PROGRAMMING GUIDE

VIASmart ETK_SDK

V0.0.16

1.00-07302015-180600

Copyright

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 the make changes to the products described in this manual at any time without prior notice.

Revision History

Version	Date	Remarks
1.00	2015/07/29	Initial draft- SmartETK_VAB-820 v0.0.16.

Table of Contents

1. Introduction

VIA Smart ETK SDK supports the hardware controlling API for GPIO, Watch Dog, and UART (RS-232) 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 such as bss_vab820 to meet the request from Smart ETK API.

2. System Requirements

2.1. Hardware Requirements

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

VAB-820 Android BSP 2.0 above

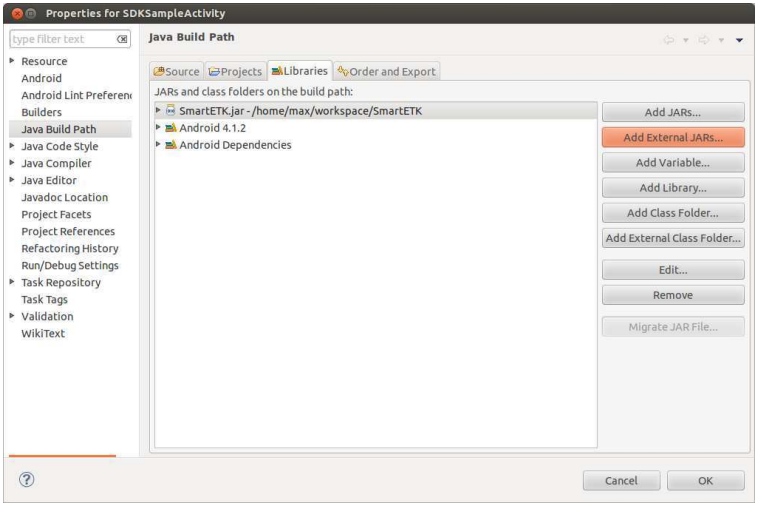
2.2. Software Requirements

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

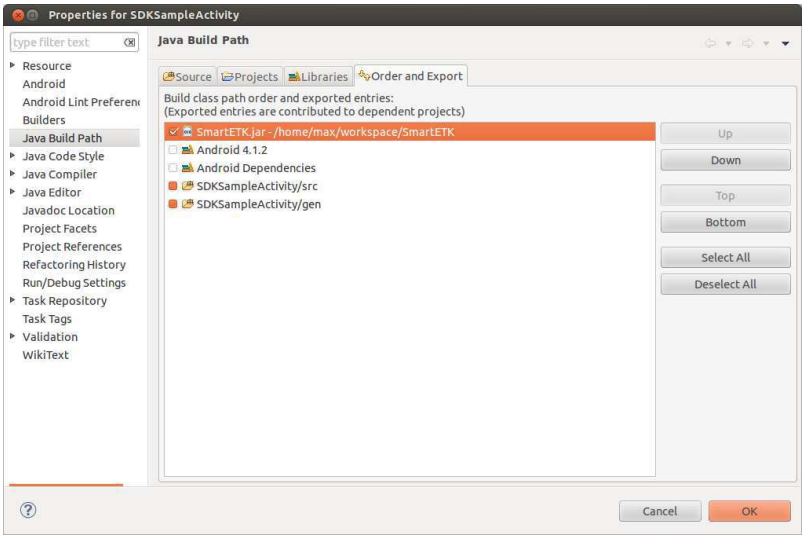
3. Installation and Usage

3.1 Installation on the development computer

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



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



3.2 Installation on the target board

Install the firmware released by VIA and it is done.

4. Smart ETK SDK API

4.1 Class definitions

GPIO, Watch Dog and UART modules are placed in class named `com.viaembedded.smartetk`, and returned values are 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 of 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 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 no corresponding data has been 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.

4.2.10SmartETK.E_UART_NOT_OPEN

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

4.2.11SmartETK.E_UART_ALREADY_OPENED

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

4.2.12SmartETK.E_UART_TTY_BEEN_USED

When Uart.open() returns the `E_UART_TTY_BEEN_USED` value, it indicates that the tty device has 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.

4.2.13SmartETK.E_UART_BAUDRATE_NOT_SUPPORT

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

Start from here

4.2.14SmartETK.E_CAN_OPENFAIL

When Can.open() returns the `E_CAN_OPENFAIL` value, it indicates that the CAN device can't be opened successfully. Please make sure the name of the CAN device exists.

4.2.15SmartETK.E_CAN_NOT_OPEN

When a function returns the `E_CAN_NOT_OPEN` value, it indicates that can object cannot be operated normally. The reason might be that the application doesn't open can device before calling other operating function.

4.2.16SmartETK.E_CAN_ALREADY_OPENED

When Can.open() returns the `E_CAN_ALREADY_OPENED` value, it indicates that the can object has been opened. If you need to open other can device, please call close function to close the current device, then open the other can again.

4.2.17SmartETK.E_CAN_BAUDRATE_NOT_SUPPORT

When Can.setBitrate() returns the `E_CAN_BAUDRATE_NOT_SUPPORT` value, it indicates that bit 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: 1, 2, 4, 5, 7, 8, 9, 16.

Parameters:

`pinID` – GPIO's pin ID.

Example:

Create a GPIO object with pin ID 5.

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

4.3.2 GPIO.setEnable()

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 succeeded.

`E_*` – if the function fails.

4.3.3 GPIO.getEnable()

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 succeeded.

`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 succeeded.

`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 succeeded.

`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 succeeded.

`E_*` – if the function fails.

4.4 WatchDog Class

4.4.1 WatchDog Class

Syntax:

```
WatchDog();
```

Description:

Create a new WatchDog object.

Example:

Create a WatchDog object.

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

4.4.2 WatchDog.setEnable()

Syntax:

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

Description:

Enable or disable watch dog function. SmartETK service will feed the watch dog within a period automatically. Once watch dog function is enabled, it must be fed within a period(keepAlive()), otherwise the system will reboot.

Parameters:

`bEnable` – enable or not to enable watch dog function.

Return:

S_OK – if the function succeeded.

E_* – if the function fails.

4.4.3 WatchDog.getEnable()

Syntax:

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

Description:

Get enable state of the watch dog function.

Parameters:

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

Return:

S_OK – if the function succeeded.

E_* – if the function fails.

4.4.4 WatchDog.keepAlive()

Syntax:

```
int keepAlive ();
```

Description:

Keep watch dog alive to avoid rebooting the system.

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

4.4.5 WatchDog.setTimeout()

Syntax:

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

Description:

Set watch dog timeout value

Parameters:

`iTimeout`–an integer(1~128) representing the timeout in seconds

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

4.4.6 WatchDog.getTimeout()

Syntax:

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

Description:

Get watch dog timeout value.

Parameters:

iTimeout– return timeout value(sec.).

Return:

S_OK – if the function succeeded.

E_* – if the function fails.

4.5 Uart Class

4.5.1 Uart Class

Syntax:

```
Uart();
```

Description:

Create a new Uart object.

Example:

Create an Uart object.

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

4.5.2 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 succeeded.

E_UART_OPENFAIL – open device has failed.

E_UART_ALREADY_OPENED – object already has been opened.

E_UART_TTY_BEEN_USED – device has been used by other object.

E_* – if the function fails.

4.5.3 Uart.close()

Syntax:

```
int close();
```

Description:

Close the Uart device that is currently opened.

Return:

S_OK – if the function succeeded.

E_* – if the function fails.

4.5.4 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

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

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

4.5.5 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

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

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

}

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

Example:

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

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

4.5.6 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 succeeded.

`E_*` – if the function fails.

4.5.7 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 succeeded.

`E_*` – if the function fails.

Example:

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

if(SmartETK.S_OK != m_uart.getTimeout(T))
{
        cleanStatus();
        return;
}
```

4.5.8 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 succeeded.

`E_*` – if the function fails.

4.5.9 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 succeeded.

`E_*` – if the function fails.

Example:

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

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

4.5.10Uart.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 succeeded

`E_*` – if the function fails.

4.5.11Uart.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 succeeded, return the actual number of bytes received.

`<0(E_*)` – if the function fails.

4.5.12Uart.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 succeeded.

E_* – if the function fails.

4.5.13Uart.reset()

Syntax:

```
int reset();
```

Description:

reset the opened or open failed Uart device.

If the uart device has 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 succeeded.

E_* – if the function fails.

4.6 Can Class

4.6.1 Can Class

Syntax:

```
Can();
```

Description:

Create a new Can object.

Example:

Create a Can object.

```
Can m_can = new Can();
```

4.6.2 Can.open()

Syntax:

```
int open(String sName);
```

Description:

Open the specified Can device.

Parameters:

`sName` – Can device name. (Ex. can0, can1, ...)

Return:

S_OK – if the function succeeded.

E_CAN_OPENFAIL – open device has failed.

E_CAN_ALREADY_OPENED – object already had been opened.

E_* – if the function fails.

4.6.3 Can.close()

Syntax:

```
int close();
```

Description:

Close the Can device that is currently opened.

Return:

S_OK – if the function succeeded.

E_* – if the function fails.

4.6.4 Can.setBitrate()

Syntax:

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

Description:

Set the bitrate of the opened Can device.

Parameters:

`iBitrate` – bit rate (Ex. 125000)(default : 500000)

Return:

`S_OK` – if the function succeeded.

`E_CAN_BAUDRATE_NOT_SUPPORT`– bitrate is not supported.

`E_*` – if the function fails.

4.6.5 Can.getBitrate()

Syntax:

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

Description:

Get the bitrate of the opened Can device.

Parameters:

`iBitrate` – return bit rate (Ex. 125000)(default : 500000)

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

4.6.6 Can.setTimeout()

Syntax:

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

Description:

Set the timeout of the opened Can 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 succeeded.

`E_*` – if the function fails.

4.6.7 Can.getTimeout()

Syntax:

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

Description:

Get the timeout configuration of the opened Can 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 succeeded.
`E_*` – if the function fails.

Example:

Import com.viaembedded.smartetk.SmartETK.Timeout;

```
Can m_can = new Can();
Timeout timeout = new Timeout();

if(SmartETK.S_OK != m_can.getTimeout(timeout))
{
        cleanStatus();
        return;
}
```

4.6.8 Can.setLoopback()

Syntax:

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

Description:

The loopback functionality is enabled by default to reflect standard networking behavior for CAN applications. A local loopback functionality is similar to the local echo e.g. of tty devices.

bEnable = true (if setRecvOwnMsgs() also set to true, it will receive its own msgs after transmit)

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

Parameters:

`bEnable` – true or false to set.

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

4.6.9 Can.getLoopback()

Syntax:

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

Description:

Get loopback state.

Parameters:

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

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

Example:

```
boolean[] bEnable_getlbk = null;

if(SmartETK.S_OK != m_uart.getLoopback(bEnable_getlbk))
{
        cleanStatus();
        return;
}
```

4.6.10Can.setRecvOwnMsgs()

Syntax:

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

Description:

Set CAN_RAW_RECV_OWN_MSGS flag to decide whether the socket receives frames its own sent or not. As the local loopback is enabled, the reception of the CAN frames on the same socket that was sending the CAN frame is assumed to be unwanted and therefore disabled by default

bEnable = true (if setLoopback() set to false, it won't receive its own msgs after sending Can frame)

bEnable = false (default)

Parameters:

`bEnable` – true or false to set.

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

4.6.11Can.getRecvOwnMsgs()

Syntax:

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

Description:

Get the state of receiving its own sent frames or not.

Parameters:

`bEnable`– get true/false for enable/disable.

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

Example:

```
boolean[] bEnable_recvOwn = null;

if(SmartETK.S_OK != m_uart.getRecvOwnMsgs(bEnable_recvOwn))
{
        cleanStatus();
        return;
}
```

4.6.12Can.setFilter()

Syntax:

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

Description:

The reception of CAN frames can be controlled by defining 0 .. n filters with the CanFilter object array buffer. A filter matches, when [received_can_id] & CanFilter.iCanMask == CanFilter.iCanID & CanFilter.iCanMask

To disable the reception of CAN frames: setFilter(null, 0);

Parameters:

`canFilter`– CanFilter object array to set.

class CanFilter

{

`static final int PAYLOAD_SIZE =8` – Payload data size

`static final int CAN_INV_FILTER = 0x20000000` – The filter can be inverted (CAN_INV_FILTER bit is set in can_id)

`int iCanID` – Can ID

`int iCanMask` – Valid bits in CAN ID for frame formats

}

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

Return:

`S_OK` – if the function succeeded.

`E_*` – if the function fails.

4.6.13Can.readFrame()

Syntax:

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

Description:

Reading CanFrame from the opened Can device.

Parameters:

`canFrame`– CanFrame object to read.

class CanFrame

{

`static final int PAYLOAD_SIZE =16` – Payload data size

`int iCanID` – 32 bit CAN_ID + EFF/RTR flags

`final byte[] byData = new byte[8]` – Frame payload data. The object had been created by byte[8] array buffer. Users can modify data byte array, but cannot modify the object.

}

Return:

S_OK – if the function succeeded.

E_* – if the function fails.

4.6.14Can.writeFrame()

Syntax:

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

Description:

Writing CanFrame to the opened Can device.

Parameters:

canFrame– CanFrame object to write.

Return:

S_OK – if the function succeeded.

E_* – if the function fails.

Annex A: Internet Permission

Smart ETK is programmed with the socket IO as the communication between JAVA and C language to control the hardware modules, therefore you need to make sure that you have *android.permission.INTERNET* in *AndroidManifest.xml:*

<uses-permission android:name="*android.permission.INTERNET*"/>

Annex B: GPIO

B1. GPIO: for user's own design

GPIO1, GPIO2, GPIO4, GPIO5, GPIO7, GPIO8, GPIO9 and GPIO203 are the external GPIO pin for user's own design.

An example of setting GPIO1 as input pin and getting its value is shown here.

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(1);                    // Create GPIO1 object

gpio5.setEnable(true);                       // Enable GPIO1
gpio5.setDirection(GPIO.GM_GPI);             // Set GPIO1 as input
direction

gpio5.getEnable(bEnable);                    // Get GPIO1's enable
status
gpio5.getDirection(nDirection);          // Get GPIO1's input/output
direction
gpio5.getValue(nValue);                      // Get GPIO1's input value
```

An example of setting GPIO5 as output pin and changing its value is shown here.

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(5);                    // Create GPIO5 object

gpio6.setEnable(true);                       // Enable GPIO5
gpio6.setDirection(GPIO.GM_GPO);             // Set GPIO5 as output
direction
gpio6.setValue(1);                           // Set GPIO5's output to
high

gpio6.getEnable(bEnable);                    // Get GPIO5's enable
status
gpio6.getDirection(nDirection);          // Get GPIO5's input/output
direction
gpio6.getValue(nValue);                  // Get GPIO5's output value

<Note>: Create GPIO203 by following method
 GPIO gpio203 = new GPIO(16);
```

Annex C: Watch Dog

C1. 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;
```

C2. Get Watch Dog Status

The following is the sample code:

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

boolean[] bGetEnable = new boolean[1];

if(SmartETK.S_OK != m_watchdog.getEnable(bGetEnable))
{
        return false;
}

return bGetEnable[0];
```

C4. Keep Feeding Watch Dog

The following is the sample code:

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

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

Annex D: UART

D1. 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;
}
```

D2. UART Write Data

The following is the sample code:

Notice that "mETWrite" is the EditView to store writing texts.

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

D3. 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;
}
```

Annex E: CAN

E1. CAN Initialize Communication

The following is the sample code:

```
private Can m_can0 = null;
EditText mETDev0;
EditText mETBitRate0;

if(null == m_can0)
{
        m_can0=new Can();
}

if(SmartETK.S_OK!=m_can0.open((mETDev0.getText().toString())))
{
        cleanStatus();
        return;
}

if(SmartETK.S_OK!=m_can0.setBitrate(Integer.valueOf(mETBitRate0.getText().t
oString())))
{
        cleanStatus();
        return;
}
```

E2. CAN Write Frame

The following is the sample code:

```
Notice that "mETWrite0" is the EditText to store writing texts.
byte[] byWriteData = mETWrite0.getText().toString().getBytes();
 CanFrame can0WriteFrame = new CanFrame();

can0WriteFrame.byDataLen = (byte) byWriteData.length;
System.arraycopy(byWriteData, 0, can0WriteFrame.byData, 0,
can0WriteFrame.byDataLen);

if(SmartETK.S_OK!=m_can0.writeFrame(can0WriteFrame))
{
        return;
}
```

E3. CAN Read Frame

The following is the sample code:

```
int iActualLen = 0;
byte[] zero_byRead = new byte[8];
Arrays.fill(zero_byRead, (byte) 0x0);
CanFrame can0ReadFrame = new CanFrame();

if(SmartETK.S_OK == m_mainThreadCan0.readFrame(can0ReadFrame))
{
        if(can0ReadFrame.byData!=zero_byRead){ /* there is data from Can */
                iActualLen=8;

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

```
        m_mainThreadHandler0.post(new
        StrRunnable(can0ReadFrame.byData,iActualLen));
            iActualLen = 0;
}
```

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

USA
940 Mission Court
Fremont, CA 94539
USA

TEL: 1.510.683.3300
FAX: 1.510.687.4654
Email: 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

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

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

Korea
2F, Sangjin Bldg., 417
Dogok Dong, Gangnam-Gu
Seoul 135-854
South Korea

TEL: 82.2.571.2986
FAX: 82.2.571.2987
Email: embedded@via-korea.com