Android

For

TM1

User Manual

Document Reference: TM1 Android Guide

Document Issue: 1.10

Associated SDK release: 1.10

Associated Image release: BCT-TM1-V1.25

Associated Update Utility release: V1.25

Author: D Robinson

Contents

1. Introduction

Android was originally designed as a mobile operating system for phones and tablets, however many of Androids features are desirable in the context of embedded systems. These include:

- Standard SDK API's for most hardware interfaces
- Free feature rich development tools
- Royalty free software distribution without requirement for disclosing source code
- Large developer base
- Rich native multimedia capabilities
- Standard user interface
- Quick time to market
- Native Java and C++ language support. Other languages supported through third party tools.

Where Android falls short in the embedded space, is lack of support for embedded hardware interfaces like serial ports, i2c buses, and GPIO's. As a rule of thumb, unless hardware is defined in the Compatibility Definition Document (CDD) there is unlikely to be a native Android API available.

Blue Chip Technology have overcome this limitation by making a custom API available to customers, which allows access to nonstandard hardware interfaces.

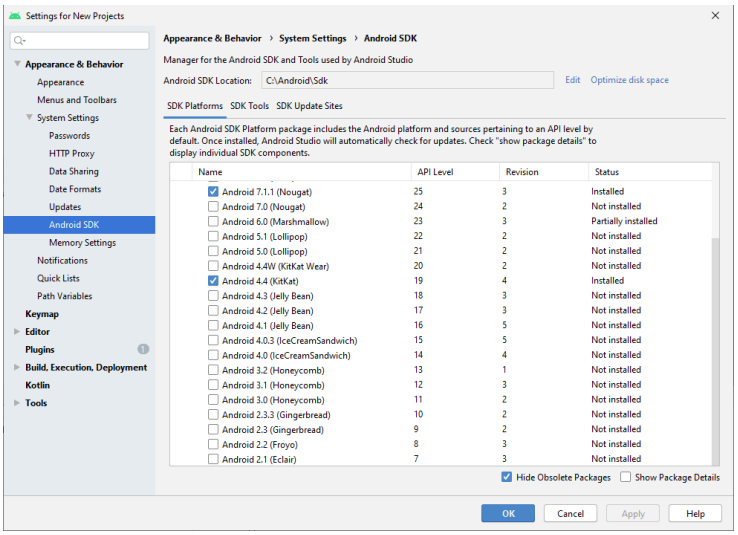
The content of this document provides information required to start building Android applications for the BCT TM1 / HB5 platform. It covers:

- Development environment requirements
- How to Enable debugging on the TM1/HB5 platform
- How to setup a simple hello world application in Android Studio
- How to import the BCTAPI hardware library into Android Studio
- Definitions of the BCTAPI hardware library class structure

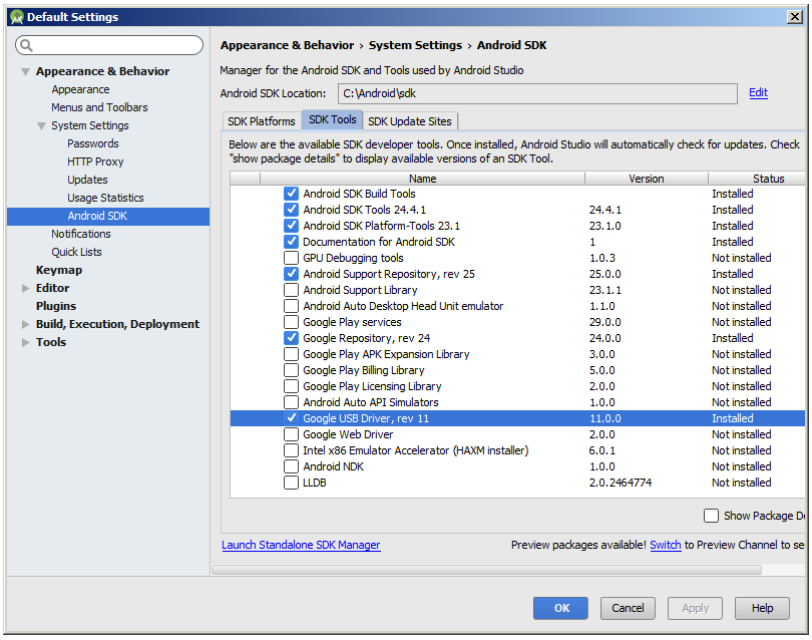
2. Environment

Android applications for TM1/ HB5 can be implemented in either Android Studio or with the older Eclipse ADT plugin. The examples in this document focus on the Android Studio environment.

At the time of writing, TM1 / HB5 supports Android 4.4.3 (Kit Kat) which corresponds to Android API version 19 and Android 7.1 (Nougat) which corresponds to Android API version 25. It is important that at least API 19 is installed in the Android SDK manager as this will allow applications targeting both Kit Kat Nougat to be developed.



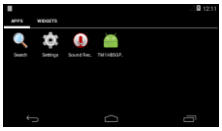
Debugging of Android applications is typically performed over the Android ADB USB interface. To enable this feature within Android Studio the USB debug feature must be installed in the Android SDK manager.



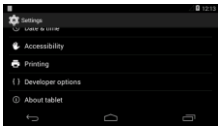
2. Enable USB ADB Debugging on TM1/HB5

By default the USB ADB Debugging interface is turned off. To enable the debug interface follow the below steps. Note in Android 7 the screen layout will be slightly different.

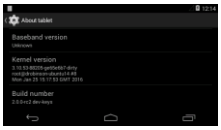
1. Navigate the Android settings control panel.



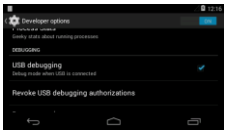
2. Scroll to the bottom of the list and click, “About tablet”



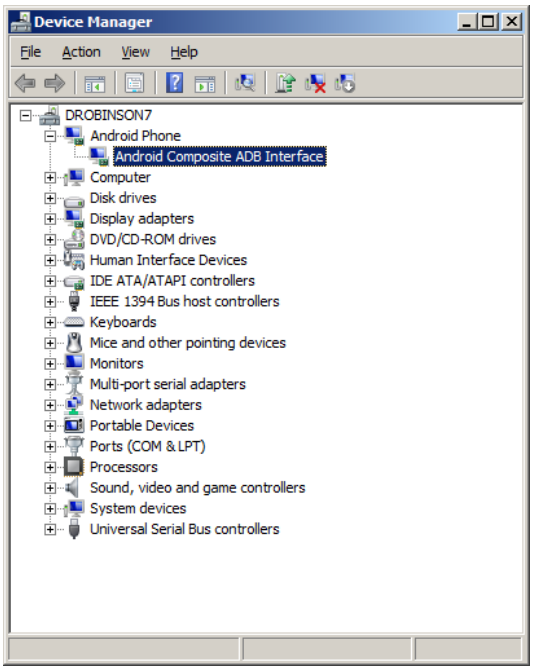
3. Scroll to the bottom of the list and click, “Build Number” repeatedly until a message is displayed saying, “you are now a developer”.



4. Press the back button, and click on, “Developer Options”.
5. Scroll down to the option, “USB Debugging”, and click to enable the feature. You may be prompted to confirm that debugging is allowed.



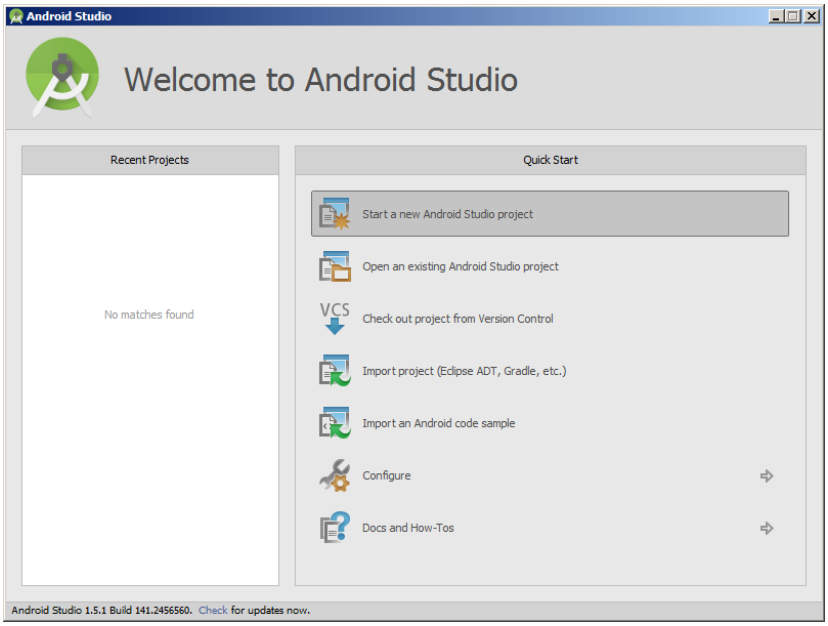
6. Connect a USB cable between the development PC and TM1/HB5.
7. Windows should detect a new ADB USB device and search for drivers. If a driver cannot be found automatically it may be necessary to point Windows device manager at the following location. <Android SDK root>\extras\google\usb_driver.
8. Upon successfully loading the ADB driver, Windows device manager should display an Android ADB device.



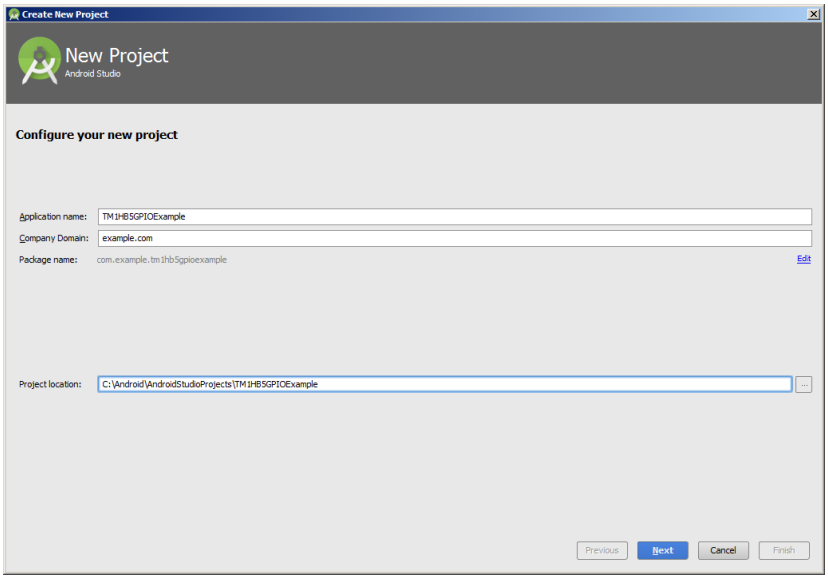
3. Simple GPIO light switch example

The following steps describe how to setup and deploy a basic Android App to TM1/HB5. The app has a simple toggle button that controls a GPIO output. The walkthrough presumes that Android Studio is installed, and that the SDK manager is setup as per the previous section.

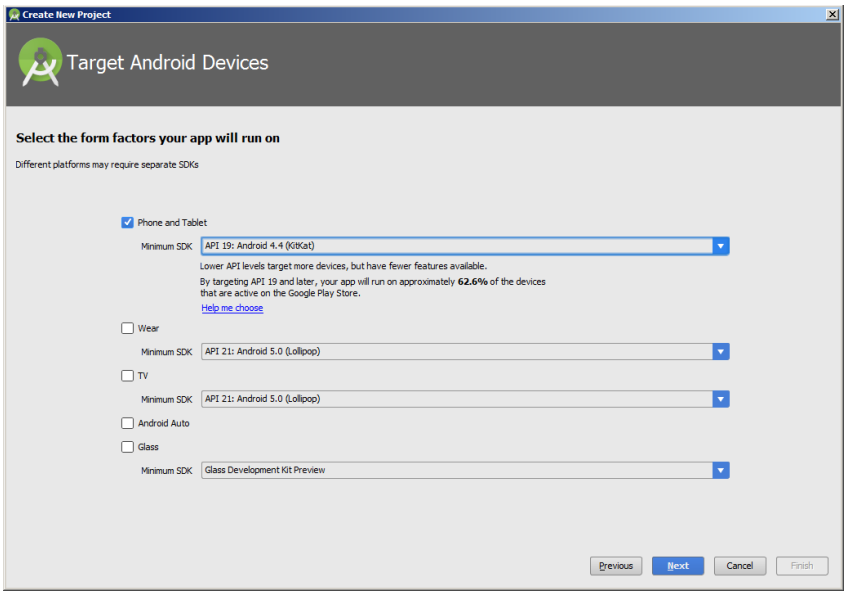
1. Start Android Studio
2. Click “Start a new Android Studio project“, in the “Quick Start” menu.



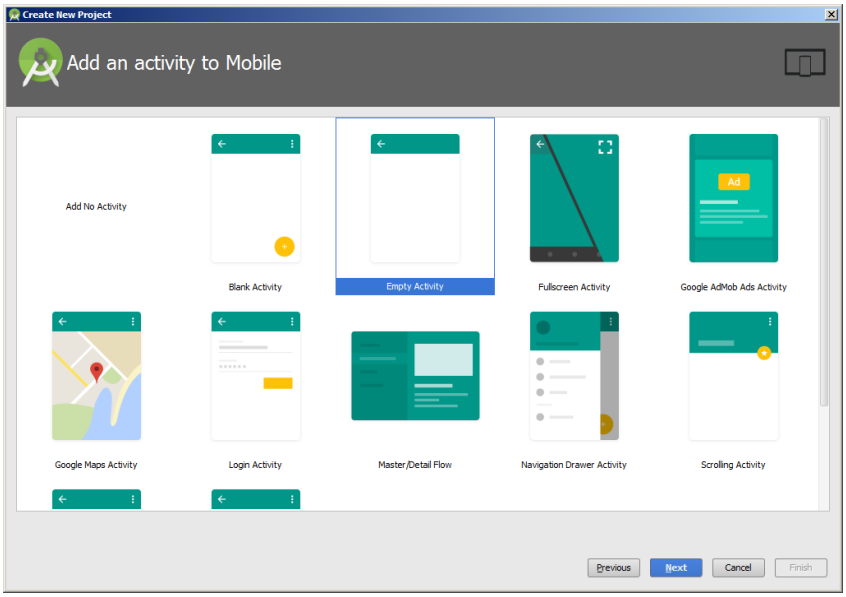
3. Give the project a name and namespace.



4. Tell the wizard that the app is targeting API 19 for a Phone / Tablet device.

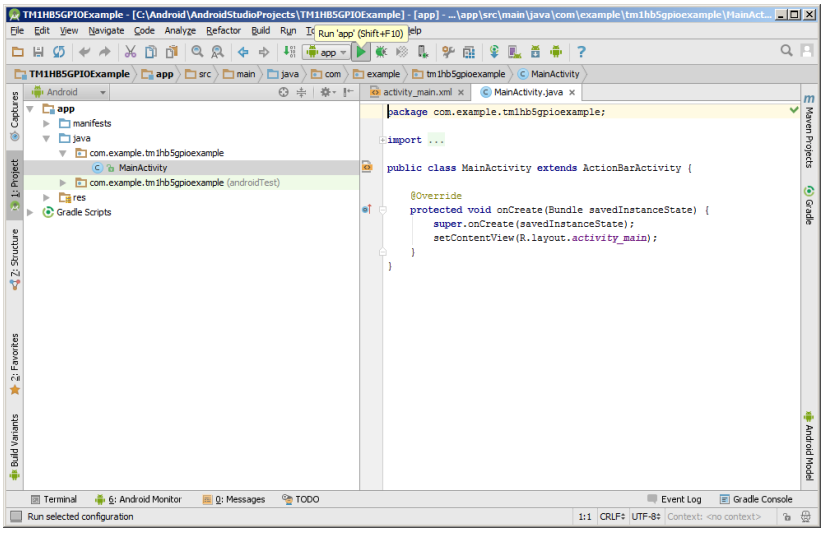


5. Select Empty Activity

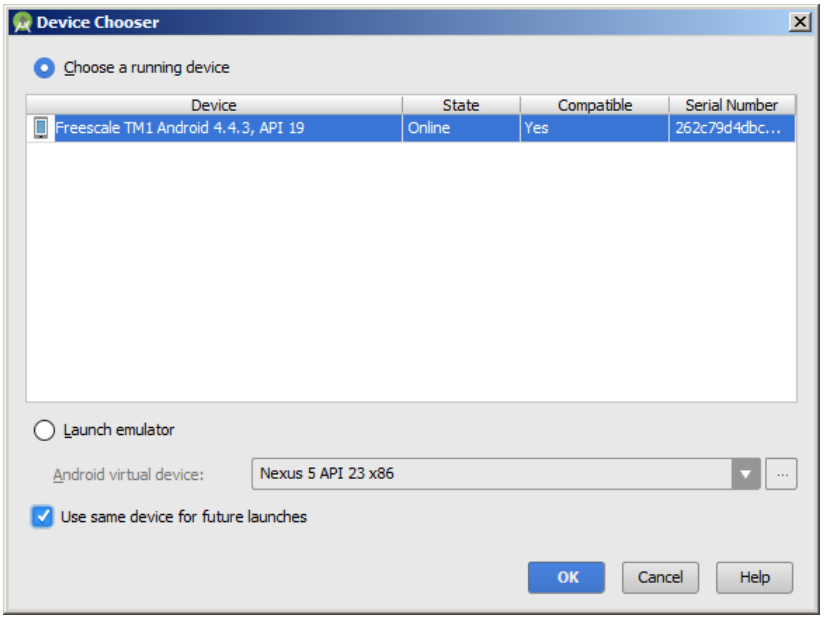


6. Use the default activity name and click finish. Android studio will now initialise the development environment.
7. At this stage it is advisable to build and deploy the app in its default state. Ensure that the TM1/HB5 device has been setup for debug over USB and that the appropriate driver has been installed on the development PC. See previous section for details.

8. Click the “Run App” button to deploy the app

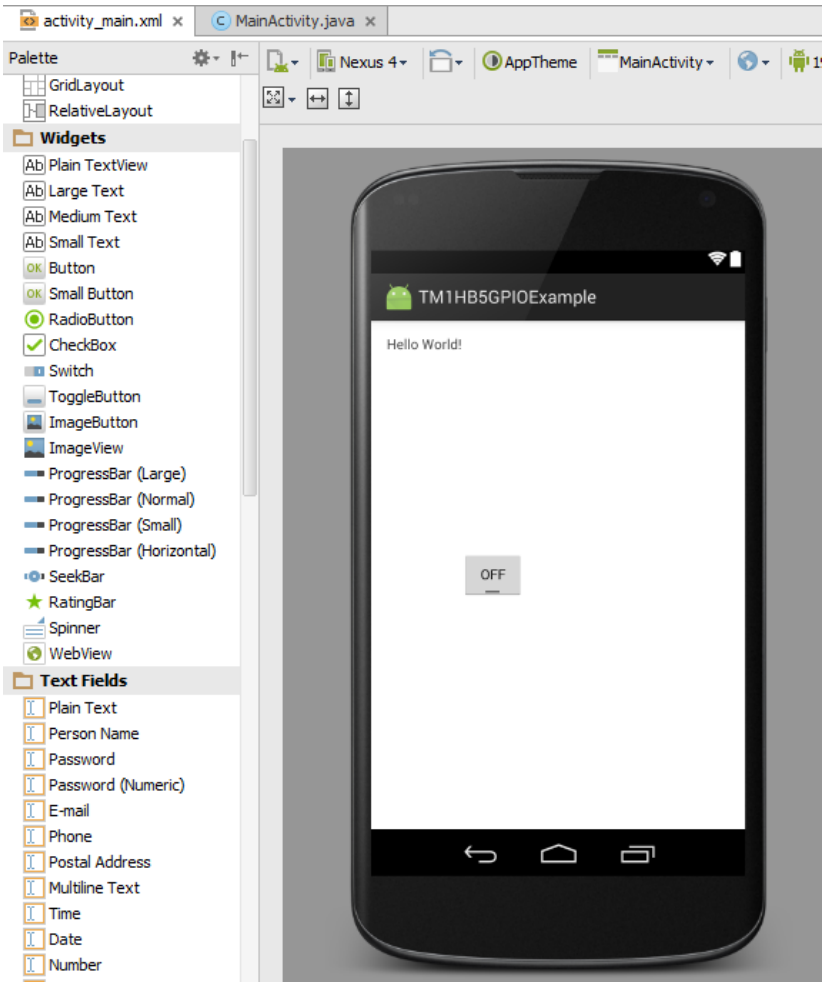


9. Select the TM1/HB5 device in the “Device Chooser” dialogue box and press ok. If the device is displayed as unauthorised, check the TM1 /HB5 display for an authorisation request.

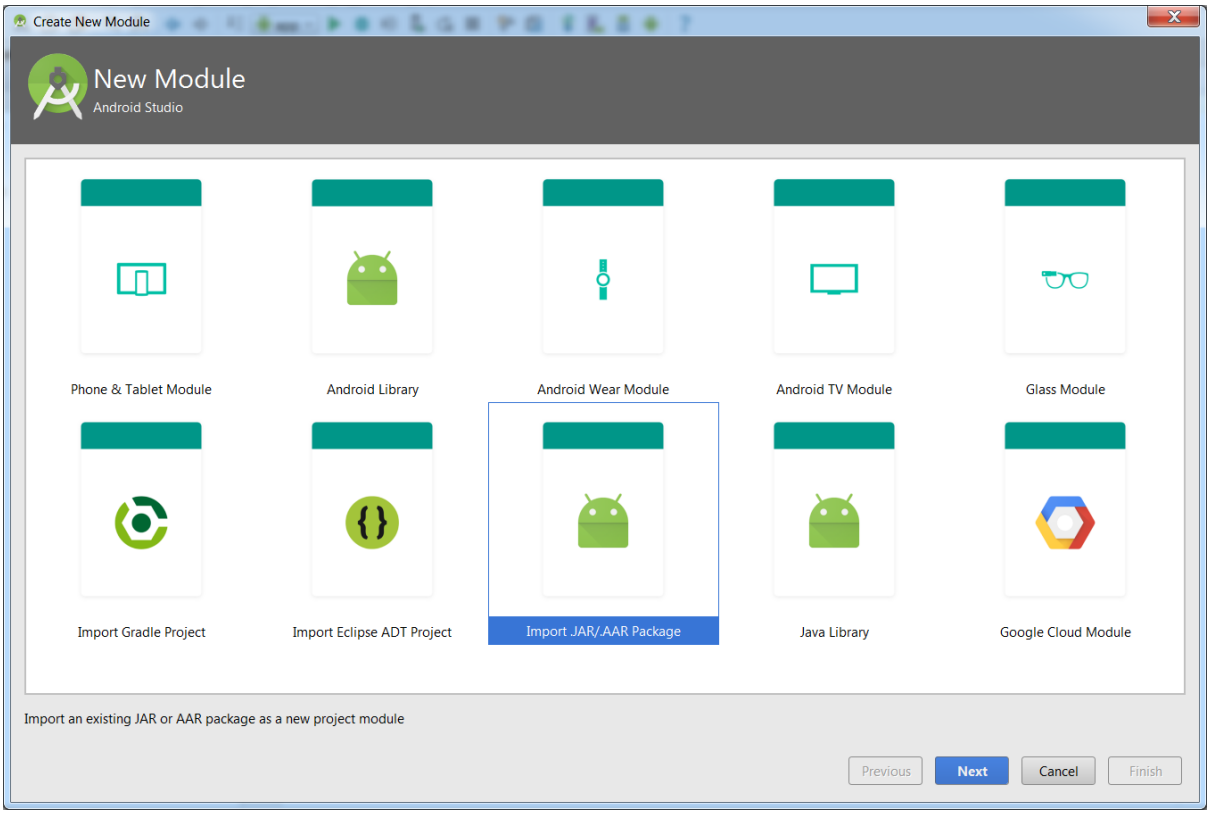


10. The Hello world application should automatically deploy to the device and execute.

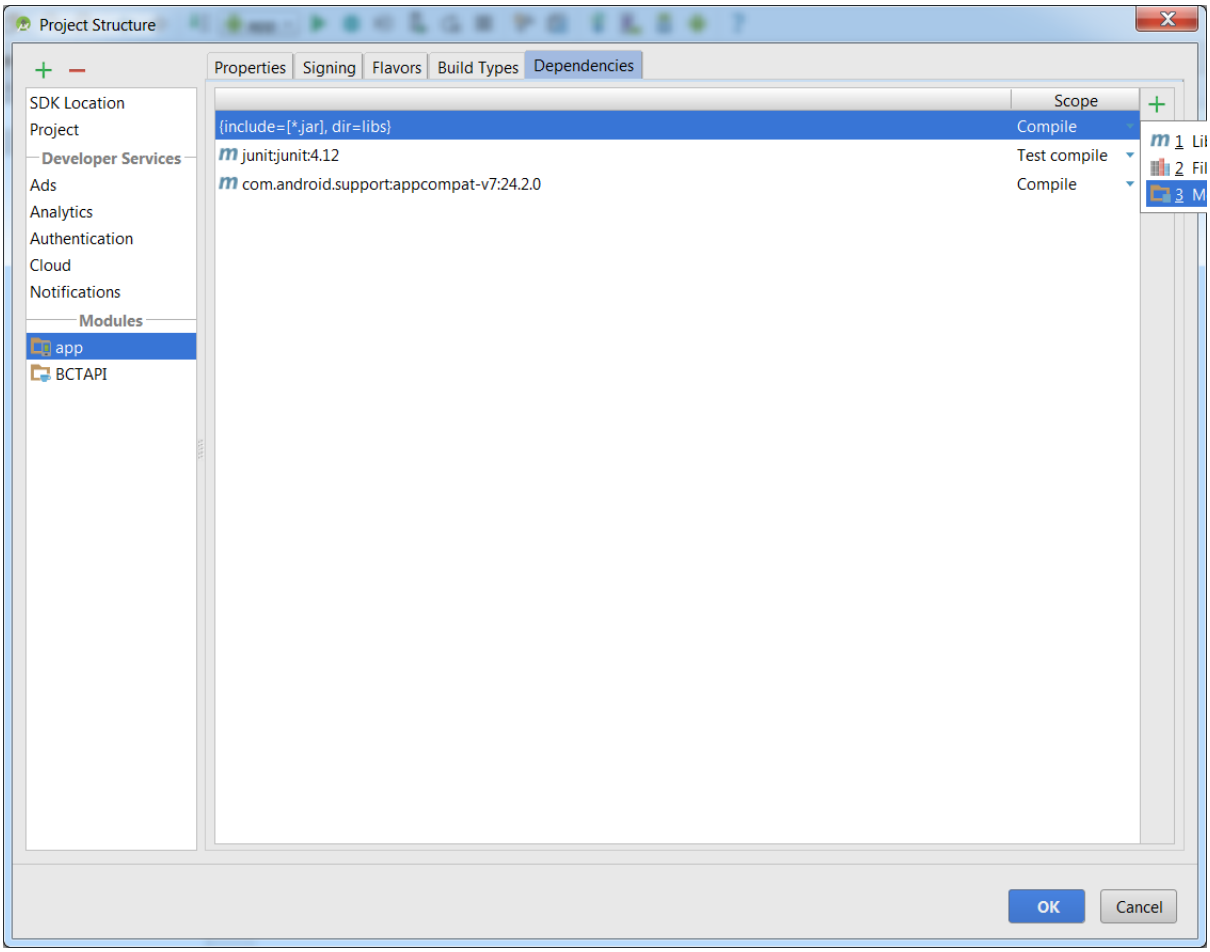
11. Add a toggle button control to the activity_main.xml gui designer.

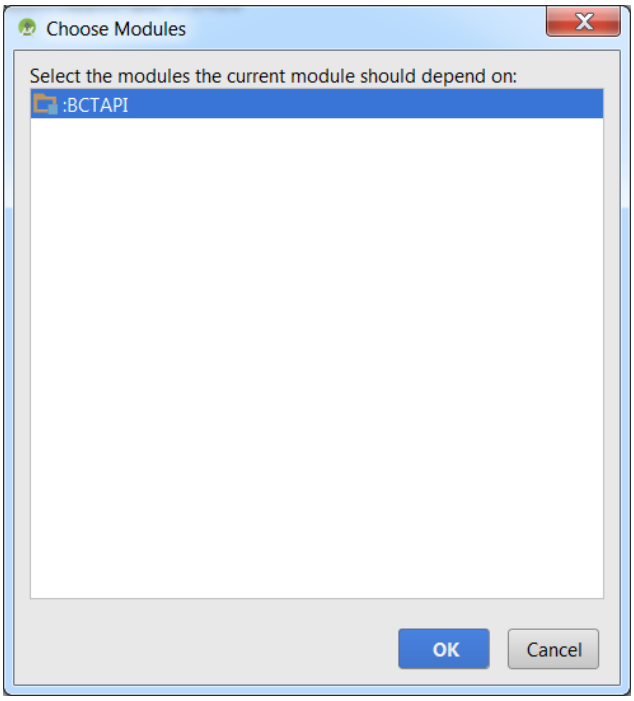


12. Import the BCTAPI.aar library into the Android project.
    a. Obtain the BCTAPI.aar library from Blue Chip Technology
    b. Choose File : New : New Module
    c. Select Import JAR/AAR option (as shown below)



13. Add the BCTAPI.aar library as a dependency.
    a. Choose File : Project Structure
    b. Select the Dependencies tab for the app and choose add Module dependency (as shown below)
    c. Choose BCTAPI from the list of modules presented (as shown below)


Project Structure
Properties
Signing
Flavors
Build Types
Dependencies
SDK Location
Project
Developer Services
Ads
Analytics
Authentication
Cloud
Notifications
Modules
app
BCTAPI
Scope
{include=[*.jar], dir=libs}
Compile
junit:junit:4.12
Test compile
com.android.support:appcompat-v7:24.2.0
Compile
OK
Cancel


Choose Modules
Select the modules the current module should depend on:
:BCTAPI
OK
Cancel

14. Modify the MainActivity.java source code so that it contains the following.

```
package com.example.tm1hb5gpioexample;

import bct.hwapi.*;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.CompoundButton;
import android.widget.ToggleButton;

public class MainActivity extends ActionBarActivity {

    private static final String TAG = "tm1hb5gpioexample";
    GPIO gpio = null;
    ToggleButton toggle = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        try {
            gpio = new GPIO(GPIODefinitions.GPIO_USER_LED); //Create GPIO object for user gpio on P12
            gpio.InitialiseGPIO(GPIO.GPIODirection.OUTPUT); //Set GPIO as an output
        }
        catch(Exception ex)
        {
            Log.e(TAG, "Failed to initialise GPIO: " + ex.getMessage());
        }

        toggle = (ToggleButton) findViewById(R.id.toggleButton);
        toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
            {
                try {
                    if (isChecked) {
                        gpio.SetOutput(1);
                    } else {
                        gpio.SetOutput(0);
                    }
                }
                catch(Exception ex)
                {
                    Log.e(TAG, "Failed to set GPIO: " + ex.getMessage());
                }
            }
        });

    }
}
```

15. Run the app in the same way as step 8. The LED on p12 (Ethernet Connector) will indicate the state of the toggle button.

Install APK on TM1

To install a compiled APK on the TM1 outside of Android Studio do the following:

On PC:

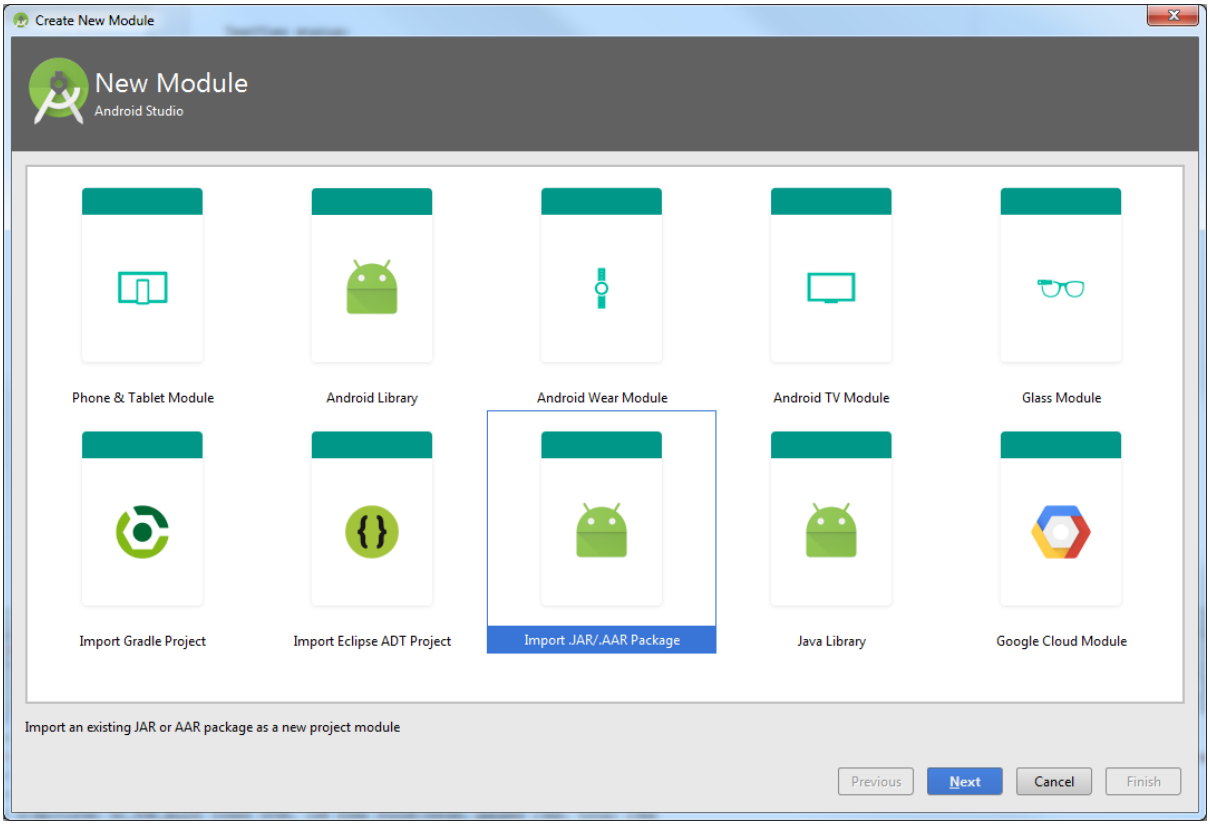
1. Connect the TM1 to a PC using the USB device port on the TM1.
2. Wait for the TM1 to appear as a drive on the PC
3. Navigate to the folder: ‘Computer\tm1hb5\Internal shared storage\Download’
4. Copy the APK into the Download folder

On TM1:

1. Navigate to the folder: ‘Setting->Storage->Explore->Download’
2. Tap the APK file to install and follow on screen instructions.

4. BCTAPI

The BCT API for Android is distributed in the form of a java AAR file with filename BCTAPI.aar. The library can be imported into an Android Studio project by using the New Module wizard. E.g.



The BCT API contains class definitions which allow Android apps to control serial ports, i2c ports, GPIO pins, and other hardware bespoke to the TM1/HB5 platform.

BCTAPI Namespace

All BCT API class definitions are implemented in the namespace bct.hwapi. By including the following import definition in an Android Studio source file, all defined namespaces will be available to the developer.

```
import bct.hwapi.*;
```

SerialPort Class

Class Namespace:

```
bct.hwapi.SerialPort
```

Constructor:

```
SerialPort(File serialportfile, int baudrate, int wordlendth, int stopbits, int
parity, boolean enablers485, boolean enablers485localloopback) throws
SecurityException, IOException
```

Description:

Opens a serial port with the specified parameters.

Parameters:

serialportfile – The filename of the serial port to open. TM1 / HB5 supports two serial ports by default these are:

/dev/ttymxc1 - RS232 levels on P4

/dev/ttymxc2 - RS232 or RS422/485 levels on P4.

baudrate – The baud rate that the serial port will operate at.

wordlength – The length of each word transmitted or received. Valid values are 5, 6, 7, and 8.

stopbits – The number of stop bits included with each word. Valid values are 1 and 2.

parity - 0 = no parity, 1 = even parity, 2 = odd parity

enablers485 - Enable automatic transmit control for RS485 operation

enablers485localloopback - receive transmitted data. Only valid when enablers485 is true.

SerialPort.Close()

Definition:

```
void Close();
```

Description:

Closes an open serial port.

SerialPort.WriteString

Definition:

```
void WriteString(String text) throws IOException
```

Description:

Write a string of characters in UTF-8 format to the serial port synchronously.

Parameters:

text – String of characters to transmit.

SerialPort.ReadString

Definition:

```
String ReadString() throws IOException
```

Description:

Read a string of characters in UTF-8 format from the serial port synchronously.

SerialPort.getInputStream

Definition:

```
InputStream getInputStream()
```

Description:

Function to retrieve the InputStream of an open serial port. This is useful if byte level access to a serial port is required. The return value will be null is the serial port failed to open.

SerialPort.getOutputStream

Definition:

```
OutputStream getOutputStream()
```

Description:

Function to retrieve the OutputStream of an open serial port. This is useful if byte level access to a serial port is required. The return value will be null is the serial port failed to open.

I2C Class

Class Namespace:

```
bct.hwapi.I2C
```

Constructor:

```
I2C(File device) throws SecurityException, IOException
```

Description:

Opens an I2C port with the specified parameters.

Parameters:

device – The filename of the I2C port to open. TM1 / HB5 supports two I2C ports by default these are:

/dev/i2c-0 – 1.8V bus on TM1

/dev/i2c-1 - 3.06V bus on HB5.

I2C.Close()

Definition:

```
void Close();
```

Description:

Closes an open I2C port.

I2C.ReadByte

Definition:

```
byte ReadByte(byte slaveaddress, byte offset) throws IOException
```

Description:

Read a byte of data from an I2C slave device.

Parameters:

slaveaddress – Address of I2C slave

offset – address offset to read data from.

I2C.WriteByte

Definition:

```
void WriteByte(byte slaveaddress, byte offset, byte data) throws IOException
```

Description:

Write a byte of data from to an I2C slave device.

Parameters:

slaveaddress – Address of I2C slave

offset – address offset to write data to.

Data – data to be written to I2C slave

I2C.BufferedRead

Definition:

```
byte[] BufferedRead(byte slaveaddress, byte offset, byte count) throws IOException
```

Description:

Read (n) bytes of data from an I2C device into a byte array.

Parameters:

slaveaddress – Address of I2C slave

offset – address offset to read data from.

count – number of bytes to read

I2C.BufferedRead

Definition:

```
byte[] BufferedRead(byte slaveaddress, byte count) throws IOException
```

Description:

Read (n) bytes of data from an I2C device into a byte array without writing an offset.

Parameters:

slaveaddress – Address of I2C slave

count – number of bytes to read

I2C.BufferedWrite

Definition:

```
void BufferedWrite(byte slaveaddress, byte offset, byte[] buffer) throws
IOException
```

Description:

Write (n) bytes of data to an I2C slave.

Parameters:

slaveaddress – Address of I2C slave

offset – address offset to read data from.

buffer – array of bytes to be written to an I2C slave.

I2C.BufferedWrite

Definition:

```
void BufferedWrite(byte slaveaddress, byte[] buffer) throws IOException
```

Description:

Write (n) bytes of data to an I2C slave without writing an offset.

Parameters:

slaveaddress – Address of I2C slave

offset – address offset to read data from.

buffer – array of bytes to be written to an I2C slave.

GPIO Class

Class Namespace:

```
bct.hwapi.GPIO
```

Constructor:

```
GPIO(GPIODefinitions gpio) throws Exception
```

Description:

Opens a GPIO pin.

Parameters:

gpio – A GPIO pin defined in GPIODefinitions enum. Values include:

GPIO_0 - GPIO_11 – These correspond to GPIO pins on the HB5 P5 GPIO connector.

GPIO_USER_LED – This GPIO controls an amber LED on the HB5 P12 connector (Magjack).

GPIO_422_485_TXEN – Control of the transmit control signal for the RS422 / 485 transeiver.

GPIO.InitialiseGPIO

Definition:

```
void InitialiseGPIO(GPIODirection direction) throws SecurityException, IOException
```

Description:

Initialise a GPIO pin, and set the pins initial direction

Parameters:

direction – Either GPIODirection.INPUT or GPIODirection.OUTPUT. GPIO inputs on HB5 are configured without any pull-up or pull down resistors.

GPIO.ReadInput

Definition:

```
int ReadInput() throws IOException
```

Description:

Read the logical state of a GPIO input. This function will not return the state of a GPIO output pin.

GPIO.SetOutput

Definition:

```
public void SetOutput(int value) throws IOException
```

Description:

Set the output value of a GPIO pin.

Parameters:

value – 0 = low

1 = high

GPIO. SetDirection

Definition:

```
void SetDirection(GPIODirection direction) throws IOException
```

Description:

Controls whether a pin is setup as an input or output.

Parameters:

Direction – Either GPIODirection.OUTPUT or GPIODirection.INPUT

Audio Class

Class Namespace:

```
bct.hwapi.Audio
```

Constructor:

```
Audio() throws Exception
```

Description:

Creates in instance of the audio class

Audio.EnableClassD

Definition:

```
void EnableClassD() throws IOException
```

Description:

Enable the Class D speaker output on HB5

Audio.DisableClassD

Definition:

```
void DisableClassD() throws IOException
```

Description:

Disable the Class D speaker output on HB5

Watchdog API

Class Namespace:

```
bct.hwapi.Watchdog
```

Constructor:

```
Watchdog() throws Exception
```

Description:

Creates an instance of the watchdog class.

Watchdog.EnableWatchdog

Definition:

```
void EnableWatchdog(byte timeout);
```

Description:

Enables the watchdog and sets the timeout period. The watchdog will reset the system if the timeout expires.

Parameters:

timeout – Watchdog timeout period in seconds.

Watchdog.DisableWatchdog

Definition:

```
void DisableWatchdog();
```

Description:

Disables the watchdog.

Watchdog. RefreshWatchdog

Definition:

```
void RefreshWatchdog();
```

Description:

Refreshes the watchdog countdown timer, to prevent a system reset.

Watchdog. SystemResetSource

Definition:

```
SystemResetSource SystemResetSource ();
```

Description:

This function allows software to determine if the last system reset was caused by a watchdog timeout. A return value of SYSTEM_RESET_SOURCE_POR indicates the system has booted up normally. A return value of SYSTEM_RESET_SOURCE_WD indicates that the system has booted as the result of a watchdog timeout.

PWM API – Requires a special build. Contact BCT.

Class Namespace:

```
bct.hwapi.PWM
```

Constructor:

```
PWM(int PWMController, int PWMIndex) throws Exception
```

Description:

Creates an instance of the PWM class.

Parameters:

PWMController – Index of the PWM controller in the system

PWMIndex - Index of the PWM instance

Note: PWM capability is only available on certain processor module, and host board configurations, and requires a custom Android image installing. Please contact your sales representative for details.

PWM. IsPWMEnabled

Definition:

```
boolean IsPWMEnabled();
```

Description:

Returns the current enabled state of the PWM.

PWM. EnablePWM

Definition:

```
void EnablePWM(boolean bEnable)
```

Description:

Allows the PWM to be enabled or disabled. When the PWM is disabled the logic level of the PWM is low.

Parameters:

bEnable – true = enabled, false = disabled.

PWM. SetPWMPeriod

Definition:

```
void SetPWMPeriod(int value)
```

Description:

Allows the PWM period to be modified.

Parameters:

value – Duration of the PWM period in nanoseconds. Values between 1000, and 1000000000 are valid.

PWM. ReadPWMPeriod

Definition:

```
int ReadPWMPeriod();
```

Description:

Allows the current PWM period to be read.

PWM. SetPWMDutyCycle

Definition:

```
void SetPWMDutyCycle (int value)
```

Description:

Allows the PWM duty cycle to be modified.

Parameters:

value – Duration within a PWM period that the PWM signal is logic high. Values between 0 and the PWM period are valid.

PWM. ReadPWMDutyCycle

Definition:

```
int ReadPWMDutyCycle();
```

Description:

Allows the current PWM duty cycle to be read.

CAN Socket API

The following section details the API for using the CAN interfaces which become available by attaching a CB3 module to HB5. The two physical CAN interfaces are called "can0" and "can1".

CAN Bitrate

Each interface is initialised at boot time with a bitrate of 125000. The bitrate for each interface can be changed by issuing a global broadcast to, "bct.netconfigservice.CAN_BITRATE" with extra parameters defined as follows.

CANINTERFACE - The interface to configure (0 or 1)
CANBITRATE - The desired bitrate to setup.

By issuing an ordered broadcast it is possible to receive notification when the command completes along with a result code. A result code of 1 represents command success, and a result code of 0 represents error. In the event of an error, the result data field hold information related to the failure. See the TM1HB5CanSocketSample application for an example of using the CAN interfaces.

CanSocket Class

Class Namespace:

```
bct.hwapi.CanSocket
```

Constructor:

```
CanSocket(Mode mode)throws IOException
```

Description:

Opens a CAN socket in the specified mode.

Parameters:

mode – SocketCAN mode :

CanSocket.Mode.RAW
Cansocket.Mode.BCM

CanSocket.bind

Definition:

void bind(CanInterface canInterface) throws IOException

Description:

Bind the socket to the CAN interface

CanSocket.send

Definition:

void send(CanFrame frame) throws IOException

Description:

Send the supplied CAN Frame

CanSocket.recv

Definition:

CanFrame recv() throws IOException

Description:

Block for a received CAN frame on the socket

CanSocket.close

Definition:

void close() throws IOException

Description:

Close the socket

CanSocket.getMtu

Definition:

int getMtu(final String canif) throws IOException

Description:

Get the MTU for the named CAN interface

Parameters:

canif - the physical name of the CAN interface (E.g. "can0")

CanSocket.setLoopbackMode

Definition:

void setLoopbackMode(final boolean on) throws IOException

Description:

Set CAN loopback mode

Parameters:

true - on
false - off

CanSocket.getLoopbackMode

Definition:

boolean getLoopbackMode() throws IOException

Description:

Query CAN loopback mode

Return:

true - on
false - off

CanSocket.setRecvOwnMsgsMode

Definition:

void s setRecvOwnMsgsMode (final boolean on) throws IOException

Description:

Set CAN receive own messages mode

Parameters:

true - on
false - off

CanSocket.getRecvOwnMsgsMode

Definition:

boolean getRecvOwnMsgsMode () throws IOException

Description:

Query CAN receive own messages mode

Return:

true - on
false – off

CanSocket.setFilter

Definition:

void setFilter (CanFilter[] filters) throws IOException

Description:

Sets the filters for the socket

Return:

None

CanSocket.setErrFilter

Definition:

void seErrFilter (int mask) throws IOException

Description:

Sets the error filter mask for the socket

Return:

None

CanSocket.CanId Class

Class Namespace:

`bct.hwapi.CanSocket.CanId`

Constructor:

```
CanSocket.CanId(final int address)throws IOException
```

Description:

Create a CAN Id.

Parameters:

address – Identity for a CAN frame:

CanSocket.CanId.setEFFSFF

Definition:

CanId setEFFSFF() () throws IOException

Description:

Set data frame format in CanId

Return:

Resulting CanId

CanSocket.CanId.setRTR

Definition:

CanId setRTR() () throws IOException

Description:

Set remote transmission request in CanId

Return:

Resulting CanId

CanSocket.CanId.setERR

Definition:

CanId setERR() () throws IOException

Description:

Set error flag in CanId

Return:

Resulting CanId

CanSocket.CanId.isSetEFFSFF

Definition:

boolean isSetEFFSFF() () throws IOException

Description:

Test if data frame

Return:

True if data frame, false otherwise

CanSocket.CanId.isSetRTR

Definition:

boolean isSetRTR() () throws IOException

Description:

Test if RTR set

Return:

True if RTR set, false otherwise

CanSocket.CanId.isSetERR

Definition:

boolean isSetERR() () throws IOException

Description:

Test if error flag is set

Return:

True if error flag set, false otherwise

CanSocket.CanId.clearEFFSFF

Definition:

CanId clearEFFSFF() () throws IOException

Description:

Clear data frame flag in CanId

Return:

Resulting CanId

CanSocket.CanId.clearRTR

Definition:

CanId clearRTR() () throws IOException

Description:

Clear remote transmission request in CanId

Return:

Resulting CanId

CanSocket.CanId.clearERR

Definition:

CanId clearERR() () throws IOException

Description:

Clear error flag in CanId

Return:

Resulting CanId

CanSocket.CanId.getCanId_SFF

Definition:

int getCanId_SFF () () throws IOException

Description:

Get raw address from CanId

Return:

Raw address

CanSocket.CanId.getCanId_EFF

Definition:

int getCanId_EFF () () throws IOException

Description:

Get raw address from CanId

Return:

Raw address

CanSocket.CanId.clearERR

Definition:

CanId clearERR() () throws IOException

Description:

Clear error flag in CanId

Return:

Resulting CanId

CanSocket.CanInterface Class

Class Namespace:

```
bct.hwapi.CanSocket.CanInterface
```

Constructor:

```
CanInterface(final CanSocket socket, final String ifName) throws IOException
```

Description:

Create a CAN interface

Parameters:

socket – CAN socket that will be associated with this interface
ifName – Name of physical interface (E.g. "can0")

CanSocket. CanFrame Class

Class Namespace:

```
bct.hwapi.CanSocket.CanFrame
```

Constructor:

```
CanFrame(final CanInterface canIf, final CanId canId, bytes[] data) throws
IOException
```

Description:

Create a CAN interface

Parameters:

canif – interface over which frame will be sent
canId - address of frame
bytes - frame data (limit is 8 octets)

CanSocket.CanFrame.getCanId()

Definition:

CanId getCanId() () throws IOException

Description:

Get the CAN identity of the frame

Return:

CanId

CanSocket.CanFrame.getData()

Definition:

byte[] getData() () throws IOException

Description:

Get the frame data

Return:

Frame data

CanSocket. CanFilter Class

Class Namespace:

`bct.hwapi.CanSocket.CanFilter`

Constructors:

CanFilter(CanId id)

Description:

Creates a filter to exactly matches the given ID.

CanFilter(CanId id, int mask)

Description:

Creates a filter for id and mask.

Mask Values:

CanSocket.EFF_FLAG
CanSocket.RTR_FLAG
CanSocket.ERR_FLAG

CanSocket.SFF_MASK
CanSocket.ERR_MASK
CanSocket.ERR_TX_TIMEOUT_MASK
CanSocket.ERR_LOSTARB_MASK
CanSocket.ERR_CRTL_MASK
CanSocket.ERR_PROT_MASK
CanSocket.ERR_TRX_MASK
CanSocket.ERR_ACK_MASK
CanSocket.ERR_BUSOFF_MASK
CanSocket.ERR_BUSERROR_MASK
CanSocket.ERR_RESTARTED_MASK

CanSocket. CanFilter.getId

Definition:

CanId getId() ()

Description:

Get the canId for the filter

Return:

CanId

CanSocket. CanFilter.getMask

Definition:

int getMask() ()

Description:

Get the mask for the filter

Return:

Int (see mask values)

CanSocket. CanFilter.isInverted

Definition:

boolean isInverted() ()

Description:

Checks if this filter is inverted

Return:

True if this filter is inverted

CanSocket. CanFilter.isExact

Definition:

boolean isExact() ()

Description:

Checks if this filter is exact

Return:

True if this filter is exact

CanSocket. CanFilter.matchId

Definition:

boolean matchId(CanId id) ()

Description:

Matches this filter against the given CAN ID

Return:

True if the given CAN ID would be accepted by this filter

6. Sample Applications

TM1HB5GPIOExample

This sample demonstrates how to control the GPIO pins on HB5 using the GPIO API. The sample also demonstrates how an Android app can be made to operate in full screen mode, and even take the place of the default desktop to create a more embedded experience.

TM1HB5-AC1-RS485Demo

This sample demonstrates how to use /dev/ttymxc2 in RS232 and RS485 mode using the SerialPort API. The UART is configured in auto RS-485 transmit mode. The application is written using Xamarin and C#.

TM1HB5SerialportSample

This sample demonstrates how to use /dev/ttymxc2 in RS232 and RS485 mode using the SerialPort API. The GPIO API is used for transmit enable control.

TM1HB5CanSocketSample

This sample demonstrates how to use the Socket CAN API to send and receive messages over the CAN bus. Note that can0 and can1 must be physically linked together for this to work as expected.

SetTimeExample

This sample demonstrates how to set the date and time from within and Android application targeting the TM1 platform running image BCT-TM1-V1.11 or later.

AndroidPdfViewer

In Android 4.4 there is no native PDF viewing capability. AndroidPdfView has been downloaded and tested to work on the TM1 platform. It can be embedded into an is distributed under the Apache 2.0 licence.

https://github.com/barteksc/AndroidPdfViewer

There are other third party solutions available for viewing PDF files in Android

ConfigureEthernetSettings

This sample demonstrates how to setup the Ethernet interface with a static IP address.

TM1HB5PWMExample

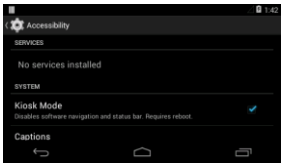
This sample demonstrates how to control a PWM signal. The PWM feature is only available on certain processor module, and host board combinations, and requires a custom Android image.

GuiSample

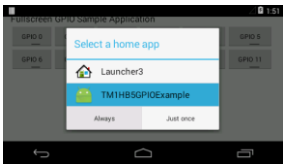
This sample demonstrates an app that can automatically update the boot animation, enable kiosk mode, reboot the unit, and also configure the backlight to fully turn off while the system is fully operational.

7. KIOSK Mode

Blue Chip Technology has made some OEM customisations to Android which allows a developer to lock down the operating system by hiding the system user interface. This can be achieved by navigating to the Settings -> Accessibility page, and checking the “Kiosk Mode” option. After selecting this option wait 5 seconds to allow time for the operating system to flush the setting to disk, and reboot the device. After selecting “Kiosk Mode” and rebooting the device, the software navigation buttons, and status bar will be disabled.



To complete the process of locking down the unit, a customer application must be installed which overrides the android “HOME” intent. See the TM1HB5GPIOExample for an example of how to do this.

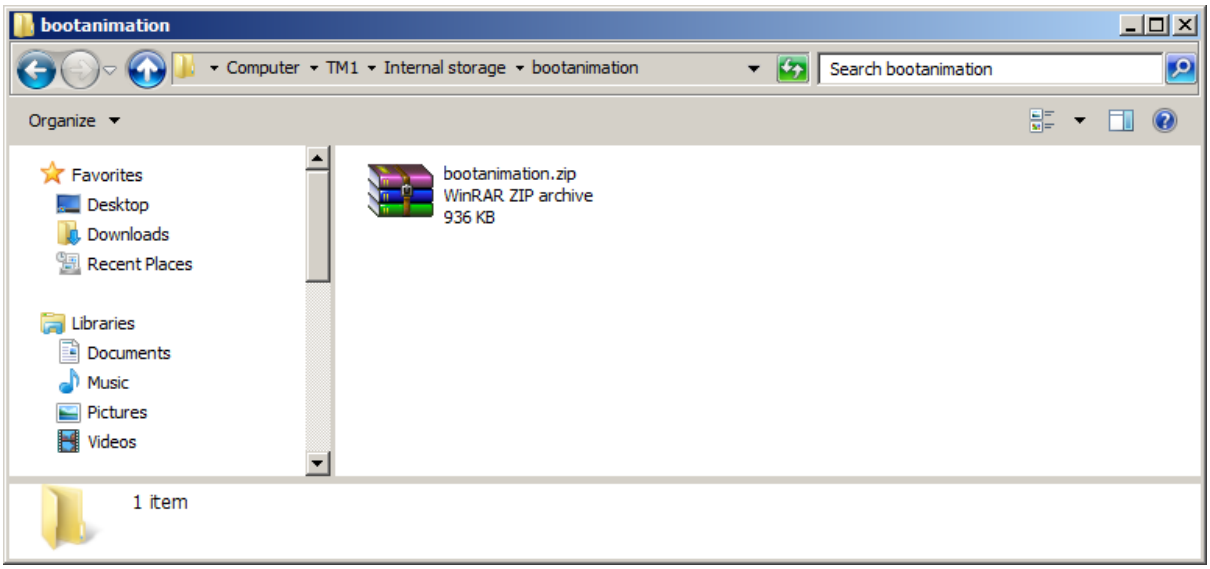


8. Custom Boot Animation

The default Android boot animation can be overridden by copying a custom bootanimation.zip file into the bootanimation directory of the internal storage. There are various sources on the internet that describe the format of bootanimation.zip. E.g.

http://www.addictivetips.com/mobile/how-to-change-customize-create-android-boot-animation-guide/

A sample bootanimation.zip file downloaded from the internet is including in the TM1 Android SDK download.



9. Setting the date/time

By default Android limits setting the date and time to system applications. This typically means using the built-in settings control panel, however when using Android for embedded application development this is often not desirable.

From Build BCT-TM1-V1.11 onwards the permission requirements for setting the system time (android.permission.SET_TIME) have been changed from "signature|system", to "dangerous". This allows an application to set the system time by requesting permission, "android.permission.SET_TIME".

See the SetTimeExample for details on how to set the system date/time from within an android app.

10. BCT.NETCONFIGSERVICE

The configuration of network interfaces from within an Android application is not permitted due to permission constraints. To allow configuration of the Ethernet and CAN interfaces from within an Android app Blue Chip Technology have implemented a system service that performs such configuration on an applications behalf.

The service is designed to receive configuration requests from applications in the form of global broadcasts.

By issuing an, "Ordered Broadcast " it is possible to receive notification when the command completes along with a result code. A result code of 1 represents command success, and a result code of 0 represents error. In the event of an error, the result data field holds information related to the failure. The following broadcast intents are supported.

bct.netconfigservice.UDPATE_ETH0_IP_SETTINGS

Description

Setup the eth0 interface with either a static IP or DHCP.

Intent Extras

DHCP - Set to "dhcp" to enable a DHCP address, or "manual" to enable a static address

IPADDRESS - IP address to setup in static IP mode. Should be in the form "X.X.X.X".

IPMASK - Subnet mask to setup in static IP mode. Should be in the form "X.X.X.X".

IPGATEWAY - Gateway address to setup in static IP mode. Should be in the form "X.X.X.X".

IPDNS - DNS server address to setup in static IP mode. Should be in the form "X.X.X.X".

bct.netconfigservice.ETH0_UP

Description

Enable the eth0 interface.

Intent Extras

NONE

bct.netconfigservice.ETH0_DOWN

Description

Disable the eth0 interface.

Intent Extras

NONE

bct.netconfigservice.CAN_DOWN

Description

Disable the canx interface.

Intent Extra

CANINTERFACE - The CAN interface to disable (0 or 1)

bct.netconfigservice.CAN_UP

Description

Enable the canx interface.

Intent Extra

CANINTERFACE - The CAN interface to enable (0 or 1)

bct.netconfigservice.CAN_BITRATE

Description

Setup the canx interface bitrate.

Intent Extra

CANINTERFACE - The CAN interface to setup (0 or 1)

CANBITRATE - The CAN bitrate to setup.

11. Document History

Issue Level	Issue Date	Author	Amendment Details
1.5		DR	
1.6	01/04/2019	DR	Add Watchdog API Add PWM API Change formatting
1.7			
1.8	22/10/2019		Add CAN filter API Add Nougat Errata Add APK installation instructions

Appendix A: TM1 Android 7.1.2 (Nougat) Known Issues

- Installing APK takes a long time as the system translates DEX file format to OAT file format.
- P2P is not enabled for Wifi.