



PROGRAMMING GUIDE

VIA Smart ETK SDK

v1.0.0



Copyright

Copyright © 2017 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 (including use or connection of extra device/equipment/add-on card) 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	Remark
1.00	08/11/2017	Initial programming guide for VIA Smart ETK SDK v1.0.0



Table of Contents

1. Introduction.....	1
2. Install Development Environment	2
3. Smart ETK SDK API.....	7
3.1. Function Return Values.....	7
3.2. Information.....	9
3.2.1. INFO Class	9
3.2.2. Query Supported Modules	9
3.2.3. Query Supported Board ID	9
3.2.4. Get Board ID.....	10
3.2.5. Get Board Version.....	10
3.2.6. Get Board Serial Number	10
3.2.7. Get Board Information.....	11
3.2.8. Get Smart ETK SDK Version	11
3.2.9. Get Board Name.....	12
3.3. CAN Bus	13
3.3.1. CAN Class.....	13
3.3.2. Query CAN Device List.....	13
3.3.3. Open CAN Device.....	13
3.3.4. Close UART Device	14
3.3.5. Set Bitrate for CAN Device	14
3.3.6. Get Bitrate for CAN Device.....	14
3.3.7. Set Timeout Value for CAN Device	15
3.3.8. Get Timeout of CAN Device.....	15
3.3.9. Enable/Disable Loopback.....	15
3.3.10. Get Loopback.....	16
3.3.11. Receive Messages from the CAN Device.....	16
3.3.12. Receive Message Status from the CAN Device.....	17
3.3.13. Set Filter of CAN Device	17
3.3.14. Read the Frame from CAN Device	18
3.3.15. CAN Device be written with the Frame	18
3.4. GPIO	19
3.4.1. GPIO Class.....	19
3.4.2. Query GPIO List.....	19
3.4.3. Enable GPIO Pin	20
3.4.4. Get GPIO Pin Status	20
3.4.5. Set GPIO Pin Direction.....	20
3.4.6. Get GPIO Pin Direction.....	21



3.4.7. Set GPO Value.....	21
3.4.8. Get GPI Value	21
3.5. Network Wake-On-LAN	22
3.5.1. Network Class.....	22
3.5.2. Query Wake-on-LAN Support.....	22
3.5.3. Enable/Disable Wake-on-LAN	22
3.5.4. Get Wake-on-LAN Status	23
3.6. UART	24
3.6.1. UART Class	24
3.6.2. Query UART Device List.....	24
3.6.3. Query UART Mode	24
3.6.4. Query Baud Rate List	25
3.6.5. Open UART Device	25
3.6.6. Close UART Device	26
3.6.7. Configure UART Device	26
3.6.8. Get Configurations of UART Device.....	27
3.6.9. Set Timeout of UART Device	27
3.6.10. Get Timeout of UART Device.....	28
3.6.11. Read Data from UART Device	28
3.6.12. Send Data to UART Device	29
3.6.13. Reset UART Device	29
3.6.14. Enable/Disable of RS-485 Mode.....	29
3.6.15. Get Enable/Disable Status for RS-485 Mode	30
3.7. Watchdog Timer	31
3.7.1. Watchdog Timer Class.....	31
3.7.2. Query Minimum Timeout.....	31
3.7.3. Query Maximum Timeout.....	31
3.7.4. Query Default Timeout.....	32
3.7.5. Enable/Disable the Watchdog Timer.....	32
3.7.6. Get Watchdog Timer Status.....	32
3.7.7. Refresh Watchdog Timer	33
3.7.8. Set Watchdog Timer Timeout Value.....	33
3.7.9. Get Watchdog Timer Timeout Value.....	33
Appendix A. INFO.....	34
A.1. The Usage of querySupportedModules().....	34
A.2. The Usage of getSmartETKSDKVersion()	34
Appendix B. GPIO	35
B.1. The Usage of queryGPIOList()	35
B.2. The Usage of setDirection()	36



Appendix C. Watchdog Timer	37
C.1. The Usage of queryTimeoutMin and queryTimeoutMax.....	37
Appendix D. UART	38
D.1. The Usage of queryUarts	38
D.2. The Usage of queryBaudRateList	38



1. Introduction

This section provides an overview of the Smart ETK program which is designed with enhanced features including APIs for CAN Bus, GPIO, Network (Wake-on-LAN), UART and Watchdog Timer function.

The Smart ETK is programmed with binder as a communication between JAVA and C language to control the hardware modules. The bbservice (Board Support Service) is implemented in order to perform a requested function from a Smart ETK API.

The Smart ETK service defines the internal listening IP address as 127.0.0.1. The purpose of this IP address is to keep it from establishing a connection with external networks.

The VIA Smart ETK SDK is compatible with the following mainboards and system:

- ALTA DS 4K Android BSP v1.0 and above
- VAB-820 Android BSP v5.0 and above
- AMOS-820 Android BSP and v5.0 above
- ARTiGO A820 Android BSP and v5.0 above

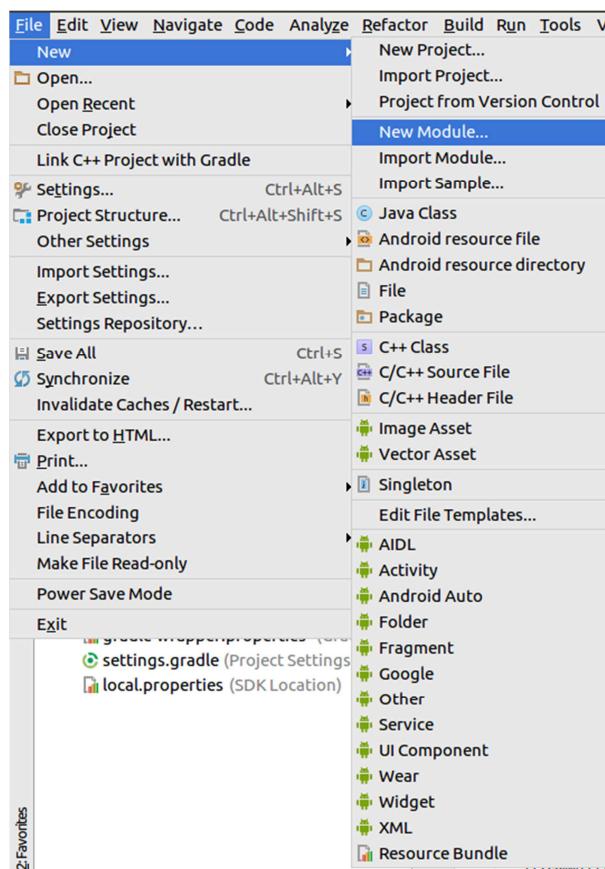


2. Install Development Environment

The Android Studio is an officially recommended and integrated development environment (IDE) in place of the old Eclipse for the Android application. It requires importing the specific JAR library, which can be extracted from the Smart_ETK_v1.0_SourceCode.zip file, in order to integrate the Smart ETK enhanced features in your development.

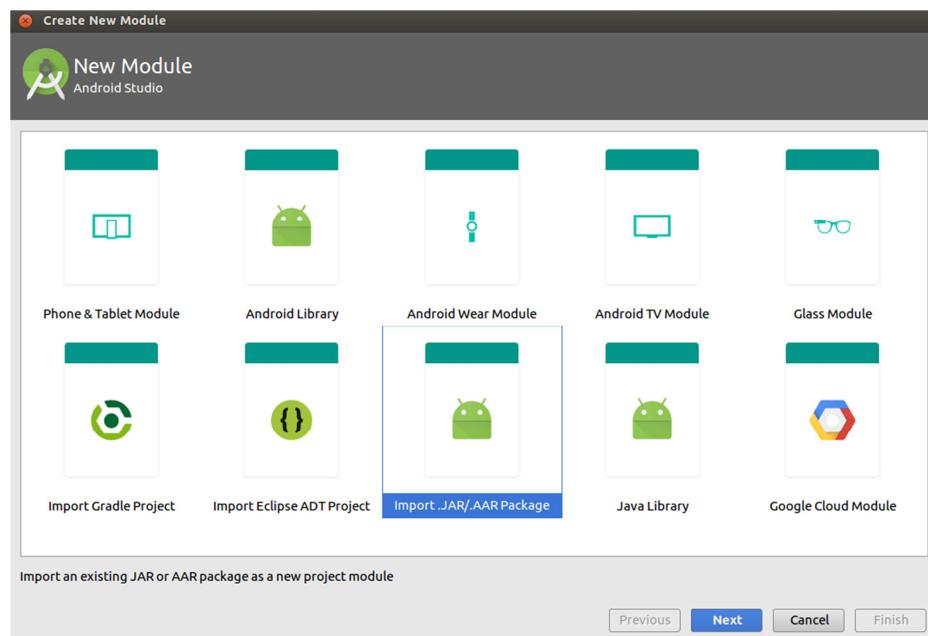
After creating a new project in the Android Studio, import the JAR library from the Smart ETK to the project. Please follow the steps below to import the JAR library.

1. Start Android Studio and create a new module in the project.

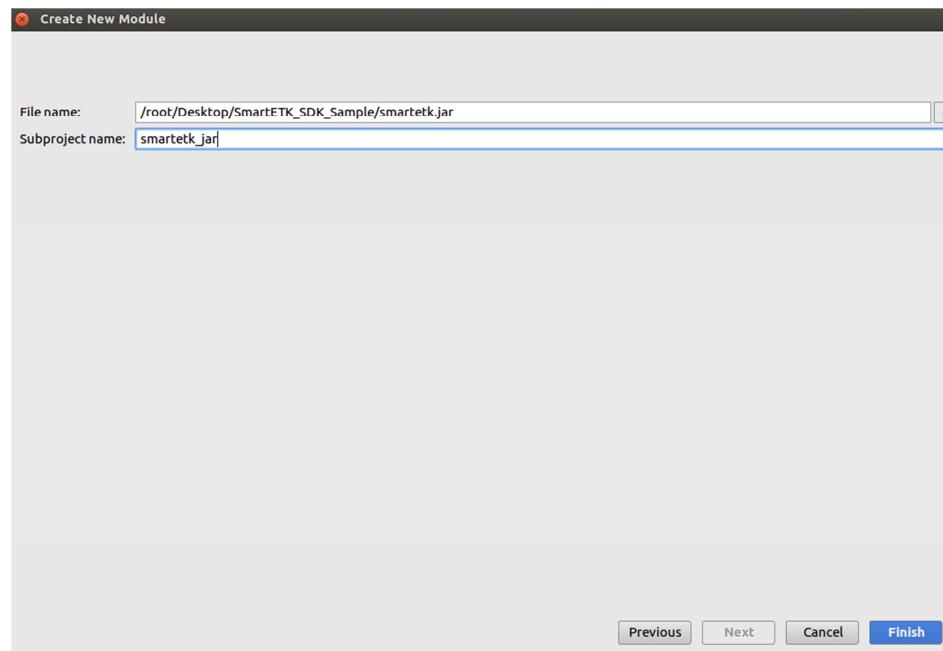




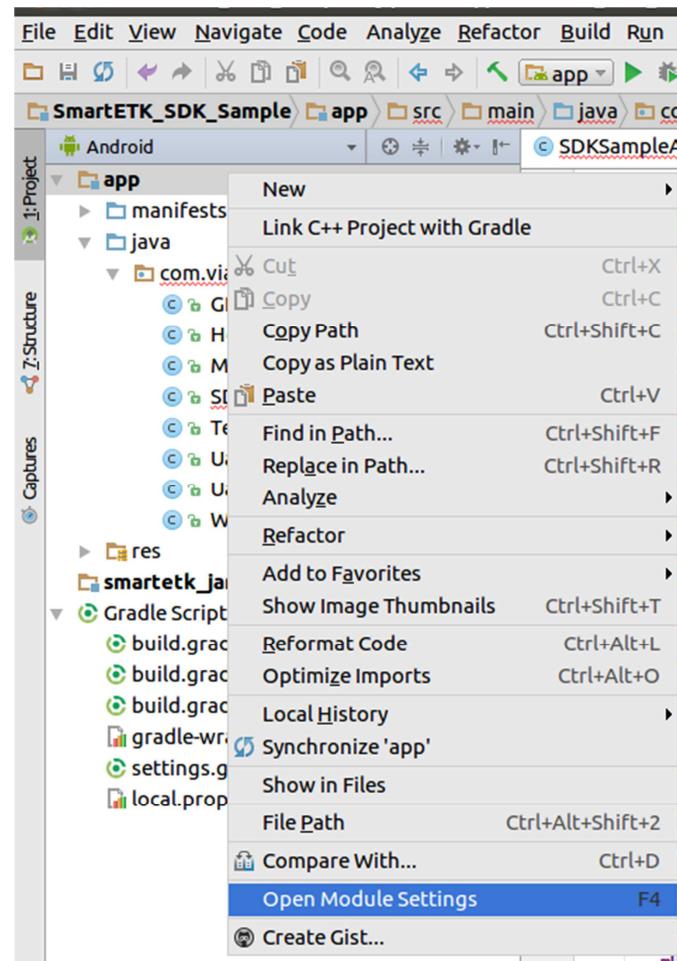
2. Choose **Import.JAR/.AAR Package** in the pop-up window and then click Next.



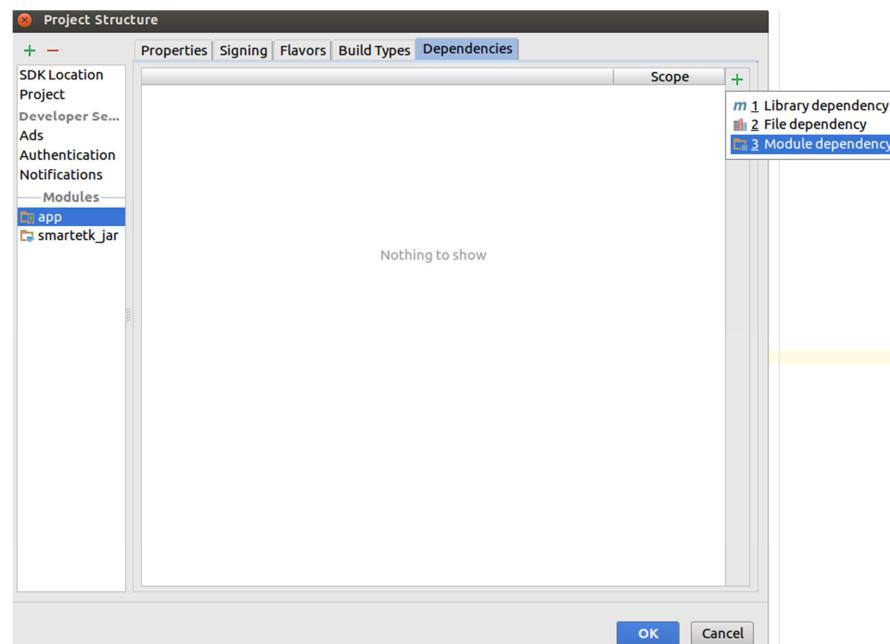
3. Specify the file name (SmartETK_SDK_Smaple) and path of the JAR library, and then click Finish.



4. Open the module settings.

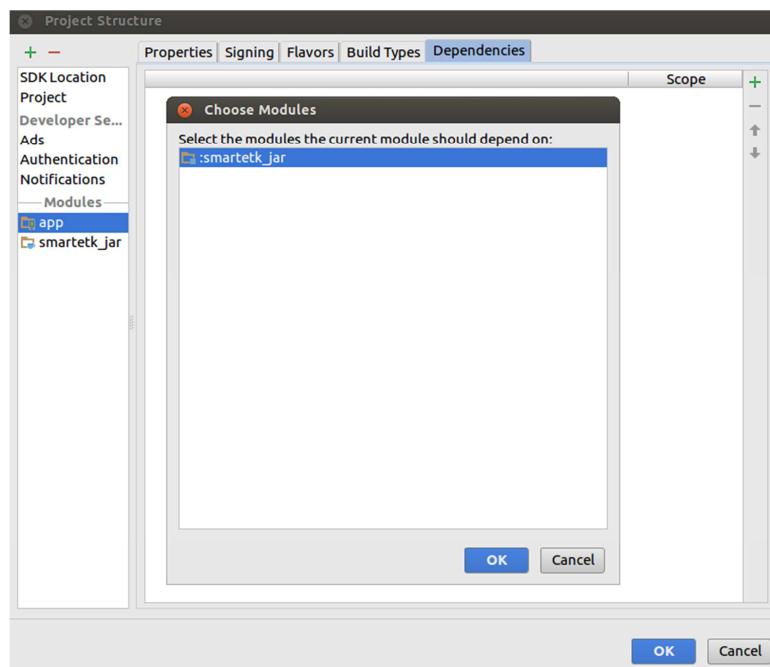


5. Select number 3. Module dependency in the pop-up window.

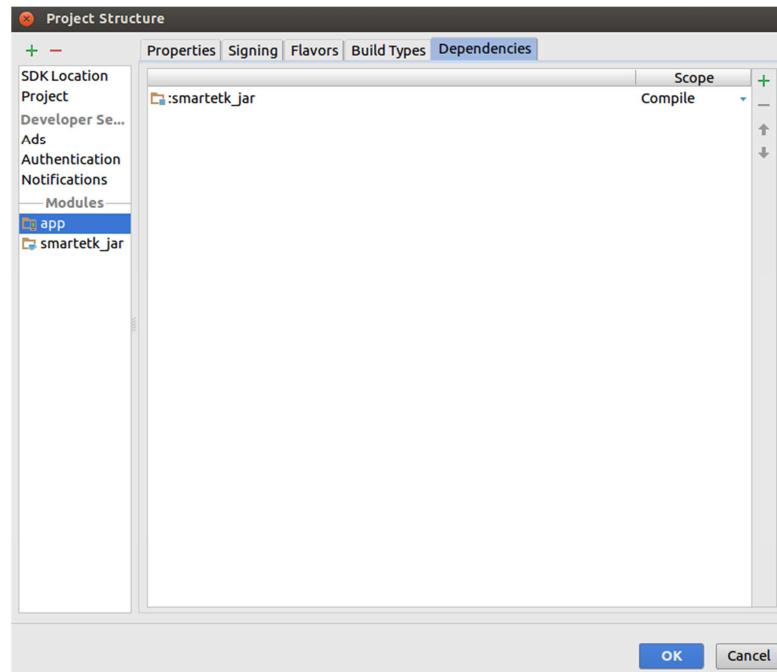




6. Choose the module to import the: smartetk_jar.



7. After importing the module (smartetk_jar), it will be shown in the window, then click OK to complete this setting.





*Important: For the project to communicate with the **bbservice** in VIA BSP using TCP protocol, please add the red highlighted area below in the AndroidManifest.xml before the element application:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.vab820standardeexample"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="21"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name="viaembedded.vab820.examples.smartetk.SDKSampleActivity"
            android:configChanges="orientation|screenSize|keyboardHidden|keyboard"
            >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```



3. Smart ETK SDK API

This section explains the seven types of Classes that are found in the smartetk.jar. The smartetk.jar should be imported in the Smart ETK demo APP source code with a package name "com.viaembedded.smartetk".

The INFO, GPIO, Watchdog timer, Network WOL, UART, and CAN classes are placed in the package "com.viaembedded.smartetk" and the returned values are placed in Java class "com.viaembedded.smartetk.SmartETK".

3.1. Function Return Values

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

Static Int Variable	Return Value	Description
SmartETK.S_OK	0x00	When a function returns the S_OK, it indicates that the function has been completed successfully.
SmartETK.E_FAIL	-0x01	When a function returns the E_FAIL, it indicates that the function has failed to complete.
SmartETK.E_VERSION_NOT_SUPPORT	-0x02	When a function returns the E_VERSION_NOT_SUPPORT, it indicates that the versions of SmartETK.jar and bbservice are not compatible.
SmartETK.E_INVALID_ARG	-0x04	When a function returns the E_INVALID_ARG, it indicates that the arguments are invalid.
SmartETK.E_FUNC_NOT_SUPPORT	-0x06	When a function returns the E_FUNC_NOT_SUPPORT, it indicates that the function is not supported on this board.
SmartETK.E_CONNECTION_FAIL	-0x08	When a function returns the E_CONNECTION_FAIL, it indicates that the bbservice didn't respond to the request. Please make sure bbservice is running successfully.
SmartETK.E_NOT_RESPOND_YET	-0x09	When a function returns the E_CONNECTION_FAIL, it indicates that the bbservice function is still running and hasn't been finished yet.



SmartETK.E_TIMEOUT	-0x0A	When a function returns the E_TIMEOUT, it indicates that there is no corresponding data received during the period.
--------------------	-------	---



3.2. Information

This section explains how to use the Information object and its functions.

3.2.1. INFO Class

Description:

Create an info object.

Syntax:

```
Info()
```

Example:

```
Info m_info = new Info();
```

3.2.2. Query Supported Modules

Description:

Query whether the board/system support GPIO, WDT, NETWORK, UART, and CAN functions.

Syntax:

```
int querySupportedModules(long[] lmodules);
```

Parameters:

`lmodules` – return the Module byte value.

If `(1 << (SmartETK.FUN_GPIO - 1)) & bModules == 1`, it supports GPIO.

If `(1 << SmartETK.FUN_WDT - 1) & bModules == 1`, it supports WDT.

If `(1 << SmartETK.FUN_NETWORK - 1) & bModules == 1`, it supports WOL.

If `(1 << SmartETK.FUN_UART - 1) & bModules == 1`, it supports UART.

If `(1 << SmartETK.FUN_CAN - 1) & bModules == 1`, it supports CAN.

Return:

`S_OK` – The function has succeeded;

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.2.3. Query Supported Board ID

Description:

Query whether the board/system support board ID function.

Syntax:

```
int querySupportedBID (boolean[] bSupported);
```

Parameters:

`bSupported` – if it's supported, return `true`; otherwise return `false`.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.2.4. Get Board ID

Description:

Get the board/system ID.

Syntax:

```
int getBoardID (String[] sBID);
```

Parameters:

sBID – return the board ID.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.2.5. Get Board Version

Description:

Get the board/system version number.

Syntax:

```
int getBoardVersion(String[] sVersion);
```

Parameters:

sVersion – return the board/system version number.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.2.6. Get Board Serial Number

Description:

Get the board/system serial number.

Syntax:

```
int getBoardSerialNumber(String[] sSN);
```

Parameters:

sSN – return the board/system serial number.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.2.7. Get Board Information

Description:

Get the board/system information. The information is board/system ID, version number, and serial number.

Syntax:

```
Int getBoardInfo(byte[] sBuffer);
```

Parameters:

sBuffer – return the board/system information.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.2.8. Get Smart ETK SDK Version

Description:

Get the board/system Smart ETK SDK version.

Syntax:

```
int getSmartETKSDKVersion(int[] iVersion);
```

Parameters:

iVersion – return the Smart ETK SDK version number.

There are four bytes for iVersion parameter.

For example:

The v1.10.12.00, number “1” is major version, number “10” is minor version, number “12” is update version, and number “00” is other information. “1” is stored in the highest byte. Followed by decreasing, “00” is stored in lowest byte.

To get major version: (int)((iVersion[0] & 0xff000000) >> 24);

To get minor version: (int)((iVersion[0] & 0x00ff0000) >> 16);

To get update version: (int)((iVersion[0] & 0x0000ff00) >> 8);

To get other version information: (int)(iVersion[0] & 0x000000ff).

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.2.9. Get Board Name

Description:

Get the board/system name.

Syntax:

```
int getBoardName(String[] sBoardName);
```

Parameters:

`sBoardName` – return the board/system name.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.3. CAN Bus

This section explains how to use the CAN Bus object and its functions.

3.3.1. CAN Class

Description:

Create a new CAN object.

Syntax:

```
CAN();
```

Example:

```
CAN m_can= new CAN();
```

3.3.2. Query CAN Device List

Description:

Query supported CAN devices list.

Syntax:

```
int queryCans(String[] sName);
```

Parameters:

`sName` – return name list of supported CAN devices.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.3.3. Open CAN Device

Description:

Open specified CAN device.

Syntax:

```
int open(String sCan);
```

Parameters:

`sCan` – the name of CAN device to open.

Return:

`S_OK` – The function has succeeded.

`E_CAN_OPENFAIL` – The device has failed to open.

`E_CAN_ALREADY_OPENED` – The object has been opened already.

`E_*` – The function has failed by other causes. Please refer to section 3.1 “Function Return Values”.



3.3.4. Close UART Device

Description:

Close specified CAN device.

Syntax:

```
int close();
```

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.3.5. Set Bitrate for CAN Device

Description:

Set the bitrate for specified CAN device.

Syntax:

```
int setBitrate(int iBitrate)
```

Parameters:

iBitrate – bitrate (Ex. 125000)(Default: 500000)

Return:

S_OK – The function has succeeded.

E_CAN_BAUDRATE_NOT_SUPPORT –The function has failed for the bitrate is not supported.

E_* – The function has failed by other causes. Please refer to section 3.1 “Function Return Values”.

3.3.6. Get Bitrate for CAN Device

Description:

Get the bitrate for specified CAN device.

Syntax:

```
int getBitrate(int[] iBitrate);
```

Parameters:

iBitrate – return bitrate. (Ex. 125000)(Default: 500000)

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.3.7. Set Timeout Value for CAN Device

Description:

Set the timeout value for opened CAN device.

Syntax:

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

Parameters:

bEnable – enable or disable timeout function.

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

bEnable = true, iTimeout = 0 – polling read

bEnable = true, iTimeout > 0 – read with timeout

bEnable = false – blocking read

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.3.8. Get Timeout of CAN Device

Description:

Get the timeout configuration of the opened CAN device.

Syntax:

```
int getTimeout(Timeout timeout);
```

Parameters:

timeout – Get timeout configuration through the object of **Timeout** object.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.3.9. Enable/Disable Loopback

Description:

Enable/Disable loopback function.

Syntax:

```
int setLoopback(boolean bEnable);
```

Parameters:

bEnable = true for enable, **false** for disable.

bEnable = true (only when **setRecvOwnMsgs()** is also set to true, it will receive its own messages after transmit).



bEnable = false (no matter setRecvOwnMsgs() set to true or false, it won't receive its own messages after transmit)

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 "Function Return Values".

Note:

For demo APK source code, the loopback function is enabled by default.

3.3.10. Get Loopback

Description:

Get loopback status from opened CAN device.

Syntax:

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

Parameters:

bEnable – return true for enabled, return false for disabled.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 "Function Return Values".

3.3.11. Receive Messages from the CAN Device

Description:

Set CAN_RAW_RECV_OWN_MSGS flag to decide whether the CAN device will receive data frames which is sent by itself.

Syntax:

```
int setRecvOwnMsgs(boolean bEnable);
```

Parameters:

bEnable – enable or disable.

bEnable = true (only if setLoopback() is set to true, it will receive its own messages after sending CAN frame)

bEnable = false (it won't receive its own messages after sending CAN frame)

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 "Function Return Values".



3.3.12. Receive Message Status from the CAN Device

Description:

Get the status of receiving frames sent by itself.

Syntax:

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

Parameters:

bEnable – return `true` for enabled, `false` for disabled.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.3.13. Set Filter of CAN Device

Description:

Set CAN device's filters.

Syntax:

```
int setFilter(CanFilter[] canFilter, int iLength);
```

Parameters:

`canFilter` – `CanFilter` object array to set.

`iLength` – number of `CanFilter` object to set, `0` represents to disable the reception of CAN frames.

A filter matches when `received_can_id` is not null and `CanFilter.iCanMask == CanFileter.iCanMaster`.

To disable the reception of CAN frames, `setFilter(null, 0)`.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.3.14. Read the Frame from CAN Device

Description:

Read CAN Frame from the opened CAN devices.

Syntax:

```
int readFrame(CanFrame canFrame);
```

Parameters:

canFrame – CanFrame object to read.

Return:

[S_OK](#) – The function has succeeded.

[E_*](#) – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.3.15. CAN Device be written with the Frame

Description:

Write CAN Frame to opened CAN device.

Syntax:

```
int writeFrame(CanFrame canFrame);
```

Parameters:

canFrame – CanFrame object to write.

Return:

[S_OK](#) – The function has succeeded.

[E_*](#) – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.4. GPIO

This section explains how to use the GPIO object and its functions.

3.4.1. GPIO Class

Description:

Create a new GPIO object.

Syntax:

```
GPIO();
```

Example:

```
GPIO m_gpio = new GPIO();
```

3.4.2. Query GPIO List

Description:

Query the GPIO pin and type list.

Syntax:

```
int queryGPIOList(ArrayList<Integer> listGPIO, ArrayList<Integer> listStatus);
```

Parameters:

`listGPIO` – return the GPIO pin list, such as 1,2,4,8...

`listStatus` – return the GPIO type list, such as 0x1, 0x0, 0xff...

Note:

0x1 is a GPO device, 0x0 is a GPI device, and 0xff is a GPIO device.

The `setDirection` API can set GPIO pin direction (GPI or GPO).

The length of `listStatus` is the same as the length of `listGPIO`, and they're one by one mapping.

For example:

When pin 8 position in `listGPIO` is 0, its type value is the one at position 0 in `listStatus`.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.4.3. Enable GPIO Pin

Description:

Enable specified GPIO pin.

Syntax:

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

Parameters:

iGPIO – GPIO pin number.

enable – To set `true` is enable and `false` is disable.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.4.4. Get GPIO Pin Status

Description:

Get the specified GPIO pin status.

Syntax:

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

Parameters:

iGPIO – GPIO pin number.

enable – return `true` is enable and `false` is disable.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.4.5. Set GPIO Pin Direction

Description:

Set input/output direction for specified GPIO pin.

Syntax:

```
int setDirection(int iGPIO, int direction);
```

Parameters:

iGPIO – GPIO pin number.

direction – `GPIO.GM_GPI` for input direction.

`GPIO.GM_GPO` for output direction.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.4.6. Get GPIO Pin Direction

Description:

Get the specified GPIO pin direction.

Syntax:

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

Parameters:

iGPIO – GPIO pin number.

direction – return GPIO.GM_GPI for input direction;

return GPIO.GM_GPO for output direction.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.4.7. Set GPO Value

Description:

Set output signal for specified GPIO pin.

Syntax:

```
int setValue(int iGPIO, int value);
```

Parameters:

iGPIO – GPIO pin number.

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

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.4.8. Get GPI Value

Description:

Get input signal for specified GPIO pin.

Syntax:

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

Parameters:

iGPIO – GPIO pin number.

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

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.5. Network Wake-On-LAN

This section explains how to use the Network Wake-On-LAN object and its functions.

3.5.1. Network Class

Description:

Create a Network object.

Syntax:

```
Network()
```

Example:

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

3.5.2. Query Wake-on-LAN Support

Description:

Query whether board/system support Wake-on-LAN function.

Syntax:

```
int querySupportedWOL(boolean[] bSupported);
```

Parameters:

bSupported – return **true** is support and **false** is not support.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.5.3. Enable/Disable Wake-on-LAN

Description:

Enable/Disable the Wake-on-LAN function.

Syntax:

```
int setWakeOnLan(boolean iEnable);
```

Parameters:

iEnable – **true** for enable, **false** for disable.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.5.4. Get Wake-on-LAN Status

Description:

Get wake-on-LAN status.

Syntax:

```
Int getWakeOnLan(boolean[] iEnable);
```

Parameters:

iEnable – true for enabled, false for disabled.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.6. UART

This section explains how to use the UART object and its functions.

3.6.1. UART Class

Description:

Create a new UART object.

Syntax:

```
Uart();
```

Example:

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

3.6.2. Query UART Device List

Description:

Query supported UART device list.

Syntax:

```
Int queryUarts(String[] sName)
```

Parameters:

sName – return supported UART device list.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.3. Query UART Mode

Description:

Query supported mode for specified UART device.

Syntax:

```
int queryCapability(String sName, long[] iSupportedModes);
```

Parameters:

sName – the UART device name.

iSupportedModes – return the supported mode for specified UART device.

For example, if `UART.MODE_UART & iSupportedModes == 1`, it shows this UART supports UART mode;

If `UART.MODE_COM & iSupportedModes == 1`, it shows this UART supports COM mode;

If `UART.MODE_RS485 & iSupportedModes == 1`, it shows this UART supports RS485 mode;



If `UART.MODE_RS422 & iSupportedModes == 1`, it shows this UART supports RS422 mode.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.4. Query Baud Rate List

Description:

Query supported baud rate for specified UART device.

Syntax:

```
int queryBaudRateList(String sName, ArrayList<Integer> baudrate_list);
```

Parameters:

`sName` – the UART device name.

`baudrate_list` – return the supported baudrate value; `baudrate_list` is arraylist data, such as {0, 50, 75, 110}.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.5. Open UART Device

Description:

Open specified UART device.

Syntax:

```
int open(String sUart);
```

Parameters:

`sUart` – Open specified UART device name such as `ttyUSB0`

Return:

`S_OK` – The function has succeeded.

`E_UART_OPENFAIL` – The device has failed to open.

`E_UART_ALREADY_OPENED` – The object has been opened already.

`E_UART_TTY_BEEN_USED` – The device is being used by other object.

`E_*` – The function has failed by other cause. Please refer to section 3.1 “Function Return Values”.

Important:

If the UART device has been opened already, the open function will return fault value

`E_UART_ALREADY_OPENED`. You need to call reset function to release the UART

device and call open function to open UART device again.



3.6.6. Close UART Device

Description:

Close specified UART device.

Syntax:

```
int close();
```

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.7. Configure UART Device

Description:

Configure the opened UART device.

Syntax:

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

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.

byFlowControl – flow control. **0**: none; **1**: CTS/RTS.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.6.8. Get Configurations of UART Device

Description:

Get the configurations of the opened UART device.

Syntax:

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

Parameters:

UC – Get UART Configuration through the object of UartConfig class.

```
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

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

 byte byFlowControl – flow control. 0: none; 1: CTS/RTS

```
}
```

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

Example:

```
UartConfig UC = m_uart.new UartConfig()
if (SmartETK.S_OK != m_uart.getConfig(UC))
{
    cleanStatus();
    return;
}
```

3.6.9. Set Timeout of UART Device

Description:

Set the timeout value of receiving Data for the opened UART device.

Syntax:

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

Parameters:

bEnable – enable or disable timeout function.

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

bEnable = true, iTimeout = 0 (polling read)

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

bEnable = false (blocking read)

**Return:**

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.10. Get Timeout of UART Device

Description:

Get the timeout configuration of the opened UART device.

Syntax:

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

Parameters:

T – Get timeout configuration through the object of Timeout class.

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 – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.11. Read Data from UART Device

Description:

Receive data from opened UART device.

Syntax:

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

Parameters:

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

byRead – the byte array buffer to read.

iActualLen – the actual number of bytes received.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.6.12. Send Data to UART Device

Description:

Send data to opened UART device.

Syntax:

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

Parameters:

`byWrite` – pointer to data buffer.

`iByteOffset` – the start position in `byWrite` from where to get bytes.

`iWriteLen` – number of bytes from buffer to transmit, maximum 1024 bytes per transmission.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.13. Reset UART Device

Description:

Reset specified UART device.

Syntax:

```
int reset();
```

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.6.14. Enable/Disable of RS-485 Mode

Description:

Enable/Disable RS-485 mode and this function only support ARTiGO A820.

Please note to check hardware switch.

Syntax:

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

Parameters:

`bEnable` – if `true`, RS-485 software mode is enabled; if `false` RS-485 software mode is disabled.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.6.15. Get Enable/Disable Status for RS-485 Mode

Description:

Get the enabled/disabled status for RS-485 mode.

Syntax:

```
int getRS485(boolean[] bEnable);
```

Parameters:

bEnable – return `true` for enabled, return `false` for disabled.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.7. Watchdog Timer

This section explains how to use the Watchdog timer object and its functions.

3.7.1. Watchdog Timer Class

Description:

Create a new watchdog object.

Syntax:

```
WatchDog();
```

Example:

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

3.7.2. Query Minimum Timeout

Description:

Query the minimum timeout value of watchdog timer.

Syntax:

```
int queryTimeoutMin(int[] iMinValue);
```

Parameters:

iMinValue – return the minimum timeout value of watchdog timer.

Return:

S_OK – The function has succeeded.

*E_** – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.7.3. Query Maximum Timeout

Description:

Query the maximum timeout value of watchdog timer.

Syntax:

```
int queryTimeoutMax(int[] iMaxValue);
```

Parameters:

iMaxValue – return the maximum timeout value of watchdog timer.

Return:

S_OK – The function has succeeded.

*E_** – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.7.4. Query Default Timeout

Description:

Query the default timeout value of watchdog timer.

Syntax:

```
int queryDefaultTimeout(int[] iDefaultValue);
```

Parameters:

`iDefaultValue` – return the default timeout value of watchdog timer.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.7.5. Enable/Disable the Watchdog Timer

Description:

Enable/Disable watchdog timer function.

Syntax:

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

Parameters:

`bEnable` – Set `true` to enable watchdog timer, `false` to disable it.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.

Note:

When the watchdog timer function is enabled, you need to refresh the timer before the time is out, otherwise the system will reboot.

3.7.6. Get Watchdog Timer Status

Description:

Get the watchdog timer status.

Syntax:

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

Parameters:

`bEnable` – return `true` for enabled, return `false` for disabled.

Return:

`S_OK` – The function has succeeded.

`E_*` – The function has failed. Please refer to section 3.1 “Function Return Values”.



3.7.7. Refresh Watchdog Timer

Description:

Refresh the watchdog timer.

Syntax:

```
int keepAlive();
```

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.7.8. Set Watchdog Timer Timeout Value

Description:

Set the watchdog timer timeout value.

Syntax:

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

Parameters:

iTimeout – timeout value in seconds.

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.

3.7.9. Get Watchdog Timer Timeout Value

Description:

Get the watchdog timer timeout value.

Syntax:

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

Parameters:

iTimeout – timeout value in seconds

Return:

S_OK – The function has succeeded.

E_* – The function has failed. Please refer to section 3.1 “Function Return Values”.



Appendix A. INFO

This section has two partial source codes for the INFO object, they are querySupportedModules() and getSmartETKSDKVersion(). All sample source codes are provided in BSP package.

A.1. The Usage of querySupportedModules()

The querySupportModules() function returns supported modules on board; the sample codes are as follows:

```
if (mINFO == null) {  
    mINFO = new INFO();  
}  
  
if (SmartETK.S_OK == mINFO.querySupportedModules(lModules)) {  
    if ((lModules[0] & (1 << (SmartETK.FUN_WDT-1))) != 0) {  
        //mMenuItems.add(new String("Watchdog"));  
        menuItem = new HashMap<String, Object>();  
        menuItem.put("values", "Watchdog");  
        menuItem.put("images", R.drawable.btn_watchdog);  
        mMenuItems.add(menuItem);  
        mMsgID.add(MI_WATCHDOG);  
    }  
    if ((lModules[0] & (1 << (SmartETK.FUN_GPIO-1))) != 0) {  
        //mMenuItems.add(new String("GPIO"));  
        menuItem = new HashMap<String, Object>();  
        menuItem.put("values", "GPIO");  
        menuItem.put("images", R.drawable.btn_gpio);  
        mMenuItems.add(menuItem);  
        mMsgID.add(MI_GPIO);  
    }  
    if ((lModules[0] & (1 << (SmartETK.FUN_UART-1))) != 0) {  
        //mMenuItems.add(new String("UART"));  
        menuItem = new HashMap<String, Object>();  
        menuItem.put("values", "UART");  
        menuItem.put("images", R.drawable.btn_uart);  
        mMenuItems.add(menuItem);  
        mMsgID.add(MI_UART);  
    }  
}
```

A.2. The Usage of getSmartETKSDKVersion()

The getSmartETKSDKVersion() function is used to get the current Smart ETK JAR package's version; it returns an int value, user could do as following codes to get major version, minor version, update version and others;

```
INFO info = new INFO();  
String[] sBoardName = new String[1];  
info.getBoardName(sBoardName);  
String ss = new String(getResources().getString(R.string.home_title_string) + " " + sBoardName[0]);  
mHomeTitle.setText(ss);  
  
int[] iVersion = new int[1];  
info.getSmartETKSDKVersion(iVersion);  
  
int majorVersion = (int)((iVersion[0] & 0xff000000) >> 24);  
int minorVersion = (int)((iVersion[0] & 0x00ff0000) >> 16);  
int updateVersion = (int)((iVersion[0] & 0x0000ff00) >> 8);  
int other = (int)(iVersion[0] & 0x000000ff);
```



Appendix B. GPIO

This section has two partial source codes for the GPIO object, they are queryGPIOList() and setDirection(). All sample source codes are provided in BSP package.

B.1. The Usage of queryGPIOList()

The queryGPIOList function returns the GPIO number and GPIO type of each GPIO;

There are three kinds of GPIO type, GPI, GPO and GPIO;

The usage of queryGPIOList is as follows:

```
if (mGPIO == null) {
    mGPIO = new GPIO();
}

if (mGPIOMapTable == null) {
    mGPIOMapTable = new GPIOMapTable();
}

ArrayList<Integer> deviceList = new ArrayList<Integer>();
ArrayList<Integer> statusList = new ArrayList<Integer>();
mGPIO.queryGPIOList(deviceList, statusList);
mGPIOMapTable.setDeviceStatus(deviceList, statusList);

public void setDeviceStatus(ArrayList<Integer> deviceList, ArrayList<Integer> statusList) {
    if (deviceList == null || statusList == null) {
        Log.e(TAG, "input deviceList or statusList is null");
        return;
    }

    if (deviceList.size() != statusList.size()) {
        Log.e(TAG, "device size is not equal status size");
        return;
    }

    for (int i = 0; i < deviceList.size(); i++) {
        switch ((statusList.get(i))) {
            case GPIO.GM_GPI:
                if (mGPIDeviceList.size() <= MAX_DEVICE_SHOWING) {
                    mGPIDeviceList.add(new GPIDevice(deviceList.get(i)));
                }
                break;
            case GPIO.GM_GPO:
                if (mGPODeviceList.size() <= MAX_DEVICE_SHOWING) {
                    mGPODeviceList.add(new GPODevice(deviceList.get(i)));
                }
                break;
            case GPIO.GM_GPIO:
                if (mGPIODeviceList.size() <= MAX_DEVICE_SHOWING) {
                    mGPIODeviceList.add(new GPIODevice(deviceList.get(i)));
                }
                break;
            default:
                break;
        }
    }
}
```



B.2. The Usage of setDirection()

To set GPIO pin to GPI or GPO, For example, call setDirection() to set GPIO direction,

```
mGPIO.setDirection(mDeviceId, GPIO.GM_GPI);
```

mDeviceId is the value got from queryGPIOList()



Appendix C. Watchdog Timer

This section has two partial source codes for the Watchdog timer object, they are queryTimeoutMin() and queryTimeoutMax(). All sample source codes are provided in BSP package.

C.1. The Usage of queryTimeoutMin and queryTimeoutMax

These two functions return the minimum or maximum value of timeout respectively;

The usage is as follows:

```
private int[] mTimeoutMin = new int[1];
private int[] mTimeoutMax = new int[1];

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

m_watchdog.queryTimeoutMin(mTimeoutMin);
m_watchdog.queryTimeoutMax(mTimeoutMax);
```



Appendix D. UART

This section has two partial source codes for the UART object, they are queryUarts() and queryBaudRateList(). All sample source codes are provided in BSP package.

D.1. The Usage of queryUarts

The queryUarts function returns the name list of all UART devices on board.

The usage is as follows:

```
/* new Uart service */
m_uart=new Uart();

/* new Uart settings */
m_setUC = m_uart.new UartConfig();
m_setT = m_uart.new Timeout();
m_setRC = m_uart.new ReturnChar();
m_getUC = m_uart.new UartConfig();
m_getT = m_uart.new Timeout();
m_getRC = m_uart.new ReturnChar();

selectMode(DISCONNECT);

//get uart names
mUartNames.clear();
m_uart.queryUarts(mUartNames);
```

D.2. The Usage of queryBaudRateList

The queryBaudRateList function returns supported baud rates of UART device, its usage is as follows

```
// get all uart baudrate;
ArrayList<Integer> baudrate_list = new ArrayList<Integer>();
for (int i = 0; i < mUartNames.size(); i++) {
    m_uart.queryBaudRateList(mUartNames.get(i), baudrate_list);
    mUartBaudrates.put(mUartNames.get(i), baudrate_list);
}
```



 Taiwan Headquarters
1F, 531 Zhong-zheng Road,
Xindian Dist., New Taipei City 231
Taiwan

Tel: 886-2-2218-5452
Fax: 886-2-2218-9860
Email: embedded@via.com.tw

 USA
940 Mission Court
Fremont, CA 94539,
USA

Tel: 1-510-687-4688
Fax: 1-510-687-4654
Email: embedded@viatech.com

 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

 China
Tsinghua Science Park Bldg. 7
No. 1 Zongguancun East Road,
Haidian Dist., Beijing, 100084
China

Tel: 86-10-59852288
Fax: 86-10-59852299
Email: embedded@viatech.com.cn

 Europe
Email: embedded@via-tech.eu