Android

For

TM1, DB1, TM3

User Manual

Document Reference: -BCT Android SDK Guide

Document Issue: 1.12

Associated SDK release: 1.12

Associated Image releases:

- TM1 Android 4.4.3 - BCT-TM1-V1.23
- TM1 Android 7.1.2 - BCT-TM1-V1.26
- DB1 Android 6.0.1 - BCT-DB1-V1.0
- TM3 Android 9.0 – BCT-TM3-V1.0.1

Associated TM1 Update Utility release: V1.27

Author: D Robinson, M Olejnik

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.

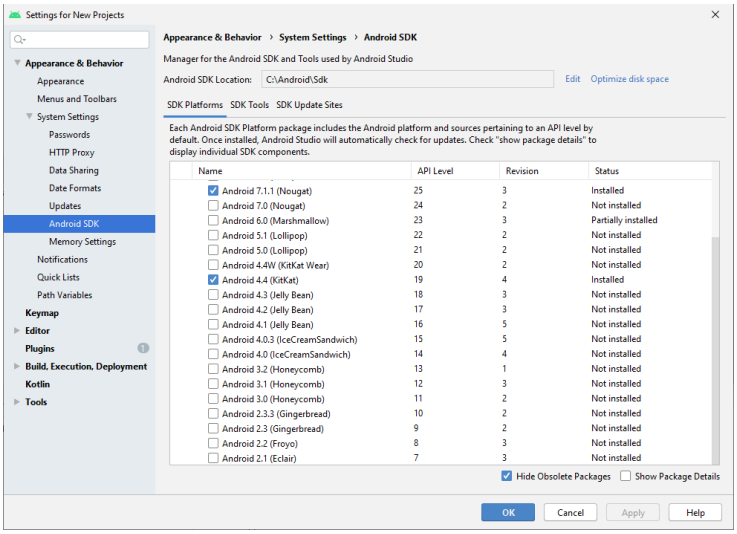
The content of this document provides information required to start building Android applications for the BCT TM1 / HB5 platform. All instructions in this document apply to BCT DB1, BCT TM3 platforms as well as TM1 platform, unless specified otherwise. This document covers:

- 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

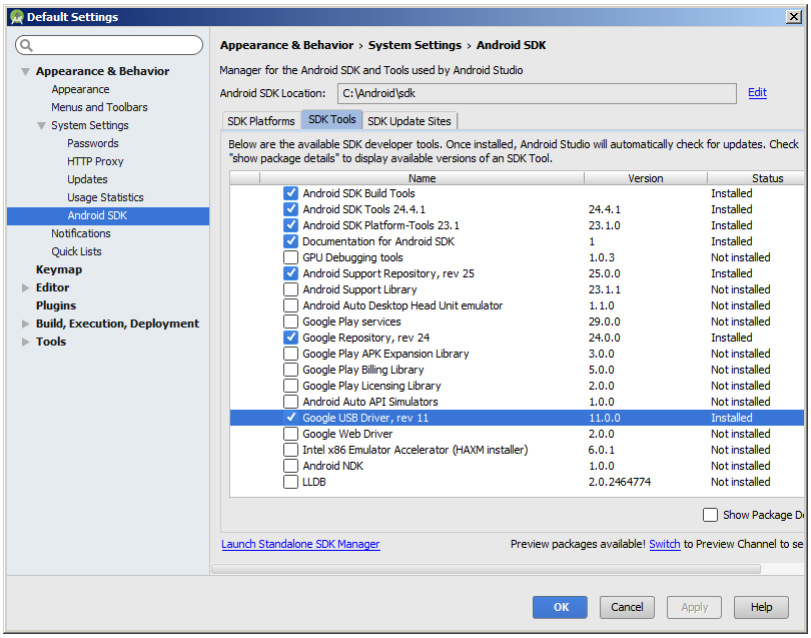
2. Environment

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. DB1 supports Android 6.0.1 (Marshmallow) which corresponds to Android API level 23. TM3 supports Android 9.0 (Pie) which corresponds to Android API level 28. 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. Using API level 19 also simplifies handling of Android permissions.



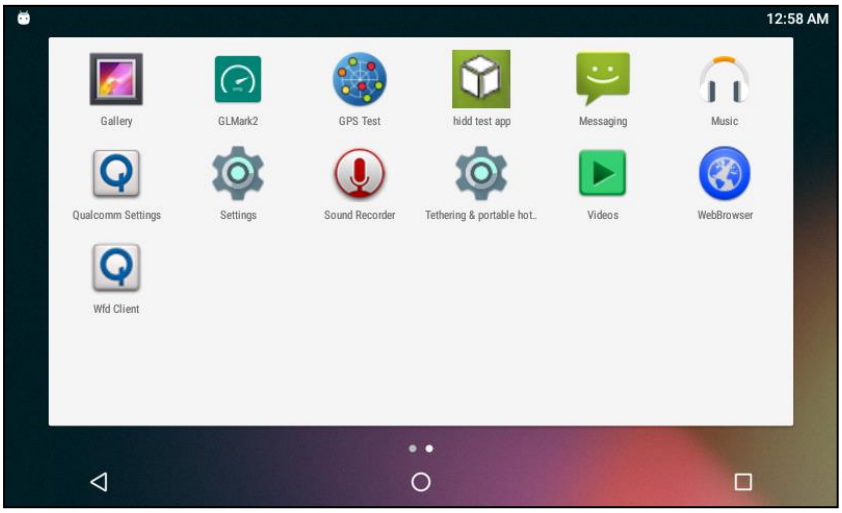
Debugging of Android applications is typically performed over the Android ADB USB interface. To enable this feature within Android Studio the USB debug feature must be installed in the Android SDK manager.



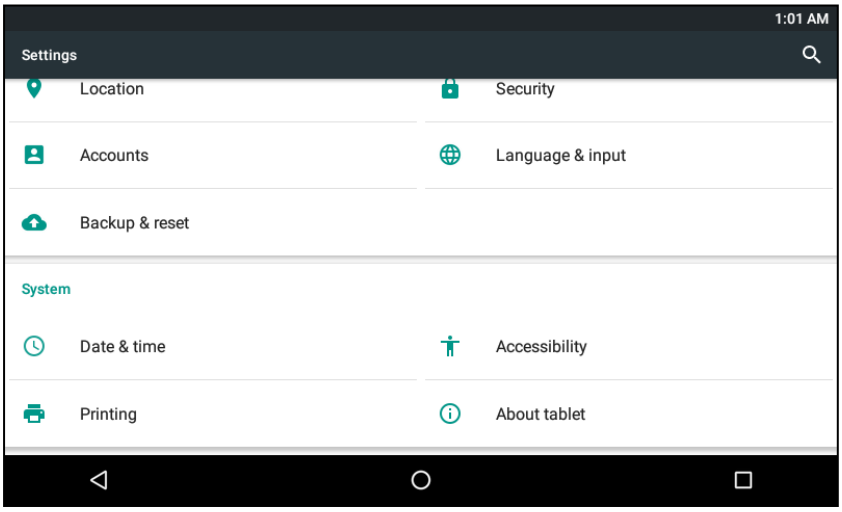
3. Enable USB ADB Debugging on TM1/HB5

By default the USB ADB Debugging interface is turned off. To enable the debug interface follow the below steps. Note in Android 4.4 the screen layout will be slightly different.

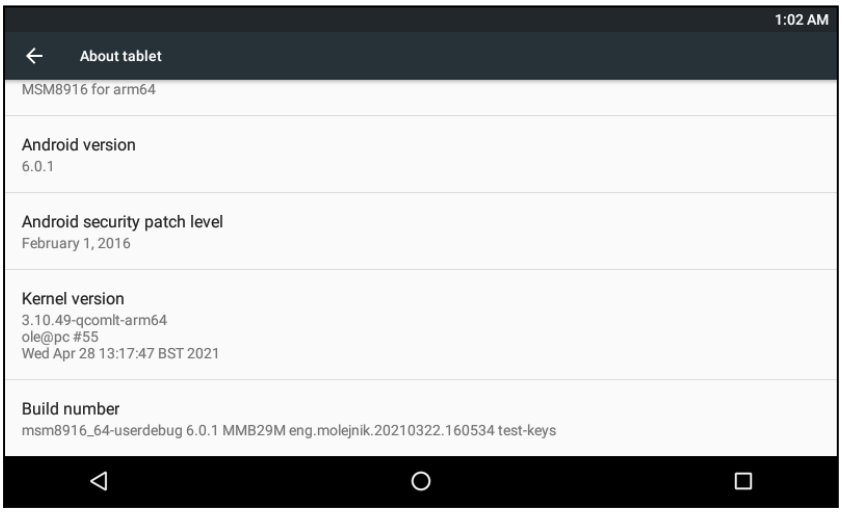
1. Navigate the Android settings control panel.



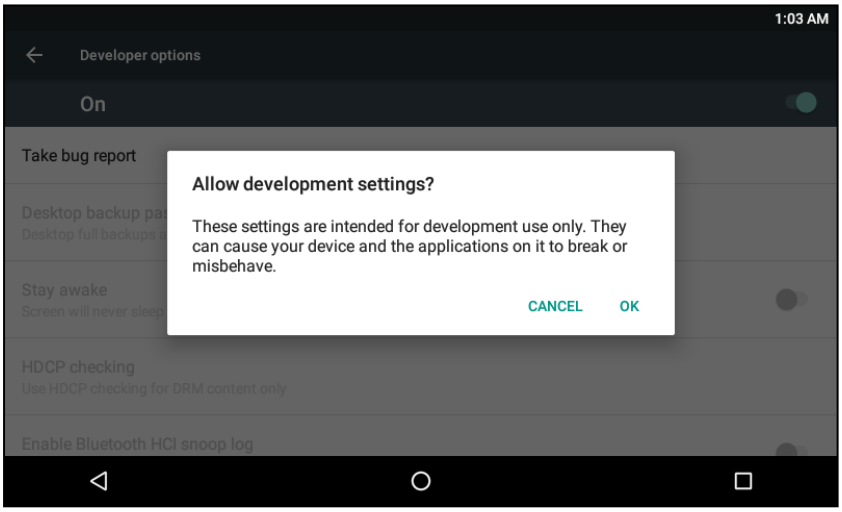
2. Scroll to the bottom of the list and click, “About tablet”



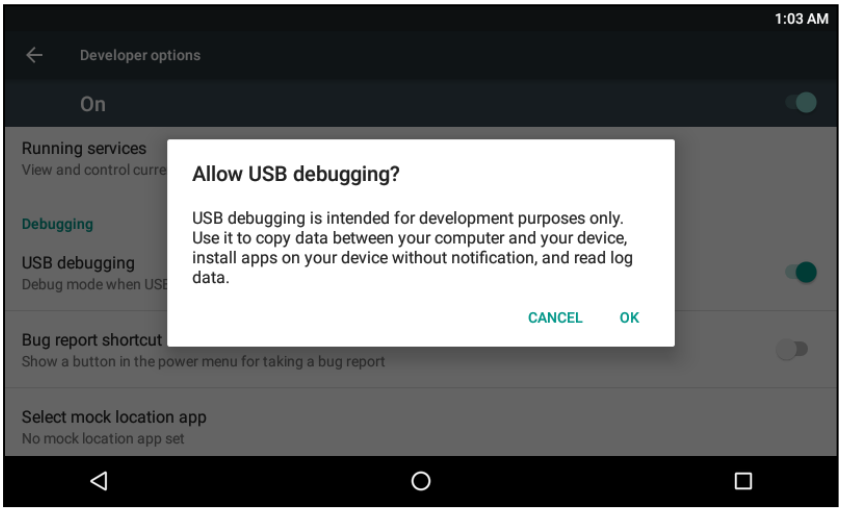
3. Scroll to the bottom of the list and click, “Build Number” repeatedly until a message is displayed saying, “you are now a developer”.



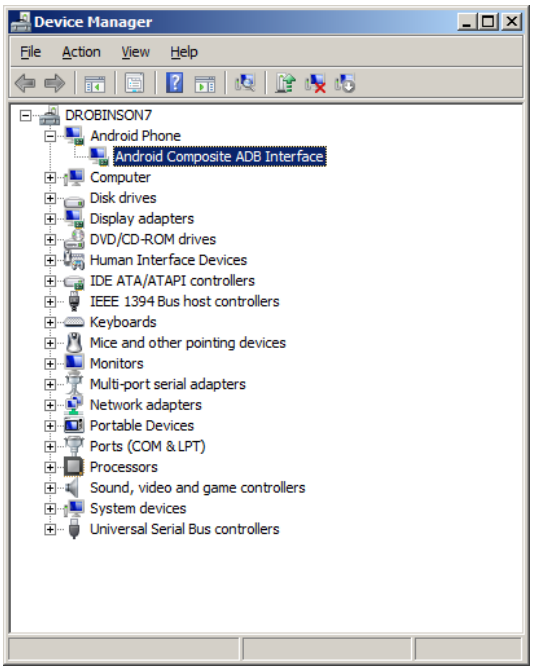
4. Press the back button, and click on, “Developer Options”. Enable the Developer Options at the top of the screen: toggle the big “On” switch.



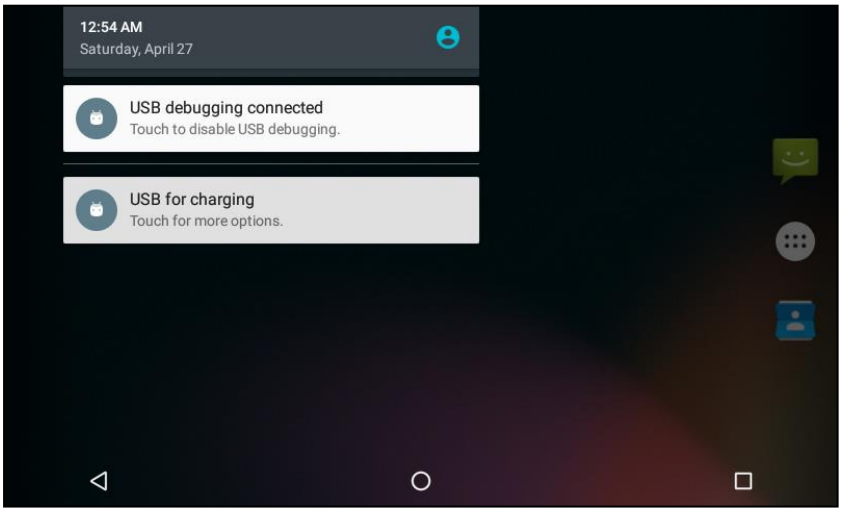
5. Scroll down to the option, “USB Debugging”, and click to enable the feature. You may be prompted to confirm that debugging is allowed.



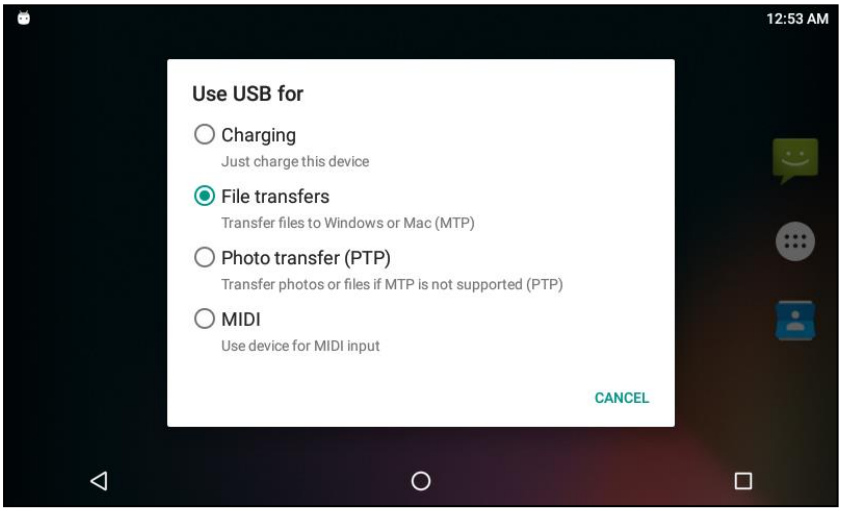
6. Connect a USB Host cable between the development PC and TM1/HB5.
7. Windows should detect a new ADB USB device and search for drivers. If a driver cannot be found automatically it may be necessary to point Windows device manager at the following location. <Android SDK root>\extras\google\usb_driver.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



9. Display the notifications in the Status bar at the top of the screen and click at “USB for charging” item.



10. In the “Use USB for” dialogue select the “File transfers” option.



3.1 Enable ADB Debugging on DB1

When USB Host cable is attached between DB1 and a PC then DB1 acts a USB device. In such state all peripherals connected to DB1's USB ports are disconnected and can't be used. That means keyboard and computer mouse connected to DB1 and also Ethernet adapter (which is connected via internal USB interface) will not work as long as the USB Host cable is connected to a PC. When the USB Host cable is disconnected then all USB devices connected to DB1 are automatically re-initialised and should work normally.

Debugging an Android application that requires USB peripherals to function (for example an Ethernet adapter or a USB camera) may need to establish ADB connection over Ethernet network. To enable ADB over Ethernet network:

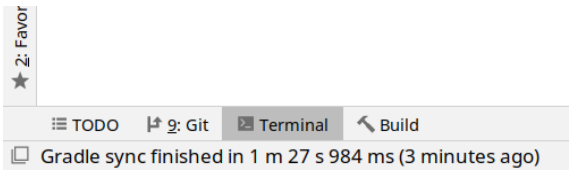
- Ensure the USB Host cable is connected to your PC and issue the following command

  ```
  adb tcpip 5555
  ```
- Disconnect the USB Host cable and wait until the Ethernet network is connected. Note the IP address of the DB1 (check Settings->About tablet->Status)
- Establish the ADB connection over Ethernet

  ```
  adb connect <db1-ip-address>:5555
  ```

Once the ADB is connected to DB1 then all subsequent ADB commands (adb shell, adb logcat etc.) should work normally as expected.

Android Studio does not have a UI element to enable ADB over Ethernet. You can however type the above commands to the Terminal panel which is opened by clicking at the 'Terminal' tab located at the bottom left of the Android Studio window.



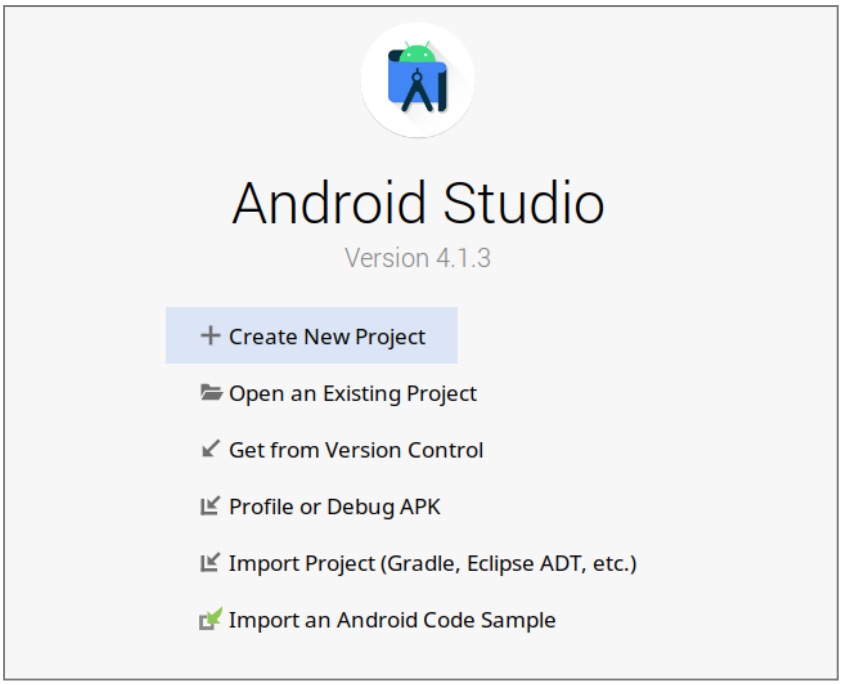
4. Simple GPIO light switch example

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 4.1.3 is installed, and that the SDK manager is setup as per the previous section.

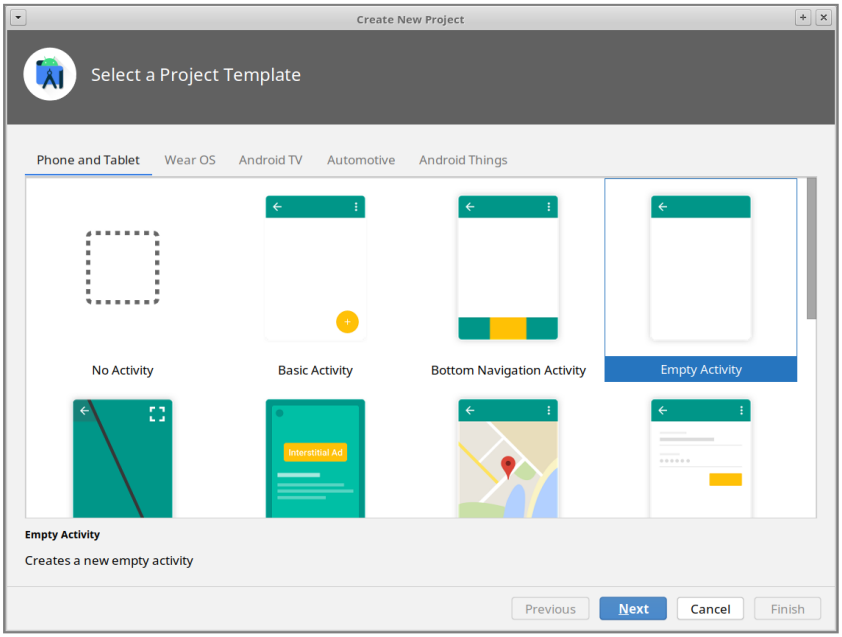
Please note that Android Studio features may change significantly in between versions. If that happens and your current version of Android Studio does not contain screens presented in the following examples please download and run version 4.1.3 (March 18, 2021) from the following link:

https://developer.android.com/studio/archive

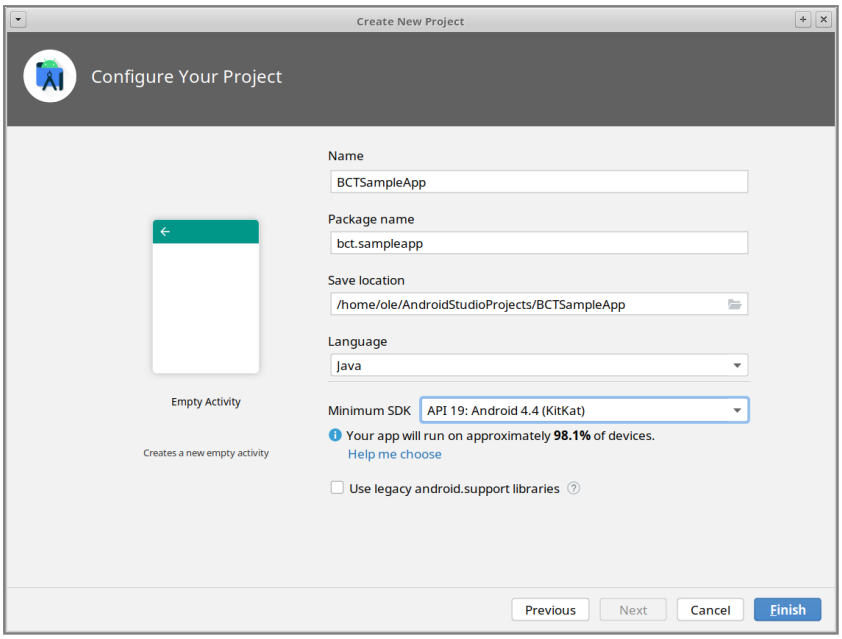
1. Start Android Studio
2. Click "Create New Project", in the "Welcome to Android Studio" menu.



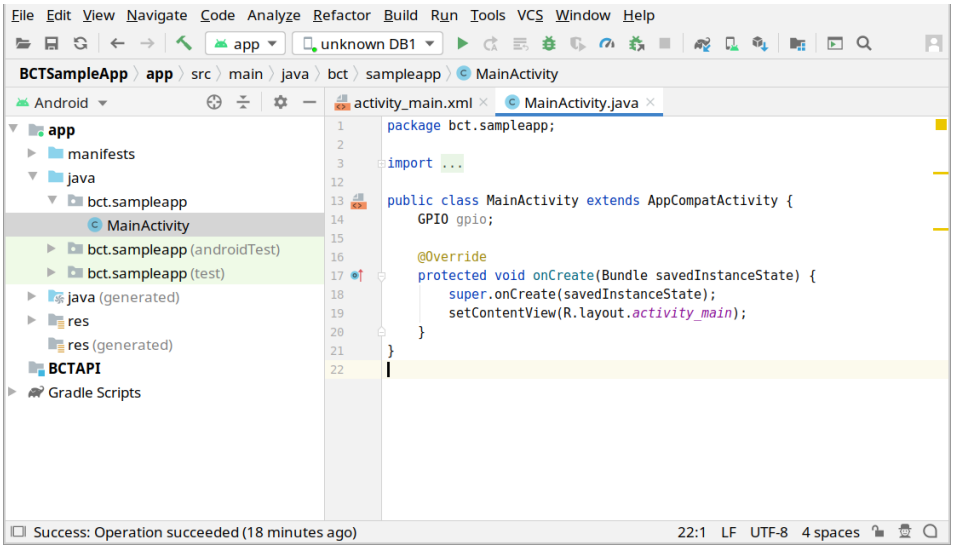
3. Chose “Empty Activity” in the “Select a Project Template” step.



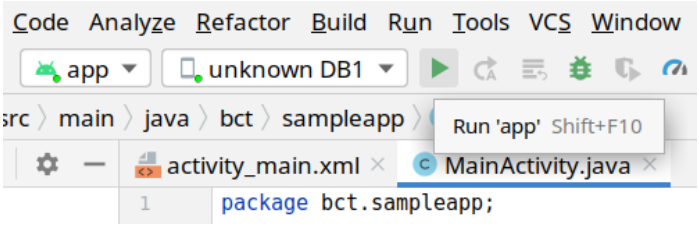
4. Give the project a name and namespace.



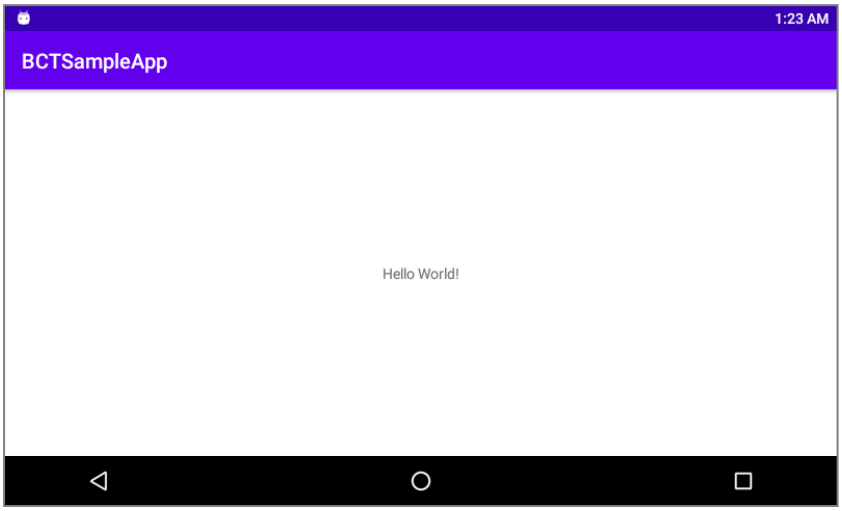
5. Tell the wizard that the app is targeting API 19 for a Phone / Tablet device in the “Minimum SDK” combo box and click finish.
6. Android studio will now initialise the development environment.



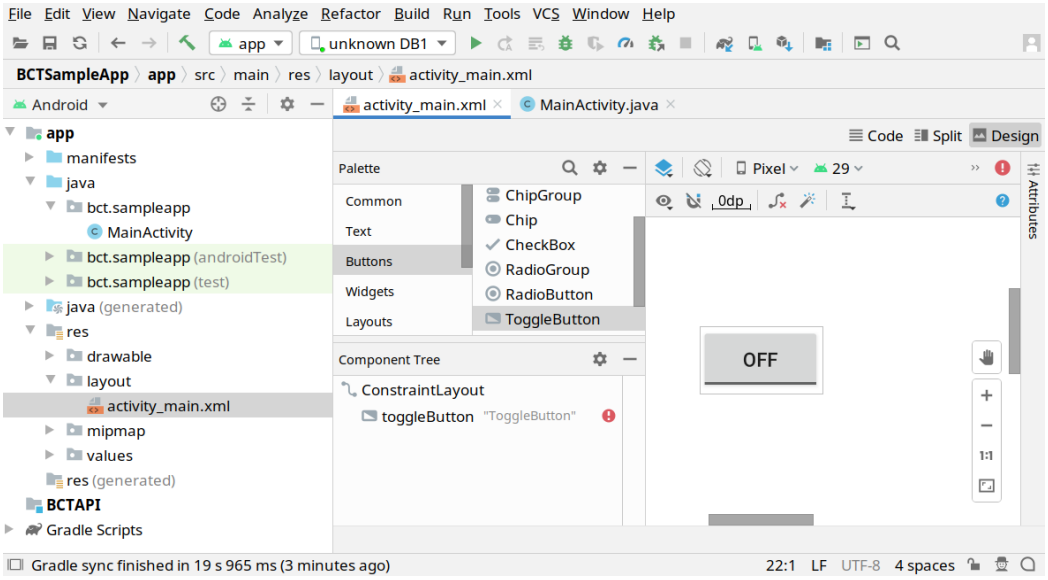
7. At this stage it is advisable to build and deploy the app in its default state. Ensure that the TM1/HB5 device has been setup for debug over USB and that the appropriate driver has been installed on the development PC. See previous section for details.
8. Select the TM1/HB5 device in the “Device Chooser” dialogue box and press ok. If the device is displayed as unauthorised, check the TM1 /HB5 display for an authorisation request.
   NOTE: BCT DB1 board is identified as “unknown DB1” device.
9. Click the “Run app” button to deploy the app



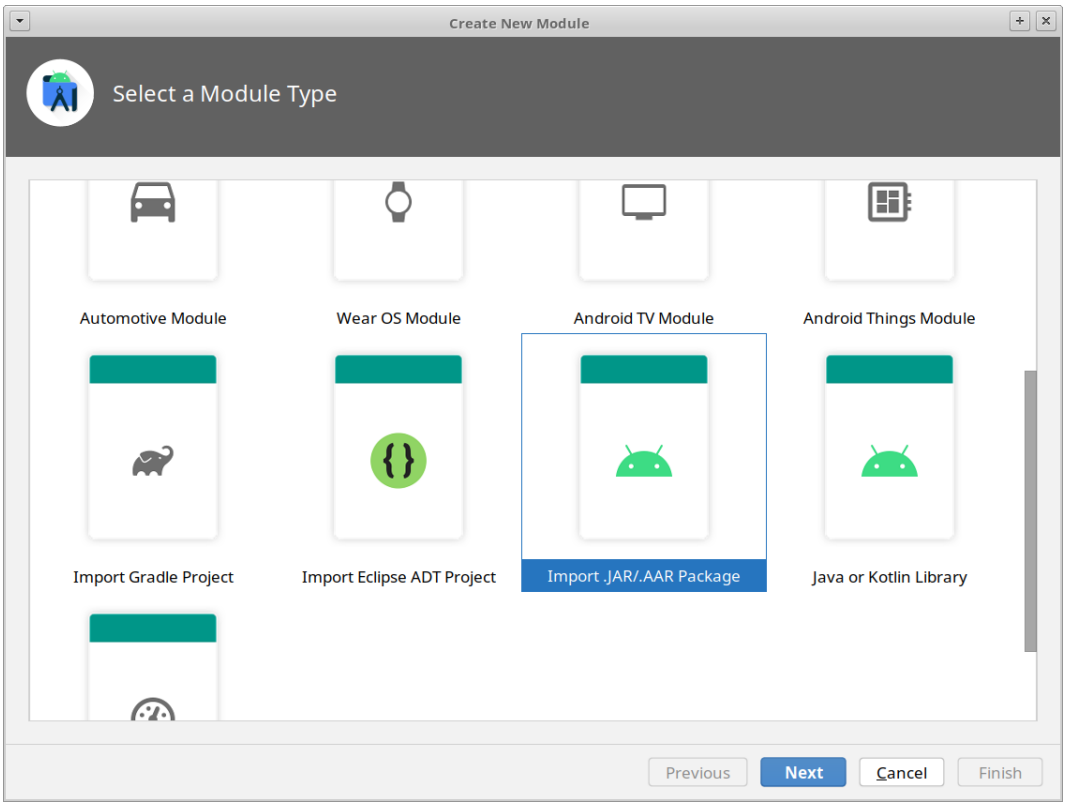
10. The Hello world application should automatically deploy to the device and execute.



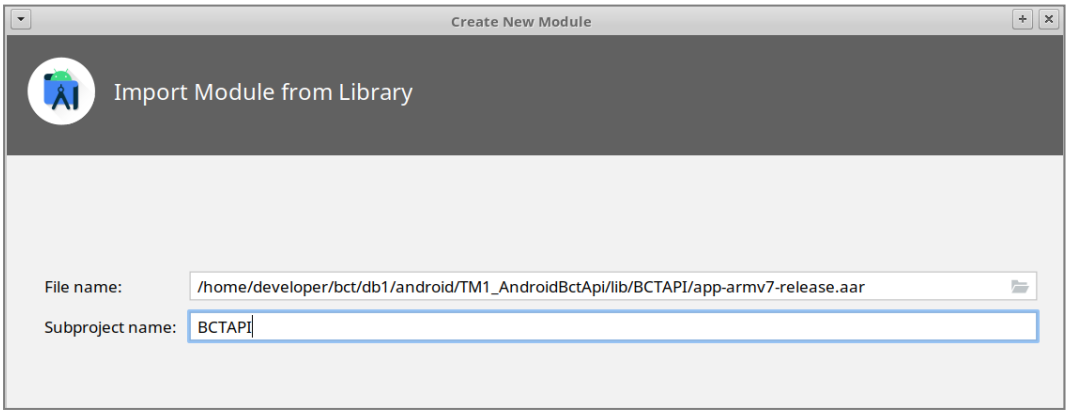
11. Add a toggle button control to the activity_main.xml gui designer.



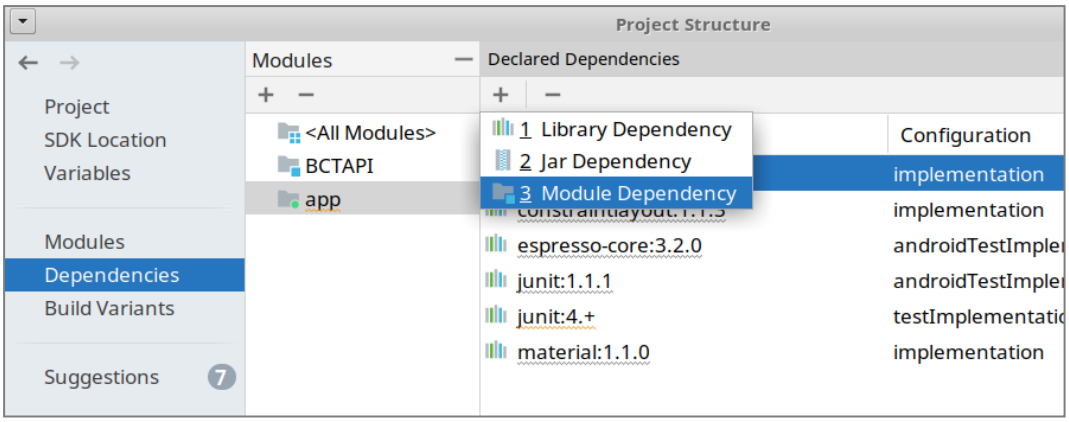
12. Import the BCTAPI.aar library into the Android project.
    a. Obtain the app-armv7-release.aar library from Blue Chip Technology
    b. Choose File : New : New Module
    c. Select Import JAR/AAR option (as shown below)



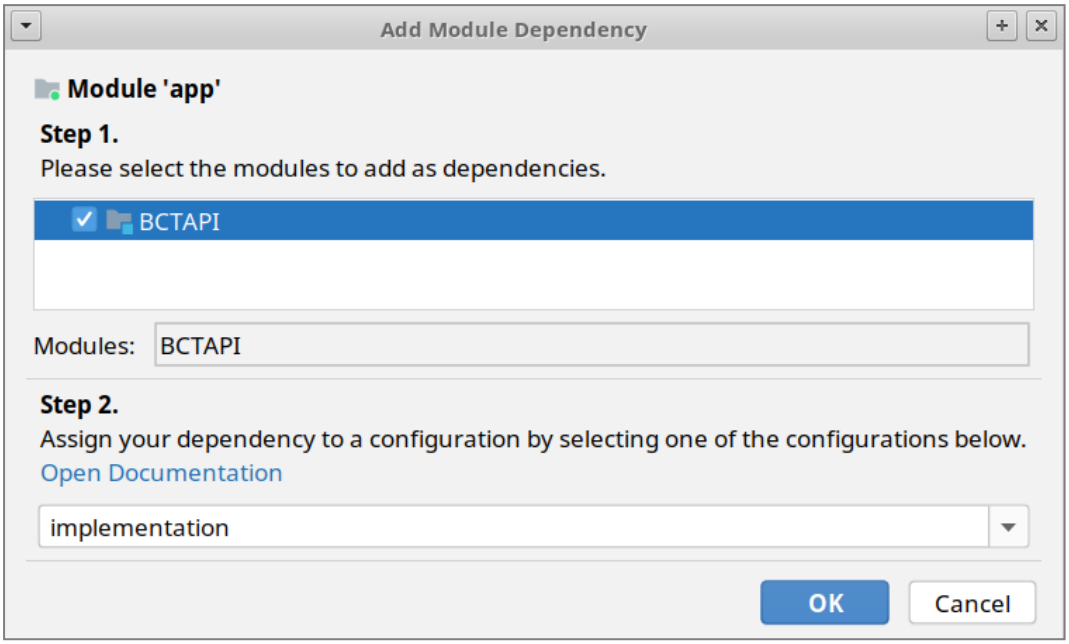
13. Locate the “app-armv7-release.aar” file (the BCT API library) on your PC. Use “BCTAPI” as the Subproject name. Then click Finish button.



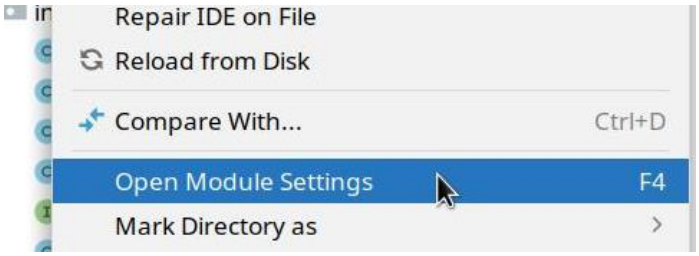
14. Add the BCTAPI module as a dependency of the BCTSampleApp.
    a. Select the Dependencies item in the left column.
    b. Click at the “app” module in the Modules list.
    c. Click at the [+] button under the “Declared Dependencies” panel.
    d. Select “Module Dependency” option.



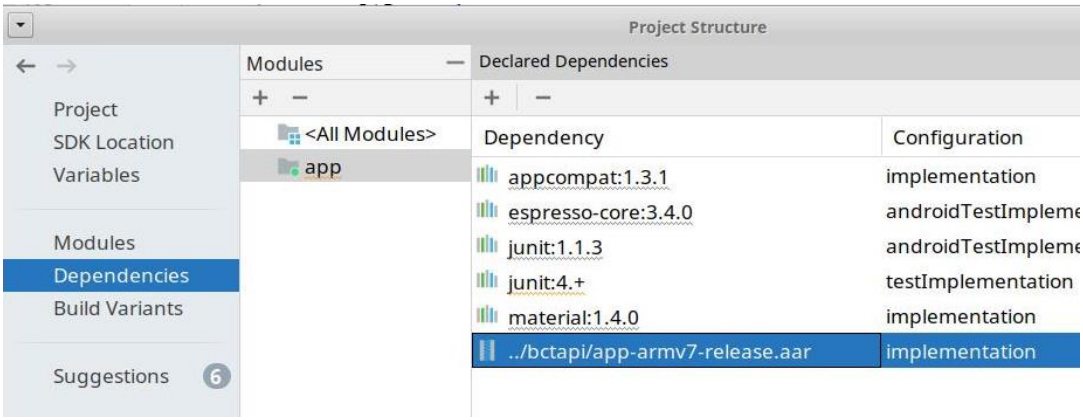
e. Choose BCTAPI from the list of modules presented (as shown below)



Note than recent versions of Android Studio (2022.02 and newer) may not display the “Import .JAR/.AAR Package” option. In such case, press the Right Mouse button on the Application top level item in the Project explorer and select option “Open Module Settings - F4”.



It will open the dependency dialogue that allows to specify the BCT API library location.



15. 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());
                }
            }
        });

    }
}
```

16. Run the app in the same way as step 9. When running on BCT TM1 board the LED on p12 (Ethernet Connector) will indicate the state of the toggle button. On BCT DB1 board the LED is located in the top corner of the board, next to the Ethernet connector.

Install APK on TM1

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

On PC:

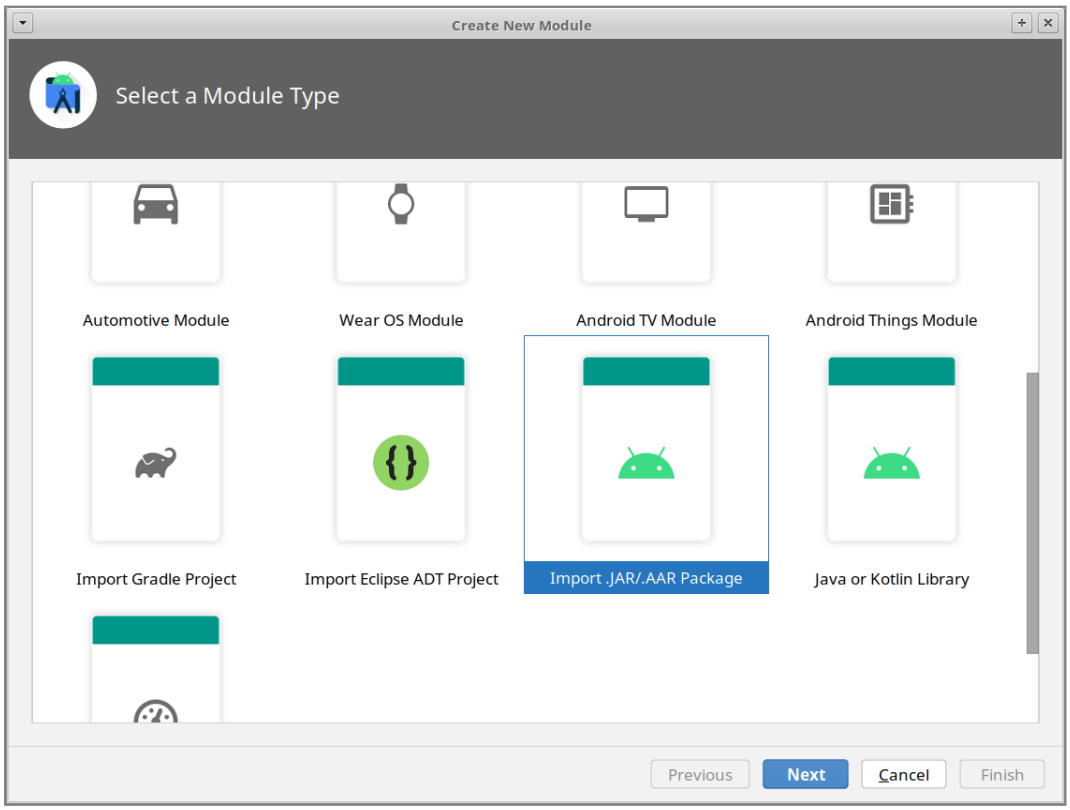
1. Connect the TM1 to a PC using the USB device port on the TM1.
2. Ensure the TM1 device uses “USB for file transfer” option. See steps 9 and 10 in Section 2.
3. Wait for the TM1 to appear as a drive on the PC
4. Navigate to the folder: ‘Computer\tm1hb5\Internal shared storage\Download’
5. Copy the 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.

5. BCTAPI

The BCT API for Android is distributed in the form of a java AAR file with filename app-armv7-release.aar. The library is compatible with 32 bit and 64 bit Android systems. The library can be imported into an Android Studio project by using the New Module wizard. E.g.



Note than recent versions of Android Studio (2022.02 and newer) may not display the “Import .JAR/.AAR Package” option. In such case, press the Right Mouse button on the Application top level item in the Project explorer and select option “Open Module Settings - F4”. It will open the dependency dialogue that allows to specify the BCT API library location.

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
```

SerialPort(...)

Definition:

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

Description:

It is a constructor of SerialPort object. It 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.

DB1 supports two serial ports:

/dev/ttyHSL1 – RS232 levels on P4 (COM2)

/dev/ttySC0 – RS232 or RS422/485 levels on P4 (COM3)

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(...)

Definition:

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

Description:

It is a constructor of SerialPort object. It opens a serial port with the specified parameters.

Parameters:

comPort – The symbolic representation of the serial port. Use SerialPortDefinitions.COM1, SerialPortDefinitions.COM2, SerialPortDefinitions.COM3. The COM1 port is connected to Android serial console terminal, COM2 is an RS232 port, COM3 can be either RS323 port or RS422/RS485 port based on the jumper configuration (physically on the board) and the type of the wiring (RS422 – 4 wires full duplex, RS485 – 2 wires half duplex). See the board hardware manual for more information related to serial ports and their locations.

The rest of the parameters are identical to the previous SerialPort constructor.

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). On BCT DB1 board it corresponds to a green LED in the corner of the board next to the Ethernet connector.

GPIO_422_485_TXEN – Control of the transmit control signal for the RS422 / 485 transceiver represented by COM3 serial port. It is applicable only to BCT TM1 boards.

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.SetInputListener

Definition:

```
void SetInputListener (IGPIOListener listener) throws IOException
```

Description:

Set a IGPIOListener instance that will be notified when the input state of the GPIO changes. The GPIO must be configured as Input before setting the input listener. Only one listener can be set to a particular GPIO. The same listener can be set to multiple GPIOs. To stop/remove the listener call SetInputListener with null listener parameter. The GPIO is internally polled every 10 ms, therefore state changes within 10 ms time frame may not be detected.

```
void SetInputListener (IGPIOListener listener, int pollMs) throws IOException
```

Description:

Same as above, but allows to set the polling time. Valid values of 'pollMs' are from 1 to 100 ms. Note that the shorter poll times may increase the CPU load especially if multiple GPIOs are sampled.

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

IGPIOListener interface

```
public void onGpioInputChanged(GPIO gpio, int value );
```

The function is called when the GPIO's input state changes.

```
public void onGpioInputStopped(GPIO gpio);
```

The function is called when the GPIO's listener is stopped.

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. Note that the maximum timeout on TM3 platforms is 16 seconds because of the hardware limitiations.

Watchdog.DisableWatchdog

Definition:

```
void DisableWatchdog();
```

Description:

Disables the watchdog. Once the watchdog is disabled it no longer resets the board even after the timeout period is expired.

Watchdog. RefreshWatchdog

Definition:

```
void RefreshWatchdog();
```

Description:

Refreshes the watchdog countdown timer, to prevent a system reset. The countdown timer is reinitalised to contain the timeout value passed to the Watchdog via EnableWatchdog() function.

The watchdog will reset the system unless the RefresWatchdog function is called again within the timeout period.

Watchdog. getLastResetSource

Definition:

```
SystemResetSource getLastResetSource();
```

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. Watchdog must be enabled prior calling this function.

PWM API – Requires a special build for TM1. Contact BCT. Not available for DB1.

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".

Note: CAN Socket API is not available for BCT DB1 boards at the time of writing this document. Existing CB3 modules for TM1 are not compatible with DB1 dues to voltage differences, please do not use CB3 module with DB1 board. If you need a CAN Socket support for DB1 boards, please contact your sales representative.

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

SysInfo Class

Class Namespace:

```
bct.hwapi.SysInfo
```

SysInfo.getHardwareConfiguration()

Definition:

```
public static HardwareConfiguration getHardwareConfiguration() throws Exception
```

Returns a HardwareConfiguration enum object representing the board and specific peripherals the code runs on. Valid return objects are: BCT_TM1_HB5, BCT_TM1_TV1, BCT_DB1, BCT_TM3_HB5, BCT_TM3_HB8, BCT_TM3_TV2. The function can be used to check the code runs on specific board or to use appropriate resources for given board.

SysInfo.getAPIVersionInfo()

Definition:

```
public SysInfo.Version getAPIVersionInfo()
```

Returns an instance of SysInfo.Version object containing the API version numbers. The function can be used for checking certain features of the BCT API are available or not.

SysInfo.getLibraryVersionInfo()

Definition:

```
public SysInfo.Version getLibraryVersionInfo()
```

Returns an instance of SysInfo.Version object containing the Library version numbers. The function can be used for identification of the BCT API native library during troubleshooting.

SysInfo.Version

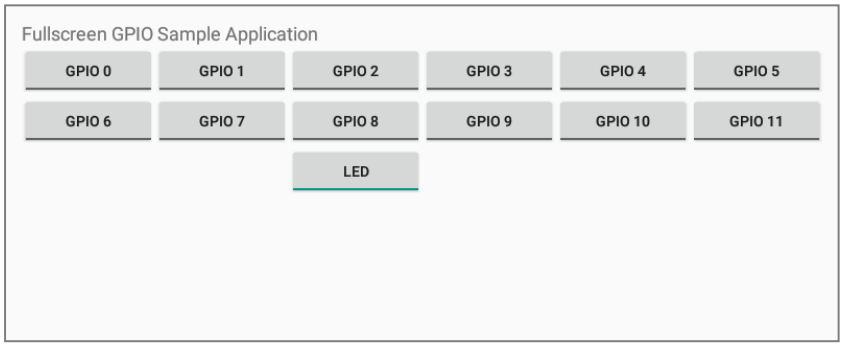
Definition:

```
public class Version {
    public int VersionMajor;
    public int VersionMinor;
}
```

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.



Features:

- Creates and configures instances of GPIO class.
- Allows toggling the GPIO pins via on screen buttons.
- Allows turning on/off onboard LED via dedicated screen button.
- Enables full screen mode.
- Registers the application as the Home App (can auto launch the app after boot). Requires restart of the board.

TM1HB5-AC1-RS485Demo

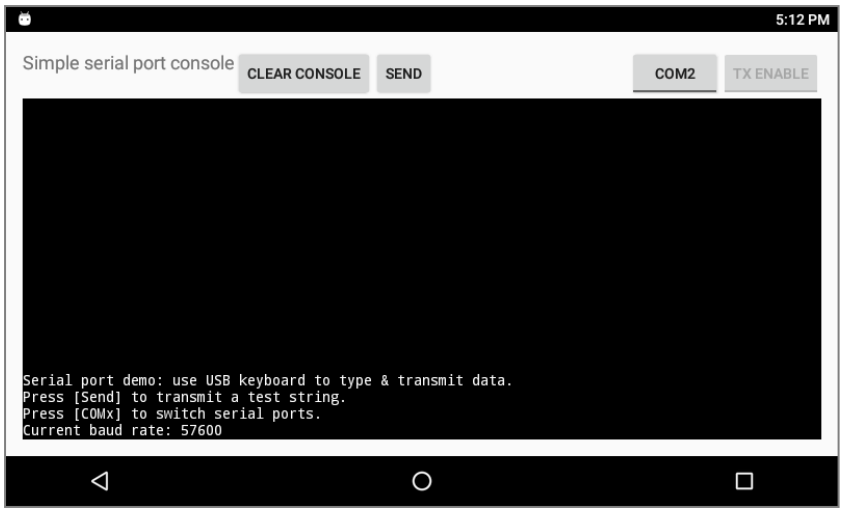
This sample demonstrates how to use /dev/ttymxc2 in RS485 mode using the SerialPort API. The UART is configured in auto RS-485 transmit mode. The application is written using Xamarin and C# and specifically targets TM1 board. The demo requires a slave device (AC1 board) connected via RS-485 half duplex bus with the TM1 board.

RS485-AC1-Example

This sample application is a Java port of the above C# based demo. It demonstrates how to use the COM3 port in RS485 mode using the SerialPort API. The UART is configured in auto RS-485 transmit mode. The demo requires a slave device (AC1 board) connected via RS-485 half duplex bus with the TM1 or DB1 board.

TM1HB5SerialportSample

This sample demonstrates how to use COM2 and COM3 in RS232 and RS485 mode using the SerialPort API. The GPIO API is used for transmit enable control.



Features:

- Transmits outgoing data over COM2 (RS232) or COM3 (RS422/RS485) serial port either via attached USB keyboard or via dedicated UI button.
- Receives incoming data over COM2 or COM3 serial port.
- Prints the received data on the screen.
- COM2 / COM3 port is selectable via UI button
- Manually controls RS485 TX enable GPIO on the boards that support such functionality.

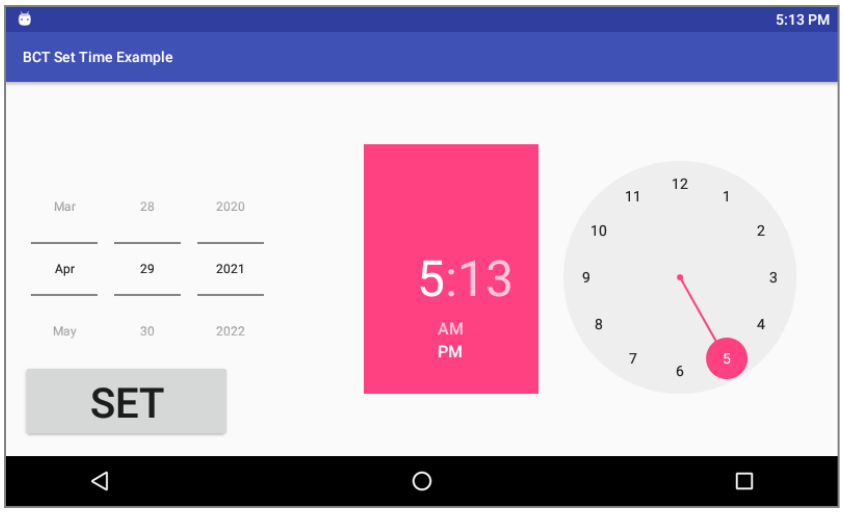
TM1HB5CanSocketSample

This sample demonstrates how to use the Socket CAN API to send and receive messages over the CAN bus. Note that the CAN interface must be physically linked to another CAN device on the bus for this example to work as expected. For example on CB3 expansion board CAN0 and CAN1 can be physically linked together.

The app will function only on TM1 boards with attached CB3 expansion board or on custom boards with CAN hardware.

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. It also works on DB1 platform.



AndroidPdfViewer

In Android 4.4 there is no native PDF viewing capability. AndroidPdfView has been downloaded and tested to work on the TM1 and DB1 platforms. It can be embedded into a 3rd party application and the library is distributed under the Apache 2.0 licence.

https://github.com/barteksc/AndroidPdfViewer

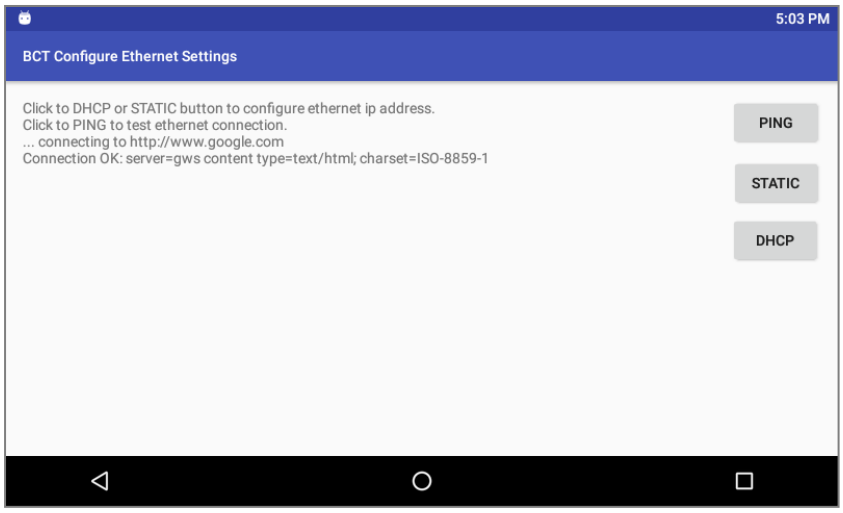
There are other third party solutions available for viewing PDF files in Android.

Features:

- Displays a PDF document and allows basic navigation within the document.
- Allows text zooming by double tapping on the screen.

ConfigureEthernetSettings

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



Features:

- Static IP Address assignment via UI button.
- Dynamic IP Address assignment via UI button.
- Internet connection test via Http request.

To verify the actual IP address numbers please use the serial console on COM1 port and enter ‘ip a’ command and check the ‘eth0’ interface information.

Note 1: The dynamic address assignment requires the board to be connected to LAN via an Ethernet cable. Also a DHCP server must be active in the LAN in order to receive a valid IP Address. It might take several seconds to acquire the dynamically assigned IP address.

Note 2: On BCT DB1 board the Ethernet interface is connected to USB bus. Please disconnect the USB OTG cable (USB debugging/development cable) to ensure the Ethernet interface is functional.

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.

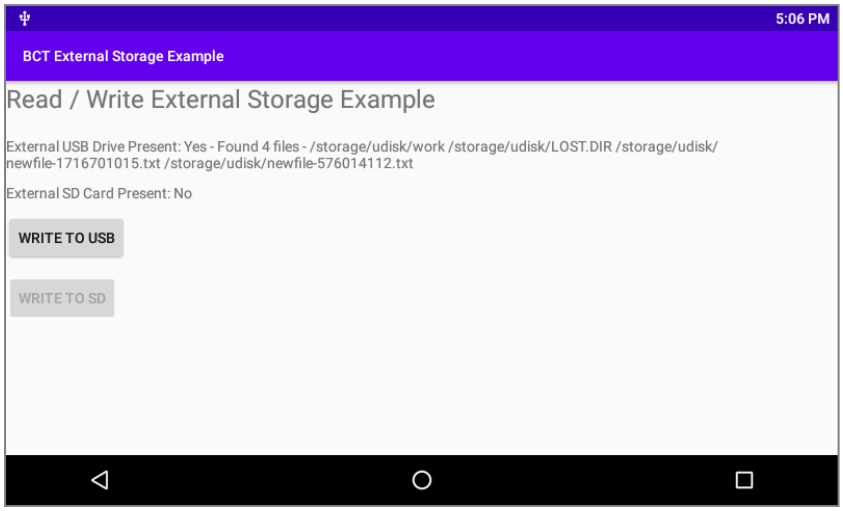
BCT DB1 does not support this example at the time of writing this document.

TM1HB5SPIExample

This sample demonstrates how to communicate with devices connected to SPI bus. The SPI functionality requires specific kernel driver and custom pin settings to communicate with a generic SPI device. Please contact your sales representative if you need to connect to external devices over SPI bus on TM1 or DB1 boards.

TM1HB5ExternalStorageExample

This example demonstrates how to access external storage on TM1 and DB1 boards.



Features:

- Detects insertion and removal of a USB disk drive.

- Detects insertion and removal of micro SD card.
- Reads the contents of the root directory of the inserted external storage.
- Prints the root directory file list on the screen.
- Writes a file to the external storage.

TM1HB5I2CExample

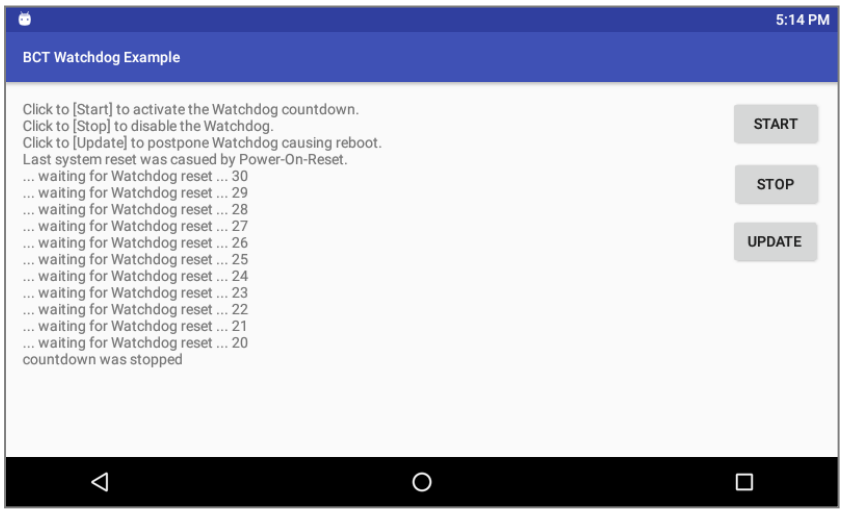
This example demonstrates how to use I2C API to communicate with I2C peripherals.

Features:

- Opens and closes a specific I2C bus.
- Uses SysInfo class for I2C bus selection.
- Writes data to specific slave device on the I2C bus based on its address.
- Read data from a slave device on the I2C bus.
- Uses existing on-board RTC I2C device to exercise the API.

WatchdogExample

This example demonstrates how to use watchdog to automatically reset the board.



Features:

- Enables / Disables the watchdog via UI button.
- Refreshes the watchdog via UI button.
- Displays the watchdog reset countdown.

AudioExample

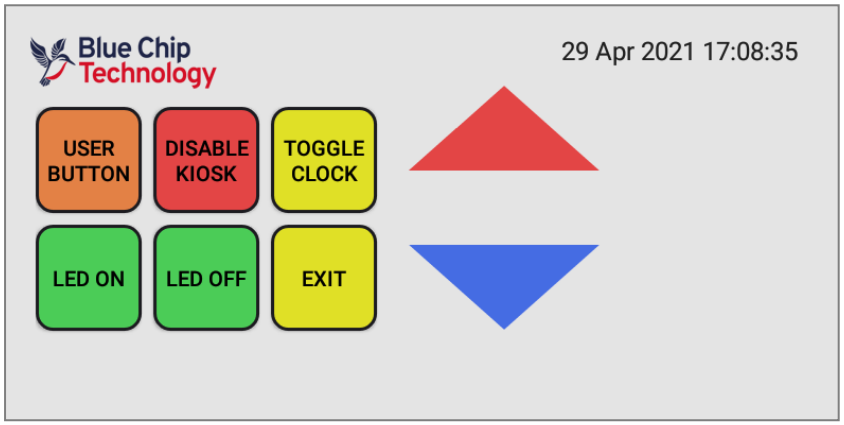
This example demonstrates how to use audio recording and playback capabilities of the TM1 andDB1 boards.

Features:

- Starts / Stops an audio recording.
- Starts / Stops an audio playback.
- Enabled /disables the onboard Class D audio amplifier (enables output on the Speaker port)

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.



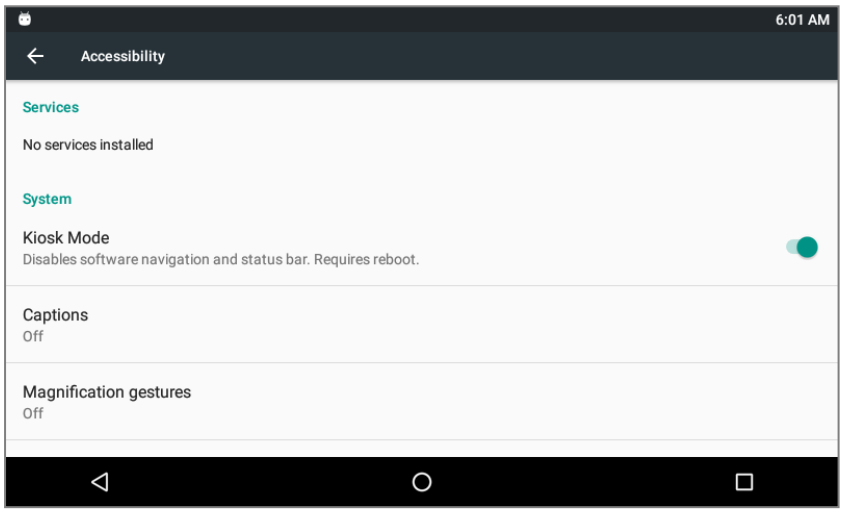
Features:

- Installs a custom Boot animation file from the internal resource during application start-up.
- Enables the Kiosk mode during application start-up (requires board reboot).
- Disables the Kiosk mode via UI button.
- Controls the Backlight level via UI buttons
- Enables / Disables the backlight to be turned completely off (requires board reboot)
- Programmatically reboots the board.
- Disable / Enable UI elements.
- Use GPIO API to control the onboard LED via UI buttons.

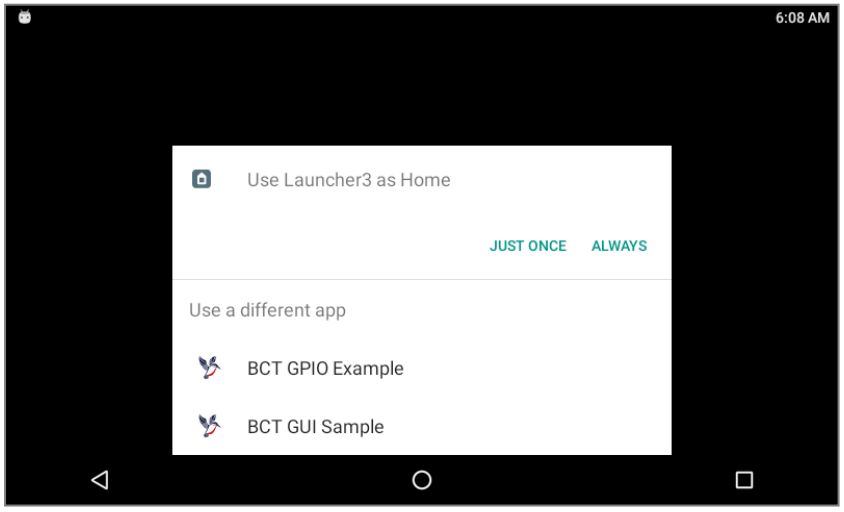
Backlight level feature explanation: by default Android OS does not let the screen backlight to go completely off without pressing the power button. While it is indeed possible to press the power button on TM1 and DB1 boards to turn the screen off, the side effect is that it puts the whole system into a suspended state where peripherals and communication channels are turned off as well. That is often undesired in industrial environment. Therefore TM1 and DB1 Android OS was customised to support reducing the screen backlight level to zero as if the screen is turned off. The touch-screen still works when the backlight level is set to zero, therefore the running application can detect the screen touch events and in turn to increase the backlight level to 'wake-up' the screen. Enabling this feature requires to set a specific system property as demonstrated in the GuiSample source code. The change is applied after the next board reboot. The feature can be reverted by setting a non-zero value in the same system property.

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.



The Home application selection dialogue is displayed after the next reboot of the device. The Home application can also be selected in Settings: Home.

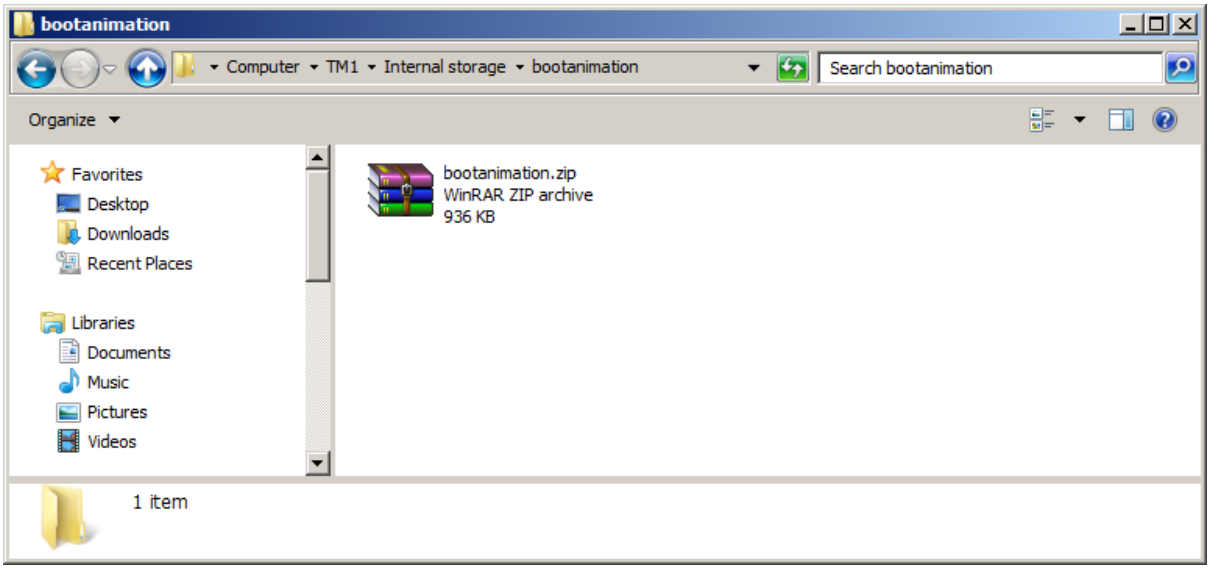
Please note: in Android 6 and 7 the Kiosk mode can be only used when there is no ‘Screen lock’ selected in Settings: Security: Screen lock.

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.



The boot animation zip file can also be programmatically changed from within your Android application. Please see the GuiSample source code for more information.

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.

Please check the ConfigureEthernetSettings sample application source code for an example how to use the EthernetSettings API. Check the section 6 for more information about the ConfigureEthernetSettings sample application.

11. Android permissions

Since Android version 5.0 the operating system requires that certain permissions are confirmed by the user via dialogues. Such permission-request dialogues appear during the first start-up of the application and once the permission is granted then the request/dialogue it is never displayed again. Note that certain permissions require additional source code changes that are beyond the scope of this document.

If the permission dialogues are not acceptable they can be eliminated by setting the 'minSdk' and 'targetSdk' to value 19 in the application's build.gradle configuration file.

12. 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
1.11	29/04/2021	MO	Updated screenshots to match Android 6 and Android studio 4.1.X. Added BCT DB1 board specific information. Updated BCT API SerialPort constructor. Updated Watchdog.getLastResetSource(). Added SysInfo API class description. Updated sample app description and added application screenshots. Added AudioExample, ExternalStorage, SPIExample and WatchdogExample app descriptions. Added backlight level information to GuiSample. Added information to Kiosk Mode about restrictions on Android 6 and 7. Added note to BCT.NETCONFIGSERVICE pointing to the ConfigureEthernetSettings app. Added issue level 1.11 to Document history. Added Appendix B for DB1 known issues.

1.11	09/12/2021	MO	Added Android studio download link. Added RS485-AC1-Example description. Added “3.1 Enable ADB on DB1” section.
1.12	16/10/2023	MO	Introduction: Added TM3 as a supported platform. Added note about importing the BCT API library in the recent Android Studio. Added GPIO.SetInputListener function documentation and IGPIOListener interface description. Watchdog: added note about TM3 maximum timeout limit. Updated CAN socket sample information. Added Android permission section. Added appendix C – TM3 issues and limitations.

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.
- Kiosk mode does not work with the Screen Lock feature.

Appendix B: DB1 Android 6.0.1 (Marshmallow) Known Issues

- WiFi AP scan takes a longer time when the WiFi is enabled for the first time.
- Kiosk mode does not work with the Screen Lock feature.

Appendix C: TM3 Android 9.0 Known Issues and limitations.

- No Bluetooth support.
- Kiosk mode does not work with the Screen Lock feature.
- Maximum timeout for watchdog is 16 seconds.