Android SDK Guide for AP6800

80187500-001 Rev 52
24 May 2024

ID TECH
10721 Walker Street, Cypress, CA 90630-4720
Tel: (714) 761-6368 Fax (714) 761-8880
www.idtechproducts.com

ID TECH
10721 Walker Street Cypress, CA 90630 USA

Warranty Disclaimer
Software, services, and hardware are provided "as is" and "as-available" and the use of the software, services, and hardware are at the user's own risk. ID TECH does not make, and hereby disclaims, any and all other express or implied warranties, including, but not limited to, warranties of merchantability, fitness for a particular purpose, title, and any warranties arising from a course of dealing, usage, or trade practice. ID TECH does not warrant that the services or hardware will be uninterrupted, error-free, or completely secure.

Revision History

Date	Rev	Changes	By
12/01/2023	50	Initial release.	Jacky
05/17/2024	51	Revision/style pass.	CB
05/24/2024	52	Alpha content pass.	CB

Table of Contents

1. OVERVIEW 6
1.1. ZSDK 7
1.2. Android Studio Environment Setup 7
2. AP6800 MAIN TRANSACTION COMMANDS 8
2.1. Transaction Flow 8
2.2. Contact Transaction Flow 8
2.3. Contactless (CTLS) Transaction Flow 9
2.4. MSR Transaction Flow 9
2.5. Transaction API 10
2.5.1. Start Transaction 10
2.5.2. Cancel Transaction 11
2.5.3. Update Balance Command 11
2.5.4. Authenticate Transaction 12
2.5.5. Complete Transaction 13
2.6. Transaction Command Examples 14
2.6.1. Contactless Example 14
2.6.2. MSR Example 14
2.6.3. EMV Example 15
2.6.4. EMV with Authenticate and Complete Example 16
3. SAMPLE PROJECT TUTORIAL 18
3.1. Create a New Project 18
3.2. Import the Necessary Libraries 19
3.2.1. Remote Binary Dependency 19
3.2.2. Local Binary Dependency 20
3.3. Payment Application 21
3.3.1. Implement the Layout 21
3.3.2. Implement Function List 23
3.3.3. Application UI 24
4. TRANSACTION FLOW DIAGRAMS 30
4.1. MSR Flow 30
4.2. CTLS Flow 31
4.3. EMV Flow 32
4.4. Cancel Flow 33
5. AP6800 ANDROID SDK 34
5.1. Proximity Sensor 34
5.2. Update LED 35
5.3. Serial Port 38
5.4. Camera 42
5.4.1. Add Camera Dependencies 42
5.4.2. Request Camera Permissions 43
5.4.3. Add Camera Preview 44
5.5. QR Codes 44
5.6. Light Sensor 46
5.7. Network Connectivity 47
6. FOR MORE INFORMATION 49

1. Overview

The ID TECH Android SDK is an Android framework provided by ID TECH as the main interface between Android applications and payment processing solutions.

This document describes the framework requirements and interface definitions and requirements for any Android applications deploying with a payment application. We also provide the AP6800 Peripheral SDK to control peripheral devices.

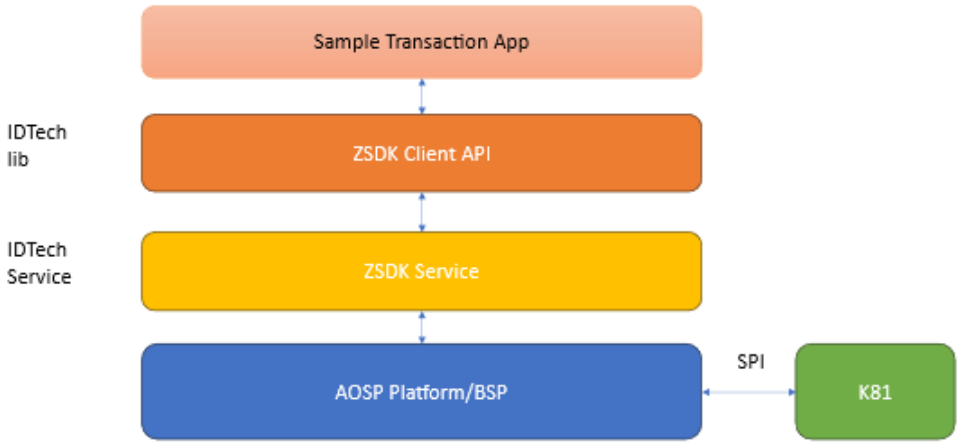
This document covers the following topics:

- Environment Setup: Android Studio installation.
- AP6800 Provision Status Check: Detail steps for initial setup using the Welcome app to help users provision the reader.
- AP6800 Main Transaction Commands.
- Detailed transaction command flows.
- Sample Project Tutorial.
- Step by step instructions for activating AP6800 and a sample project tutorial.
- AP6800 Android SDK.
- Proximity Sensor: Sample code for retrieving data from the AP6800's proximity sensor.
- Update LED: AP6800 includes LED control, provide sample code to how to light on/off LED.
- Serial Port: Sample code for reading and writing data through the AP6800's serial port.
- Camera: Sample code for reading data from both the front camera and scan camera.
- QRCode: Sample code for decoding QR Codes with MLKit.
- Light Sensor: Sample code for reading ambient light levels.
- Connectivity: Descriptions of the AP6800's connectivity options, including Wi-Fi, Ethernet, and LTE.

1.1. ZSDK

Developers who require high-level-language support should consider using ID TECH's ZSDK API when integrating AP6800 payment devices. Developers can reference the Android Client ZSDK API reference document for detailed API descriptions.

Figure 1: ZDSK Architecture



- The ZSDK Service is an Android built-in service on the AP6800 that is responsible for communication with the K81 payment chip.
- The ZSDK Client API is an Android Kotlin AAR library that communicates with the ZSDK Service using ZMQ.

Developers can write transaction applications using the ZSDK Client API. We provide detailed sample project tutorials below in the Sample Project Tutorial section.

1.2. Android Studio Environment Setup

Before you begin, make sure you have set up Android Studio (see the Android Studio website for downloads and other information).

2. AP6800 Main Transaction Commands

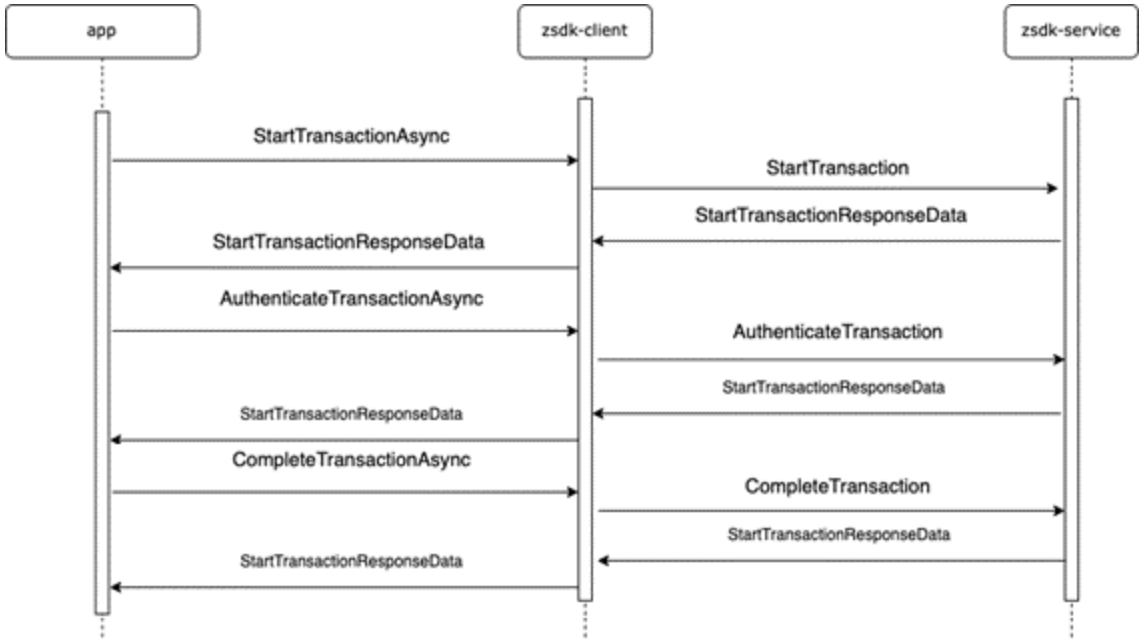
The following section describes the main transaction commands and the transaction flow for AP6800 readers.

2.1. Transaction Flow

The following section describes the transaction flow for AP6800 readers.

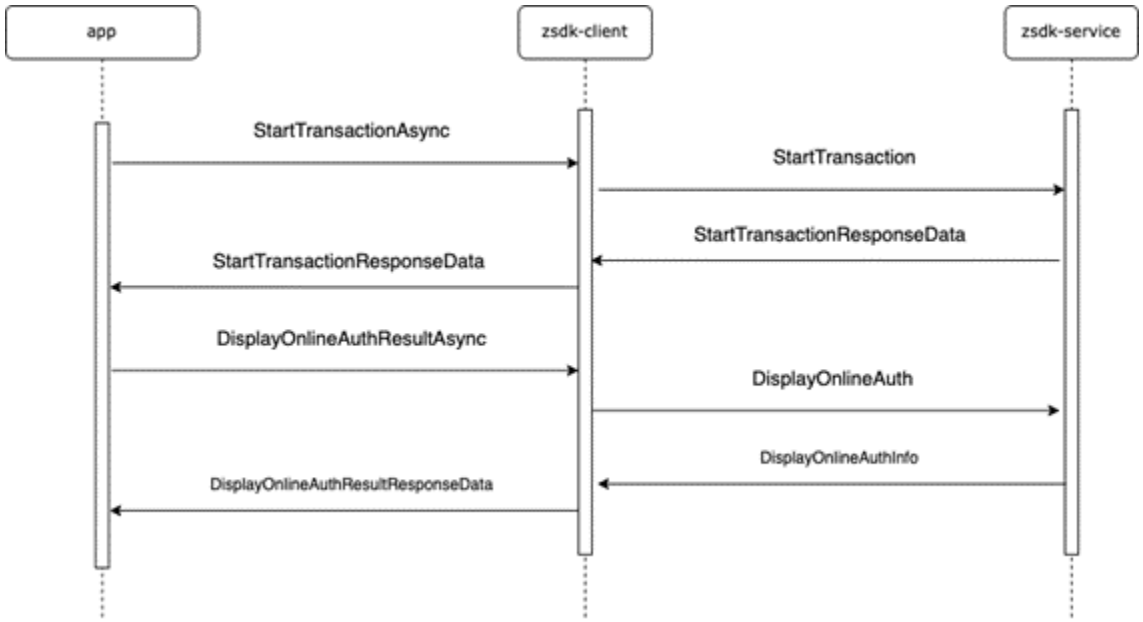
2.2. Contact Transaction Flow

Contract transactions use three commands: **StartTransactionAsync**, **AuthenticateTransactionAsync**, and **CompleteTransactionAsync**. AP6800 readers are usually set to auto-authenticate and auto-complete. This means the application can only call the **StartTransactionAsync** command to finish a transaction. However, when disabling auto-authenticate and auto-complete, the application can also call the **AuthenticateTransactionAsync** and **CompleteTransactionAsync** commands.



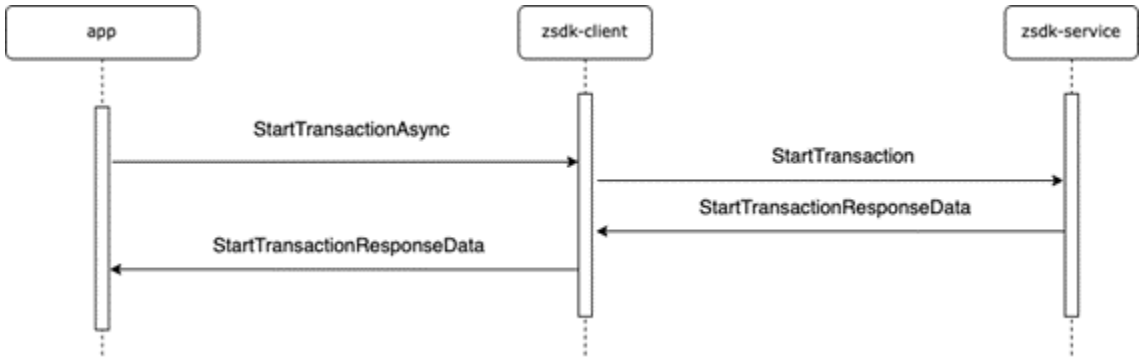
2.3. Contactless (CTLS) Transaction Flow

Contactless transactions use two commands: **StartTransactionAsync** and **DisplayOnlineAuthResultAsync**, which must be called in that order. Use the **respType** parameter to know if the application needs to call **DisplayOnlineAuthResultAsync**. If the **respType** is set to **OnlineAuthRequired**, call **DisplayOnlineAuthResultAsync** to finish the transaction.



2.4. MSR Transaction Flow

MSR transactions use one command: StartTransactionAsync.



2.5. Transaction API

The section below describes the transaction API for AP6800 readers.

2.5.1. Start Transaction

```
fun Client.StartTransactionAsync(deviceId: String, amount: Double,
amountOther: Double, transType: UByte, transTimeout: UByte,
transInterfaces: UByte, timeoutMilli: Long)
```

Starts a transaction; asynchronous.

The reader responds with an **Activate Transaction (02-40)** command. If auto-authentication is enabled, it will respond with a **Contact Authenticate Transaction (60-11)** command; if auto-completion is enabled, it will respond with a **Complete Transaction (60-12)** command.

Parameters

Name	Type	Description
deviceId	String	The device ID.
amount	Double	Transaction amount value.
amountOther	Double	Other amount value, if any (Tag value 9F03). Secondary amount associated with the transaction representing a cashback amount.
transType	UByte	Indicates the type of financial transaction. **0x00** = Purchase, **0x20** = Refund.
transTimeout	UByte	The **transTimeout** of transaction timeout time.
transInterfaces	UByte	Interface transaction type. **1**=MSR, **2**=CTLS, **4**=EMV, **7**=All of the above.
timeoutMilli	Long	The zsdk-client socket timeout time. Default is three seconds.

Response

Name	Type	Description
respType	TransactionResponseType	*Success*, FailedNack, OnlineAuthRequired, Timeout, NonDefined
attribute	UShort	Attribution byte for 02-40, 60-11 and 60-12
cardData	List<UShort>	TLV structure data

2.5.2. Cancel Transaction

```
fun Client.CancelTransactionAsync(deviceId: String, mode: UShort = 0u,
timeoutMilli: Long = 3000
```

Stops reader or card communication; asynchronous.

Parameters

Name	Type	Description
deviceId	String	The device ID.
timeoutMilli	Long	The zsdk-client socket timeout time. Default is three seconds.

Response

Name	Type	Description
data	UShort	transaction exponent

2.5.3. Update Balance Command

```
fun Client.DisplayOnlineAuthResultAsync(deviceId: String, StatusCode:
Int, OnlineAuthCode: UByteArray, TransactionDate: UByteArray,
TransactionTime: UByteArray, timeoutMilli: Long = 3000)
```

Display Online Authorization Result Async on the reader's display (OK or NOT OK)

Parameters

Name	Type	Description
deviceId	String	The device ID.
StatusCode	Int	**0:** OK, **1:** NOT OK, **2:** (ARC response 89 for Interac).
OnlineAuthCode	UByteArray	Use for authorization, can be empty to use default value.
TransactionDate	UByteArray	Binary-Coded Decimal (BCD) Code: YYMMDD, can be empty to use reader date.
TransactionTime	UByteArray	Binary-Coded Decimal (BCD) Code: HHMMSS, can be empty to use reader time.
timeoutMilli	Long	The zsdk-client socket timeout time. Default is three seconds.

Response

Name	Type	Description
respType	TransactionResponseType	*Success*, FailedNack, OnlineAuthRequired, Timeout, NonDefined
attribute	UShort	Attribution byte for 02-40, 60-11, and 60-12
cardData	List<UShort>	TLV structure data

2.5.4. Authenticate Transaction

```
fun Client.AuthenticateTransactionAsync(deviceId: String,
goOnline:Boolean, online_timeout: Long,
outputTagList:Array<ByteArray?>, timeoutMilli: Long = 3000)
```

Asynchronous version of the **Contact Authenticate Transaction** command; authenticates a contact transaction. Call this function after the **Start Transaction** command. Continues the EMV transaction flow:

1. Offline data authentication
2. Processing restrictions
3. Cardholder verification
4. Terminal risk management
5. TAA/CAA/1st Gen AC

If the response code is **00 04**, the reader needs to go online to process the transaction and must send an **Apply Host Response** command.

Parameters

Name	Type	Description
deviceId	String	The device ID.
goOnline	Bool	Force the transaction to go online or allow normal processing to occur via Terminal action analysis.
online_timeout	Long	the waiting time for host response when online in seconds.
outputTagList	Array <ByteArray?>	TLV Data of output tag list.
timeoutMilli	Long	The zsdk-client socket timeout time. Default is three seconds.

Response

Name	Type	Description
respType	TransactionResponseType	*Success*, FailedNack, OnlineAuthRequired, Timeout, NonDefined
attribute	UShort	Attribution byte for 02-40, 60-11, and 60-12.
cardData	List<UShort>	TLV structure data.

2.5.5. Complete Transaction

```
fun Client.CompleteTransactionAsync(deviceId: String, wentOnline:
Boolean, authCode: ByteArray,issuerAuthData: UByteArray, scriptsData:
UByteArray, outputData: UByteArray, timeoutMilli: Long = 3000)
```

Asynchronous version of **Complete Transaction**.

This function is the "completion" step in the EMV transaction flow. The command sends the acquirer data (if online) to the card and notifies the reader that the transaction is complete. Depending on the transaction type, this command performs the following process; the reader may or may not perform each step depending upon the acquirer's requirement and response:

1. External Authentication
2. Script Processing
3. 2nd Gen AC
4. Completion

Parameters

Name	Type	Description
deviceId	String	The device ID.
wentOnline	Boolean	If true, the reader went online with the host device.
authCode	ByteArray	Authorization response code.
issuerAuthData	UbyteArray	Issuer authentication data as TLV data format.
scriptsData	UbyteArray	Scripts as TLV data format.
outputData	UbyteArray	Output data list.
timeoutMilli	Long	The zsdk-client socket timeout time. Default is three seconds.

Response

Name	Type	Description
respType	TransactionResponseType	*Success*, FailedNack, OnlineAuthRequired, Timeout, NonDefined
attribute	UShort	Attribution byte for 02-40, 60-11, and 60-12.
cardData	List<UShort>	TLV structure data.

2.6. Transaction Command Examples

The section below provides several example transaction command flows.

2.6.1. Contactless Example

```
GlobalScope.launch {
    val deviceId = "Neo2-0" //replace to your device id
    val amount = 0.1
    val amountOther = 0.0
    val transType: UByte = 0u
    val transTimeout: UByte = 100u
    val transInterfaceType = 1.toUByte()
    val startTransCmd = Client.StartTransactionAsync(deviceId,
        amount, amountOther, transType, transTimeout,
        transInterfaceType, 200000)

    startTransCmd.waitForCompletion()
    if (startTransCmd.getCommandStatus() ==
ClientCommandStatus.Completed) {
        val startTransResult = startTransCmd.getResultData()
        if (startTransResult?.respType ==
TransactionResponseType.OnlineAuthRequired) {
            val okStatus = 0x00
            val dummy = "DEADBEAF"
            val authcode = /*dummy.hexToUByteArray()*/ubyteArrayOf()
            val tdate = ubyteArrayOf(/*0x22u, 0x06u, 0x11u*/)
            val ttime = ubyteArrayOf(/*0x12u, 0x34u, 0x56u*/)
            val displayOnlineAuthResultCmd =
Client.DisplayOnlineAuthResultAsync(deviceId, okStatus, authcode,
tdate, ttime)
            val displayOnlineAuthResult_rv =
displayOnlineAuthResultCmd.waitForCompletion()
        }
    }
```

2.6.2. MSR Example

```
GlobalScope.launch {
    val deviceId = "Neo2-0" //replace to your device id
    val amount = 0.1
    val amountOther = 0.0
    val transType: UByte = 0u
    val transTimeout: UByte = 100u
    val transInterfaceType = 2.toUByte()
    val startTransCmd = Client.StartTransactionAsync(deviceId,
        amount, amountOther, transType, transTimeout,
        transInterfaceType, 200000)

    startTransCmd.waitForCompletion()
    if (startTransCmd.getCommandStatus() ==
ClientCommandStatus.Completed) {
```

```
        startTransCmd.getResultData()?.apply {
            cardData.apply {
                val builder = StringBuilder()
                val resultData = arrayListOf<Byte>()
                forEach { uShort ->
                    resultData.add(uShort.toByte())
                }
                TLVUtils.getTags(resultData).let {
                    it.forEach { tlv ->

builder.append(tlv.tag.toHexString()).append(":").append(tlv.value.to
HexString()).append("\n")
                    }
                }
                Log.d(TAG,"Transaction card data :
${builder.toString()}")
            }
        }
    }
}
```

2.6.3. EMV Example

```
GlobalScope.launch {
    val deviceId = "Neo2-0" //replace to your device id
    val amount = 0.1
    val amountOther = 0.0
    val transType: UByte = 0u
    val transTimeout: UByte = 100u
    val transInterfaceType = 4.toUByte()

    val startTransCmd = Client.StartTransactionAsync(deviceId,
        amount, amountOther, transType, transTimeout,
        transInterfaceType, 200000)
    startTransCmd.waitForCompletion()
    if (startTransCmd.getCommandStatus() ==
ClientCommandStatus.Completed) {
        startTransCmd.getResultData()?.apply {
            cardData.apply {
                val builder = StringBuilder()
                val resultData = arrayListOf<Byte>()
                forEach { uShort ->
                    resultData.add(uShort.toByte())
                }
                TLVUtils.getTags(resultData).let {
                    it.forEach { tlv ->

builder.append(tlv.tag.toHexString()).append(":").append(tlv.value.to
HexString()).append("\n")
                    }
                }
```

```
                Log.d(TAG,"Transaction card data :
${builder.toString()}")
            }
        }
    }
}
```

2.6.4. EMV with Authenticate and Complete Example

```
GlobalScope.launch {
    val deviceId = "Neo2-0" //replace to your device id
    val amount = 0.1
    val amountOther = 0.0
    val transType: UByte = 0u
    val transTimeout: UByte = 100u
    val transInterfaceType = 4.toUByte()

    val startTransCmd = Client.StartTransactionAsync(deviceId,
amount, amountOther, transType, transTimeout, transInterfaceType,
200000)
    startTransCmd.waitForCompletion()
    if (startTransCmd.getCommandStatus() ==
ClientCommandStatus.Completed) {
        val startTransResult = startTransCmd.getResultData()
        if (startTransResult?.respType ==
TransactionResponseType.Success) {
            delay(3000)
            var outputTag: ByteArray? = null
            val outputTagList = arrayOf(outputTag)
            val authenticateTransactionCommand =
Client.AuthenticateTransactionAsync(deviceId, false, 120,
outputTagList)
            val authTran_rv =
authenticateTransactionCommand.waitForCompletionWithTimeout(40000)
            if (authTran_rv == ClientCommandStatus.Completed) {
                authenticateTransactionCommand.getResultData()?.apply
{
                    // Do fake complete for demo
                    delay(3000)
                    //change to data from PAE, example data
AC862442C8BF32893030
                    val issuerAuthData = UByteArray(0)
                    //change data from PAE, example data
DFEE1A6B4F5056575A8284878A959A9B9C5F245F2A5F305F349F029F039F069F079F0
89F099F0D9F0E9F0F9F109F119F129F159F1A9F1E9F219F269F279F339F349F359F36
9F379F389F399F3C9F409F419F4B9F5B9F6E9F7C5F205F285F2D5F56DF31DF812ADFE
E23DFEE26FFEE01
                    val outputData = UByteArray(0)

                    val completeTransCmd =
Client.CompleteTransactionAsync(deviceId, true, byteArrayOf(0x30,
0x30), issuerAuthData, ubyteArrayOf(), outputData)
```

```
                    val completeTransCmd_rv =
completeTransCmd.waitForCompletion()
                    val completeData =
completeTransCmd.getResultData()
                }
            }
        }
    }
}
```

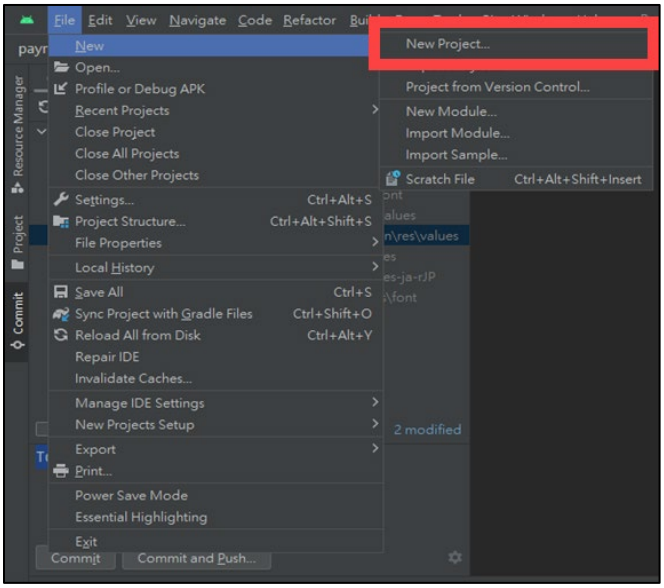
3. Sample Project Tutorial

Developing an Android application installed on the AP6800 for payment requests involves using the library ID TECH provides and Android Studio to develop a payment application.

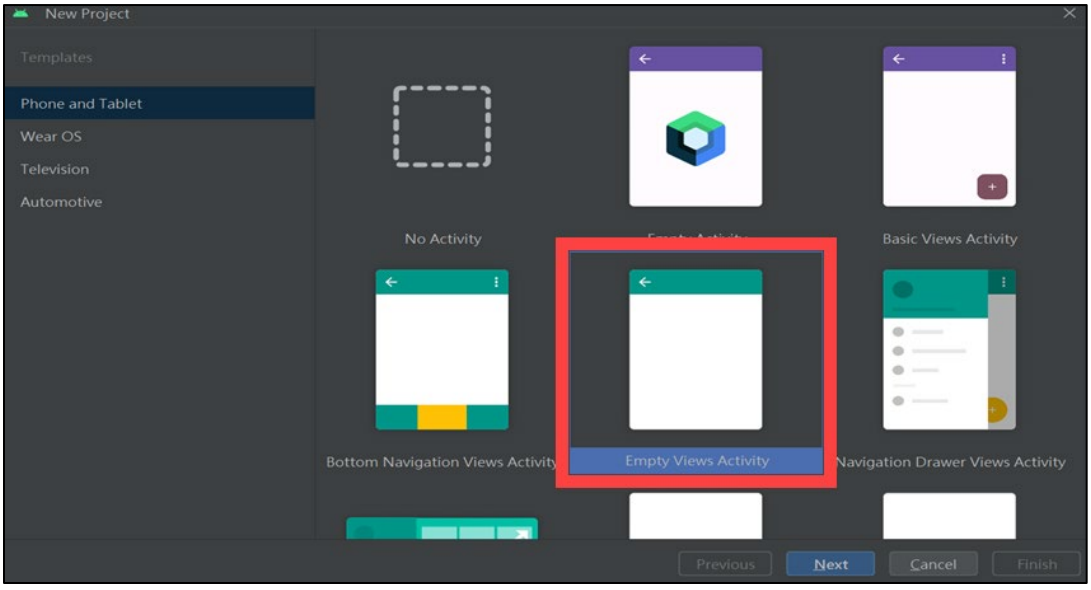
Gradle is a build tool used for managing and automating the build process. It assists in downloading project dependencies, packaging your project code, and preparing it for compilation. Visit the Gradle website to learn the tool's fundamentals and features

3.1. Create a New Project

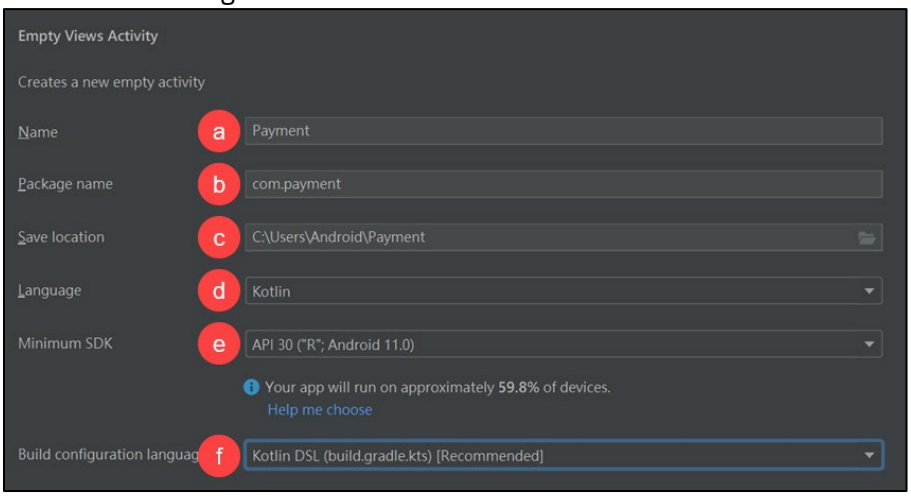
1. Select the File item on Android Studio menu bar to create a new project, **File** > **New** > **New Project**.



2. Select **Phone and Tablet** under **Templates**, then select the **Empty Views Activity** layout.



3. Enter the following information:



   a. Application name.
   b. Application package name.
   c. Save location.
   d. Select **Kotlin** for the language.
   e. Select **API 30** as the minimum SDK.
   f. Select **Kotlin DSL (build.gradle.kts)** for the Build configuration language.

4. Select **Finish**.

3.2. Import the Necessary Libraries

Integrators must import the libraries listed below.

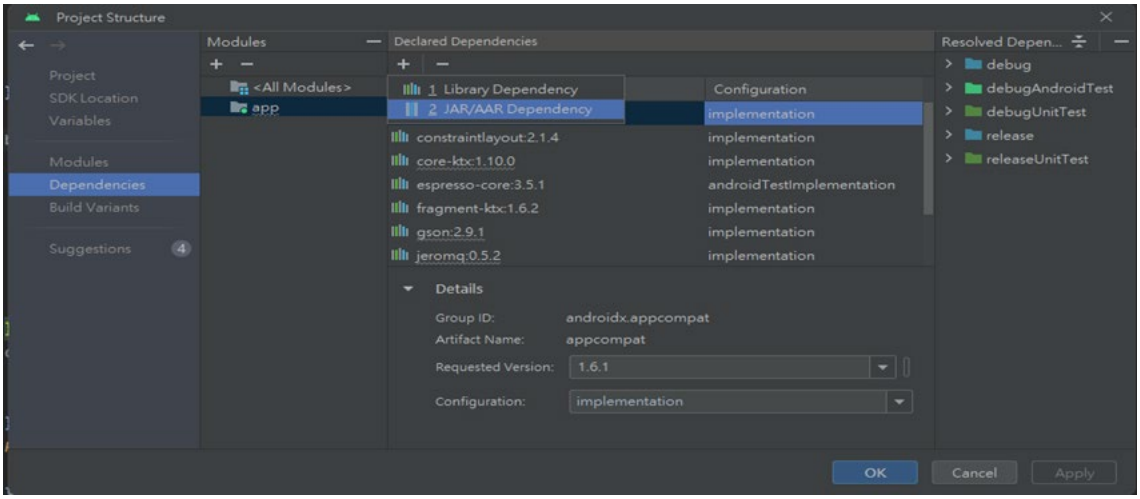
3.2.1. Remote Binary Dependency

The following libraries are required to communicate with AP6800 devices:

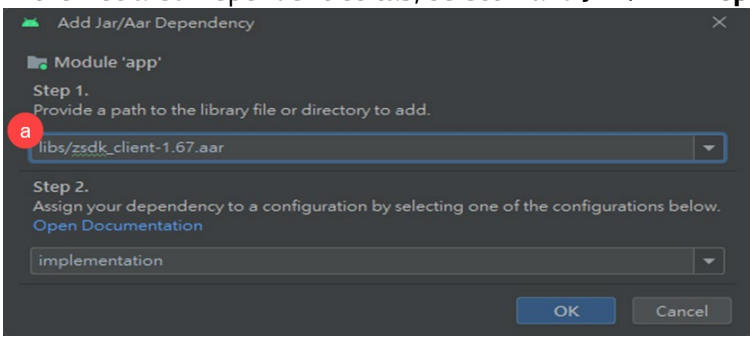
- `implementation("com.google.code.gson:gson:2.9.1")`
- `implementation("org.zeromq:jeromq:0.5.2")`
- `implementation("androidx.fragment:fragment-ktx:1.6.2")`

3.2.2. Local Binary Dependency

To use your new Android library's code in another app or library module within the same project, add a project-level dependency. For example, using **zsdk_client-1.67**:



1. Copy **zsdk_client-1.67.aar** to **\app\libs**
2. Navigate to **File** > **Project Structure** > **Dependencies**.
3. Select the module to add to the library.
4. In the Declared Dependencies tab, select **+** and **JAR/ARR Dependency**.



   a. Enter **libs/zsdk_client-1.67.aar**., then select **OK**.
5. Select **Apply** and **OK**
6. Automatically generate **implementation(files("libs/zsdk_client-1.67.aar"))**.

3.3. Payment Application

The payment application example uses the **View Model** class. The View Model class allows consistent UI design across whole application and provides access to business logic.

For example:

```
val outputs = MutableLiveData<ArrayList<String>>(arrayListOf())

class MainViewModel: ViewModel() {
fun connectToServer(ip: String, port: String = "42501", activity:
Activity) {
        Client.autoReconnectToServer(activity, ip, port,
"PaymentTest", "1.0", 1u, "")
        GlobalScope.launch(Dispatchers.IO) {
            Client.waitUntilConnected()
            val status = enumerateDevices()
            connectStatus.postValue(status)
            if (devices.isNotEmpty()) { connectedDeviceId =
devices[0]  }
        }
    }
}

var connectStatus  = MutableLiveData(false)

class MainActivity : AppCompatActivity() {
viewModel.connectStatus.observe(this) { status ->
            if (status) {
                ivConnectStatus?.setImageResource(R.drawable.connect)
            } else {

ivConnectStatus?.setImageResource(R.drawable.disconnect)
            }
            paymentButtonModel(status, edtAmount?.text)

     }
}
```

3.3.1. Implement the Layout

The layout components used are:

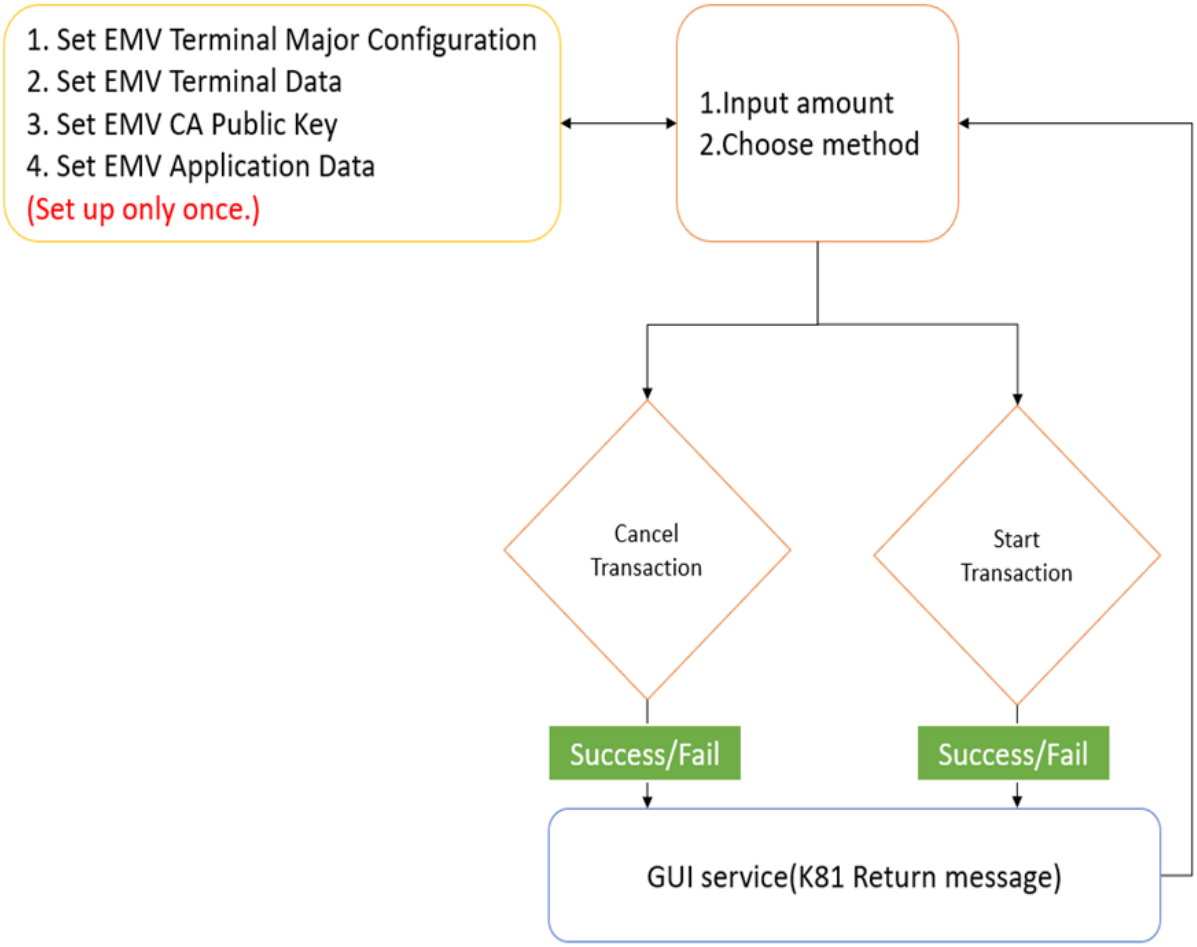
- `androidx.constraintlayout.widget.ConstraintLayout`
- `androidx.cardview.widget.CardView`
- `androidx.appcompat.widget.AppCompatImageView`
- `RadioGroup`
- `androidx.appcompat.widget.AppCompatRadioButton`
- `com.google.android.material.button.MaterialButton`
- `Androidx.recyclerview.widget.RecyclerView`

For detailed UI design, refer to the source code in **activity_main.xml**. For detailed information, refer to the URL (Recyclerview).

The sample application uses this ResponseAdapter:

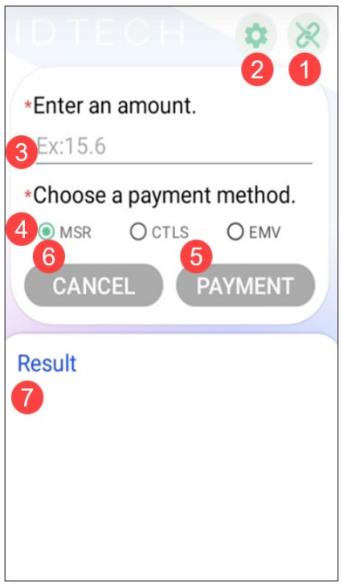
```
class ResponseAdapter(private var data: ArrayList<String>) :
    RecyclerView.Adapter<ResponseAdapter.ResponseViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): ResponseViewHolder {
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.item_response,
parent, false)
        return ResponseViewHolder(view)
    }

    override fun onBindViewHolder(holder: ResponseViewHolder,
position: Int) {
        holder.setData(data[position])
    }

    override fun getItemCount() = data.size

    fun update(newData: ArrayList<String>) {
        data = newData
        notifyItemChanged(0, newData.size)
    }

    class ResponseViewHolder(view: View) :
RecyclerView.ViewHolder(view) {
        private val tvResponse: TextView =
view.findViewById(R.id.tv_response)

        fun setData(item: String) {
            tvResponse.text = item
        }
    }
```

3.3.2. Implement Function List



3.3.3. Application UI

The section below describes the numbered visual elements of the following screenshot and, if applicable, the API commands associated with those elements.



1. **Connect**
   Connects to the server.
   ```
   Client.autoReconnectToServer(activity, “127.0.0.1”, port, "Payment Test", "1.0", 1u, "")
   Client.GetDevicesAsync() , return List<String>
   ```
2. **Setting**
   Configuration settings. You only need to set this one time.
   ```
   Client.SetEMVTerminalMajorConfigurationAsync(connectedDeviceId, 1)
   Client.SetEMVTerminalDataAsync(connectedDeviceId, data1C.decodeHex())
   Client.SetEMVCAPublicKeyAsync(connectedDeviceId, pair.first, pair.second.decodeHex())
   Client.SetEMVApplicationDataAsync(connectedDeviceId, pair.first, pair.second.decodeHex())
   ```
3. **Enter Amount**
   Required field; input is restricted to numeric values only.

4. **Choose Payment Method**
   Required field; default is set to MSR.

```
rgType?.setOnCheckedChangeListener { group, checkedId ->
    val rb = findViewById<RadioButton>(checkedId)
    when (rb.id) {
        R.id.rb_msr -> { selectType = 1 }
        R.id.rb_ctls -> {  selectType = 2  }
        R.id.rb_emv -> { selectType = 4 }
    }
```

5. **Transaction**
   Runs a transaction.

```
Client.SetAutoAuthenticateAsync(connectedDeviceId, false)
Client.SetAutoCompleteAsync(connectedDeviceId, false)
Client.StartTransactionAsync(connectedDeviceId, amount,
amountOther, transType, transTimeout,
transInterfaceType.toUByte(), 100000)
```

```
@OptIn(ExperimentalStdlibApi::class,
ExperimentalUnsignedTypes::class)
    fun startTransaction(amount: Double, transInterfaceType:
Int) {
        val amountOther = 0.0
        val transType: UByte = 0u
        val transTimeout: UByte = 100u
        val transTypeText = when (transInterfaceType) {
            1 -> "MSR"
            2 -> "CTLS"
            4 -> "EMV"
            else -> {""}
       }

    GlobalScope.launch(Dispatchers.IO) {
        Client.waitUntilConnected()
        if (connectedDeviceId == null) return@launch

        val setAutoAuthenticateCommand =
        Client.SetAutoAuthenticateAsync(connectedDeviceId!!,
false)
        val autoAuthenticateStatus=

setAutoAuthenticateCommand.waitForCompletionWithTimeout(1000)
        Log.d(TAG, "DisableAutoAuthenticate:
${autoAuthenticateStatus.name}")

        val setAutoCompleteCommand =
        Client.SetAutoCompleteAsync(connectedDeviceId!!, false)
        val autoCompleteStatus =
```

```
setAutoCompleteCommand.waitForCompletionWithTimeout(1000)

        val startTransCmd = Client.StartTransactionAsync(
        connectedDeviceId!!,amount, amountOther, transType,
transTimeout,
        transInterfaceType.toUByte(), 100000)
        startTransCmd.waitForCompletion()

    val startTransStatus = startTransCmd.getCommandStatus()
        if (startTransStatus == ClientCommandStatus.Completed)
{
            val startTransResult =
startTransCmd.getResultData()
             if (startTransResult?.respType ==
TransactionResponseType.
                OnlineAuthRequired) {
                val okStatus = 0x00
                val authCode = ubyteArrayOf()
                val tDate = ubyteArrayOf()
                val tTime = ubyteArrayOf()
                val displayOnlineAuthResultCmd =
Client.DisplayOnlineAuthResultAsync(
                connectedDeviceId!!, okStatus, authCode, tDate,
tTime )

    val displayOnlineAuthResult =
displayOnlineAuthResultCmd.waitForCompletion()
        outputInCoroutine("[StartTransaction $transTypeText]",
        "DisplayOnlineAuthResult :
${displayOnlineAuthResult.name}")
        return@launch
        }

        if (startTransResult?.respType ==
TransactionResponseType.Success &&
        transInterfaceType != 1) {
        delay(3000)
        val outputTag: ByteArray? = null
        val outputTagList = arrayOf(outputTag)
        val authenticateTransactionCommand =
Client.AuthenticateTransactionAsync(
        connectedDeviceId!!, false, 120, outputTagList)

        val authenticateTransactionStatus =

authenticateTransactionCommand.waitForCompletionWithTimeout(400
00)
        outputInCoroutine("[StartTransaction $transTypeText]",
        "AuthenticateTransaction :
${authenticateTransactionStatus.name}")
```

```
            if (authenticateTransactionStatus ==
ClientCommandStatus.Completed) {
            val authenticateTransactionData =
            authenticateTransactionCommand.getResultData()
                if (authenticateTransactionData != null) {
                Log.d(TAG, "AuthenticateTransaction
AuthAttribute:
                ${Integer.toHexString(

authenticateTransactionData.attribute.toInt())}")
                Log.d(TAG, "AuthenticateTransaction
AuthCardDataSize:
                ${authenticateTransactionData.cardData.size}")
                val authTransDataHB = StringBuilder()
                for (uShort in
authenticateTransactionData.cardData) {
                 authTransDataHB.append(
                Integer.toHexString(uShort.toInt()))
                }

                    Log.d(TAG, "AuthenticateTransaction
CardData: $authTransDataHB")
                delay(3000)
                val completeTransCmd =
Client.CompleteTransactionAsync(
                connectedDeviceId!!, true, byteArrayOf(0x30,
0x30),
                issuerAuthData.hexToUByteArray(),
ubyteArrayOf(),
                transactionOutputData.hexToUByteArray())

    val completeTransCmdStatus =
completeTransCmd.waitForCompletion()
            val transactionData =
completeTransCmd.getResultData()
            outputInCoroutine("[StartTransaction
$transTypeText]",
            "CompleteTransaction :
${completeTransCmdStatus.name}")
              if (transactionData != null) {
                Log.d(TAG, "CompleteTransaction:
${completeTransCmdStatus.name}; "
                +"RespType: ${transactionData.respType.name}; "
+
                  "Attribute:${Integer.toHexString(
                    transactionData.attribute.toInt())};"
                    +"CardDataSize:
${transactionData.cardData.size}")

                val transHB = StringBuilder()
```

```
                for (uShort in transactionData.cardData) {
                transHB.append(String.format("%02X",
uShort.toInt()))}
                Log.d(TAG, "CompleteTransaction CardData:
$transHB")
            } else {
                val error = startTransCmd.getErrorInformation()
                Log.d(TAG, "CompleteTransaction:
${completeTransCmdStatus.name}; "
                +"ErrorCode: ${error?.ErrorCode}; "
+"ErrorMessage:
                ${error?.ErrorMessage}")

        val tracker = startTransCmd.cancelCommandAsync()
        val status = tracker.waitForCompletion().name
        outputInCoroutine("[CancelTransaction]", "Status :
$status")
        }
        } else {
        outputInCoroutine("[StartTransaction $transTypeText]",
        "AuthenticateTransaction TransactionData is NULL")
      return@launch
      }
     } else {
    val error = startTransCmd.getErrorInformation()
    outputInCoroutine("[StartTransaction $transTypeText]",
"ErrorMessage :
    ${error?.ErrorMessage}")
     Log.d(TAG, "StartTransaction ErrorCode:
${error?.ErrorCode}; " +
    "ErrorMessage: ${error?.ErrorMessage}")

    val tracker = startTransCmd.cancelCommandAsync()
    val status = tracker.waitForCompletion().name
    Log.d(TAG, "CancelTransaction: $status")
    return@launch
    }
     } else { return@launch}
     } else return@launch
        } }
```

6. **Cancel Transaction**
   Cancels a transaction.

```
Client.CancelTransactionAsync(connectedDeviceId)
```

7. **Update Result**

   Updates the transaction result.

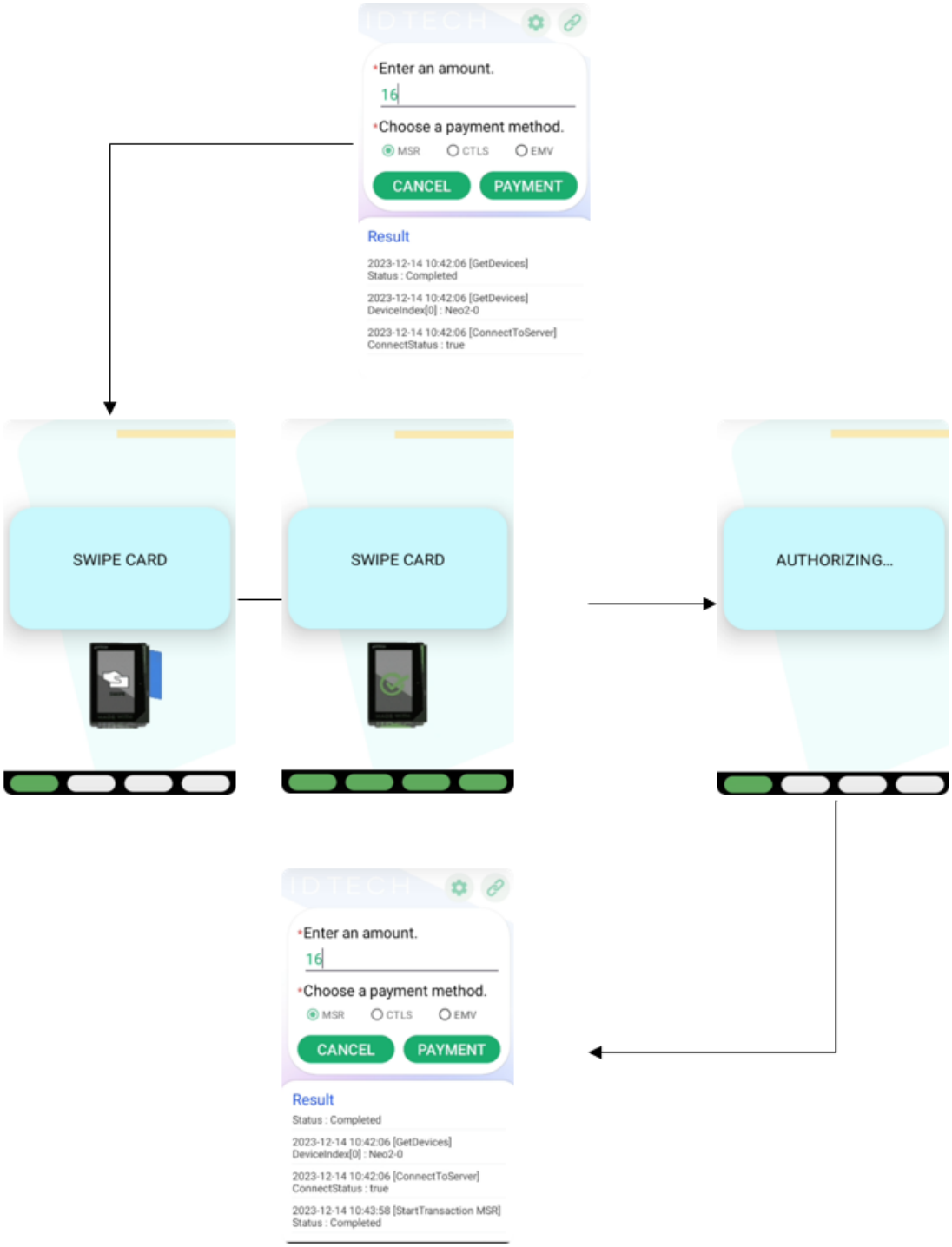
```
viewModel.outputs.observe(this) { list ->
            responseAdapter?.update(list)
            if (list.size > 0)
                rvResponse?.smoothScrollToPosition(list.size - 1)
        }
```

4. Transaction Flow Diagrams

The section below provides diagrams illustrating transaction flows.

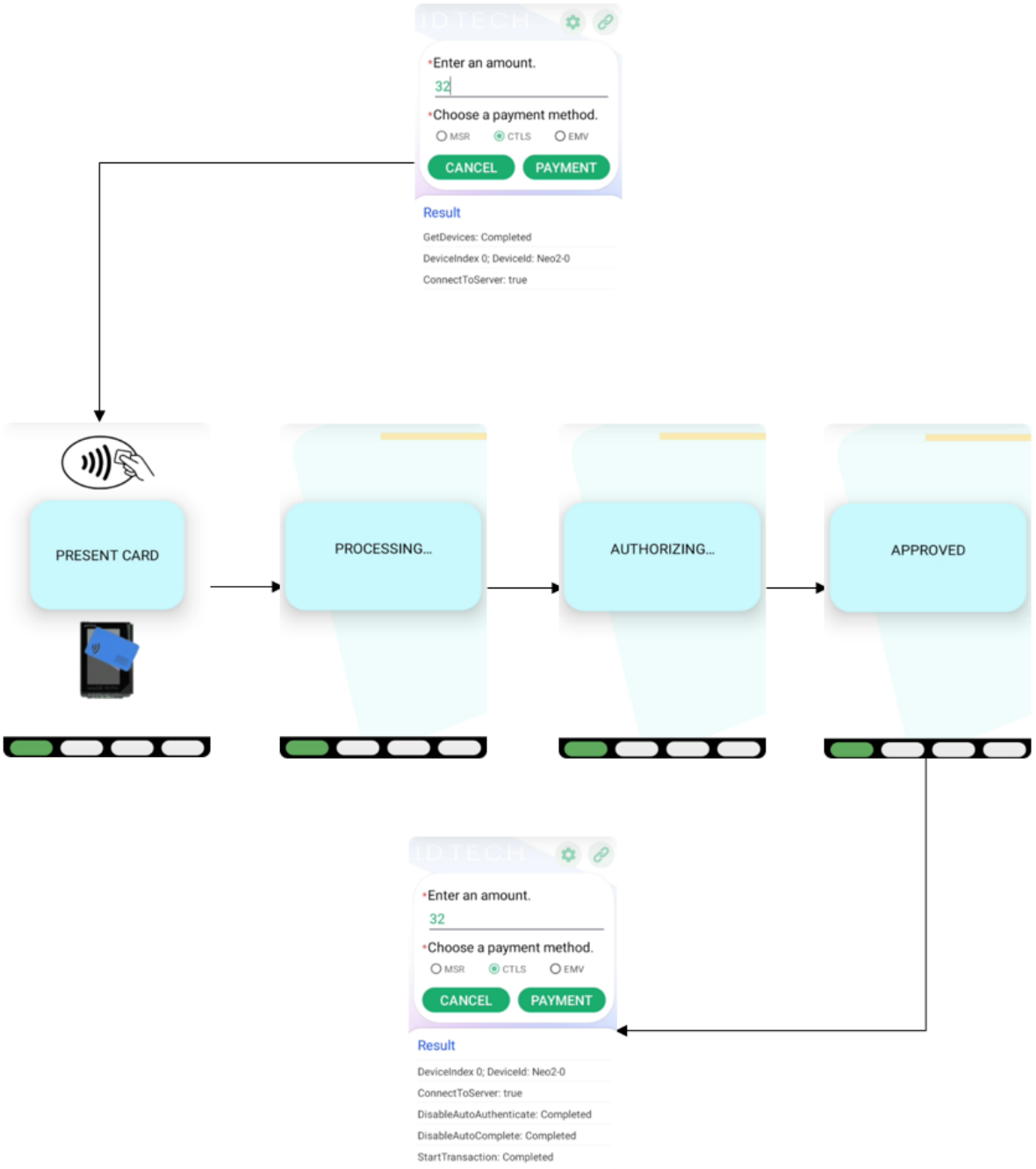
4.1. MSR Flow

The diagram below illustrates the MSR transaction flow.



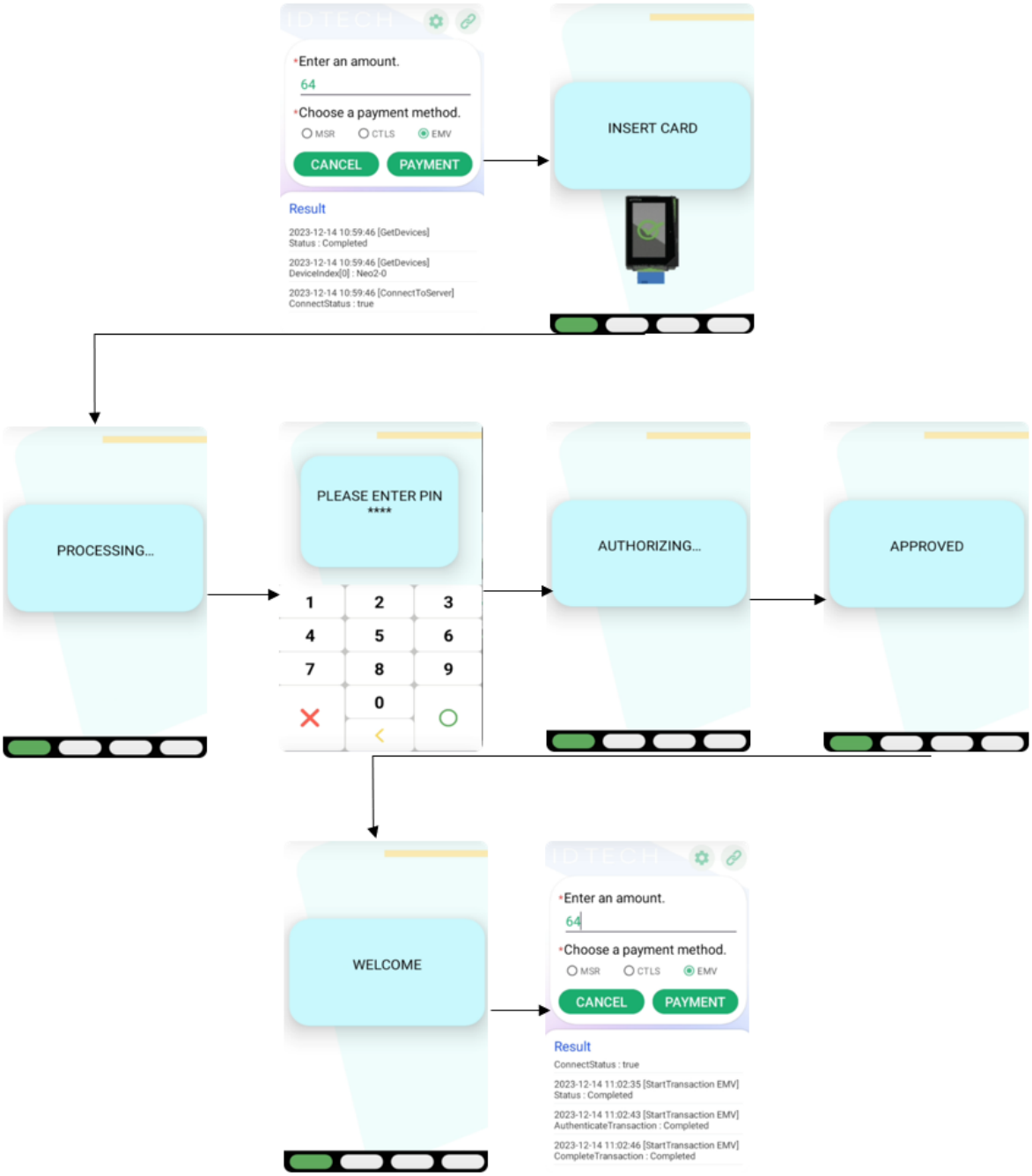
4.2. CTLS Flow

The diagram below illustrates the CTLS transaction flow.



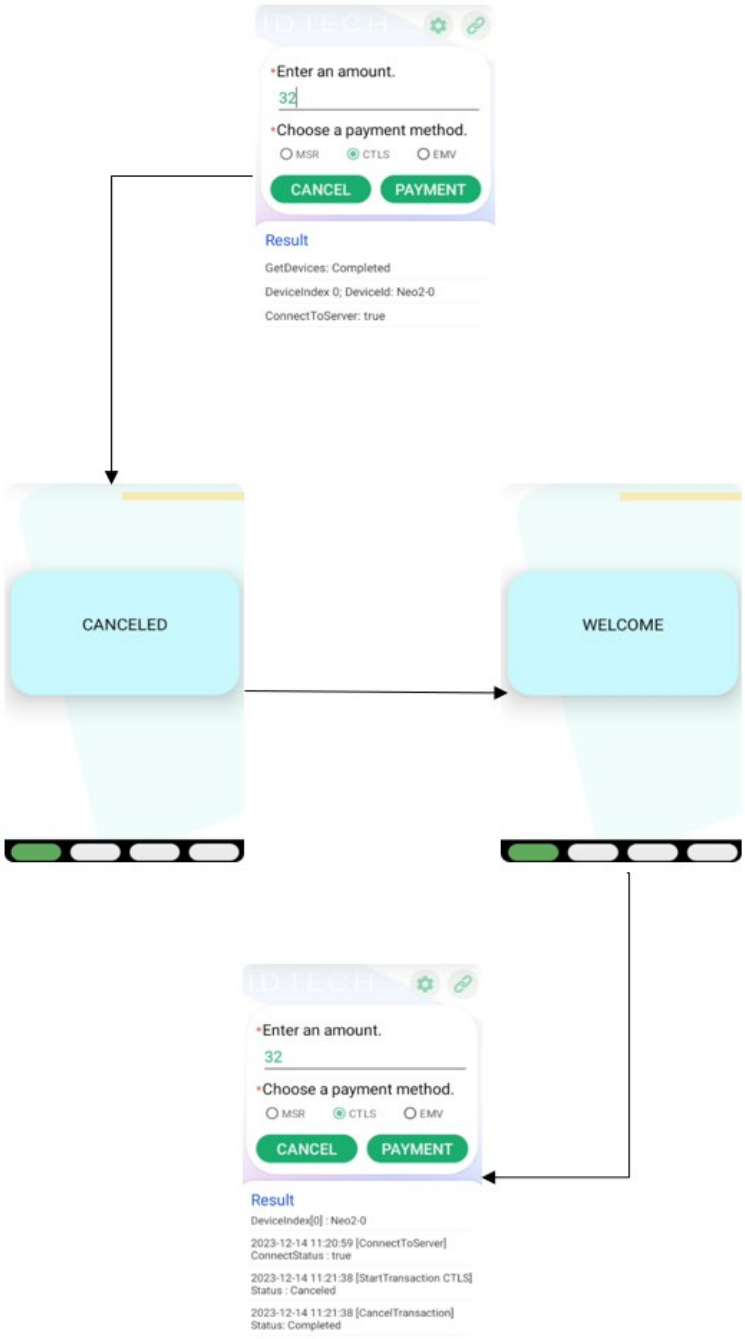
4.3. EMV Flow

The diagram below illustrates the EMV transaction flow.



4.4. Cancel Flow

The diagram below illustrates the Cancel transaction flow.

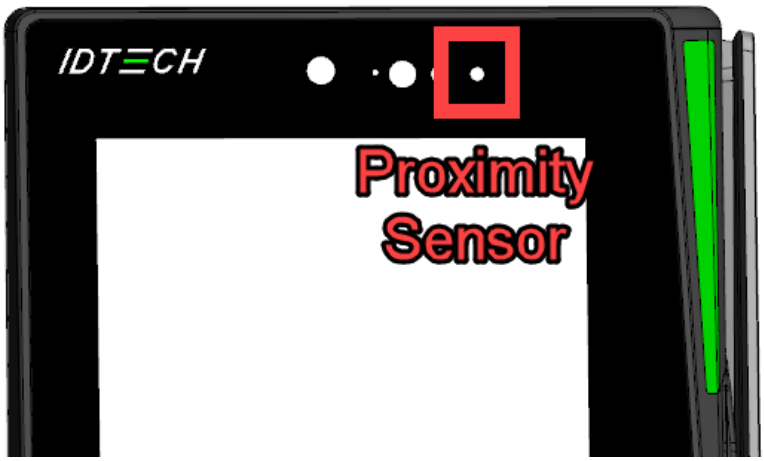


5. AP6800 Android SDK

The following section describes the Android SDK for AP6800 features.

5.1. Proximity Sensor

ID TECH AP6800 proximity sensor is located in the following image and marked as red circle. The proximity sensor determines the distance between the device and an object.



The following code sample retrieves an instance of the default proximity sensor:

```
private lateinit var sensorManager: SensorManager
private var proximity: Sensor? = null

…
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
proximity = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

Proximity sensors are often used to determine the distance between a user's head and the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance in cm.

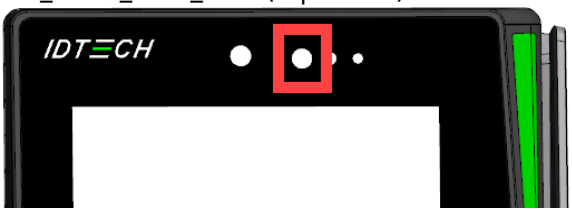
```
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var sensorManager: SensorManager
    private var proximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get
        // an instance of a particular sensor.
        sensorManager = getSystemService(Context.SENSOR_SERVICE) as
SensorManager
```

```
        proximity =
sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }

    override fun onSensorChanged(event: SensorEvent) {
        val distance = event.values[0]
        // Do something with this sensor data.
Log.d("ProximitySensor", "distance=$distance"
    }

    override fun onResume() {
        // Register a listener for the sensor.
        super.onResume()

        proximity?.also { proximity ->
            sensorManager.registerListener(this, proximity,
SensorManager.SENSOR_DELAY_NORMAL)
        }
    }

    override fun onPause() {
        // Be sure to unregister the sensor when the activity pauses.
        super.onPause()
        sensorManager.unregisterListener(this)
    }
}
```

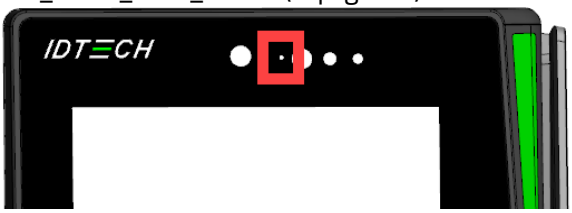
5.2. Update LED

ID TECH provides the LED SDK to let customers control the reader's LEDs.

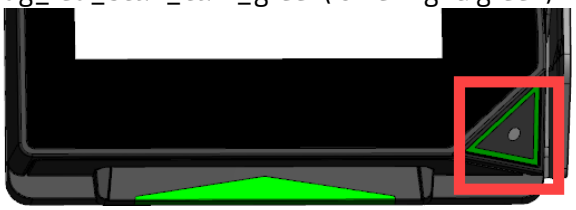
- led_front_cam_flash(top/white)



- led_front_cam_status(top/green)



- idg_led_scan_cam_white(lower right/white)
- idg_led_scan_cam_green(lower right/green)



The LED IDs are:

- `ledInfo=LedInfo(idx=1, name=front-cam:flash)`
- `ledInfo=LedInfo(idx=2, name=front-cam:status)`
- `ledInfo=LedInfo(idx=6, name=scan-cam:flash)`
- `ledInfo=LedInfo(idx=7, name=scan-cam:status)`

The ID TECH LED API provides the following AIDL interface:

```
// IPeripheralInterface.aidl
package com.idtech.peripheralmanager;

interface IPeripheralInterface {
    void setKioskLed(int idx, int onOff);
}
```

Add the LedUtils interface:

```
interface LedUtils {

   fun lightOn(context: Context, ledId: Int, lightLevel: Int)

}
```

Binding to the peripheral manager:

```
import com.idtech.peripheralmanager.IPeripheralInterface

class LedManager :  LedUtils  {

    private var peripheralManager: IPeripheralInterface? = null

    @WorkerThread
override fun lightOn(context: Context, ledId: Int, turnOn: Boolean) {
        try {
                        getPeripheralService(context)?.let {
peripheral->
If (turnOn)
            peripheral.setKioskLed(ledId, 255)
                  else
            peripheral.setKioskLed(ledId, 0)
                   }
        } catch (e: Exception) {
```

```
                Log.e(TAG, "lightOn: ", e)
        }
    }

    @WorkerThread
             private fun getPeripheralService(context: Context):
IPeripheralInterface? {
             if (peripheralManager == null) {
             try {
                   Log.d(TAG, "try to bind
com.idtech.peripheralmanager")
                   val intent =
Intent("android.intent.action.PeripheralManager")

intent.setPackage("com.idtech.peripheralmanager")
                   peripheralManager = runBlocking {
                       waitBindService(context.applicationContext,
intent)
                   }
                } catch (e: Exception) {
                   Log.e(TAG, "getPeripheralService: ", e)
                }
             }
              return peripheralManager
           }
private suspend fun waitBindService(context: Context, intent:
Intent): IPeripheralInterface? {
        return callbackFlow {
            conn = object : ServiceConnection {
                override fun onServiceConnected(name: ComponentName,
service: IBinder) {

this@callbackFlow.trySend(IPeripheralInterface.Stub.asInterface(servi
ce)).isSuccess
                }
                override fun onServiceDisconnected(name:
ComponentName) {
                    this@callbackFlow.trySend(null).isSuccess
                }
            }.also {
                context.bindService(intent, it,
Context.BIND_AUTO_CREATE)
            }
            awaitClose {}
        }.first()
    }

}
```

Sample code to turn **front-cam:flash LED** on or off:

```
private var ledManager: LedManager? = null

            ledManager = LedManager()
            GlobalScope.launch(Dispatchers.IO) {

                            ledManager?.lightOn(
                                context,
                                1,
                                true
                            )

                            delay(5000)
                            ledManager?.lightOn(
                                context,
                                1,
                                false
                            )

                        }
            }
```

5.3. Serial Port

ID TECH SerialPort API reads and writes data to the SerialPort in AP6800 readers.

SerialPort AIDL interface code:

```
package com.idtech.peripheralmanager;

interface IPeripheralInterface {
    int openSerialPort(String path, int baudrate);
    String readSerialPort(int handle, int size);
    int writeSerialPort(int handle, String data);
    int closeSerialPort(int handle);
}
```

Add the SerialPortUtils interface:

```
interface SerialPortUtils {

    fun openSerialPort(context: Context, path: String, baudrate:
Int): Int
    fun readSerialPort(context: Context,  handle: Int, size: Int):
String?
    fun writeSerialPort(context: Context, handle: Int, data:
String?): Int
    fun closeSerialPort(context: Context, handle: Int): Int

}
```

Binding to the peripheral manager:

```
class SerialPortManager : SerialPortUtils {
    companion object {
        private var TAG = SerialPortManager::class.java.simpleName
    }

    private var peripheralManager: IPeripheralInterface? = null
    private var conn: ServiceConnection? = null



    @WorkerThread
    override fun openSerialPort(context: Context, path: String,
baudrate: Int): Int {
        try {
            getPeripheralService(context)?.let { peripheral->
                val ret = peripheral.openSerialPort(path, baudrate)
                Log.d(TAG, "openSerialPort, path=$path,
baudrate=$baudrate, ret=$ret")
                return ret
            }
        } catch (e: Exception) {
            Log.e(TAG, "openSerialPort: ", e)
        }
        return 0
    }

    @WorkerThread
    override fun closeSerialPort(context: Context, handle: Int): Int
{
        try {
            getPeripheralService(context)?.let { peripheral->
                val ret = peripheral.closeSerialPort(handle)
                Log.d(TAG, "closeSerialPort, ret=$ret")
                return ret
            }
        } catch (e: Exception) {
            Log.e(TAG, "closeSerialPort: ", e)
        }
        return 0
    }
@WorkerThread
    override fun writeSerialPort(context: Context, handle: Int,
data:String?): Int {
        try {
            getPeripheralService(context)?.let { peripheral->
                val ret = peripheral.writeSerialPort(handle, data)
                Log.d(TAG, "writeSerialPort, ret=$ret")
                return ret
            }
        } catch (e: Exception) {
```

```
            Log.e(TAG, "closeSerialPort: ", e)
        }
        return 0
    }

    @WorkerThread
    override fun readSerialPort(context: Context, handle: Int, size:
Int): String {
        try {
            getPeripheralService(context)?.let { peripheral->
                val retString = peripheral.readSerialPort(handle,
size)
                Log.d(TAG, "readSerialPort, retString=$retString")
                return retString
            }
        } catch (e: Exception) {
            Log.e(TAG, "readSerialPort: ", e)
        }
        return ""
    }


    @WorkerThread
    private fun getPeripheralService(context: Context):
IPeripheralInterface? {
        if (peripheralManager == null) {
            try {
                Log.d(TAG, "try to bind
com.idtech.peripheralmanager")
                val intent =
Intent("android.intent.action.PeripheralManager")
                intent.setPackage("com.idtech.peripheralmanager")
                peripheralManager = runBlocking {
                    waitBindService(context.applicationContext,
intent)
                }
            } catch (e: Exception) {
                Log.e(TAG, "getPeripheralService: ", e)
            }
        }
        return peripheralManager
    }

@OptIn(ExperimentalCoroutinesApi::class)
    private suspend fun waitBindService(context: Context, intent:
Intent): IPeripheralInterface? {
        return callbackFlow {
            conn = object : ServiceConnection {
                override fun onServiceConnected(name: ComponentName,
service: IBinder) {
```

```
this@callbackFlow.trySend(IPeripheralInterface.Stub.asInterface(servi
ce)).isSuccess
                }
                override fun onServiceDisconnected(name:
ComponentName) {
                    this@callbackFlow.trySend(null).isSuccess
                }
            }.also {
                context.bindService(intent, it,
Context.BIND_AUTO_CREATE)
            }

            awaitClose {}

        }.first()
    }
```

Sample code for reading and writing to the SerialPort:

```
private var serialPortManager: SerialPortManager? = null
    serialPortManager = SerialPortManager()

    GlobalScope.launch(Dispatchers.IO) {

                        val handle =
serialPortManager?.openSerialPort(context, "/dev/ttyUSB4", 9600)
                        serialPortManager?.writeSerialPort(context,
handle!!, "Hello, world!!!")
                        val retString =
serialPortManager?.readSerialPort(context, handle!!,5)
                        Log.d(TAG, "readSerialPort=$retString")
    }
```

5.4. Camera

The AP6800 camera can interface with a camera application as a standard android device. AP6800 readers have two cameras. The front camera is located at the top-middle of the front side of the device. The scanner camera is located at the bottom right of the device.

To develop a camera app, follow the CameraX Android developer guide.

The following steps describe creating a camera preview app on AP6800 and are based on the guide linked above. Consult that documentation for more detail.

To implement a camera preview app, you will need to:

- Add camera libraries dependencies.
- Add camera permissions.
- Add camera preview.

5.4.1. Add Camera Dependencies

Open **build.gradle** and add the following dependencies:

```
dependencies {
  def camerax_version = "1.2.2"
  implementation "androidx.camera:camera-core:${camerax_version}"
  implementation "androidx.camera:camera-camera2:${camerax_version}"
  implementation "androidx.camera:camera-
lifecycle:${camerax_version}"
  implementation "androidx.camera:camera-video:${camerax_version}"
  implementation "androidx.camera:camera-view:${camerax_version}"
  implementation "androidx.camera:camera-
extensions:${camerax_version}"
}
```

Check the CameraX release notes page for the most recent CameraX version.

5.4.2. Request Camera Permissions

Open **AndroidManifest.xml** and add the following code:

```
<uses-feature android:name="android.hardware.camera.any" />
<uses-permission android:name="android.permission.CAMERA" />
```

In **MainActivity**, add the following code to request multiple permissions:

```
private val REQUIRED_PERMISSIONS =
            mutableListOf (
                Manifest.permission.CAMERA,
                // you can add more permissions here
            ).toTypedArray()


private val activityResultLauncher =
    registerForActivityResult(
        ActivityResultContracts.RequestMultiplePermissions())
    { permissions ->
        // Handle Permission granted/rejected
        var permissionGranted = true
        permissions.entries.forEach {
            if (it.key in REQUIRED_PERMISSIONS && it.value == false)
                permissionGranted = false
        }
// show toast to notify user that the permission request is denied.
        if (!permissionGranted) {
            Toast.makeText(baseContext,
                "Permission request denied",
                Toast.LENGTH_SHORT).show()
        } else {
            startCamera()
        }
    }
```

Be sure to request permission with:

```
activityResultLauncher.launch(REQUIRED_PERMISSIONS)
```

5.4.3. Add Camera Preview

Add the **PreviewView** class to your **MainActivity** layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.camera.view.PreviewView
        android:id="@+id/viewFinder"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Implement the Preview:

```
private fun startCamera() {
    val cameraController = LifecycleCameraController(baseContext)
    val previewView: PreviewView = viewBinding.viewFinder
    cameraController.bindToLifecycle(this)
    previewView.controller = cameraController
}
```

Execute **startCamera()** to start the camera preview.

5.5. QR Codes

To scan QR codes and barcodes, use Google's ML Kit, a free machine learning API for detecting faces and scanning QR codes and barcodes.

- The ML Kit: https://developers.google.com/ml-kit
- The ML Kit with barcode scanning: https://developers.google.com/ml-kit/vision/barcode-scanning

Android provides an easy way to integrate ML Kit with CameraX. Follow the ML Kit Analyzer and Barcode Scanning user guides to integrate ML Kit with CameraX. CameraX also offers official sample code.

This part of the sample project modifies the **startCamera()** function to scan a QR code from a camera preview.

Add ML Kit dependency to **build.gradle**. Choose one of the following to get ML Kit support.

- For bundling the model with your app:

```
dependencies {
  // Use this dependency to bundle the model with your app
  implementation 'com.google.mlkit:barcode-scanning:17.2.0'
}
```

- For using the model in Google Play Services:

```
dependencies {
  // Use this dependency to use the dynamically downloaded
model in Google Play Services
  implementation 'com.google.android.gms:play-services-mlkit-
barcode-scanning:18.3.0'
}
```

If using the Google Play model, add the following declarations to **AndroidManifest**:

```
<application ...>
      <meta-data
          android:name="com.google.mlkit.vision.DEPENDENCIES"
          android:value="barcode" >
      <!-- To use multiple models:
android:value="barcode,model2,model3" -->
</application>
```

Next, modify **startCamera()** with the following code:

```
private fun startCamera() {
        val cameraController = LifecycleCameraController(baseContext)
        val previewView: PreviewView = viewBinding.viewFinder

        val options = BarcodeScannerOptions.Builder()
            .setBarcodeFormats(Barcode.FORMAT_QR_CODE)
            .build()
        barcodeScanner = BarcodeScanning.getClient(options)

        cameraController.setImageAnalysisAnalyzer(
            ContextCompat.getMainExecutor(this),
            MlKitAnalyzer(
                listOf(barcodeScanner),
                COORDINATE_SYSTEM_VIEW_REFERENCED,
                ContextCompat.getMainExecutor(this)
            ) { result: MlKitAnalyzer.Result? ->
                val barcodeResults = result?.getValue(barcodeScanner)
                val barcodeValue =
barcodeResults?.getOrNull(0)?.displayValue
                val barcodeFormat =
barcodeResults?.getOrNull(0)?.format
                if (barcodeValue != null) {
                    // Get the scanned result!
```

```
                    Log.d(TAG, "startCamera:\nbarcodeValue:
$barcodeValue\nbarcodeFormat:$barcodeFormat")
                }
            }
        )

        cameraController.bindToLifecycle(this)
        previewView.controller = cameraController
}
```

Be sure to close the barcode scanner when the app finishes using it:

```
override fun onDestroy() {
    super.onDestroy()
    barcodeScanner.close()
}
```

With the QR code scanning app complete, users can show a QR code value to the device screen or link to a website with a valid URL. See the sample code for CameraX-MLKit for details on presenting visible QR code indicators to the screen when scanning.

5.6. Light Sensor

The AP6800's light sensor can retrieve the ambient light level via the Android sensor framework API. The light sensor is at the top front of the device, near the front camera. For more information about managing the light sensor, see the sensor framework page.

Sample code for getting the ambient light level:

```
class MainActivity : ComponentActivity(), SensorEventListener {
    private lateinit var sensorManager: SensorManager
    private var light: Sensor? = null
    private val lightLxMutable = mutableStateOf(0.0f)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme {
                Text(
                    text = lightLxMutable.value.toString(),
                    modifier = Modifier.fillMaxSize(),
                    color = Color.Black,
                    style = TextStyle(
                        fontSize = 48f.sp
                    )
                )
            }
        }

        init()
    }
```

```
    private fun init() {
        // Get an instance of the sensor service and use that to get
an instance of a particular sensor.
        sensorManager = getSystemService(Context.SENSOR_SERVICE) as
 SensorManager
        light = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }

    override fun onSensorChanged(event: SensorEvent) {
        // Do something with this sensor data.
        // values[0]: Ambient light level in SI lux units
        lightLxMutable.value = event.values[0]
    }

 override fun onResume() {
        // Register a listener for the sensor.
        super.onResume()
        sensorManager.registerListener(this, light,
 SensorManager.SENSOR_DELAY_NORMAL)
    }

    override fun onPause() {
        // Be sure to unregister the sensor when the activity pauses.
        super.onPause()
        sensorManager.unregisterListener(this)
    }

}
```

5.7. Network Connectivity

AP6800 readers can connect to Wi-Fi or ethernet for network access. The reader requires an extra hub to connect an ethernet cable. AP6800 readers also support LTE.

The reader can control the Wi-Fi or ethernet connection from local Android device UI or remotely via settings remotely via the ID TECH Remote Device Management service[1].

To download, send, or receive files via HTTP requests, implement network features on AP6800 readers as a standard Android device. For example, to download a file from a URL on an AP6800 reader, use the following code:

```
fun downloadFileFromUrl(urlString: String, file: File): File? {
    try {
        val url = URL(urlString)
        val urlConnection = url.openConnection()
```

[1] Forthcoming.

```
        urlConnection.getInputStream().use { inputStream ->
            file.outputStream().use { outputStream ->
                inputStream.copyTo(outputStream)
            }
        }
        return file
    } catch (e: Exception) {
        Log.e(TAG, "downloadFileFromUrl: ", e)
    }
    return null
}
```

6. For More Information

- To learn more about AP6800 readers, visit the ID TECH Knowledge Base.
- Visit us online at http://idtechproducts.com.
- Find more Tech Support resources at the ID TECH Tech Support home page.