Six15 HUD

Android Software Development Kit User Guide

Version 2.6.4

8/24/2021

Document Number: 555ES0002-13

Table of Contents

Android Software Development Kit 1

User Guide 1

List of Tables 5

1. Six15 HUD Product 6
   1.1 Introduction 6
   1.2 Firmware and Service Compatibility 6
2. Software Development Kit Contents 7
   2.1 What is Included 7
   2.2 Six15 HUD Service 7
      2.2.1 Service Provided Functionality 7
      2.2.2 Six15 HUD Service Installation 7
   2.3 Development Workstation Setup 7
3. Six15 HUD Service Interface 9
   3.1 Assumptions 9
   3.2 Constraints 9
   3.3 Service API 10
      3.3.1 Android Interface Definition Language (AIDL) 10
      3.3.2 HUD Byte Frame 10
      3.3.3 HUD Image Frame 10
      3.3.4 HUD Service Callback Interface 10
      3.3.5 HUD Service 12
      3.3.6 HUD Command BYTE Definition 14
   3.4 HUD Service API Definitions 15
      3.4.1 Information and Hardware Functions 15
         3.4.1.1 Service Initialized 15
         3.4.1.2 Register Service Callback 15
         3.4.1.3 Unregister Service Callback 16
         3.4.1.4 Check Service Callback 16
         3.4.1.5 Get HUD Connected 16
         3.4.1.6 Get SDK Version 16
         3.4.1.7 Get HUD Information 17
         3.4.1.8 Get HUD Information Fast 17
         3.4.1.9 Get HUD Version Information 18
         3.4.1.10 Get HUD Settings 18
         3.4.1.11 Set HUD Settings Defaults 19
         3.4.1.12 Set HUD Settings 19
         3.4.1.13 Ping Command 20
         3.4.1.14 Query Device Mode 20
         3.4.1.15 Soft Reset 20
      3.4.2 Display Functions 21
         3.4.2.1 Get HUD Display Size 21
         3.4.2.2 Get Auto Resize of Image 21

3.4.2.3 Set Auto Resize of Image Data 21
3.4.2.4 Get Last Frame Send Time 22
3.4.2.5 Get Display Information 22
3.4.2.6 Get Display Status On/Off 22
3.4.2.7 Set Display Status On/Off 23
3.4.2.8 Get Display Brightness Level 23
3.4.2.9 Set Display Brightness Level 23
3.4.2.10 Clear HUD Display 24
3.4.2.11 Send JPEG file to HUD 24
3.4.2.12 Send Android Bitmap to HUD 24
3.4.2.13 Send Buffer to HUD 25
3.4.3 IMU Functions 25
3.4.3.1 IMU Data 25
3.4.3.2 IMU Status 26
3.4.3.3 Start IMU 26
3.4.3.4 Stop IMU 27
3.4.3.5 Start User IMU Cal 27
3.4.3.6 Get User IMU Cal 27
3.4.4 HUD Camera Functions 28
3.4.4.1 Camera Field of View Measurements 28
3.4.4.2 Set HUD Camera Enabled 28
3.4.4.3 Get HUD Camera Enabled 29
3.4.4.4 Set Camera Autofocus Mode 29
3.4.4.5 Get Camera Autofocus Mode 29
3.4.4.6 Start Camera 30
3.4.4.7 Start Camera (Raw JPEG Mode) 30
3.4.4.8 Stop Camera 30
3.4.4.9 Get Supported Resolutions 30
3.4.4.10 Get Current Camera Resolution 31
3.4.4.11 Set Current Camera Resolution 31
3.4.4.12 Get Camera Zoom 31
3.4.4.13 Set Camera Zoom 32
3.4.5 HUD Microphone Functions 32
3.4.5.1 Get HUD Microphone Enabled 32
3.4.5.2 Set HUD Microphone Enabled 32
3.4.6 HUD User Data Functions 33
3.4.6.1 Get User Data 33
3.4.6.2 Set User Data 33
3.4.7 Firmware Update 33
3.4.7.1 Get Current Firmware Version 33
4. Six15 Getting Started 35
4.1 Getting Started Overview 35
4.2 Creating a new Android application 35
4.2.1 Create new Android Studio project 35
4.2.2 Add Six15 AIDL and Supporting Classes 39
4.2.3 Add Features and Permissions 46

4.2.4 Create Callback Interface and Handler 51
4.2.5 Add Broadcast Receiver 54
4.3 Calling the HUD Service. 56
5. Six15 Presentation Mode Demo 57
5.1 Create a new application 57
5.1.1 Add a display listener 57
5.1.2 Find HUD Display and send Activity to HUD 58
6. Known Issues 59
7. Troubleshooting 60
7.1 Device Boots into Safemode 60
7.1.1 Powercycle 60
7.1.2 Reflash the Firmware 60
7.1.3 Contact Six15 60
8. SDK Changelog 61
8.1 SDK 2.1 to 2.2 61
9. Document Revision History 64

List of Tables

Table 1 – Document Revision History 64

1. Six15 HUD Product

This document is intended for developers creating Head-Up Display (HUD) applications using the Six15 HUD. It communicates all possible inputs and outputs of the Six15 HUD service interface.

1.1 Introduction

This guide describes how a developer can create applications for an Android Device (the source system) to control the HUD (the target system), as well as describing the specific interface requirements that the participating systems must meet. It describes the concept of operations for the interface and defines the message structure and protocols used by the development library.

1.2 Firmware and Service Compatibility

Each version of the service is tested and released with a corresponding version of the firmware. There are reverse compatibility breaking changes that require the firmware and service to stay in sync.

SDK Release	Android Service Version	Firmware Version
1.12 and before	4.0.1	4.25
2.00	5.21	5.21
2.1.0	5.31	5.23
2.2	5.34	5.25
2.3	5.41	5.30
2.4	5.43	5.33
2.5.0	2.5.0	5.34

Starting at release 2.5.0, all app versions track the SDK release version.

Firmware updates for the HUD can be done through Windows Service or the Android application. See 555ES0012 Windows SDK Release for software and instructions.

2. Software Development Kit Contents

2.1 What is Included

The development kit includes the HUD hardware, Android HUD Service, Demonstration Application, Presentation Mode Application, Documentation and Source Code.

2.2 Six15 HUD Service

The Six15 HUD Service is a standalone service that is installed on Android devices to provide basic functionality for your HUD device and provide the abstracted interface for programmers to develop custom applications for the HUD device.

2.2.1 Service Provided Functionality

The HUD service provides basic functionality such as remote display (screen mirroring and presentation mode) and HUD camera interface. It also provides the ability to query the HUD's information and update the HUD's firmware as well as gather information about the orientation of the HUD.

2.2.2 Six15 HUD Service Installation

The HUD service installation is provided via a zip file downloaded from the Six15 Developer website. Unzip the files and plug in the Android device to the computer. On the Android device, open the notification menu and select the "USB charging this device. Tap for more options" option. A list of options for the USB functionality will appear, select the "Transfer files" option.

Next, open a file explorer and navigate to the downloads folder of the Android device. Copy "Six15Hud_Android_Service_AIDL_v5.41.apk" into the downloads folder of the Android device.

From here on we will be operating only on the Android device. Find the device's file storage application and navigate to the downloads folder. Select "Six15Hud_Android_Service_AIDL_v5.41.apk" and run it. Install the application by following the onscreen prompts.

2.3 Development Workstation Setup

Starting development using the Six15 HUD requires that your workstation be setup with Android Studio. Android Studio is a freely available development environment that runs on Linux, Windows and Mac. Using the Android Studio user guide, install Android Studio on your workstation. The Install instructions for Android Studio can be found at the following link.

URL for Android Studio Installation:

https://developer.android.com/studio/install.html

You will also require having the Six15 HUD Service installed on the Android device where your application will be installed. It is the responsibility of the application to check for the installed service prior to trying to bind to the service via the application. Six15 HUD service installation

can be performed on a device as described in section 2.2.2. All examples and code have been written using Android Studio v4.0.

3. Six15 HUD Service Interface

3.1 Assumptions

The device controlling the HUD must run the Android operating system version 5.0 or higher and will control the HUD via the commands made available by the Six15 HUD service. Commands will be issued using the service binding between your application and the HUD service which is used to provide an abstraction layer to the hardware.

3.2 Constraints

The following Device constraints are imposed to ensure compatibility and interface reliability.

- Device to run Android operating system >=5.0 (Lollipop).
- Device to run Android operating system <= 11.0 (Android 11).
- Device must support USB HS connection Host or OTG.

3.3 Service API

Six15 HUD Android Service makes interfacing with the HUD easier by abstracting the details of the protocols and physical connections between the host system and the device, while providing some features and capabilities without the need for developing custom applications.

Using the Service interface provided by the Android Interface Definition Language (AIDL) in your application, commands can be issued to the HUD and information from the HUD can be received and processed. All commands are sent in an asynchronous fashion unless otherwise noted and all responses from the HUD will be in the form of a response packet containing the command and command ID that it is responding to. Data will be returned as a JSON object for easier parsing by the application.

3.3.1 Android Interface Definition Language (AIDL)

Six15 provides the interface for communicating to the HUD service as well as some example classes for parsing and packaging the HUD data. The AIDL is the main requirement for your project to interface to the service. All other classes are provided for example and ease of implementation.

Required interfaces and classes are delivered as the source package, com.six15.hudservice, that must be included with your application. The current release of the interface library is v5.41 and is packaged as a zip file Six15_HUD_Service_AIDL_v5.41.zip.

3.3.2 HUD Byte Frame

The HUD Byte Frame interface is a marshalling wrapper for sending a raw byte array of image data to the HUD for display. This is the type that must be used to send byte arrays to the HUD.

3.3.3 HUD Image Frame

The ImageFrame interface is the marshalling wrapper for sending Android image objects to the HUD. This type must be used to wrap an Android Image object to be sent to the HUD for display.

3.3.4 HUD Service Callback Interface

The HUD Service Callback Interface provides an easy way to handle incoming data from the HUD via the Service. Data received from the HUD is in a Message object where the data can be passed to a message handler on the Activity Thread. `msg.what` is a JSON formatted string of the response for a command or data sent by the HUD for processing, such as IMU data.

Below is the HUD Callback Interface. Notice that the onImage function is the only function with different handling requirements. The data passed to the application from the HUD is image data and the application can use that data as an image to be displayed. All other commands are identical in that they are formatted by command byte and JSON message. These functions have been separated out so that the developer can more easily target processing of the information.

```
private IHudServiceCallback mCallback = new IHudServiceCallback.Stub() {

    @Override
    public void onCamera(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onCamera Callback " + test);
```

```
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onAudio(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onAudio Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onPing(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onPing Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();

    }

    @Override
    public void onInfo(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onInfo Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onDisplay(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onDisplay Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onSettings(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onSettings Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onImuData(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onImuData Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onData(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onData Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onImage(ImageFrame imageFrame){
        if(mHandler != null)
            mHandler.obtainMessage((int)HC_IMAGE, imageFrame).sendToTarget();
```

```
    }

    @Override
    public void onJpeg(ByteFrame byteFrame){
        if(mHandler != null)
            mHandler.obtainMessage((int)HC_JPEG, byteFrame).sendToTarget();
    }

};
```

Below is an example handler. Since the interface callback is on the service thread it will not have access to the Activity while processing. The handler is used to perform the processing on the Activity thread.

```
/*
 * This handler will be passed to UsbService. Data received from serial port is displayed
through this handler
 */
private static class MyHandler extends Handler {
    private final WeakReference<MainActivity> mActivity;

    public MyHandler(MainActivity activity) {
        mActivity = new WeakReference<>(activity);
    }

    @Override
    public void handleMessage(Message msg) {
        if (HUD_CommandBYTE.forValue((byte) msg.what) == HC_IMAGE) {
            ImageFrame data = (ImageFrame) msg.obj;
            mActivity.get().responseImageHandler(data.imageBitmap);
        } else if(HUD_CommandBYTE.forValue((byte) msg.what) == HC_JPEG){
            ByteFrame data = (ByteFrame) msg.obj;
            mActivity.get().responseJpegHandler(data.jpgBuffer);
        }else{
            String data = (String) msg.obj;
            mActivity.get().responsePacketHandler(HUD_CommandBYTE.forValue((byte)
msg.what), data);
        }

    }

}
```

3.3.5 HUD Service

The HUD service API is discussed further in section 3.4. Below is the raw interface that provides the programmer access to the HUD Service capabilities.

```
package com.six15.hudservice;

import com.six15.hudservice.ImageFrame;
import com.six15.hudservice.ByteFrame;
```

```
import com.six15.hudservice.IHudServiceCallback;

interface IHudService {

     boolean isInited();
     void registerCallback(IHudServiceCallback cb);
      void unregisterCallback(IHudServiceCallback cb);
      boolean checkCallback(IHudServiceCallback cb);
      boolean setAutoResizeImage(boolean enable);
      boolean getAutoResizeEnabled();
      boolean getHudConnected();
      boolean getScreenCaptureEnabled();
      int getSplashScreenEnabled();
      int getHudDisplaySize();
      float getLastFrameBufferTime();
      int getHudVersions();
      int getHudInfo();
      int getHudSettings();
      int setHudSettings(boolean autoResize, boolean splashEnable, byte defBrightness);
      int setHudDefaultSettings();
      int getDisplayInfo();
      int getImuStatus();
      int startImu();
      int stopImu();
      int sendPing();
      int setDisplayOn(boolean onOff);
      int getDisplayOn();
      int setBrightnessLevel(byte level);
      int getBrightnessLevel();
      int getCameraSnapshot();
      int startCameraCapture();
      int stopCamera();
      boolean clearHudDisplay();
      boolean sendJpegToHud(String filePath);
      boolean sendImageToHud(in ImageFrame imageFrame);
      boolean sendBufferToHud(in ByteFrame byteFrame);
      int doQueryHudMode();
      int doHudSoftReset();
      boolean bootloaderUpdateModeLegacy();
      int doFirmwareUpdate();
      int doBootloaderUpdate();
      List<String> getAvailableFirmware();
      List<String> getAvailableBootloaders();
      String getSDKVersion();
      void setCameraEnabled(boolean cameraEnable);
      boolean getCameraEnabled();
      boolean getPresentationEnabled();
      String getFirmwareVersion();
      String getAvailableFirmwareVersion();
      boolean getMicrophoneEnabled();
      void setMicrophoneEnabled(boolean microphoneEnable);
      boolean setUserData(String data);
      String getUserData();
      String getHudInfromation();
      int startRawCameraCapture();
      int getHudOperationMode();
```

```
        boolean setHudOperationMode(int mode, boolean bStopOnly);
        boolean getCameraAutofocusMode();
        void setCameraAutofocusMode(boolean bAutofocusOn);
        List<CameraResolution> getSupportedCameraResolutions();
        CameraResolution getCurrentCameraResolution();
        boolean setCameraResolution(in CameraResolution cam_res);
        int getCameraZoom();
        boolean setCameraZoom(int zoom);

}
```

3.3.6 HUD Command BYTE Definition

The HUD command bytes are the command responses that will be returned for a specific call by the HUD Service. Use the command byte to determine the response type for your response handler.

```
HC_STATUS  ((byte)0x00),
HC_VERSIONS ((byte)0x01),
HC_DEV_INFO ((byte)0x02),
HC_DEV_SETTINGS ((byte)0x03),
HC_VERSIONS_EXTRA ((byte)0x04),
HC_MODE_UPD ((byte)0x05),
HC_MODE_QUERY ((byte)0x0A),
HC_MODE_SRST ((byte)0x0F),
HC_DISP_SIZE ((byte)0x20),
HC_DISP_BRT ((byte)0x22),
HC_DISP_ON ((byte)0x23),
HC_DISP_INFO ((byte)0x2F),
HC_CAM_START ((byte)0x30),
HC_CAM_SNAP ((byte)0x35),
HC_CAM_STOP ((byte)0x3A),
HC_CAM_INFO ((byte)0x3F),
HC_CFG_SPEN ((byte)0x40),
HC_CFG_SPDEL ((byte)0x41),
HC_CFG_SPIMAGE ((byte)0x42),
HC_CFG_AUTRESIZE ((byte)0x43),
HC_CFG_BL ((byte)0x44),
HC_CFG_APP ((byte)0x45),
HC_CFG_OEM ((byte)0x46),
HC_CFG_SETTINGS ((byte)0x47),
HC_CFG_RESET ((byte)0x4F),
HC_IMU_DATA ((byte)0x50),
HC_IMU_STATUS ((byte)0x51),
HC_IMU_START ((byte)0x52),
HC_IMU_STOP ((byte)0x5F),
HC_AUD_INFO ((byte)0x6F),
HC_DEV_METRICS ((byte)0x80),
HC_DEV_DEBUG_TERMINAL ((byte)0x81),
HC_IMAGE ((byte)0xD0),
HC_JPEG((byte)0xDE),
HC_RAW ((byte)0xDD),
HC_ERROR ((byte)0xEE),
HC_HEART_BEAT ((byte)0xFF);
```

3.4 HUD Service API Definitions

3.4.1 Information and Hardware Functions

The HUD service capabilities are exposed by the functions in the HUD service interface. All get functions return a result to determine status of the call as an integer.

Integer return values are functions that will asynchronously send data back to the application with -1 indicating that sending the command to the HUD failed, -2 indicates that the function is not supported or disabled in the service. All calls with an integer return type that return a value greater than or equal to zero are calls that were made successfully. When calling integer return type functions in the API, there can be additional data received in the form of a JSON string asynchronously. The developer should implement the Callback interface and check for data sent by the Service. When JSON data is expected the ASYNC JSON Data describes what data is expected. All integer commands can return a generic user message as below to be human readable and displayed to the user or logged:

```
{
	"msg":"Erased splash image not deleted"
}
```

3.4.1.1 Service Initialized

Function Prototype: `boolean isInited()`

Parameters: None

Return: boolean

Description:

Retrieve if the HUD service has been initialized. True indicates that it has and false has indicated that it has not.

ASYNC JSON Data: None

3.4.1.2 Register Service Callback

Function Prototype: `void registerCallback(IHudServiceCallback cb)`

Parameters: IHudServiceCallback

Return: void

Description:

Used to register your interface implementation as the callback to be called by the HUD service.

ASYNC JSON Data: None

3.4.1.3 Unregister Service Callback

Function Prototype: `void unregisterCallback(IHudServiceCallback cb)`

Parameters: IHudServiceCallback

Return: void

Description:

Used to unregister your interface implementation as the callback to be called by the HUD service.

ASYNC JSON Data: None

3.4.1.4 Check Service Callback

Function Prototype: `boolean checkCallback(IHudServiceCallback cb)`

Parameters: IHudServiceCallback

Return: boolean

Description:

Used to check if your interface implementation has been registered as a callback with the HUD service. True indicates that it is already registered, false indicates that it is not.

ASYNC JSON Data: None

3.4.1.5 Get HUD Connected

Function Prototype: `boolean getHudConnected ()`

Parameters: None

Return: boolean

Description:

Used to check if a HUD is connected to the Android device. Returns true when a HUD is connected to the Android's USB port.

ASYNC JSON Data: None

3.4.1.6 Get SDK Version

Function Prototype: `String getSDKVersion ()`

Parameters: None

Return: String

Description:

Retrieve the SDK version information in the form of "x.xx.xx"

ASYNC JSON Data: None

3.4.1.7 Get HUD Information

Function Prototype: `int getHudInfo ()`

Parameters: None

Return: Integer

Description:

Retrieves all HUD information.

Command BYTE: HC_DEV_INFO

ASYNC JSON Data:

```
{
	"man_id":"SIX15",
	"prod_id":"1234",
	"prod_name":"ST1",
	"device_id":"SD1234",
	"serial_num":"1212121",
	"hw_ver":"x.xx.xx",
	"bl_ver":"x.xx.xx",
	"fw_ver":"x.xx.xx"
}
```

3.4.1.8 Get HUD Information Fast

Function Prototype: `String getHudInfromation ()`

Parameters: None

Return: String

Description:

Retrieves all HUD information immediately and returns JSON string.

```
{
	"man_id":"SIX15",
	"prod_id":"1234",
	"prod_name":"ST1",
	"device_id":"SD1234",
	"serial_num":"1212121",
	"hw_ver":"x.xx.xx",
```

```
	"bl_ver":"x.xx.xx",
	"fw_ver":"x.xx.xx"
}
```

Command BYTE: HC_DEV_INFO

ASYNC JSON Data: NONE

3.4.1.9 Get HUD Version Information

Function Prototype: `int getHudVersions()`

Parameters: None

Return: Integer

Description:

Retrieves HUD versioning information for hardware, bootloader, and firmware

Command BYTE: HC_VERSIONS

ASYNC JSON Data:

```
{
	"hw_ver":"x.xx.xx",
	"bl_ver":"x.xx.xx",
	"fw_ver":"x.xx.xx"
}
```

3.4.1.10 Get HUD Settings

Function Prototype: `int getHudSettings ()`

Parameters: None

Return: Integer

Description:

Retrieves HUD settings information.

Command BYTE: HC_DEV_SETTINGS

ASYNC JSON Data:

```
{
	"width":640,
	"height":400,
	"resize_on":"false",
	"splash_en":"true",
	“mic_en”:”true”,
```

```
        "bright_def":2
}
```

3.4.1.11 Set HUD Settings Defaults

Function Prototype: `int setHudDefaultSettings ()`

Parameters: None

Return: Integer

Description:

Resets all settings to default values.

Command BYTE: HC_DEV_SETTINGS

ASYNC JSON Data:

```
{
        "width":640,
        "height":400,
        "resize_on":"false",
        "splash_en":"true",
        “mic_en”:”true”,
        "bright_def":2
}
```

3.4.1.12 Set HUD Settings

Function Prototype: `int setHudSettings(boolean autoResize, boolean splashEnable, byte defBrightness)`

Parameters: autoResize, splashEnable, defBrightness

Return: Integer

Description:

Allows application to save settings to the HUD device that will persist across power cycles.

Command BYTE: HC_DEV_SETTINGS

ASYNC JSON Data:

```
{
        "width":640,
        "height":400,
        "resize_on":"false",
        "splash_en":"true",
        “mic_en”:”true”,
```

```
        "bright_def":2
}
```

3.4.1.13 Ping Command

Function Prototype: `int sendPing ()`

Parameters: None

Return: Integer

Description:

Simple command response that performs a heartbeat style test.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.1.14 Query Device Mode

Function Prototype: `int doQueryHudMode ()`

Parameters: None

Return: Integer

Description:

Query device mode application or bootloader. Returns negative value on failure to send command. When command is sent successfully a response from the HUD will send json data back. Mode value is “bl” or “app”.

Command BYTE: NONE

ASYNC JSON Data:

```
{
        " mode":" bl "
}
```

3.4.1.15 Soft Reset

Function Prototype: `int doHudSoftReset ()`

Parameters: None

Return: Integer

Description:

Soft reset of HUD hardware.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.2 Display Functions

3.4.2.1 Get HUD Display Size

Function Prototype: `int getHudDisplaySize ()`

Parameters: None

Return: Integer

Description:

Retrieve HUD display width and height in pixels.

Command BYTE: HC_DISP_SIZE

ASYNC JSON Data:

```
{
	"width":640,
	"height":400
}
```

3.4.2.2 Get Auto Resize of Image

Function Prototype: `boolean getAutoResizeEnabled ()`

Parameters: None

Return: boolean

Description:

Returns true if images will be resized to fit the display automatically. Resizing only occurs when images are larger than the display width and/or height.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.2.3 Set Auto Resize of Image Data

Function Prototype: `boolean setAutoResizeImage (boolean enable)`

Parameters: enable

Return: boolean

Description:

Set auto resize of images for current session, not persisting.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.2.4 Get Last Frame Send Time

Function Prototype: `float getLastFrameBufferTime ()`

Parameters: None

Return: float

Description:

Retrieves the elapsed time for transcoding and sending the image to the HUD.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.2.5 Get Display Information

Function Prototype: `int getDisplayInfo ()`

Parameters: None

Return: Integer

Description:

Retrieves the full display information.

Command BYTE: HC_DEV_INFO

ASYNC JSON Data:

```
{
	"width":640,
	"height":400,
	"display_on":"true",
	"bright_lvl":3
}
```

3.4.2.6 Get Display Status On/Off

Function Prototype: `int getDisplayOn ()`

Parameters: None

Return: Integer

Description:

Returns true if the HUD display is on.

Command BYTE: HC_DISP_ON

ASYNC JSON Data:

```
{
	"disp_on":"true"
```

```
}
```

3.4.2.7 Set Display Status On/Off

Function Prototype: `int setDisplayOn(boolean onOff)`

Parameters: onOff

Return: Integer

Description:

Set HUD display on or off, display status will be sent back from the HUD. True will turn display on and false will turn the display off.

Command BYTE: HC_DISP_ON

ASYNC JSON Data:

```
{
	"disp_on":"true"
}
```

3.4.2.8 Get Display Brightness Level

Function Prototype: `int getBrightnessLevel ()`

Parameters: None

Return: Integer

Description:

Returns current brightness level of the HUD display. Values are integers from zero (0) to three (3), with three (3) being the highest.

Command BYTE: HC_DISP_BRT

ASYNC JSON Data:

```
{
	"bright_lvl":3
}
```

3.4.2.9 Set Display Brightness Level

Function Prototype: `int setBrightnessLevel (byte level)`

Parameters: None

Return: Integer

Description:

Sets brightness level of the HUD display for current session. Returns current brightness level of the HUD display. Values are integers from zero (0) to three (3), with three (3) being the highest.

Command BYTE: HC_DISP_BRT

ASYNC JSON Data:

```
{
	"bright_lvl":3
}
```

3.4.2.10 Clear HUD Display

Function Prototype: `boolean clearHudDisplay ()`

Parameters: None

Return: Boolean

Description:

Clears HUD display. Returns true when the clear command was sent successfully and false when there is an error.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.2.11 Send JPEG file to HUD

Function Prototype: `boolean sendJpegToHud (String filePath)`

Parameters: filePath

Return: Boolean

Description:

Send a JPG file to the HUD directly. Developer must provide a string path to the JPEG file to send. Developer must also ensure that permissions where implicitly granted by the user for accessing storage on Android version 6.0 and later. Returns true when the JPEG formatted image data was sent successfully and false when there is an error. Data will be checked and verified.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.2.12 Send Android Bitmap to HUD

Function Prototype: `boolean sendImageToHud(in ImageFrame imageFrame)`

Parameters: ImageFrame

Return: Boolean

Description:

Send a bitmap to the HUD. Returns true when the bitmap data was sent successfully and false when there is an error. Developer must provide an ImageFrame wrapped

bitmap object to this function to martial the data to the remote service. Data will be checked and verified.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.2.13 Send Buffer to HUD

Function Prototype: `boolean sendBufferToHud (in ByteFrame byteFrame)`

Parameters: ImageFrame

Return: Boolean

Description:

Send a raw JPEG buffer to the HUD. Returns true when the buffer data was sent successfully and false when there is an error. Developer must provide a ByteFrame wrapped byte array to this function to martial the data to the remote service. This is provided as a performance enhancement and data is not checked or verified. This command requires that the developer ensure that the JPEG data is properly formatted for the HUD.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.3 IMU Functions

IMU functions are located on the HUD can asynchronously stream factory calibrated sensor data to the connected device in the form of JSON data. The sensor returns accelerometer, gyroscope, magnetometer, quaternion, and temperature data.

3.4.3.1 IMU Data

Function Prototype: None

Parameters: None

Return: None

Description:

The IMU sensor data is sent to the HUD every 33ms.

This data is sent via the HUD Service callback onImuData function. Data is in JSON format defined below. All floats are carried out to a maximum of three decimal places except for temperature which is two decimal places.

Accelerometer data is in the form of a float vector consisting of X, Y, Z values signified by the A element.

Gyroscope data is in the form of an array of floats X, Y, Z values signified by the G element.

Magnetometer data is in the form of an array of floats X, Y, Z values signified by the M element.

A Quaternion vector is provided signified by the Q element and consists of X, Y, Z, W values.

Temperature is in the format of degrees Celsius and is signified by the T element in the data structure and consists of a single float value.

ASYNC JSON Data:

```
{
	"A":[%.3f,%.3f,%.3f],
	"G":[%.3f,%.3f,%.3f],
	"M":[%.3f,%.3f,%.3f],
	"Q":[%.3f,%.3f,%.3f,%.3f],
	"T":%.2f
}
```

3.4.3.2 IMU Status

Function Prototype: `Integer getImuStatus ()`

Parameters: None

Return: Integer

Description:

Return the status of the IMU streaming, whether it is on or off.

Command BYTE: NONE

ASYNC JSON Data:

```
{
	"imu_on":"true"
}
```

3.4.3.3 Start IMU

Function Prototype: `Integer startImu ()`

Parameters: None

Return: Integer

Description:

Starts the IMU data stream and asynchronously returns a JSON message indicating the state of the IMU data stream.

Command BYTE: NONE

ASYNC JSON Data:

```
{
```

```
        "imu_on":"true"
}
```

3.4.3.4 Stop IMU

Function Prototype: `Integer stopImu ()`

Parameters: None

Return: Integer

Description:

Stop the IMU data Stream, and asynchronously returns JSON message indicating the state of the IMU data stream.

Command BYTE: NONE

ASYNC JSON Data:

```
{
        "imu_on":"false"
}
```

3.4.3.5 Start User IMU Cal

Function Prototype: `bool startUserImuCal(int imuSensor);`

Parameters: int imuSensor

Return: bool

Description:

Before using this method, make sure the IMU data Stream has been started using startImu() (3.4.3.3), otherwise the calibration will not be able to complete. This starts the process of calibrating the gyroscope and MAG BIAS in the HUD. Only way to get status is by calling get status function.

During the calibration process, make sure the HUD is stationary while the gyroscope is being calibrated and moving while MAG BIAS. It is important that the MAG BIAS rotates slowly in three different directions: forward over itself, pivoting around itself, and rotating end over end.

3.4.3.6 Get User IMU Cal

Function Prototype: `bool getUserImuCal(int imuSensor)`

Parameters: int imuSensor

Return: bool

Description:

Get the status of the IMU's calibration of the system corresponding to the given byte value. In the asynchronous JSON that is returned there are three indexes: imu_en, cal_status, and cal_error.

Imu_en is the current status of the imu. If the imu is connected and enabled this should have a value of one. Otherwise, this will have a value of zero.

Cal_status is what the system is currently looking at. While calibrating the gyroscope it will have a value of two and for MAG BIAS it will have a value of three. Once the calibration is complete this value will become zero.

Cal_error will read zero if there are no errors.

ASYNC JSON Data:

```
{
	"imu_en":1,
	"cal_status":3,
	"cal_error":0
}
```

3.4.4 HUD Camera Functions

3.4.4.1 Camera Field of View Measurements

The camera field of view is given in the table below.

Aspect Ratio	Supported Resolutions	Field of View	Tolerance
4:3	640 x 480 2592 x 1944 (5MP)	65.9°	±1°
16:9	1280 x 720 (720p) 1920 x 1080 (1080p) 2560 x 1440 (QHD)	61.6°	±1°

3.4.4.2 Set HUD Camera Enabled

Function Prototype: `void setCameraEnabled(boolean cameraEnable)`

Parameters: cameraEnable

Return: void

Description:

Enables or disables the camera feature.

Command BYTE: NONE

ASYNC JSON Data: None

NOTE: This feature is not available in the 2.03 SDK release. Contact info@six-15.com for details on this feature.

3.4.4.3 Get HUD Camera Enabled

Function Prototype: `boolean getCameraEnabled ()`

Parameters: void

Return: boolean

Description:

Returns whether or not the HUD camera is enabled(true) or disabled(false).

Command BYTE: NONE

ASYNC JSON Data: None

NOTE: This feature is not available in the 2.03 SDK release. Contact info@six-15.com for details on this feature.

3.4.4.4 Set Camera Autofocus Mode

Function Prototype: `void setCameraAutofocusMode (boolean bAutofocusOn)`

Parameters: bAutofocusOn

Return: void

Description:

Enable or disable camera autofocus. Setting bAutofocusOn to true turns autofocus on and false turns autofocus off.

Command BYTE: NONE

ASYNC JSON Data: None

NOTE: This feature is not available in the 2.03 SDK release. Contact info@six-15.com for details on this feature.

3.4.4.5 Get Camera Autofocus Mode

Function Prototype: `boolean getCameraAutofocusMode ()`

Parameters: void

Return: boolean

Description:

Returns whether or not the HUD camera has autofocus enabled(true) or disabled(false).

Command BYTE: NONE

ASYNC JSON Data: None

NOTE: This feature is not available in the 2.03 SDK release. Contact info@six-15.com for details on this feature.

3.4.4.6 Start Camera

Function Prototype: `Boolean startCameraCapture ()`

Parameters: None

Return: Boolean

Description:

Start camera streaming. Function will decode the JPEG data to an Android image and send each frame as an image to the application.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.4.7 Start Camera (Raw JPEG Mode)

Function Prototype: `Boolean startRawCameraCapture ()`

Parameters: None

Return: Boolean

Description:

Start camera streaming and raw JPEG frames will be sent to the application from HUD service via onJpeg. The data sent is taken from the motion-JPEG stream on the UVC camera.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.4.8 Stop Camera

Function Prototype: `Boolean stopCamera ()`

Parameters: None

Return: Boolean

Description:

Stop camera streaming.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.4.9 Get Supported Resolutions

Function Prototype: `List<CameraResolution> getSupportedCameraResolutions ()`

Parameters: None

Return: List<CameraResolution>

Description:

Only supported on SDK 2.0 and above (5.xx firmware). Returns a list of supported camera resolutions.

Command BYTE: NONE

ASYNC JSON Data: NONE

3.4.4.10 Get Current Camera Resolution

Function Prototype: `CameraResolution getCurrentCameraResolution()`

Parameters: None

Return: CameraResolution

Description:

Only supported on SDK 2.0 and above (5.xx firmware). Returns the current camera resolution.

Command BYTE: NONE

ASYNC JSON Data: NONE

3.4.4.11 Set Current Camera Resolution

Function Prototype: `boolean setCurrentCameraResolution(CameraResolution cam_res)`

Parameters: CameraResolution

Return: boolean

Description:

Only supported on SDK 2.0 and above (5.xx firmware). Changes the current camera resolution, returns true on success and false on failure.

Command BYTE: NONE

ASYNC JSON Data: NONE

3.4.4.12 Get Camera Zoom

Function Prototype: `int getCameraZoom ()`

Parameters: None

Return: int

Description:

Only supported on SDK 2.0 and above (5.xx firmware). Returns the current camera zoom setting with zero(0) being no zoom and any positive integer being the level of zoom. Supported zoom levels can be found in the cameraresolution object.

Command BYTE: NONE

ASYNC JSON Data: NONE

3.4.4.13 Set Camera Zoom

Function Prototype: `boolean setCameraZoom(int zoom)`

Parameters: int

Return: boolean

Description:

Only supported on SDK 2.0 and above (5.xx firmware). Sets the camera zoom setting with to the desired zoom level. Returns true on success and false on failure. Supported zoom levels can be found in the cameraresolution object.

Command BYTE: NONE

ASYNC JSON Data: NONE

3.4.5 HUD Microphone Functions

3.4.5.1 Get HUD Microphone Enabled

Function Prototype: `boolean getMicrophoneEnabled ()`

Parameters: NONE

Return: boolean

Description:

Get whether or not the HUD Microphone is enabled or disabled.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.5.2 Set HUD Microphone Enabled

Function Prototype: `void setMicrophoneEnabled (boolean microphoneEnable)`

Parameters: microphoneEnable

Return: void

Description:

Enable or disable the microphone feature.

This is used if Android system uses the HUD microphone as the default microphone. when disabled, the Android system will use the normal default microphone.

If the microphone state is changed in the HUD, the HUD will restart so the new USB descrinptiors will be sent. The set of microphone enable is stored in non-volatile memory in the HUD, it does not need to be called everytime an app is started.

Command BYTE: NONE

ASYNC JSON Data: None

3.4.6 HUD User Data Functions

The HUD allows the application to store data in the HUD devices non-volatile memory. There is 256K bytes available for the developer. The only requirement is that the data must be stored in a string.

3.4.6.1 Get User Data

Function Prototype: `String getUserData()`

Parameters: NONE

Return: String

Description:

Gets user data string. Typically developers will use a JSON string. This is an asynchronous call, the data stored in the HUD will be sent from the HUD to the application.

Command BYTE: NONE

ASYNC JSON Data: Returns JSON string of user data stored on HUD device.

3.4.6.2 Set User Data

Function Prototype: `boolean setUserData (String data)`

Parameters: data

Return: boolean

Description:

Sets user data string. Typically developers will use a JSON string.

Command BYTE: NONE

ASYNC JSON Data: Returns a JSON string indicating success or failure of saving user data.

3.4.7 Firmware Update

3.4.7.1 Get Current Firmware Version

Function Prototype: `String getFirmwareVersion ()`

Parameters: None

Return: String

Description:

Retrieves version of firmware currently running in HUD. In the format “x.xx”.

Command BYTE: NONE

ASYNC JSON Data: None

4. Six15 Getting Started

4.1 Getting Started Overview

To get started with your first Android project utilizing the Six15HUD you will need to create a new project or use an existing project and add the Six15HUD AIDL to the project that will be used.

Once the interface definition is added to the project the HUD service needs to be installed prior to running your application on an Android device.

4.2 Creating a new Android application

In this section we will walk through setting up an android project and adding the Six15 library.

First thing is to ensure that Android Studio is installed on the device you will be programming on. It can be installed free of charge from:

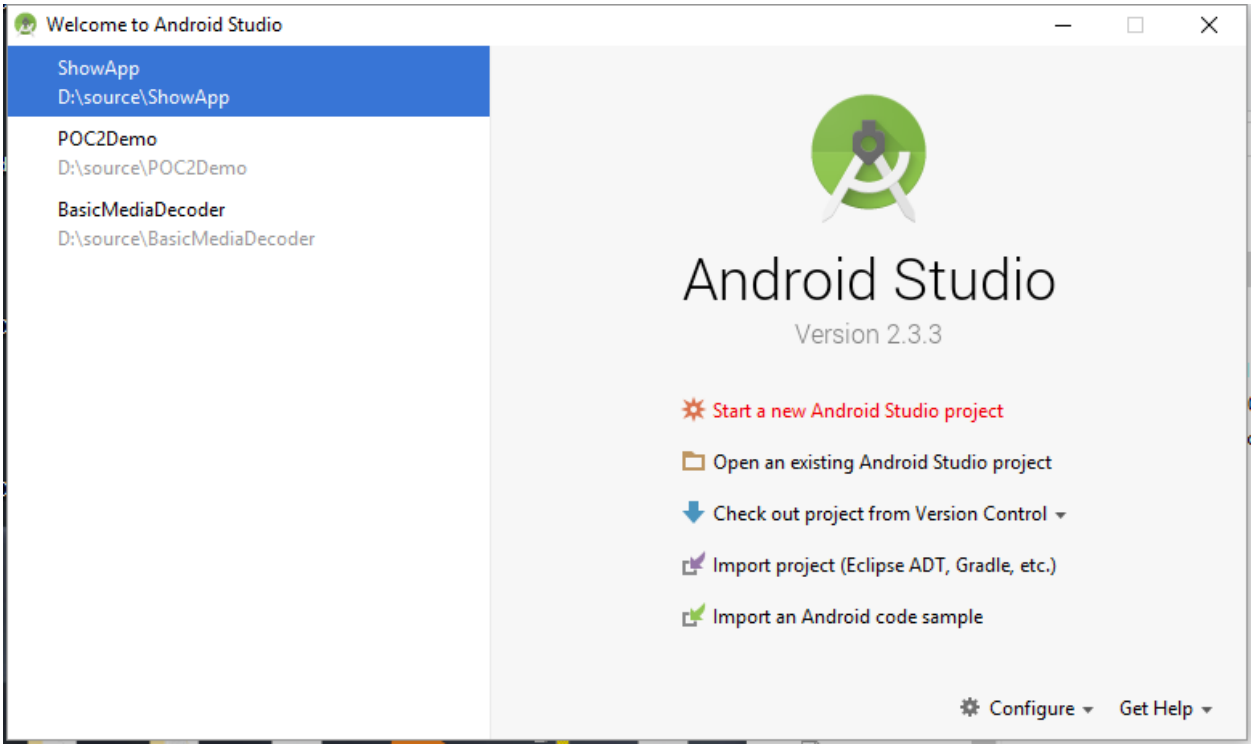
https://developer.android.com/studio/

Information on installing Android Studio can be found here:

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

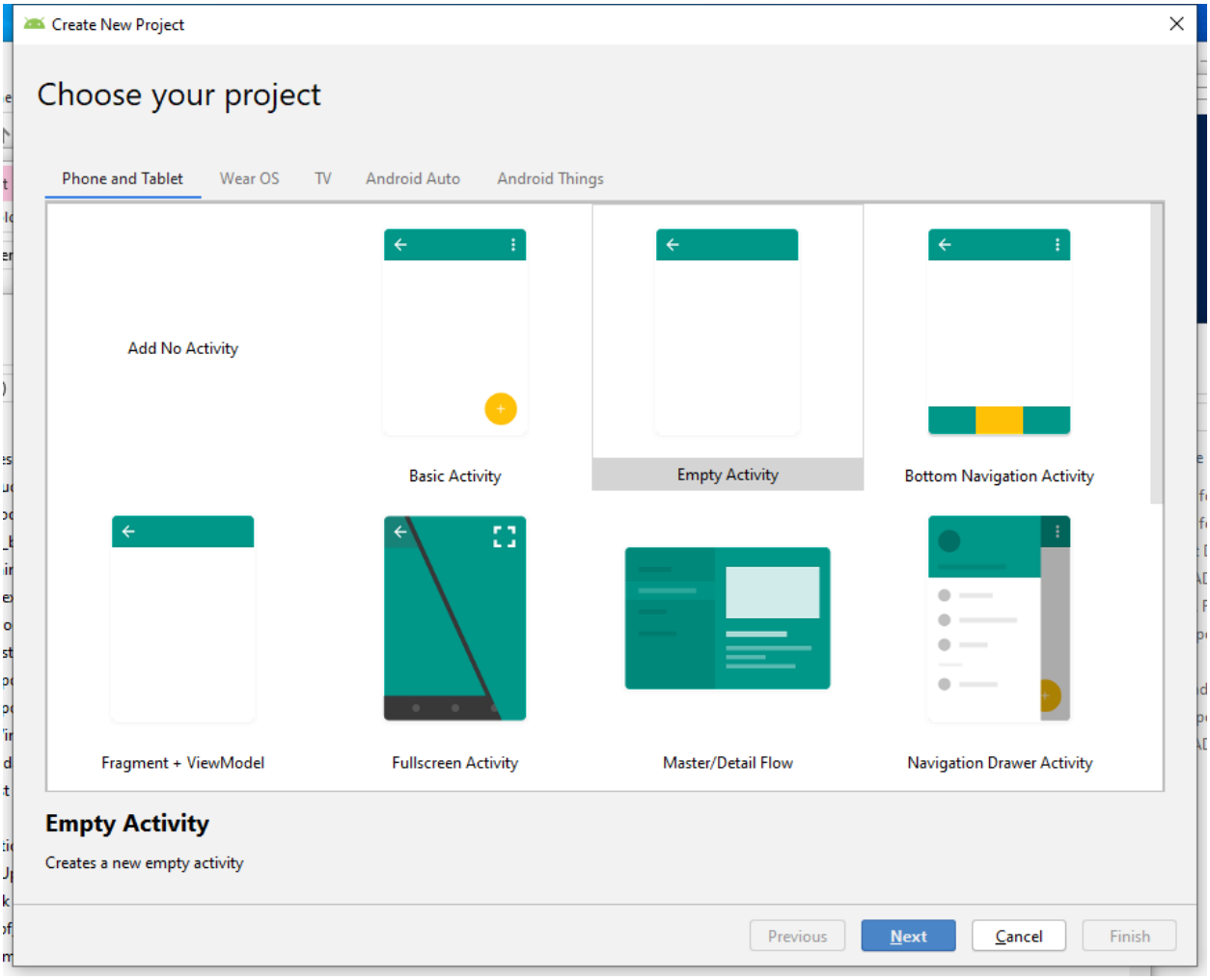
4.2.1 Create new Android Studio project

Launch Android Studio and select Start a New Android Studio project.



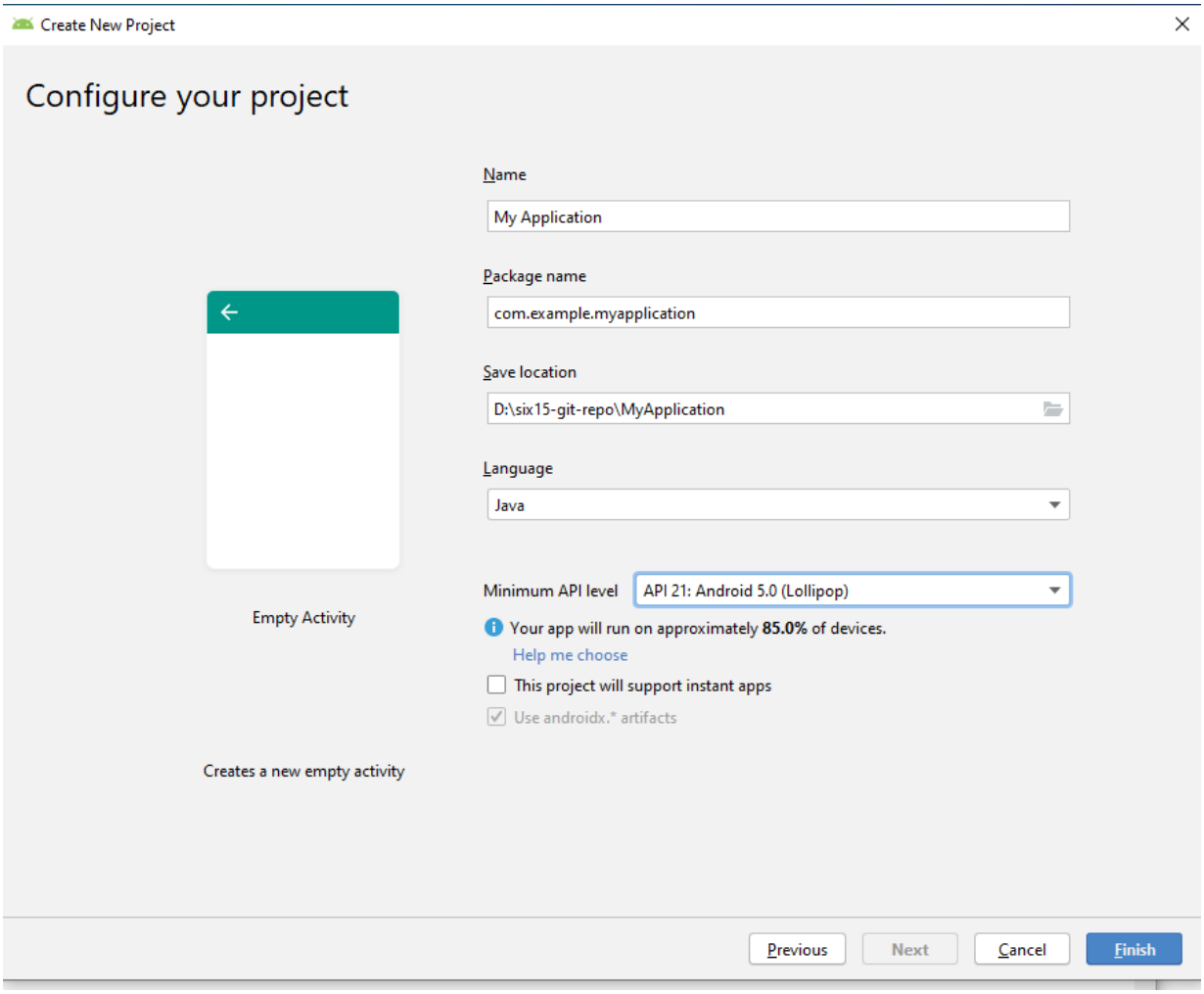
Select a type of project and click next.

For this example, it will be Empty Activity.

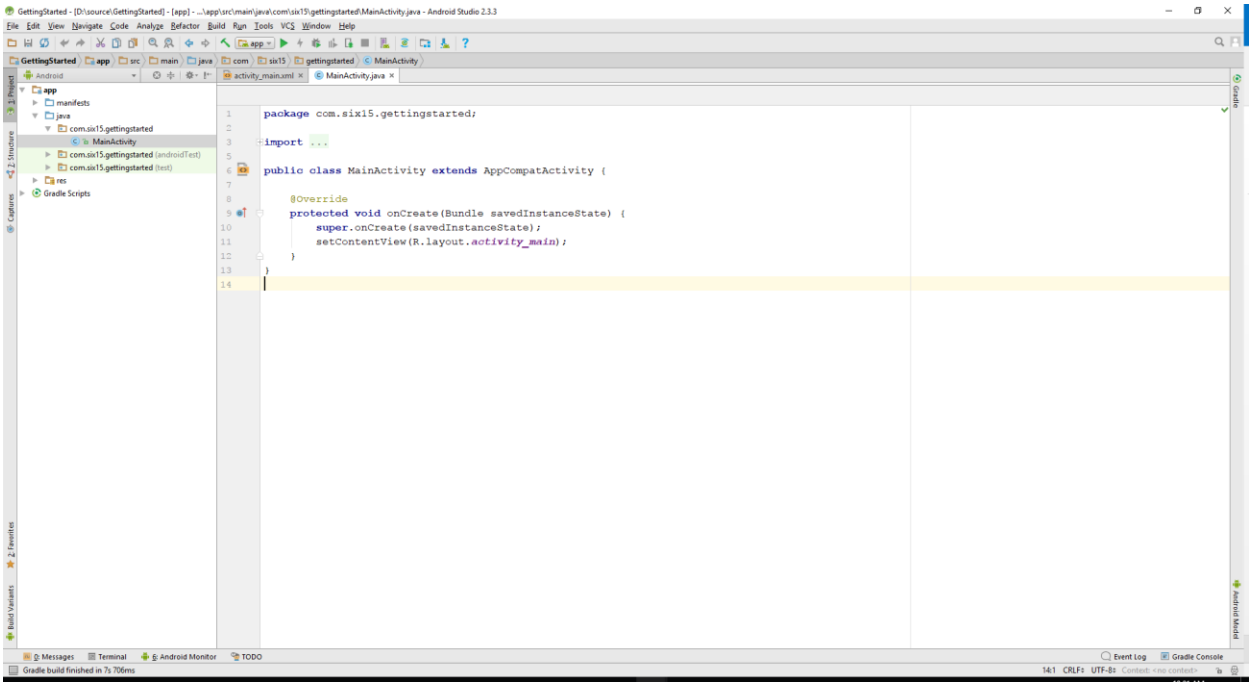

Create New Project
Choose your project
Phone and Tablet
Wear OS
TV
Android Auto
Android Things
Add No Activity
Basic Activity
Empty Activity
Bottom Navigation Activity
Fragment + ViewModel
Fullscreen Activity
Master/Detail Flow
Navigation Drawer Activity
Empty Activity
Creates a new empty activity
Previous
Next
Cancel
Finish

Select a name for your project, language to use, and then click finish.

For this example, we are using java and the minimum SDK which is 21



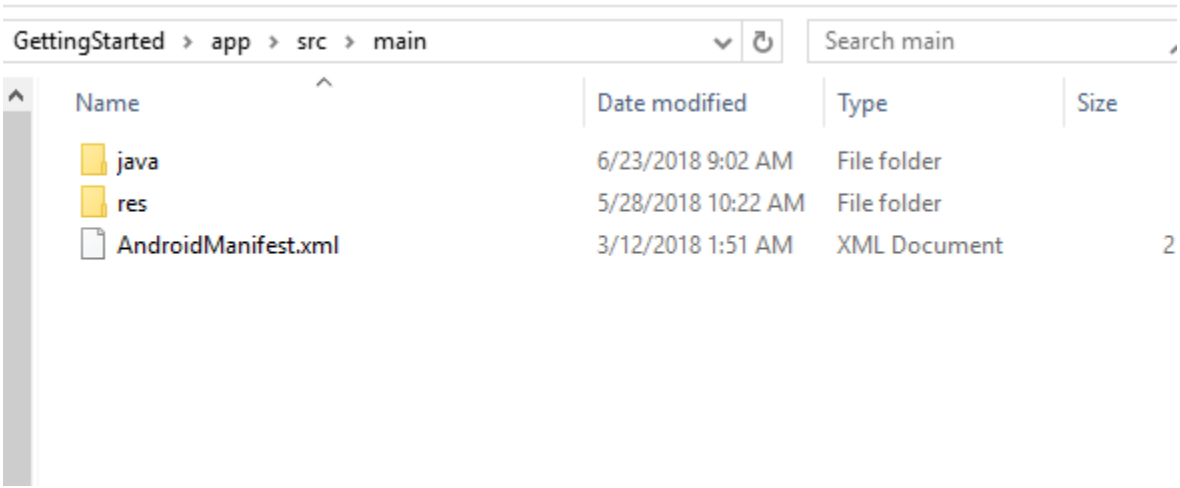
You should now be in your project.



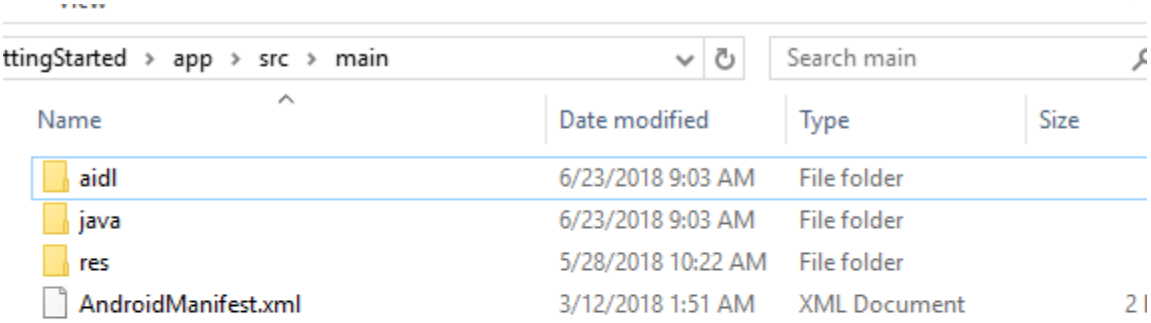
4.2.2 Add Six15 AIDL and Supporting Classes

In this section, we will add required interface and classes. Classes and interface definitions can be found in the Six15_HUD_Service_AIDL_v5.34.zip. Navigate to your source directory and enter your main directory where the AndroidManifest.xml resides. Unzip the contents of the Six15_HUD_Service_AIDL_v5.34.zip to this directory.

Navigate to main folder of project source and extract zip file.

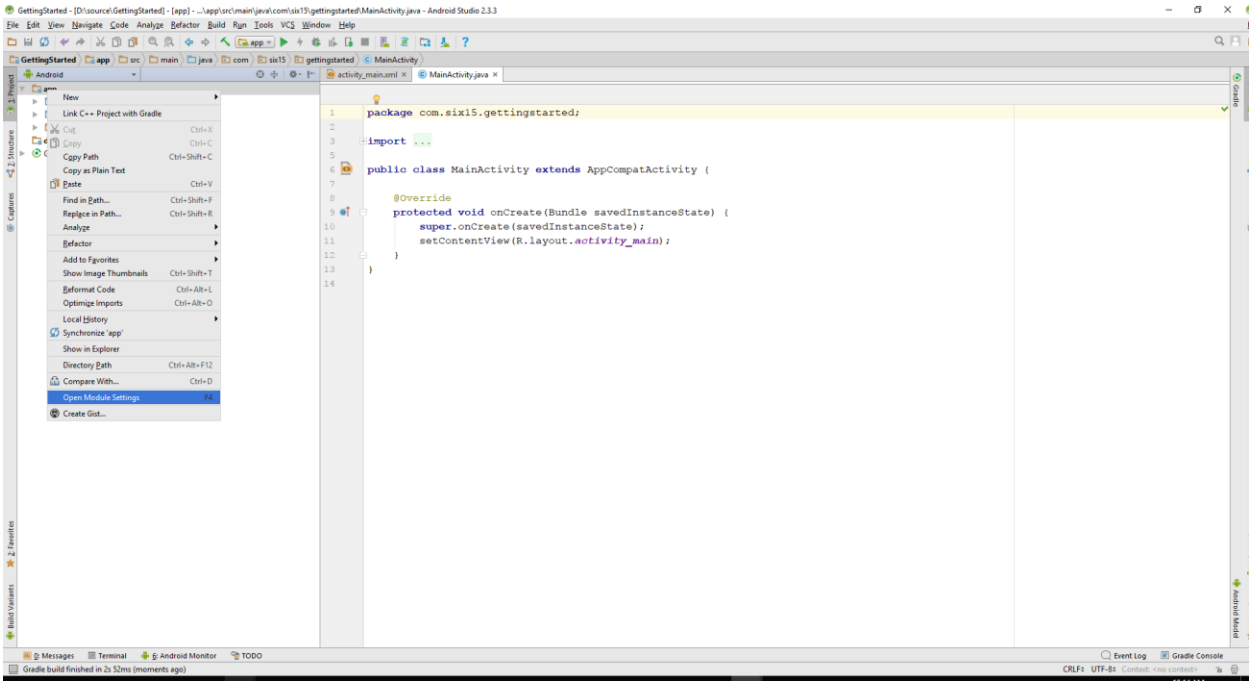


After extraction of the zip file content, the directory will look like the image below. It should now contain the aidl folder as well as six15 classes in the java directory.

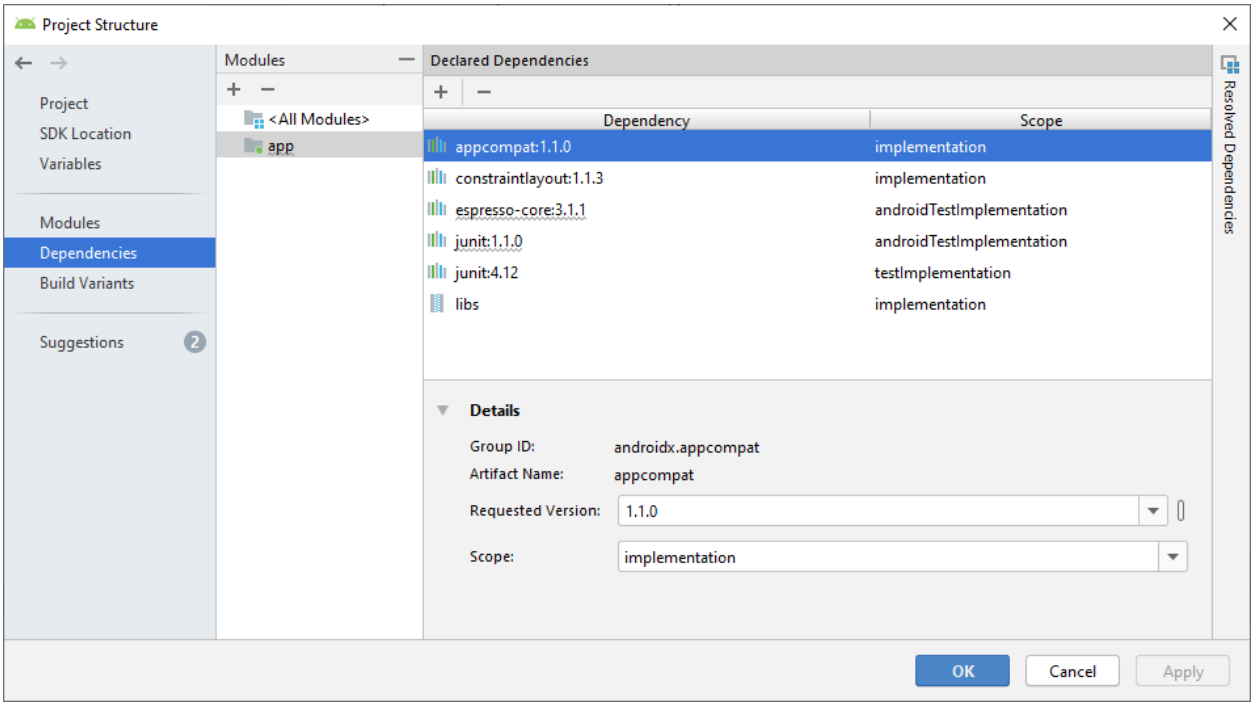


Now to add the GSON dependency for the included source.

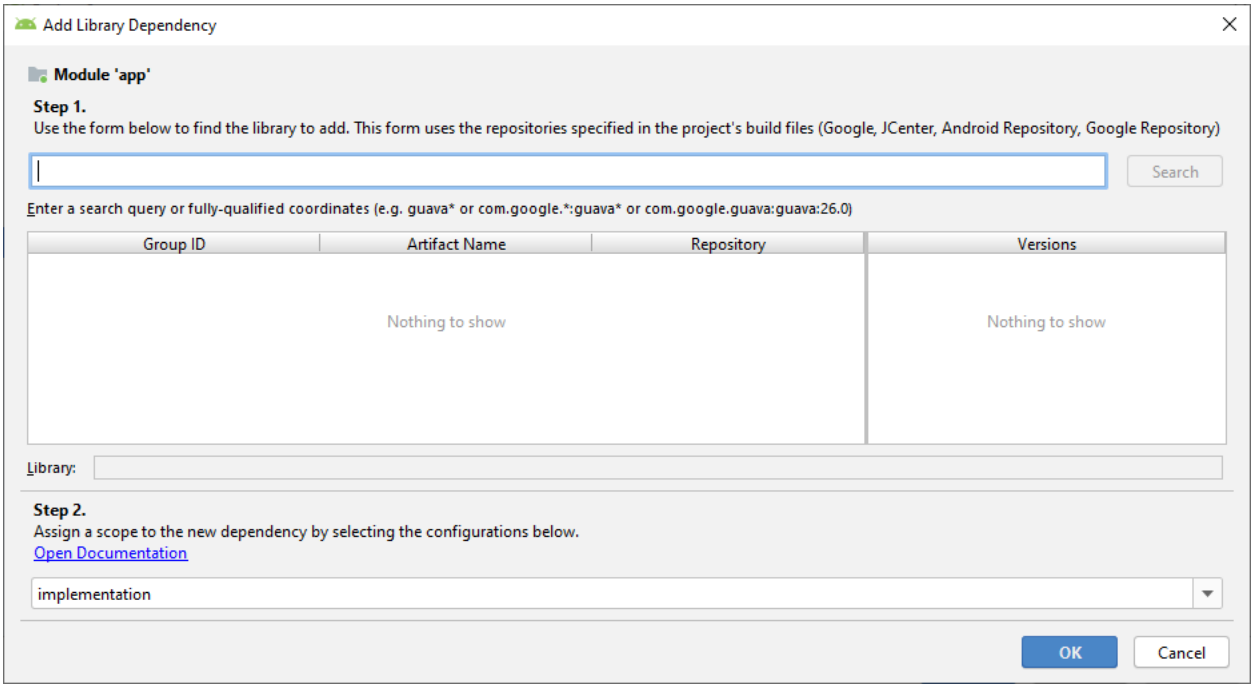
Right Click on project and select Open Module Settings.



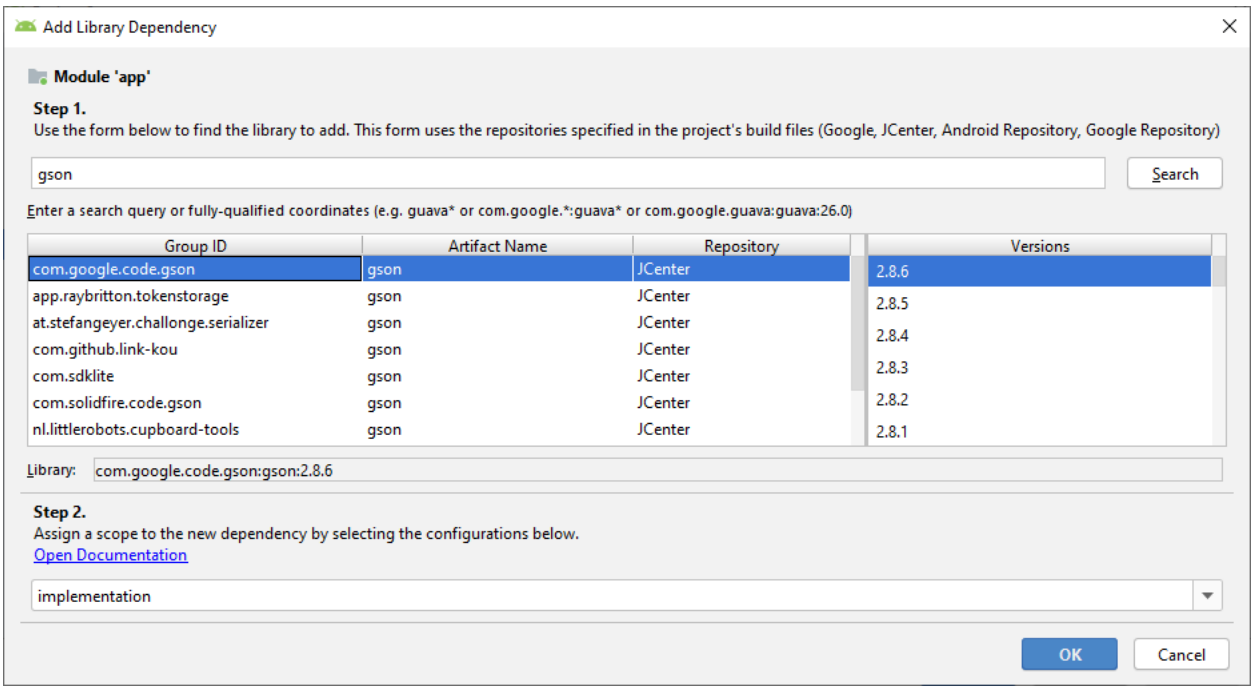
Select the Dependencies tab.



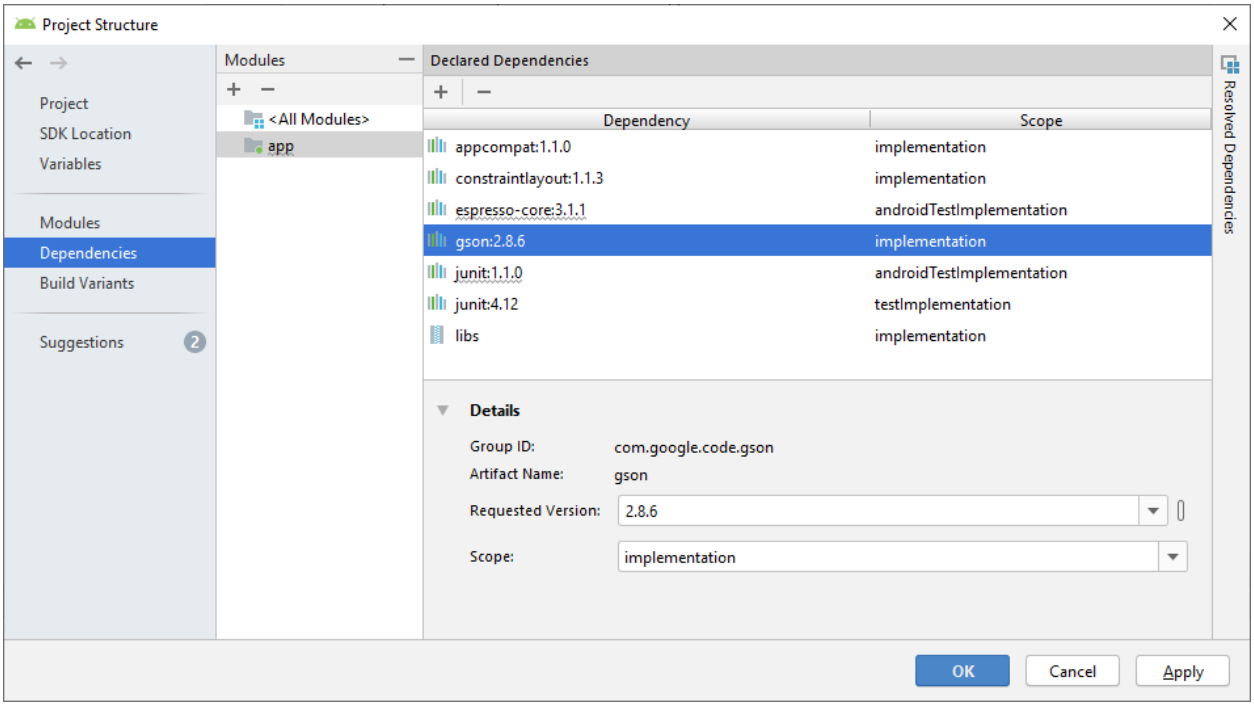
Click the plus sign and select Library dependency. Enter “gson” and click search.



Select the highlighted library and click OK.

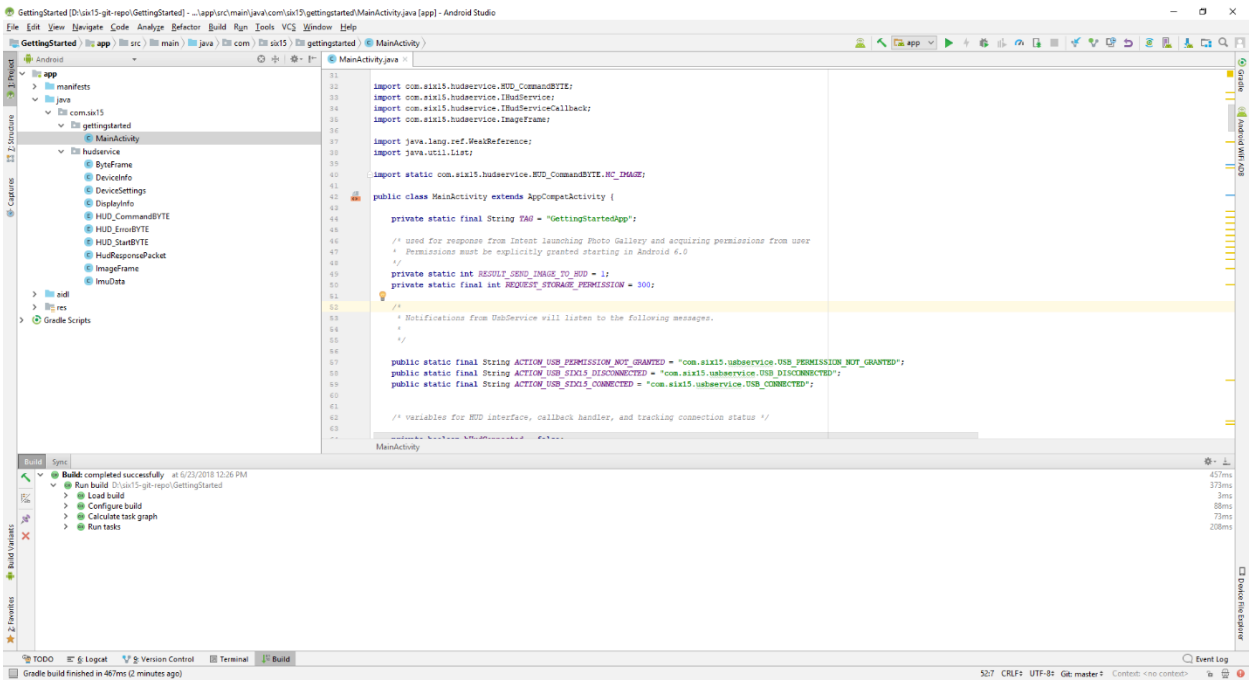


Click OK to return to IDE.



Add the following import statements to your activity.

```
import com.six15.hudservice.HUD_CommandBYTE;
import com.six15.hudservice.IHudService;
import com.six15.hudservice.IHudServiceCallback;
import com.six15.hudservice.ImageFrame;
import android.os.Bundle;
import android.util.Log;
import android.content.ComponentName;
import android.content.Intent;
import java.lang.ref.WeakReference;
import android.os.Message;
import static com.six15.hudservice.HUD_CommandBYTE.HC_IMAGE;
import android.graphics.Bitmap;
import android.view.Display;
```



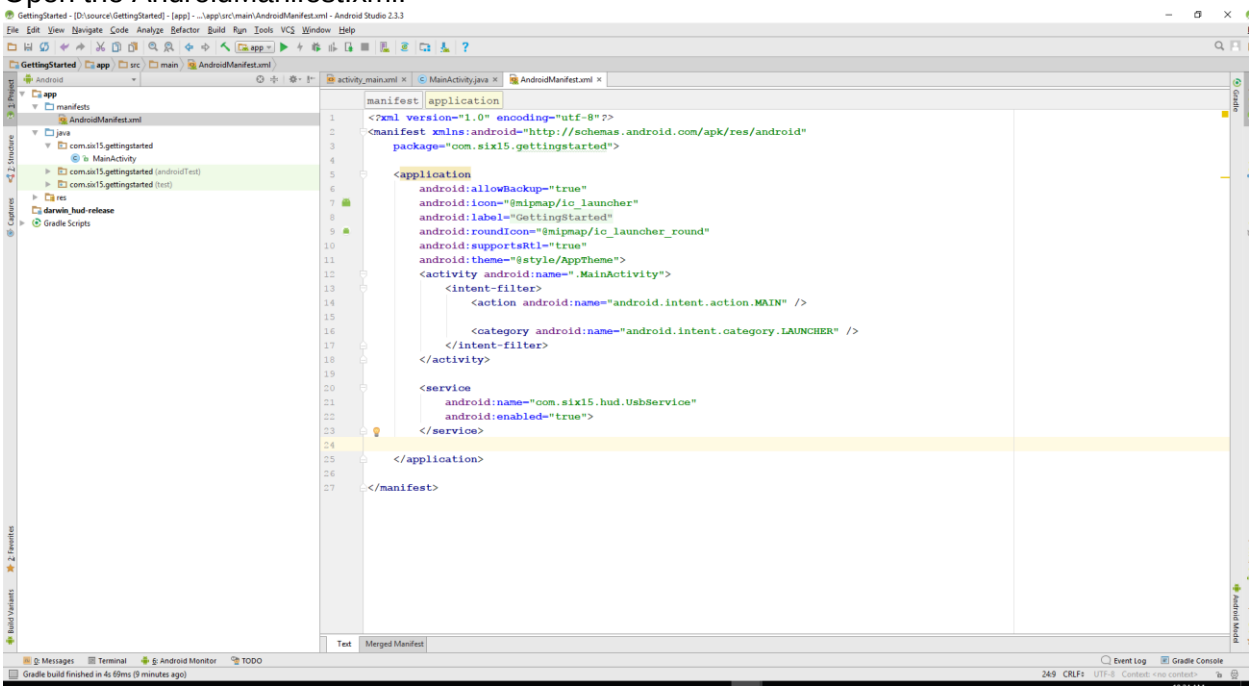
Afterwords, add the following variables.

```
public static final String ACTION_USB_PERMISSION_NOT_GRANTED =
        "com.six15.usbservice.USB_PERMISSION_NOT_GRANTED";
public static final String ACTION_USB_SIX15_DISCONNECTED =
        "com.six15.usbservice.USB_DISCONNECTED";
public static final String ACTION_USB_SIX15_CONNECTED =
        "com.six15.usbservice.USB_CONNECTED";
private MyHandler mHandler;
private boolean bBounded = false;
private static final String TAG = “Six15 Example”;
```

4.2.3 Add Features and Permissions

We will now add permissions for our application to access specific libraries on the Android Device. As of Android 6.0 permission must be explicitly granted by the user. This is demonstrated in the getting started source code.

Open the AndroidManifest.xml.



Any permissions that need to be added will be done in here.

Back in MainActivity.java, add Service and Service Connection.

```
private IHudServiceCallback mCallback = new IHudServiceCallback.Stub() {

    @Override
    public void onCamera(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onCamera Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onAudio(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onAudio Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onPing(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onPing Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();

    }

    @Override
    public void onInfo(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onInfo Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onDisplay(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onDisplay Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onSettings(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onSettings Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onImuData(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onImuData Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
```

```
    public void onData(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onData Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onImage(ImageFrame imageFrame){
        if(mHandler != null)
            mHandler.obtainMessage((int)0xD0, imageFrame).sendToTarget();
    }

    @Override
    public void onJpeg(ByteFrame byteFrame) throws RemoteException {
        if(mHandler != null)
            mHandler.obtainMessage((int)0xDE, byteFrame).sendToTarget();
    }

    @Override
    public void onDebugTerminal(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onDebugTerminal Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

};

private IHudService usbService = null;

/* unregister callback and unbind service */
private void exitService() {
    try {
        if(usbService != null)
        {
            Log.d(TAG, "Unregister Callback");
            usbService.unregisterCallback(mCallback);
        }
    } catch (RemoteException e) {
        e.printStackTrace();
    }

    if(usbConnection != null && bBounded) {
        unbindService(usbConnection);
    }
}

private void usbConnectDisplay(boolean enabled) {
    if (enabled) {
        lblStatus.setTextColor(Color.WHITE);
        lblStatus.setText(getString(R.string.usb_connected));
        bHudConnected = true;
        //setDisplayBrightness(4);
        try{
            //Need to set the brightness in brightnessValue to what is returned here.
            usbService.getBrightnessLevel();
            Log.d(TAG, "Brightness: "+usbService.getBrightnessLevel());
```

```
        } catch(RemoteException e){
            e.printStackTrace();
        }
    } else {
        lblStatus.setTextColor(Color.RED);
        lblStatus.setText(getString(R.string.usb_disconnected));
        if(tvValue != null)
            tvValue.setText("-".toCharArray(), 0, 1);
        bHudConnected = false;

        final Fragment currFrag = (Fragment)
getSupportFragmentManager().findFragmentById((R.id.fragment));
        if ( currFrag != null && (currFrag instanceof OnActivityInteractionListener))
{
            ((OnActivityInteractionListener) currFrag).onReceivedCommand(0);
        }
    }
}


private final ServiceConnection usbConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName arg0, IBinder arg1) {
        usbService = IHudService.Stub.asInterface(arg1);


        try {
            if(usbService.isInited()) {
                Log.d(TAG,"Register Callback");
                usbService.registerCallback(mCallback);

                if(!usbService.checkCallback(mCallback)){
                    Log.d(TAG,"Register Callback Second Try");
                    usbService.registerCallback(mCallback);
                }

                if(usbService.getHudConnected()){
                    usbConnectDisplay(true);
                }
            }
            else {
                Log.e(TAG,"Service not initialized");
                exitService();
            }
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onServiceDisconnected(ComponentName arg0) {
        Log.e(TAG, "Service has unexpectedly disconnected");
        usbConnectDisplay(false);
        usbService = null;
    }
};
```

Add HUD Service Connect and Disconnect to onPause and onResume.

```
private void setFilters() {
    IntentFilter filter = new IntentFilter();
    filter.addAction(ACTION_USB_PERMISSION_NOT_GRANTED);
    filter.addAction(ACTION_USB_SIX15_CONNECTED);
    filter.addAction(ACTION_USB_SIX15_DISCONNECTED);
    registerReceiver(mUsbReceiver, filter);
}

public static Intent createExplicitFromImplicitIntent(Context context, Intent
implicitIntent) {
    // Retrieve all services that can match the given intent
    PackageManager pm = context.getPackageManager();
    List<ResolveInfo> resolveInfo = pm.queryIntentServices(implicitIntent, 0);

    // Make sure only one match was found
    if (resolveInfo == null || resolveInfo.size() != 1) {
        return null;
    }

    // Get component info and create ComponentName
    ResolveInfo serviceInfo = resolveInfo.get(0);
    String packageName = serviceInfo.serviceInfo.packageName;
    String className = serviceInfo.serviceInfo.name;
    ComponentName component = new ComponentName(packageName, className);

    // Create a new intent. Use the old one for extras and such reuse
    Intent explicitIntent = new Intent(implicitIntent);

    // Set the component to be explicit
    explicitIntent.setComponent(component);

    return explicitIntent;
}

private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "Action received");
        switch (intent.getAction()) {
            case ACTION_USB_SIX15_CONNECTED: // USB PERMISSION GRANTED
                Toast.makeText(context, "USB Connected", Toast.LENGTH_SHORT).show();
                usbConnectDisplay(true);
                break;
            case ACTION_USB_PERMISSION_NOT_GRANTED: // USB PERMISSION NOT GRANTED
                Toast.makeText(context, "USB Permission not granted",
Toast.LENGTH_SHORT).show();
                usbConnectDisplay(false);
                break;
            case ACTION_USB_SIX15_DISCONNECTED: // USB DISCONNECTED
                Toast.makeText(context, "USB disconnected",
Toast.LENGTH_SHORT).show();
                usbConnectDisplay(false);
```

```
                break;
            default:
                Log.d(TAG, intent.getAction());
                break;
        }
    }
};

@Override
public void onPause() {
    super.onPause();
    /* unregister for USB events and unBind Service */
    unregisterReceiver(mUsbReceiver);
    if(usbConnection != null && bBounded) {
        unbindService(usbConnection);
    }
}

@Override
public void onResume() {
    super.onResume();
    /* set USB filter and Bind to Service */
    setFilters();
    Intent i = createExplicitFromImplicitIntent(this, new
Intent("com.six15.hudservice.SERVICE"));
    bindService(i, usbConnection, Context.BIND_AUTO_CREATE);
    bBounded = true;
}
```

4.2.4 Create Callback Interface and Handler

Create a handler to handle HUD responses on the Activity thread.

```
/*
 * This handler will be passed to UsbService. Data received from serial port is displayed
through this handler
 */
private static class MyHandler extends Handler {
    private final WeakReference<MainActivity> mActivity;

    public MyHandler(MainActivity activity) {
        mActivity = new WeakReference<>(activity);
    }

    @Override
    public void handleMessage(Message msg) {
        if (HUD_CommandBYTE.forValue((byte) msg.what) == HC_IMAGE) {
            ImageFrame data = (ImageFrame) msg.obj;
            mActivity.get().responseImageHandler(data.imageBitmap);
        } else {
            String data = (String) msg.obj;
            mActivity.get()
                  .responsePacketHandler(HUD_CommandBYTE.forValue((byte) msg.what), data);
        }
```

```
        }
}
```

Create the callback interface and response packet handlers.

```
/*
 * This function handles repsonse packets from the UsbService.
 */

public void responseImageHandler(Bitmap bmp){
    Log.d(TAG, "Image data not handled");
}

public void responsePacketHandler(HUD_CommandBYTE command_type, String data) {

    switch (command_type){
        case HC_HEART_BEAT:
            Toast.makeText(this, "Ping Response Received", Toast.LENGTH_LONG).show();
            break;
        default:
            Log.i(TAG, "Response not handled");
            Log.i(TAG, "Data Received: " + data);
            break;
    }

}



/**
 * This implementation is used to receive callbacks from the remote
 * service.
 */
private IHudServiceCallback mCallback = new IHudServiceCallback.Stub() {

    @Override
    public void onCamera(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onCamera Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onAudio(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onAudio Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onPing(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onPing Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();

    }

    @Override
    public void onInfo(byte cmd, String test) throws RemoteException {
```

```
        Log.d(TAG, "onInfo Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onDisplay(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onDisplay Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onSettings(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onSettings Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onImuData(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onImuData Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    @Override
    public void onData(byte cmd, String test) throws RemoteException {
        Log.d(TAG, "onData Callback " + test);
        if(mHandler != null)
            mHandler.obtainMessage((int)cmd, test).sendToTarget();
    }

    public void onImage(ImageFrame imageFrame){
        if(mHandler != null)
            mHandler.obtainMessage((int)0xD0, imageFrame).sendToTarget();
    }

    @Override
    public void onJpeg(ByteFrame byteFrame){
        if(mHandler != null)
            mHandler.obtainMessage((int)HC_JPEG, byteFrame).sendToTarget();
    }

};
```

4.2.5 Add Broadcast Receiver

Add BroadcastReceiver to handle HUD connections. The broadcast receiver handler added to the application will detect connections and disconnections from the HUD. It is possible for the HUD to be connected to an Android device prior to the applications execution. In this scenario the HUD connected event will not fire since the HUD is already connected. For best practice, check for the HUD's presence on start up using the getHudConnected() command.

```
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "Action received");
        switch (intent.getAction()) {
            case ACTION_USB_SIX15_CONNECTED: // USB PERMISSION GRANTED
                Toast.makeText(context, "USB Connected", Toast.LENGTH_SHORT).show();
                usbConnectDisplay(true);
                break;
            case ACTION_USB_PERMISSION_NOT_GRANTED: // USB PERMISSION NOT GRANTED
                Toast.makeText(context, "USB Permission not granted",
Toast.LENGTH_SHORT).show();
                usbConnectDisplay(false);
                break;
            case ACTION_USB_SIX15_DISCONNECTED: // USB DISCONNECTED
                Toast.makeText(context, "USB disconnected", Toast.LENGTH_SHORT).show();
                usbConnectDisplay(false);
                break;
            default:
                Log.d(TAG, intent.getAction());
                break;
        }
    }
};
```

4.3 Calling the HUD Service.

Add calls to the service for interfacing with the HUD by inserting the following code into your MainActivity after Intent createExplicitFromImplicit. The service handle is required, and it is a good practice to check if the HUD is connected prior to sending commands. See example prototype below.

```
/* check if HUD service is available and HUD is connected */
if(usbService != null && bHudConnected) {
    try {
        /* Send clear HUD display command */
        usbService.clearHudDisplay();
        Log.d(TAG, String.format("Clear Frame Time: %.2f ms",
usbService.getLastFrameBufferTime()));
    } catch (RemoteException rex) {
        Log.e(TAG, rex.getMessage());
        Toast.makeText(getApplicationContext(), "Remote USB Service Failed",
Toast.LENGTH_SHORT).show();
    }
}else {
    Toast.makeText(getApplication(), "Hud Not Connected",
            Toast.LENGTH_SHORT).show();
}
```

5. Six15 Presentation Mode Demo

The presentation mode of the HUD can be enabled by the HUD Service User Interface. Once presentation mode is enabled, any Android application can use the HUD display as a secondary screen for sending activities to. Presentation mode's demonstration project is available in the six15_pres_demo_app_src.zip file distributed with the SDK. There is a compiled APK version that is also released. To demonstrate the capabilities of presentation mode, plug HUD device into Android system with HUD Service previously installed, access the HUD Service User Interface menu and enable presentation mode. Install and launch the presentation mode demo. A hello world screen will show on the Android device while the HUD will display a blue screen with spinning text and a count timer in the lower right-hand corner.

5.1 Create a new application

Create a new Android application using an empty activity.

5.1.1 Add a display listener

Add a display listener to your application for managing the display.

```
private final DisplayManager.DisplayListener mDisplayListener = new
DisplayManager.DisplayListener() {
    @Override
    public void onDisplayAdded(int i) {
        Display display = mDisplayManager.getDisplay(i);
        if(display != null) {
            Log.d(TAG, "onDisplayAdded " + display.getName() + " ID: " + i);
            if (display.getName().compareTo("SIX-15 HUD") == 0) {
                Log.d(TAG, "Found SIX-15 display enable presentation");

            }
        }
    }

    @Override
    public void onDisplayRemoved(int i) {
        Display display = mDisplayManager.getDisplay(i);
        Log.d(TAG, "onDisplayRemoved ID: " + i);
        if(i==hudDisplayID){
            Log.d(TAG, "Found SIX-15 display sending slide to it");
            if(mPresentSlide != null){
                mPresentSlide.dismiss();
                mPresentSlide = null;
            }
            hudDisplayID =  -1;
        }
    }

    @Override
    public void onDisplayChanged(int i) {
        Display display = mDisplayManager.getDisplay(i);
        if(display != null)
```

```
            Log.d(TAG, "onDisplayChanged " + display.getName() + " ID: " + i);
        else
            Log.d(TAG, "onDisplayChanged ID: " + i);
    }
};
```

5.1.2 Find HUD Display and send Activity to HUD

In order to use presentation mode in your application, put the following code in your onStart function or where you would like it to detect the SIX15 HUD display and send an activity to the HUD.

```
mDisplayManager = (DisplayManager) getSystemService(Context.DISPLAY_SERVICE);
mDisplayManager.registerDisplayListener(mDisplayListener,null);

Display[] displays = mDisplayManager.getDisplays();
for(int x = 0; x<displays.length; x++){
    Log.d(TAG, displays[x].getName());
    if(displays[x].getName().compareTo("SIX-15 HUD")==0){
        hudDisplayID = displays[x].getDisplayId();
        Log.d(TAG, "Found SIX-15 display sending slide to it ID: " + hudDisplayID);
        mPresentSlide = new PresentSlide(MainActivity.this, displays[x]);
        mPresentSlide.show();
    }
}
```

The HUD display is 640x400 so developing your layouts will be necessary to have the displayed content look correctly.

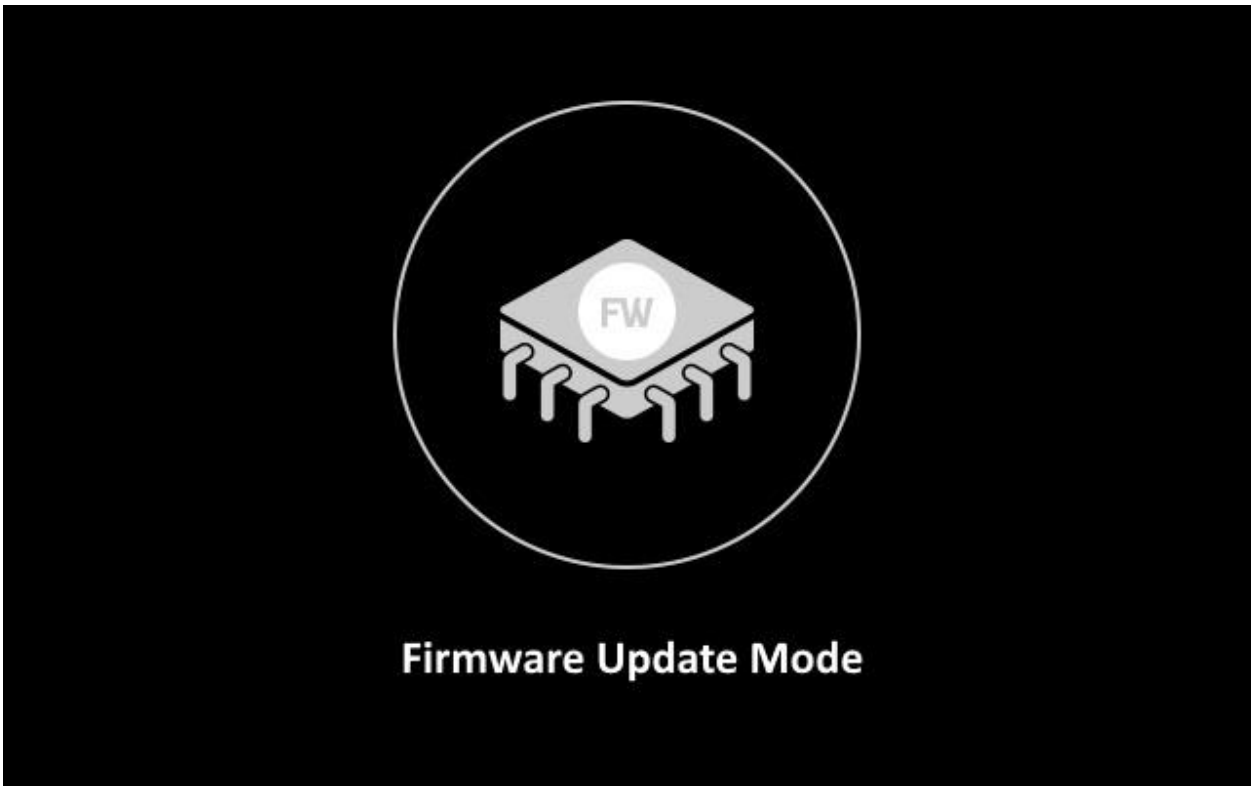
6. Known Issues

- Camera: setCameraEnabled()/ getCameraEnabled() not available
- Camera: setCameraAutofocusMode() / getCameraAutofocusMode() not available

7. Troubleshooting

7.1 Device Boots into Safemode

Your device may occasionally display an image like the one below. This is because the device has entered bootloader mode.



7.1.1 Powercycle

When this happens, the first step you should take is to power cycle your device. Make sure to close the application connecting the device to your computer before unplugging and reconnected the HUD.

7.1.2 Reflash the Firmware

If power cycling does not work, your next step is trying to reflash the firmware. To do this, refer to section 4 Updater App for instructions on installing and using the updater app to update the HUD's firmware.

7.1.3 Contact Six15

If all else fails and you are unsure why your device does not appear to be working, please contact Six15.

Email: Info@six-15.com

8. SDK Changelog

8.1 SDK 2.1 to 2.2

Firmware (5.25):

- Power Usage:
  - Reduced idle power: 450mW
  - Static image display power: 500mW
- Tracker (IMU):
  - Added calibration routines
  - Corrected quaternion output
- Stability fixes

Android Service (5.34):

- General stability fixes.

Android Demo (9.05):

- Added graph for roll/pitch/yaw in IMU.

8.2 SDK 2.2 to 2.3

Firmware (5.30)

- Display:
  - Improved image pipeline to support framerates up to 30 fps
- Microphone:
  - Firmware now restarts when mic enable state is changed (Used to be done in service)
- Stability fixes:
  - Corrected issue: IMU caused reset when not initialized correctly
  - Corrected issue where camera did not work on Zebra TC51
  - Corrected issue where microphone did not work on Android 10 (updated sample rate to 48kHz)

Android Service

- Removed screen mirroring and created separate app for this functionality
- Created two builds:
  - camSDK – Includes Six15-created camera interface
  - uvc – Does not connect to camera and allows a UVC app to connect to the camera
- Stability fixes:
  - Corrected bug where COMMS channel would lock-up
  - Improved callback handling, previously service could crash if receiving many callback (example: comms message, imu and camera)

Android Screen Mirror

- Created App

Android Demo:

- Improved encoding quality on microphone test

8.3 SDK 2.3 to 2.4

Firmware (5.33)

- Tracker (IMU) - Added support for ICM20948 (rev C support)
- Camera
    - Changed UVC to version 1.0 to support Linux/Android native driver
    - Added autofocus (Only accessible from UVC interface)
    - Added UVC commands for brightness, contrast, hue, saturation, and sharpness

Android Service (5.43)

- Fixed permissions bug where they need to be accepted every time USB is connected. Now is able to save permission and it only needs to be accepted once.

8.4 SDK 2.4 to 2.5.0

Firmware:

- Tracker (IMU)
    - Improved calibration routine
    - Added calibration progress reporting.
    - Improved filter for calculating pose (roll, pitch, yaw)
    - Samples are output at 60Hz (from 30Hz).
    - **Breaking Change**: magnetometer readings are in units of uT (instead of mG). This is to conform to the Android sensor specification.
    - **Breaking Change**: The orientation of the accel/gyro/magnetometer is changed to adhere to Android standard (see: https://developer.android.com/reference/android/hardware/SensorEvent#values)
- Camera
    - Corrected bug where video locked up on highest resolution.

Android Service:

- Re-enabled presentation mode and screen mirroring through the API
- (Internal) Automatic build of project
- Target SDK 30 (Android 11)

Android Demo:

- UI Clean-up
- Example slideshow images are included.
- Example slideshow images may come from galley instead of a single folder.
- Example slideshow supports Bluetooth remote next/previous.
- Corrected microphone bug where mic didn’t work on Android 11
- Added check for HMD microphone in microphone screen

Android Calibration:

- Project UI Clean-up
- Added progress bar when calibrating.

8.5 SDK 2.5.0 to 2.6.3

Firmware (5.39):

- Fixed crash when microphone is enabled and disabled ~20 times.
- Fixed bug where screen flicker when using screen mirror and camera
- Fixed bug where image display is sometimes delayed when recovering from low-power mode

Android Service:

- Added button for screen mirror and presentation mode
- Fixed bug where old camera frames are returned when restarting camera

Android Demo:

- Code cleanup/ stability fixes

8.6 SDK 2.6.3 to 2.6.4

Android Service:

- Corrected bug that caused camera to stop when calling startRawCameraCapture() with highest resolution

9. Document Revision History

Table 1 – Document Revision History

Version Number	Date	Author/Owner	Description of Change
1.00	7/21/2017	George Vigelette	Initial Release
1.01	8/9/2017	Ted Ricks	Review updates
1.02	8/16/2017	George Vigelette	Added API Commands
1.03	3/05/2018	George Vigelette	Added API Commands
1.04	6/22/2018	George Vigelette	Added API Commands, Service information, updated entire guide
1.05	06/29/2018	Ted Ricks	Review updates
1.06	06/29/18	Ted Ricks	Release to SDK
1.07	07/27/18	George Vigelette Geoffrey Furman	2.2.4.7 Device Information. Updated images reflect version 4.20 of the firmware. 3.3.1 AIDL (Android Interface Definition Language). Updated to version 1.01 3.3.4 HUD Service Callback Interface. Added JPEG camera mode. 3.3.5 HUD Service. Added JPEG camera mode. 3.4.5.4 Start Camera (Raw JPEG Mode). Section Inserted. 4.2.4 Create Callback Interface and Handler. Added JPEG camera mode. 4.2.5 Add Broadcast Receiver. Text added.
2.00	11/04/2019	A Sojda	Initial release of SDK version 2.0
2.01	01/23/2020	A Sojda	Release 2.1, see Section 8.1 for changelog.
2.2	03/04/2020	John Martel	Correcting spelling and grammar issues throughout. Made style changes for ease of reading. Rearranged to flow better.
2.3	07/06/2020	A Sojda	Release 2.3, See Section 8.2 for changelog
2.4	12/07/2020	A Sojda	Release 2.4, See Section 8.3 for changelog.
2.5.0	3/31/2021	A Sojda	Release 2.5.0, See section 8.4 for changelog
2.6.3	8/13/2021	A Sojda	Release 2.6.3, See section 8.5 for changelog
2.6.4	8/24/2021	A Sojda	Release 2.6.4, See section 8.6 for changelog