. This is the opposite of the ReadByte command.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	ucData	Data to send
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**Int SmBus_WriteWord(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned short usData)**	
Description	This is the opposite operation of the ReadWord command. 16 bits of data is read from a device, from a designated register that is specified through the Command code.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	usData	Data to send
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**int SmBus_WriteBlock(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned char ucByteCnt,unsigned char *ucpDataBuf)**	
Description	The opposite of the Block Read command, this writes upto 32 bytes to a device, to a designated register that is specified through the Command code. The amount of data is specified in the ucByteCnt.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	ucByteCnt	Numbers bytes send
	ucpDataBuf	Pointer to data buffer
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

Function	**Int SmBus_PorcessCall(unsigned char ucDevAddr,unsigned char ucCommandCode, unsigned short *uspData)**	
Description	This command selects a device register (through the Command code), sends 16 bits of data to it, and reads 16 bits of data in return.	
Arguments	**ucDevAddr**	Addressee of the device in 8 bit (includes R/W bit) witch shall be set to 0
	ucCommandCode	Command code
	uspData	**In** : Data to send. **Out** : Data received
Return	If the function succeeds the return value is "_API_se_OK" otherwise it's "_API_se_Error".	

6. ReadHWMonitorItems - description of values

KT965

_HWM_Name	Note	BIOS text (if available)
Last	(Internal use)	-
VCORE	Core voltage	VCORE
VCOREb	Core voltage b	-
VCC2_5	2.5V	-
VCC3	3.3V	3VCC
VCC5	5V	+5VIN
V12	+12V	+12VIN
SB3	Standby 3.3V	VSB
V_12	-12V	-12VIN
SB5	Standby 5V	-
VBATT	Battery voltage	VBAT
VCC1_2	1.2V	-
VCC1_8	1.8V	Core 1.8V
VIN	Single Voltage Input	-
VCC1_5	1.5V	Core 1.5V
CPU_Temp	CPU temperature	CPU temperature
System_Temp	System temperature	System temperature
FC_Temp	Temperature via Feature Connector	-
CPU_Fan	CPU Fan RPM	CPUFAN0
System_Fan	System Fan RPM	SYSFAN
FC_Fan	Fan RPM via Feature Connector	AUXFAN
Type_Mask	(Internal use)	-
Type_Voltage	(Internal use)	-
Type_Temperature	(Internal use)	-
Type_Fan	(Internal use)	-

KTUS15

_HWM_Name	Note	BIOS text (if available)
Last	(Internal use)	-
VCORE	Core voltage	VCORE
VCOREb	Core voltage b	-
VCC2_5	2.5V	-
VCC3	3.3V	3VCC
VCC5	5V	+5V
V12	+12V	+12
SB3	Standby 3.3V	VSB
V_12	-12V	-
SB5	Standby 5V	-
VBATT	Battery voltage	VBAT
VCC1_2	1.2V	-
VCC1_8	1.8V	Core 1.8V
VIN	Single Voltage Input	Vin board supply
VCC1_5	1.5V	-
CPU_Temp	CPU temperature	CPU temperature
System_Temp	System temperature	System temperature
FC_Temp	Temperature via Feature Connector	VTIN temperature
CPU_Fan	CPU Fan RPM	CPUFAN0
System_Fan	System Fan RPM	-
FC_Fan	Fan RPM via Feature Connector	AUXFAN
Type_Mask	(Internal use)	-
Type_Voltage	(Internal use)	-
Type_Temperature	(Internal use)	-
Type_Fan	(Internal use)	-

7. Examples - Source Code

The following table specifies the different API's being used in the source code examples available in the KT-API-V2 package.

API's	Apitestex.c	Apirtc.c	Apitest.c						
ClrIntruderStatus									
DisableWD	x								
EnableWD	x								
GetBoardName	x	x	x						
GetIntruderStatus									
GetMac	x								
KT_API_Open	x	x	x						
KT_API_Close	x	x	x						
KT_API_GetFunction	x	x	x						
ReadBoardHeader	x								
ReadGPIO	x								
ReadMonitor	x		x						
SelectFanTempTacChannel									
SetBKLControl	x								
SetClrGPIO	x								
SetCPUThrottle									
SetFanMode	x								
SetFanSpeed	x								
SetFanTarget	x								
SetGPIODir	x								
SetWDTimer	x								
SmBus_PorcessCall									
SmBus_ReadBlock									
SmBus_ReadByte		x							
SmBus_ReadBytes									
SmBus_RecvByte									
SmBus_ReadWord									
SmBus_SendByte									
SmBus_WriteBlock									
SmBus_WriteByte									
SmBus_WriteWord									
StartThermalCruise	x								

Appendix A: How to read the IIH

IHH is Integrated Info Header from the BIOS of Kontron Technology SBC's contains board identification.

The IIH is implemented for KT Motherboards.

You can use DMI (Desktop Management Interface) or API function Readboardheader() .

Disclaimer: KONTRON Technology A/S reserves the right to make changes without notice.

Field	Size	Contents	Offset
Magicscan	4 bytes	'$IIH' (24h, 49h, 49h, 48h)	0-3
Infosize	1 byte	Amount of info in bytes, exclude header and this byte	4
Boardinfo	1 byte	40 = 886LCD-M/Flex 48 = 886LCD/mITX 50 = 886LCD-M/ATX 60 = 786LCD/mITX 90 = 986LCD-M/mITX 91 = 986LCD-M/Flex 92 = 986LCD-M/ATXP 93 = 986LCD-M/ATXE 94 = KT965/Flex 95 = KT965/ATXE 96 = KT965/ATXP 98 = KT690/mITX 100 = KT780/ATX 104 = KTUS15/mITX 108 = KTG41/ATXU 110 = KTGM45/mITX 112 = KTGM45/Flex 114 = KTGM45/ATXU 118 = KTQ45/Flex 120 = KTQ45/ATXE 122 = pITX-SP 124 = JREX-DC 126 = MOPSlcdLX (PLX8) 128 = JREX-690	5
BIOSmjr	1 word	BIOS MAJORVERSION (in Hex value)	6-7
BIOSmnr	1 word	BIOS MINORVERSION (in Hex value)	8-9
Reserved	1 byte	N/A	10
S/N	4 bytes	S/N in BCD	11-14
P/N	4 bytes	P/N in BCD	15-18