

PROGRAMMING GUIDE  
VIA Smart ETK SDK

## Copyright

Copyright © 2015 VIA Technologies Incorporated. All rights reserved.

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise without the prior written permission of VIA Technologies, Incorporated.

## Trademarks

All brands, product names, company names, trademarks and service marks are the property of their respective holders.

## Disclaimer

VIA Technologies makes no warranties, implied or otherwise, in regard to this document and to the products described in this document. The information provided in this document is believed to be accurate and reliable as of the publication date of this document. However, VIA Technologies assumes no responsibility for the use or misuse of the information in this document and for any patent infringements that may arise from the use of this document. The information and product specifications within this document are subject to change at any time, without notice and without obligation to notify any person of such change.

VIA Technologies, Inc. reserves the right to make changes to the products described in this manual at any time without prior notice.

## Revision History

Version	Date	Remarks
0.0.1	2014/4/3	Initial draft - SmartETK v0.0.11
0.0.2	2014/8/1	Removed network Ethernet functions Modified watch dog scenario SmartETK v0.0.17 <ul style="list-style-type: none"><li>• added timeout and keep alive functions</li></ul>
0.03	2014/8/22	SmartETK v0.0.18 <ul style="list-style-type: none"><li>• added system reboot and suspend functions</li></ul>
1.00	2015/7/22	SmartETK v0.0.18 <ul style="list-style-type: none"><li>• Refined document version</li></ul>

## Table of Contents

<b>1. Introduction</b> .....	<b>1</b>
1.1. Overview.....	1
<b>2. System Requirements</b> .....	<b>1</b>
2.1. Hardware Requirements.....	1
2.2. Software Requirements.....	1
<b>3. Installation and Usage</b> .....	<b>2</b>
3.1. Installation on development computer.....	2
3.2. Installation on target board.....	3
<b>4. Smart ETK SDK API</b> .....	<b>4</b>
4.1. Class definitions.....	4
4.2. Function return values.....	4
4.2.1. SmartETK.S_OK.....	4
4.2.2. SmartETK.E_FAIL.....	4
4.2.3. SmartETK.E_VERSION_NOT_SUPPORT.....	4
4.2.4. SmartETK.E_INVALID_ARG.....	4
4.2.5. SmartETK.E_FUNC_NOT_SUPPORT.....	4
4.2.6. SmartETK.E_CONNECTION_FAIL.....	4
4.2.7. SmartETK.E_NOT_RESPOND_YET.....	5
4.2.8. SmartETK.E_TIMEOUT.....	5
4.2.9. SmartETK.E_UART_OPENFAIL.....	5
4.2.10. SmartETK.E_UART_NOT_OPEN.....	5
4.2.11. SmartETK.E_UART_ALREADY_OPENED.....	5
4.2.12. SmartETK.E_UART_TTY_BEEN_USED.....	5
4.2.13. SmartETK.E_UART_BAUDRATE_NOT_SUPPORT.....	5
4.3. GPIO Class.....	6
4.3.1. GPIO Class.....	6
4.3.2. GPIO.setEnabled().....	6
4.3.3. GPIO.getEnabled().....	7
4.3.4. GPIO.setDirection().....	7
4.3.5. GPIO.getDirection().....	8
4.3.6. GPIO.setValue().....	8
4.3.7. GPIO.getValue().....	9
4.4. Network Class.....	10
4.4.1. Network Class.....	10
4.4.2. Network.setWakeOnLan().....	10

4.4.3.	Network.getWakeOnLan().....	11
4.5.	WatchDog Class.....	12
4.5.1.	WatchDog Class.....	12
4.5.2.	WatchDog.setEnabled().....	12
4.5.3.	WatchDog.isEnabled().....	13
4.5.4.	WatchDog.setTimeout().....	14
4.5.5.	WatchDog.getTimeout().....	14
4.5.6.	WatchDog.keepAlive().....	15
4.6.	RTC Class.....	16
4.6.1.	RTC Class.....	16
4.6.2.	RTC.setWakeUpTime().....	16
4.6.3.	RTC.getWakeUpTime().....	17
4.6.4.	RTC.setEnabled().....	18
4.6.5.	RTC.isEnabled().....	19
4.7.	I2C Class.....	20
4.7.1.	I2C Class.....	20
4.7.2.	I2C.read().....	21
4.7.3.	I2C.write().....	21
4.8.	UART Class.....	22
4.8.1.	UART Class.....	22
4.8.2.	Uart.open().....	22
4.8.3.	Uart.close().....	23
4.8.4.	Uart.setConfig().....	23
4.8.5.	Uart.getConfig().....	24
4.8.6.	Uart.setTimeout().....	25
4.8.7.	Uart.getTimeout().....	26
4.8.8.	Uart.setReturnChar().....	27
4.8.9.	Uart.getReturnChar().....	27
4.8.10.	Uart.setRS485().....	28
4.8.11.	Uart.getRS485().....	29
4.8.12.	Uart.readData().....	29
4.8.13.	Uart.readData().....	30
4.8.14.	Uart.writeData().....	30
4.8.15.	Uart.reset().....	31
4.9.	SystemETK Class.....	32
4.9.1.	SystemETK Class.....	32
4.9.2.	SystemETK.reboot().....	32
4.9.3.	SystemETK.suspend().....	33
<b>Appendix A. GPIO.....</b>		<b>34</b>
A.1.	GPIO5 ~ GPIO8: for user's own design.....	34

<b>Appendix B. Network</b> .....	<b>36</b>
B.1. Set Wake On LAN From Suspend mode.....	36
B.2. Get Wake On LAN From Suspend mode Status.....	36
<b>Appendix C. Watch Dog</b> .....	<b>37</b>
C.1. Enable Watch Dog .....	37
C.2. Disable Watch Dog.....	37
C.3. Set Watch Dog Timeout Value.....	37
C.4. Get Watch Dog Status.....	38
C.5. Keep Watch Dog Alive .....	38
<b>Appendix D. RTC</b> .....	<b>39</b>
D.1. Set RTC Wake Up From Suspend mode .....	39
D.2. Get RTC Wake Up Status.....	39
D.3. Set RTC Wake Up Time.....	40
D.4. Get RTC Wake Up Time.....	40
<b>Appendix E. I2C</b> .....	<b>42</b>
E.1. I2C Initialize .....	42
E.2. I2C Read Data .....	42
E.3. I2C Write Data .....	43
<b>Appendix F. UART</b> .....	<b>44</b>
F.1. UART Initialize Communication .....	44
F.2. UART Write Data.....	45
F.3. UART Read Data .....	45
<b>Appendix G. SystemETK</b> .....	<b>46</b>
G.1. Reboot the Machine .....	46
G.2. Suspend the Machine .....	46

# 1. Introduction

## 1.1. Overview

VIA Smart ETK SDK supports the hardware controlling API for GPIO, Network, Watch Dog, RTC, I2C, and UART (RS-232/RS485) modules.

Smart ETK is programmed with the socket IO as the communication between JAVA and C language to control the hardware modules. We implemented the board support service like `bss_a900` to listen the request from Smart ETK API. We bind `127.0.0.1` as the internal listening IP, to keep it from establishing the connection with the external network.

# 2. System Requirements

## 2.1. Hardware Requirements

VIA Smart ETK SDK is compatible with the following main board:

- ARTiGO A900 with Android BSP

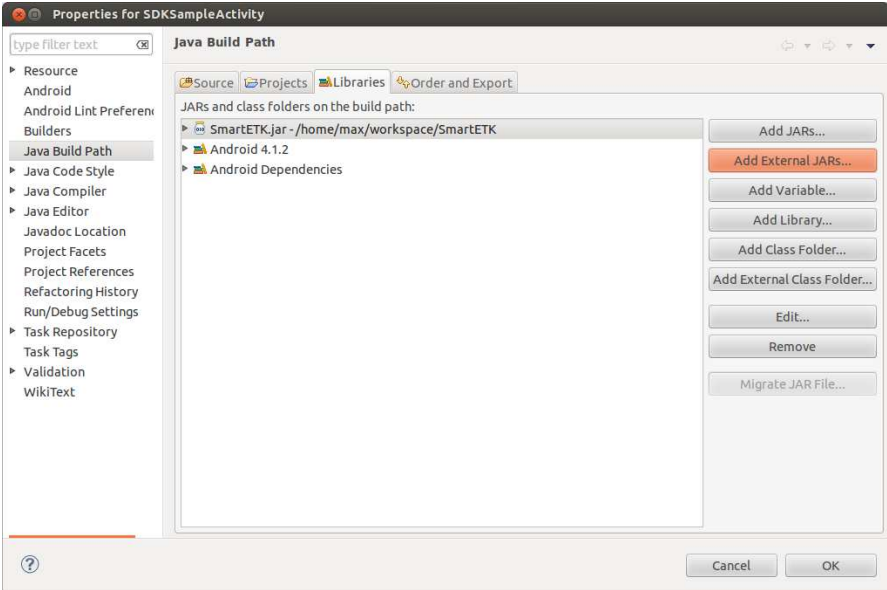
## 2.2. Software Requirements

- Microsoft Windows or Ubuntu Linux
- Eclipse IDE with Android Development Tools (ADT) installed

# 3. Installation and Usage

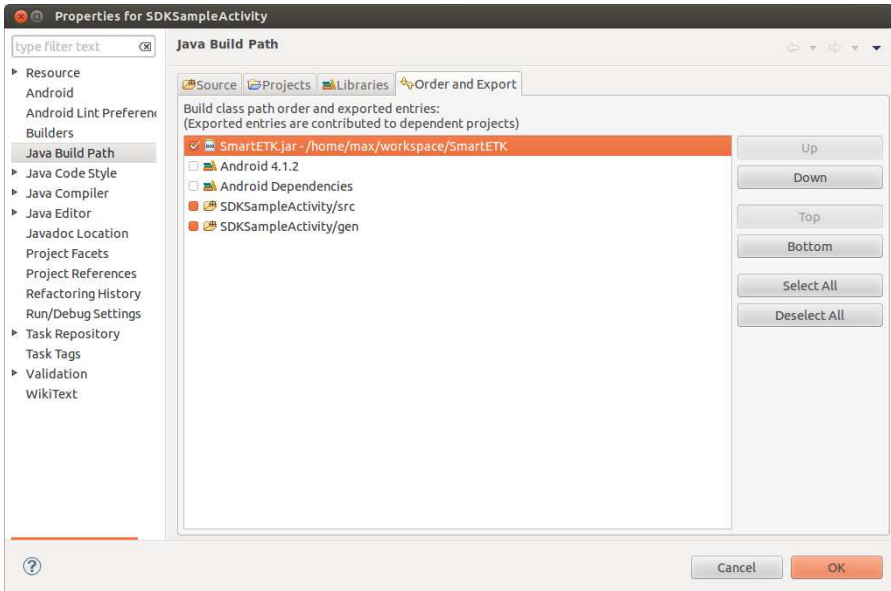
## 3.1. Installation on development computer

Open Eclipse IDE and create an Android project. In project properties, import SmartETK.jar by pressing the button “Add External JARs...” into project.



Select “Order and Export” tab, move SmartETK.jar to the top and choose it.





## 3.2. Installation on target board

Install VIA released firmware and it is done.

## 4. Smart ETK SDK API

### 4.1. Class definitions

GPIO, Network, Watch Dog, RTC, I2C, and UART modules are placed in class named `com.viaembedded.smartetk`, and returned values is placed in class named `com.viaembedded.smartetk`. Import this package `com.viaembedded.smartetk.*` into Java code to use them.

### 4.2. Function return values

There are some types of return values found throughout the Smart ETK SDK API.

#### 4.2.1. SmartETK.S\_OK

The `S_OK` return value has the constant value 0. When a function returns the `S_OK` value, it indicates that the function has successfully completed.

#### 4.2.2. SmartETK.E\_FAIL

When a function returns the `E_FAIL` value, it indicates that the function has failed to complete.

#### 4.2.3. SmartETK.E\_VERSION\_NOT\_SUPPORT

When a function returns the `E_VERSION_NOT_SUPPORT` value, it indicates that the versions of SmartETK.jar and bsservice are not compatible.

#### 4.2.4. SmartETK.E\_INVALID\_ARG

When a function returns the `E_INVALID_ARG` value, it indicates that the arguments are invalid.

#### 4.2.5. SmartETK.E\_FUNC\_NOT\_SUPPORT

When a function returns the `E_FUNC_NOT_SUPPORT` value, it indicates that the function is not supported by this board.

#### 4.2.6. SmartETK.E\_CONNECTION\_FAIL

When a function returns the `E_CONNECTION_FAIL` value, it indicates that the bsservice doesn't respond to the request. Please make sure bsservice is running successfully.

#### 4.2.7. SmartETK.E\_NOT\_RESPOND\_YET

When a function returns the `E_NOT_RESPOND_YET` value, it indicates that the `bsservice` function is still running and has not finished yet.

#### 4.2.8. SmartETK.E\_TIMEOUT

When a function returns the `E_TIMEOUT` value, it indicates that there is no corresponding data received within the period.

#### 4.2.9. SmartETK.E\_UART\_OPENFAIL

When `Uart.open()` returns the `E_UART_OPENFAIL` value, it indicates that the UART device can't be opened successfully. Please make sure the name of the tty device exists.

#### 4.2.10. SmartETK.E\_UART\_NOT\_OPEN

When a function returns the `E_UART_NOT_OPEN` value, it indicates that `uart` object cannot be operated normally. It might represent the application doesn't open `uart` device before calling other operating function; or it was reset by the other `uart` object.

#### 4.2.11. SmartETK.E\_UART\_ALREADY\_OPENED

When `Uart.open()` returns the `E_UART_ALREADY_OPENED` value, it indicates that the `uart` object had been opened. If you need to open other `uart` device, please call `close` function to close the current device, and then open the other `uart` again.

#### 4.2.12. SmartETK.E\_UART\_TTY\_BEEN\_USED

When `Uart.open()` returns the `E_UART_TTY_BEEN_USED` value, it indicates that the `tty` device had been used by other `uart` object. If you want to use it, you can call `reset` function to release the resource and open it again.

#### 4.2.13. SmartETK.E\_UART\_BAUDRATE\_NOT\_SUPPORT

When `Uart.setConfig()` returns the `E_UART_BAUDRATE_NOT_SUPPORT` value, it indicates that baud rate is not supported.

## 4.3. GPIO Class

### 4.3.1. GPIO Class

Syntax:

```
GPIO (int pinID);
```

Description:

Create a new GPIO object with specified pin ID. Ex: 5, 6, 7, 8.

Parameters:

`pinID` – GPIO's pin ID.

Example:

Create a GPIO object with pin ID 5.

```
GPIO gpio5 = new GPIO(5);
```

### 4.3.2. GPIO.setEnabled()

Syntax:

```
int setEnable(boolean enable);
```

Description:

Enable the specific GPIO pin.

Parameters:

`enable` – `true` for enable, `false` for disable.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

### 4.3.3. GPIO.enable()

Syntax:

```
int getEnable(boolean[] enable);
```

Description:

Get enable state of the specific GPIO pin.

Parameters:

`enable` – return `true` for enable, return `false` for disable.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

### 4.3.4. GPIO.setDirection()

Syntax:

```
int setDirection(int direction);
```

Description:

Set input/output direction for the specific GPIO pin.

Parameters:

`direction` – `GPIO.GM_GPI` for input direction,  
`GPIO.GM_GPO` for output direction.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

### 4.3.5. GPIO.getDirection()

Syntax:

```
int getDirection(int[] direction);
```

Description:

Get direction state of the specific GPIO Pin.

Parameters:

`direction` – return `GPIO.GM_GPI` for input direction,  
return `GPIO.GM_GPO` for output direction.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

### 4.3.6. GPIO.setValue()

Syntax:

```
int setValue(int value);
```

Description:

Set output signal for the specific GPIO Pin.

Parameters:

`value` – GPIO signal, `0` for logic low, `1` for logic high.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

### 4.3.7. GPIO.getValue()

Syntax:

```
int getValue(int[] value);
```

Description:

Get input signal of the specific GPIO Pin.

Parameters:

**value** – GPIO signal, return 0 for logic low, return 1 for logic high.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails

## 4.4. Network Class

### 4.4.1. Network Class

Syntax:

```
Network();
```

Description:

Create a new Network object.

Example:

Create a Network object.

```
Network m_network = new Network();
```

### 4.4.2. Network.setWakeOnLan()

Syntax:

```
int setWakeOnLan(boolean bEnable);
```

Description:

Enable or disable Network wake on LAN function from suspend mode.

Parameters:

**bEnable** – enable or disable Network wake on LAN function from suspend mode.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.



### 4.4.3. Network.getWakeOnLan()

Syntax:

```
int getWakeOnLan(boolean[] bEnable);
```

Description:

Get the status if Network wake on LAN function from suspend mode is enabled or disabled.

Parameters:

**bEnable** – return **true** for enable, return **false** for disable.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

## 4.5. WatchDog Class

### 4.5.1. WatchDog Class

Syntax:

```
WatchDog();
```

Description:

Create a new WatchDog object.

Example:

Create a WatchDog object.

```
WatchDog m_watchdog = new WatchDog();
```

### 4.5.2. WatchDog.setEnabled()

Syntax:

```
int setEnable(boolean bEnable);
```

Description:

Enable or disable watch dog function. If watch dog function was enabled, it should be fed within a period, otherwise the system will reboot.

Parameters:

bEnable – enable or disable watch dog function.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

### 4.5.3. WatchDog.getEnable()

Syntax:

```
int getEnable(boolean[] bEnable);
```

Description:

Get the status if watch dog function is enabled or disabled.

Parameters:

bEnable – return true for enable, return false for disable.

Return:

S\_OK – if the function succeeds.

E\_\* – if the function fails.

#### 4.5.4. WatchDog.setTimeout()

Syntax:

```
int setTimeout(int iTimeout);
```

Description:

Set watch dog timeout value. The argument is an integer representing the timeout in seconds.

Parameters:

`iTimeout` – timeout value. (only support timeout in 2, 4, 8, 16, 32, and 64 seconds).

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

#### 4.5.5. WatchDog.getTimeout()

Syntax:

```
int getTimeout(int[] iTimeout);
```

Description:

Get watch dog timeout value.

Parameters:

`iTimeout` – return timeout value.

**Return:**

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

## 4.5.6. WatchDog.keepAlive()

Syntax:

```
int keepAlive();
```

Description:

Keep watch dog alive to avoid rebooting the system.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

## 4.6. RTC Class

### 4.6.1. RTC Class

Syntax:

```
RTC();
```

Description:

Create a new RTC object.

Example:

Create an RTC object.

```
RTC m_rtc = new RTC();
```

### 4.6.2. RTC.setWakeUpTime()

Syntax:

```
int setWakeUpTime(byte byMode, int iYear, byte byMonth, byte byDay, byte  
byHour, byte byMin, byte bySec);
```

Description:

Set the wake up time and mode in RTC. The behavior of wake up from suspend mode will start at the wake up time, and it must loop according to the wake up mode.

Parameters:

byMode –

RTC.ARG\_RTC\_MODE\_DAY for wake up every day.

RTC.ARG\_RTC\_MODE\_MONTH for wake up every month.

RTC.ARG\_RTC\_MODE\_WEEK for wake up every week.

iYear – Year since 1900 ~ 2155 for wake up time

byMonth – Month (1 ~ 12) for wake up time

byDay – Day of the month (1 ~ 31) for wake up time

byHour – Hours (0 ~ 23) for wake up time

byMin – Minutes (0 ~ 59) for wake up time

bySec – Seconds (0 ~ 59) for wake up time

**Return:**

S\_OK – if the function succeeds.

E\_\* – if the function fails.

### 4.6.3. RTC.getWakeUpTime()

Syntax:

```
int getWakeUpTime(RTCStatus RS);
```

Description:

Get the wake up time and mode set in RTC.

Parameters:

RTCStatus – Wake up time and mode set in RTC

class RTCStatus

{

byMode –

RTC.ARG\_RTC\_MODE\_DAY for wake up every day.

RTC.ARG\_RTC\_MODE\_MONTH for wake up every month.

RTC.ARG\_RTC\_MODE\_WEEK for wake up every week.

iYear – Year since 1900 ~ 2155 for wake up time

byMonth – Month (1 ~ 12) for wake up time

byDay – Day of the month (1 ~ 31) for wake up time

```

byHour – Hours (0 ~ 23) for wake up time
byMin – Minutes (0 ~ 59) for wake up time
bySec – Seconds (0 ~ 59) for wake up time
    
```

```

}
    
```

Return:

S\_OK – if the function succeeds.

E\_\* – if the function fails.

#### 4.6.4. RTC.setEnabled()

Syntax:

```
int setEnable(boolean bEnable);
```

Description:

Enable or disable RTC wake up function from suspend mode.

Parameters:

bEnable – enable or disable RTC wake up function from suspend mode.

Return:

S\_OK – if the function succeeds.

E\_\* – if the function fails.



### 4.6.5. RTC.getEnable()

Syntax:

```
int getEnable(boolean[] bEnable);
```

Description:

Get the status if wake up function from suspend mode is enabled or disabled.

Parameters:

bEnable – return true for enable, return false for disable.

Return:

S\_OK – if the function succeeds.

E\_\* – if the function fails

## 4.7. I2C Class

### 4.7.1. I2C Class

Syntax:

```
I2C(int iI2CBusNum, byte byI2CAddress, int iOffsetLen);
```

Description:

Create a new I2C object with specified bus number, slave address and the length of the offset address.

Parameters:

**iI2CBusNum** – I2C bus number. Ex: 0 is for i2c-0 bus, 1 is for i2c-1 bus.

**byI2CAddress** – I2C slave address. Support 7 bits slave address data.

**iOffsetLen** – The length of the registers' offset only support 0 ~ 4 bytes. (0: no registers / 1: 8 bit registers / 2: 16 bit registers / 3: 24 bit registers / 4: 32 bit registers)

Example:

Create an I2C object in I2C bus 1 and I2C slave address 10, and the offset length is 0.

```
I2C m_i2c = new I2C(1,10,0);
```

Create an I2C object in I2C bus 1 and I2C slave address 52, and the offset length is 2 (16 bit registers).

```
I2C m_i2c = new I2C(1,52,2);
```

### 4.7.2. I2C.read()

Syntax:

```
int read(byte[] byBuf, int iOffset, int iReadLen);
```

Description:

Read data from specified offset with given length, and store data in buffer.

Parameters:

**byBuf** – The buffer to store read data

**iOffset** – The registers' offset to read from specified I2C bus number and slave address. (Valid data is 0 ~ 7FFFFFFF)

**iReadLen** – number of bytes to read, maximum 255 bytes per transfer.

### 4.7.3. I2C.write()

Syntax:

```
int write(byte[] byBuf, int iOffset, int iWriteLen);
```

Description:

Write data to specified offset with given length.

Parameters:

**byBuf** – The buffer of the written data

**iOffset** – The registers' offset of writing to specified I2C bus number and slave address. (valid data is 0 ~ 7FFFFFFF)

**iWriteLen** – The written data length, maximum 255 bytes per transfer.

## 4.8. UART Class

### 4.8.1. UART Class

Syntax:

```
Uart();
```

Description:

Create a new UART object.

Example:

Create a UART object.

```
Uart m_uart = new Uart();
```

### 4.8.2. Uart.open()

Syntax:

```
int open(String sDev);
```

Description:

Open the specified UART device.

Parameters:

`sDev` – UART device name. (Ex. ttyUSB0)

Return:

`S_OK` – if the function succeeds.

`E_UART_OPENFAIL` – open device is failed.

`E_UART_ALREADY_OPENED` – object already had been opened.

`E_UART_TTY_BEEN_USED` – device had been used by other object.

`E_*` – if the function fails.

### 4.8.3. Uart.close()

Syntax:

```
int close();
```

Description:

Close the current opened UART device.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

### 4.8.4. Uart.setConfig()

Syntax:

```
int setConfig(int iBaudRate, byte byDataBits, byte byStopBits, byte byParity,
byte byFlowCtrl);
```

Description:

Set the configurations of the opened UART device.

Parameters:

`iBaudRate` – baud rate (Ex. 115200)

`byDataBits` – data bits. 7: 7-bit data bits; 8: 8-bit data bits

`byStopBits` – stop bits. 1: 1-stop bits; 2: 2-stop bits

`byParity` – parity. 0: none; 1: odd; 2: even

`byFlowCtrl` – flow control. 0: none; 1: CTS/RTS

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

## 4.8.5. Uart.getConfig()

Syntax:

```
int getConfig(UartConfig UC);
```

Description:

Get the configurations of the opened UART device and store them in passed UartConfig Class.

Parameters:

**UartConfig** – UART Configuration

```
class UartConfiguration
```

```
{
    int iBaudRate – baud rate (Ex. 115200)
    byte byDataBits – data bits. 7: 7-bit data bits; 8: 8-bit data bits
    byte byStopBits – stop bits. 1: 1-stop bits; 2: 2-stop bits
    byte byParity – parity. 0: none; 1: odd; 2: even
    byte byFlowCtrl – flow control. 0: none; 1: CTS/RTS
}
```

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

Example:

```
UartConfig UC = m_uart.new UartConfig();
```

```
if(SmartETK.S_OK != m_uart.getConfig(UC))
```

```
{
```

```
cleanStatus();
    return;
}
```

#### 4.8.6. Uart.setTimeout()

Syntax:

```
int setTimeout(boolean bEnable, int iTimeout);
```

Description:

Set the timeout of the opened Uart device.

bEnable = true, iTimeout = 0 (polling read)

bEnable = true, iTimeout > 0 (read with timeout)

bEnable = false (blocking read)

Parameters:

**bEnable** – enable or disable timeout function.

**iTimeout** – timeout value. Range 0 – 255 (0 ~ 25.5 seconds), unit is 0.1 second.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

### 4.8.7. Uart.getTimeout()

Syntax:

```
int getTimeout(Timeout T);
```

Description:

Get the timeout configuration of the opened UART device and store them in passed Timeout Class.

Parameters:

**Timeout** – timeout configuration

```
class Timeout
```

```
{
```

```
    boolean bEnable – enable or disable timeout function
```

```
    int iTimeout – timeout value. Range 0 – 255 (0 ~ 25.5 seconds), unit is 0.1 second.
```

```
}
```

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

Example:

```
Timeout T = m_uart.new Timeout();
```

```
if(SmartETK.S_OK != m_uart.getTimeout(T))
```

```
{
```

```
    cleanStatus();
```

```
    return;
```

```
}
```



### 4.8.8. Uart.setReturnChar()

Syntax:

```
int setReturnChar(boolean bEnable, byte byReturnChar);
```

Description:

Set the termination character of the opened UART device.

bEnable = true (blocking until byReturnChar is received, or read buffer is full.)

bEnable = false (ignore byReturnChar checking when reading data)

Parameters:

**bEnable** – enable or disable the termination character function.

**byReturnChar** – the termination character

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

### 4.8.9. Uart.getReturnChar()

Syntax:

```
int getReturnChar(ReturnChar RC);
```

Description:

Get the termination character configuration of the opened UART device and store them in passed ReturnChar Class.

Parameters:

**ReturnChar** – termination character configuration

class ReturnChar

```
{
```

```

        boolean bEnable – enable or disable the termination character
function
        byte byReturnChar – the termination character
    }
    
```

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

Example:

```
ReturnChar RC = m_uart.new ReturnChar();
```

```

if(SmartETK.S_OK != m_uart.getReturnChar(RC))
{
    cleanStatus();
    return;
}
    
```

#### 4.8.10. Uart.setRS485()

Syntax:

```
int setRS485(boolean bEnable);
```

Description:

Enable or disable RS485 software mode, only supports to ttyUSB0 device. Please check if hardware jumper had been set to RS485 mode.

Parameters:

`bEnable` – enable for RS485 software mode, or disable for RS232 software mode.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

### 4.8.11. Uart.getRS485()

Syntax:

```
int getRS485(BOOLEAN bEnable);
```

Description:

Get the status if RS485 software mode is enabled or disabled.

Parameters:

**bEnable** – return true for enable, return false for disable.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

### 4.8.12. Uart.readData()

Syntax:

```
int readData(int iReadLen, byte[] byRead, int[] iActualLen);
```

Description:

Receive data from the opened UART device.

Parameters:

**iReadLen** – number of bytes to read, maximum 1024 bytes per transfer.

**byRead** – pointer to the buffer pointer.

**iActualLen** – the actual number of bytes received.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

### 4.8.13. Uart.readData()

Syntax:

```
int readData(int iReadLen, byte[] byRead);
```

Description:

Receive data from the opened UART device.

Parameters:

`iReadLen` – number of bytes to read, maximum 1024 bytes per transfer.

`byRead` – pointer to the buffer pointer.

Return:

`>=0` – if the function succeeds, return the actual number of bytes received.

`<0(E_*)` – if the function fails.

### 4.8.14. Uart.writeData()

Syntax:

```
int writeData(int iWriteLen, byte[] byWrite);
```

Description:

Send the data to the opened UART device.

Parameters:

`iWriteLen` – number of bytes to transmit, maximum 1024 bytes per transfer.

`byWrite` – pointer to data buffer.

Return:

[S\\_OK](#) – if the function succeeds.

[E\\_\\*](#) – if the function fails.

### 4.8.15. Uart.reset()

Syntax:

```
int reset();
```

Description:

Reset the opened or open failed UART device.

If the UART device had been used by other object, open function will return [E\\_UART\\_ALREADY\\_OPENED](#) fails. The object could call reset function to release the uart resource and call open UART device again.

Return:

[S\\_OK](#) – if the function succeeds.

[E\\_\\*](#) – if the function fails.

## 4.9. SystemETK Class

### 4.9.1. SystemETK Class

Syntax:

```
SystemETK();
```

Description:

Create a new SystemETK object.

Example:

Create a SystemETK object.

```
SystemETK m_system = new SystemETK();
```

### 4.9.2. SystemETK.reboot()

Syntax:

```
int reboot();
```

Description:

Reboot the machine.

Return:

**S\_OK** – if the function succeeds.

**E\_\*** – if the function fails.

### 4.9.3. SystemETK.suspend()

Syntax:

```
int suspend();
```

Description:

Suspend the machine.

Return:

`S_OK` – if the function succeeds.

`E_*` – if the function fails.

# Appendix A. GPIO

## A.1. GPIO5 ~ GPIO8: for user's own design

GPIO5 ~ GPIO8 is the external GPIO pin for user's own design.

Here it presents an example of set GPIO5 as input pin and get its value.

The following is the sample code:

```

/* Declare variables to get GPIO5 values */
boolean[] bEnable = new boolean[1];
int[] nDirection = new int[1];
int[] nValue = new int[1];

GPIO gpio5 = new GPIO(5);           // Create GPIO5 object

gpio5.setEnabled(true);           // Enable
GPIO5
gpio5.setDirection(GPIO.GM_GPI);  // Set GPIO5 as input
direction

gpio5.setEnabled(bEnable);        // Get GPIO5's enable
status
gpio5.getDirection(nDirection);  // Get GPIO5's
input/output direction
gpio5.getValue(nValue);          // Get GPIO5's
input value
    
```



Here it presents an example of set GPIO6 as output pin and change its value.

The following is the sample code:

```

/* Declare variables to get GPIO6 values */
boolean[] bEnable = new boolean[1];
int[] nDirection = new int[1];
int[] nValue = new int[1];

GPIO gpio6 = new GPIO(6);           // Create GPIO6 object

gpio6.setEnabled(true);             // Enable
GPIO6
gpio6.setDirection(GPIO.GM_GPO);    // Set GPIO6 as
output direction
gpio6.setValue(1);                 // Set GPIO6's output
to high

gpio6.getEnable(bEnable);           // Get GPIO6's enable
status
gpio6.getDirection(nDirection);     // Get GPIO6's
input/output direction
gpio6.getValue(nValue);             // Get GPIO6's output
value
    
```

# Appendix B. Network

## B.1. Set Wake On LAN From Suspend mode

The following is the sample code:

```

boolean bSetEnable = true;

if(null == m_network)
    m_network = new Network();

if(SmartETK.S_OK != m_network.setWakeOnLan(bSetEnable))
{
    return false;
}
    
```

## B.2. Get Wake On LAN From Suspend mode Status

The following is the sample code:

```

if(null == m_network)
    m_network = new Network();

boolean[] bGetEnable = new boolean[1];

if(SmartETK.S_OK != m_network.getWakeOnLan(bGetEnable))
{
    return false;
}

return bGetEnable[0];
    
```

# Appendix C. Watch Dog

## C.1. Enable Watch Dog

The following is the sample code:

```

if(null == m_watchdog)
    m_watchdog = new WatchDog();

if(SmartETK.S_OK != m_watchdog.enable(true))
    return false;
    
```

## C.2. Disable Watch Dog

The following is the sample code:

```

if(null == m_watchdog)
    m_watchdog = new WatchDog();

if(SmartETK.S_OK != m_watchdog.enable(false))
    return false;
    
```

## C.3. Set Watch Dog Timeout Value

The following is the sample code:

```

if(null == m_watchdog)
    m_watchdog = new WatchDog();

if(SmartETK.S_OK != m_watchdog.setTimeout(32))
    return false;
    
```

## C.4. Get Watch Dog Status

The following is the sample code:

```

if(null == m_watchdog)
    m_watchdog = new WatchDog();

boolean[] bGetEnable = new boolean[1];
int[] iTimeout = new int[1];

if(SmartETK.S_OK == m_watchdog.getEnable(bGetEnable))
{
    /* Do something ... */
}

if(SmartETK.S_OK == m_watchdog.getTimeout(iTimeout))
{
    /* Do something ... */
}
    
```

## C.5. Keep Watch Dog Alive

The following is the sample code:

```

if(null == m_watchdog)
    m_watchdog = new WatchDog();

if(SmartETK.S_OK != m_watchdog.keepAlive())
    return false;
    
```

# Appendix D. RTC

## D.1. Set RTC Wake Up From Suspend mode

The following is the sample code:

```

boolean bSetEnable = true;

if(null == m_rtc)
    m_rtc = new RTC();

if(SmartETK.S_OK != m_rtc.setEnabled(bSetEnable))
{
    return false;
}
    
```

## D.2. Get RTC Wake Up Status

The following is the sample code:

```

if(null == m_rtc)
    m_rtc = new RTC();

boolean[] bGetEnable = new boolean[1];

if(SmartETK.S_OK != m_rtc.setEnabled(bGetEnable))
{
    return false;
}
    
```

## D.3. Set RTC Wake Up Time

Wake up from suspend since 2014/5/1, every day at 12:00.

The following is the sample code:

```

byte byMode = RTC.ARG_RTC_MODE_DAY;
int iYear = 2014;
byte byMonth = IntToByte(5);
byte byDay = IntToByte(1);
byte byHour = IntToByte(12);
byte byMin = IntToByte(0);
byte bySec = IntToByte(0);

if(null == m_rtc)
    m_rtc = new RTC();

if(SmartETK.S_OK != m_rtc.setWakeUpTime(byMode, iYear ,
byMonth , byDay , byHour , byMin , bySec))
{
    return false;
}
    
```

## D.4. Get RTC Wake Up Time

The following is the sample code:

```

if(null == m_rtc)
{
    m_rtc = new RTC();
    m_RS = m_rtc.new RTCStatus();
}
    
```

```
if(SmartETK.S_OK != m_rtc.getWakeUpTime(m_RS))  
{  
    return false;  
}
```

# Appendix E. I2C

## E.1. I2C Initialize

Create an I2C object in I2C bus 1 and I2C slave address 52, and the offset length is 2.

The following is the sample code:

```

int iBusNum = 1;
byte byAddress = IntToByte(52);
int iOffsetLen = 2;

if(iBusNum < 0 || byAddress < 0 || iOffsetLen < 0)
    return false;

m_i2c = new I2C(iBusNum, byAddress, iOffsetLen);
    
```

## E.2. I2C Read Data

Read data from offset "0" with length "2" bytes, and store data in byRead byte array buffer.

The following is the sample code:

```

byte[] byRead = new byte[255]
int iOffset = 0;
int iReadLen = 2;

Arrays.fill(byRead, 0);

if(SmartETK.S_OK != m_i2c.read(byRead, iOffset, iReadLen) || null
    == byRead)
    return false;
    
```



### E.3. I2C Write Data

Write data to offset "0" with length "2" bytes and data value "0x0101". The written data is stored in byWrite byte array buffer.

The following is the sample code:

```
byte[] byWrite = new byte[2]
```

```
byWrite[0] = 0x01;
```

```
byWrite[1] = 0x01;
```

```
int iOffset = 0;
```

```
int iWriteLen = 2;
```

```
if(SmartETK.S_OK != m_i2c.write(byWrite, iOffset, iWriteLen))
    return false;
```

# Appendix F. UART

## F.1. UART Initialize Communication

The following is the sample code:

```

private Uart m_uart = null;

m_uart = new Uart();
if(null == m_uart)
{
    cleanStatus();
    return;
}

if(SmartETK.S_OK != m_uart.open((m_sDev =
mETDev.getText().toString()))
{
    cleanStatus();
    return;
}

if(SmartETK.S_OK != m_uart.setConfig((m_iBaudRate =
Integer.valueOf(mETBaudRate.getText().toString()), (byte)8,
(byte)1, (byte)0, (byte)0))
{
    cleanStatus();
    return;
}
    
```

## F.2. UART Write Data

The following is the sample code:

Notice that “mETWrite” is the EditText to store writing texts.

```
if(SmartETK.S_OK !=
m_uart.writeData(mETWrite.getText().toString().getBytes().length,
mETWrite.getText().toString().getBytes()))
{
    return;
}
```

## F.3. UART Read Data

The following is the sample code:

```
int iReadLen = LENGTH;
byte[] byRead = new byte[LENGTH];
int[] iActualLen = new int[1];

while(SmartETK.S_OK == m_mainThreadUart.readData(iReadLen,
byRead, iActualLen))
{
    if(0 == iActualLen[0])
        continue;

    /* Process received byRead byte array ... */

    for(int i = 0; i < byRead.length; i++)
        byRead[i] = 0;

    iActualLen[0] = 0;
}
```

# Appendix G. SystemETK

## G.1. Reboot the Machine

The following is the sample code:

```
private SystemETK m_system = null;

if(null == m_system)
    m_system = new SystemETK();

if(SmartETK.S_OK != m_system.reboot())
    return;
```

## G.2. Suspend the Machine

The following is the sample code:

```
private SystemETK m_system = null;

if(null == m_system)
    m_system = new SystemETK();

if(SmartETK.S_OK != m_system.suspend())
    return;
```

 **Taiwan Headquarters**


1F, 531 Zhong-Zheng Road  
Xindian, Taipei, 23148  
Taiwan

TEL: 886.2.2218.5452  
FAX: 886.2.2218.5453  
Email: [embedded@via.com.tw](mailto:embedded@via.com.tw)

 **USA**


940 Mission Court  
Fremont, CA 94539  
USA

TEL: 1.510.683.3300  
FAX: 1.510.687.4654  
Email: [embedded@viatech.com](mailto:embedded@viatech.com)

 **Europe**

In den Dauen 6  
53117 Bonn  
Germany

TEL: 49.228.688565.0  
FAX: 49.228.688565.19  
Email: [embedded@via-tech.eu](mailto:embedded@via-tech.eu)

 **China**

Tsinghua Science Park Bldg. 7  
No. 1 Zongguancun East Road  
Haiden District, Beijing, 100084  
China

TEL: 86.10.59852288  
FAX: 86.10.59852299  
Email: [embedded@viatech.com.cn](mailto:embedded@viatech.com.cn)

 **Japan**

3-15-7 Ebisu MT Bldg. 6F  
Higashi, Shibuya-ku  
Tokyo 150-0011  
Japan

TEL: 81.3.5466.1637  
FAX: 81.3.5466.1638  
Email: [embedded@viatech.co.jp](mailto:embedded@viatech.co.jp)