Android SDK API Documentation

1. Project Integration

The **MetaBluetoothSDK.aar** library is a Bluetooth development framework for Android development. It provides a series of APIs and tools to help developers quickly connect and communicate with Bluetooth devices. This document will introduce how to integrate **the MetaBluetoothSDK.aar** library into your Android project:

1. Download **MetaBluetoothSDK.aar** library You can download **MetaBluetoothSDK.aar** from the official website of Mojie Open Platform Library files.

2. Add the **MetaBluetoothSDK.aar** library to your Android project. Add the **MetaBluetoothSDK.aar** library to your Android project.

Copy code

```
android
        { compileSdk 32
        defaultConfig { ndk { //
        Set
              the supported SO library architecture (developers can select one or more platforms as needed)
                 abiFilters "arm64-v8a"
        }
}
```

The minimum supported version of this SDK is Android 26, and it only supports arm64-v8a phones.

2. Initialization

Currently, the SDK is managed by **MetaGlassManager .** This class is a singleton. Please call **the getInstance(Application context)** method at the appropriate location to initialize it and return a **MetaGlassManager** object.

For example:

Copy code

```
private val metaGlassManager: MetaGlassManager init

{ metaGlassManager = MetaGlassManager.getInstance(applicationContext)
}
```

3. Scan Devices

Add a listener through **the MetaGlassManager** object **setMetaGlassScanListener(listener)** method. The listener method is:

Copy code

```
public interface MetaGlassScanListener { //Start scanning void

        onMetaGlassScanStarted(); //Discover the
        device void
        onMetaGlassScanScaned(MetaGlassDevice var1); //Scan completed void

        onMetaGlassScanFinished();
}
```

Start Scan

Call the MetaGlassManager object startScan() method to start scanning. Before starting scanning, make sure you have obtained

Manifest.permission.BLUETOOTH_CONNECT,

Manifest.permission.BLUETOOTH_SCAN,

Manifest.permission.ACCESS_FINE_LOCATION,

Manifest.permission.BLUETOOTH_ADVERTISE (target>=31) permission

End Scan

Call **the stopScan()** method of **the MetaGlassManager** object to end the scan.

4. Bind device

MetaGlassDevice object, which is the operation object of the glasses. All subsequent command information communication with the glasses is through this object.

object.

4.1. Connecting devices

1). Add a listener callback by listening to **the MetaGlassDevice** object **setMetaGlassConnectListener (MetaGlassConnectListener listener)** method

Copy code

```
public interface MetaGlassConnectListener extends Serializable {
        //bluetoothAddress : Device Bluetooth address:
        state //                         Device connection
        state void onMetaGlassConnectStateChange(String bluetoothAddress, MetaGlassState state); }
```

MetaGlassState

Copy code

```
//State
public enum State{
        DISCONNECT,
        CONNECTING,
        CONNECTED,
        CONNECT_FAILED,
}

//Error
public enum Error{
        //Connect Success
        NO_ERROR,
        //SE comparison failed
        ERROR_SE_CHECK,
        //SE value error
        ERROR_SE_VALUE,
        //SE binding error
        ERROR_SE_BIND, //
        Bluetooth abnormality
        ERROR_GATT, //
        Bluetooth service exception
        ERROR_SERVICE, //
        Connection timeout
        ERROR_TIMEOUT,

}


private final State state; private final
Error error;

public MetaGlassState(State state, Error error) {
        this.state = state; this.error
        = error;
}
```

2). Add the glasses-side command set response callback through **the MetaGlassDevice** object **setMetaGlassCmdRspMsgListener (MetaGlassCmdRspMsgListener listener)** method

Copy code

```
public interface MetaGlassCmdRspMsgListener { ÿDevice
    bluetoothAddress ÿ              Bluetooth address //
    //CmdRsp           Command set return object
    void onMetaGlassCmdRspMsg(String bluetoothAddress, CmdRsp rsp);
}
```

3). Add the glasses-side command set request callback through **the MetaGlassDevice** object **setMetaGlassCmdReqMsgListener (MetaGlassCmdReqMsgListener listener)** method

Copy code

```
public interface MetaGlassCmdReqMsgListener { ÿDevice
    bluetoothAddress ÿ              Bluetooth address //
    //CmdRsp           Command set request object
    void onMetaGlassCmdReqMsg(String bluetoothAddress, CmdReq req);
}
```

4.2 Get Token

First through the cloud http://api.cloud.meta-bounds.com/v1/terminals/check Get device token (see api cloud api documentation),

Call the **MetaGlassDevice** object **connect(ConnectParam param)** method to connect the glasses:

ConnectParams Copy code

```
//MetaSEToken :SE Token //
autoConnect : Whether to connect
Connection        automatically :
timeout (ms) //timeOut public ConnectParam(MetaSEToken token, boolean autoConnect, long timeOut)
    { this.token = token;
    this.autoConnect = autoConnect; this.timeOut
    = timeOut;
}
```

MetaSEToken Copy code

```
//key : device token //id :
user id
```

```
public MetaSEToken(int token, int userId) { this.key = key;
        this.id = id;

}
```

4.3 Binding Results

After successfully connecting to the device, send it through **the MetaGlassDevice** object **sendCmd (final MetaGlassCmd cmd)** method

The CmdReqBind command initiates binding. After the command is successfully sent, the glasses will jump to the confirmation page. After the user performs the operation, the device will report a

CmdReqBind request command, this command will have the following situations

```
when (req.status)
    { CmdError.STATUS_SUCCESS -> { //
        Bind Success //
        Upload cloud
    }

    CmdError.ERR_BIND_USER_REFUSE -> {
        //Bind Refuse
    }

    CmdError.ERR_BIND_USER_BINDED -> {
        //The device has already been bound
    }

    else -> {
        //Bind fail
    }
}
```

Note: Please save the device information immediately after binding is successful. It is recommended to save the relevant information through http:// api.cloud.meta-bounds.com/v1/terminals/bind The interface is uploaded to the cloud (see the cloud API documentation for details).

If the device has been bound, please follow the steps below to connect the device:

Get the **MetaGlassDevice** object through the **MetaGlassManager** object **getMetaGlassDevice (final String bluetoothAddress)** method , and then call **the MetaGlassDevice** object **connect (ConnectParam para m)** method to connect the glasses. Make sure that the connection parameters and binding parameter values are consistent. For details, see point 6.

5. Unbind the device

Step 1:

First call http://api.cloud.meta-bounds.com/v1/terminals/unbind Interface to unbind the cloud (see the cloud API documentation for details)

Step2:

Send the **CmdReqUnbind** command to unbind via **the MetaGlassDevice** object **sendCmd(final MetaGlassCmd cmd)** method

Note: Please make sure that the unbinding device is connected to the network and the glasses.

6. MetaGlassDevice class usage instructions

MetaGlassDevice is an abstract class. SDK has created a unique proxy class and all listeners are single listeners. In actual development, different listeners are used in many places. It is recommended to create a proxy class on the APP to uniformly manage all listeners for interface distribution, which is convenient for APP and The SDK is isolated. The following is an example:

MetaGlassDeviceGlobal Demo

Copy code

```
class MetaGlassDeviceGlobal(context: Context) : MetaGlassScanListener,
MetaGlassConnectListener,
        MetaGlassCmdReqMsgListener, MetaGlassCmdRspMsgListener,
MetaGlassMtuListener,
        MetaGlassUpdateFwListener, MetaGlassBtConnectListener {

        private val mContext = context private var
        metaGlassDeviceConnect: MetaGlassDevice? = null private val mMetaGlassManager:
        MetaGlassManager =
                MetaGlassManager.getInstance(mContext) private val mLock
        = Object() private val gson = Gson() //req
        response listener setprivate val
        mCmdReqListeners =
        ArrayList<GlassDeviceCmdReqListener>() //rsp response listener setprivate val mCmdRspListeners =

        ArrayList<GlassDeviceCmdRspListener>() //Version informationprivate val glassDeviceVersion =

        MutableLiveData<MetaGlassVersion>() //Configuration informationprivate val glassDeviceConfig =

        MutableLiveData<NewDeviceConfigBean>() //Brightness informationprivate val glassDeviceLight =

        MutableLiveData<DeviceLightBean>() //Volume informationprivate val glassDeviceVolume =

        MutableLiveData<DeviceVolumeBean>()
        //Battery
        informationprivate val glassDeviceBattery = MutableLiveData<MetaGlassBattery>() //Connection
        statusprivate
        val glassDeviceState = MutableLiveData<MetaGlassState>()
        //BT connection
        status private val glassDeviceBTState = MutableLiveData<MetaGlassState>()
        //OTA upgrade
        listening private var glassDeviceUpdateFwListener: MetaGlassUpdateFwListener? = null //MTU listening private var

        glassDeviceMtuListener: GlassDeviceMtuListener? = null

        private var mScheduledThreadPoolExecutor: ScheduledThreadPoolExecutor? = null


        companion object { private
            var mInstance: MetaGlassDeviceGlobal? = null private val lock = Any() fun
            getInstance(context: Context):
            MetaGlassDeviceGlobal {
                if (mInstance == null)
                    { synchronized(lock) { if
                        (mInstance == null) {
                            mInstance = MetaGlassDeviceGlobal(context)
                        }
                    }
                }
                return mInstance!!
            }
        }


        private var mGlassDeviceScanListener: GlassDeviceScanListener? = null

        //Get the constructed device
        object fun getMetaGlass(): MetaGlassDevice? { return
            metaGlassDeviceConnect
        }

        //Construct scan
        listener fun build()
            { mMetaGlassManager.setMetaGlassScanListener(this)
        }

        //Set the scan
        listener fun setOnGlassDeviceScanListener(owner: LifecycleOwner, listener: GlassDeviceScanListener?)
{
```

```
            if (owner.lifecycle.currentState == Lifecycle.State.DESTROYED) {
                return
            }
            this.mGlassDeviceScanListener = listener
            mMetaGlassManager.setMetaGlassScanListener(this)
            owner.lifecycle.addObserver(object : LifecycleEventObserver { override fun
                onStateChanged(source: LifecycleOwner, event:
Lifecycle.Event) {
                    if (source.lifecycle.currentState ==
Lifecycle.State.DESTROYED)
                        { owner.lifecycle.removeObserver(this)
                        this@MetaGlassDeviceGlobal.mGlassDeviceScanListener = null
                        mMetaGlassManager.setMetaGlassScanListener(null)
                        stopScan()
                    }
                }
            })


        } //Start
        scanning fun
            startScan() { mMetaGlassManager.startScan()
        }

        //Stop
        scanning fun
            stopScan() { mMetaGlassManager.stopScan()

        } //Has the device
        been constructed? fun isBuildDevice(bleAddress: String?):
            Boolean { if (bleAddress == null || metaGlassDeviceConnect == null) {
                return false
            }
            return bleAddress == metaGlassDeviceConnect?.address
        }

        //Bind
        device fun buildGlass(address: String, name: String?)
            { synchronized(mLock) { if
                (address.isEmpty()) { throw
                    NullPointerException("BLE Address is not null")
                }


                if (metaGlassDeviceConnect == null) {
```

```
                                metaGlassDeviceConnect =
mMetaGlassManager.getMetaGlassDevice(address)

                        } metaGlassDeviceConnect?.setMetaGlassCmdReqMsgListener(this)
                        metaGlassDeviceConnect?.setMetaGlassCmdRspMsgListener(this)
                        metaGlassDeviceConnect?.setMetaGlassConnectListener(this)
                        metaGlassDeviceConnect?.setMetaGlassBtConnectListener(this)
                }

        }

        //Unbind the
        device fun unBindGlass()
                { synchronized(mLock)
                        { metaGlassDeviceConnect?.disconnect()
                        metaGlassDeviceConnect?.setMetaGlassCmdReqMsgListener(null)
                        metaGlassDeviceConnect?.setMetaGlassCmdRspMsgListener(null)
                        metaGlassDeviceConnect?.setMetaGlassConnectListener(null)
                        metaGlassDeviceConnect?.setMetaGlassBtConnectListener(null) metaGlassDeviceConnect = null

                }

        } //Remove the BT pairing relationship
        between the APP and the
                glasses fun unpairDevice() { synchronized(mLock) {
                        metaGlassDeviceConnect?.unpairDevice()
                }
        }

        //Scan start
        callback override fun onMetaGlassScanStarted()
                { mGlassDeviceScanListener?.onGlassDeviceScanStarted()

        } //Scan result
        callback override fun onMetaGlassScanScaned(p0: MetaGlassDevice?) {
                p0?.let
                        { mGlassDeviceScanListener?.onGlassDeviceScaned(it)
                }

        } //Scan
        completion callback override fun
                onMetaGlassScanFinished() { mGlassDeviceScanListener?.onGlassDeviceScanFinished()

        } // Glasses connection status callback
        override fun onMetaGlassConnectStateChange(p0: String, p1: MetaGlassState)
{
                glassDeviceState.postValue(p1) if (!
                p1.isConnected)
                        { glassDeviceBattery.postValue(MetaGlassBattery(0, 0)) } else { //
                Initialize
                        repeatable data sending task
                        initSendReplaceCommDataTask() //
                        Connect BT
                        Bluetooth enableBtConnect(true)
                }

        } //Glass-side Req
        request callback override fun onMetaGlassCmdReqMsg(p0: String, p1:
                CmdReq<*>)
                { doCmdReq(p0, p1) doBaseCmdReq(p0, p1)

        } // Glasses-side Rsp
        response callback override fun onMetaGlassCmdRspMsg(p0: String, p1: CmdRsp<*>) {
                MainScope().launch
                        { doBaseCmdRsp(p0, p1)
                }
        }
        //MTU value
        change callback override fun onMtuChanged(p0: String?, p1: Int, p2: Int)
                { glassDeviceMtuListener?.onMtuChanged(p0, p1, p2)

        } //Start OTA upgrade
        callback override fun onUpdateFw(p0: String?, p1: TaskStatus)
                { glassDeviceUpdateFwListener?.onUpdateFw(p0, p1)
        }


        //Start
        connecting fun connect(token: Long, userid: Long, autoConnect: Boolean, timeout:
Long)
                if (isConnected()) {
                        return
                }
                val sekey = MetaSEToken(token.toInt(), userid.toInt())
                metaGlassDeviceConnect?.connect( ConnectParam( sekey,
                        autoConnect, timeout
                )
            )
        }

        //Disconnect
        fun disConnect()



            { glassDeviceState.postValue( MetaGlassState( MetaGlassState.State.DISCONNECT, MetaGlassState.Error.N
                )

            ) glassDeviceConfig.value?.run {
                enableDND = false
                enableWeather = false


            } metaGlassDeviceConnect?.disconnect()
        }

        //Send
        command fun sendCmd(cmd: MetaGlassCmd<*>) {
            metaGlassDeviceConnect?.sendCmd(cmd)
        }

        //Process the command
        reported by the glasses private fun doBaseCmdReq(address: String, p1: CmdReq<*>) {
            when (p1.type)
                { CmdType.CMD_AUTOCUE_START_SPEECH -> { //
                    The glasses report the start speech command
                }

                CmdType.CMD_NAVIGATION_ENTER ->
                    { sendCmd(CmdRspStatus(CmdType.CMD_NAVIGATION_ENTER,
    CmdError.STATUS_SUCCESS)
                      //The glasses report the start navigation command
                }

                CmdType.CMD_GET_BATT_INFO -> { //
                    The glasses report
                    battery information parserBattery((p1 as CmdReqGlassBattInfo).value)
                }

                CmdType.CMD_PAIR_STATE_SYNC -> { //
                    The glasses report the BT
                    connection pairing command val result = (p1 as CmdReqPairStateSync).value
                    if (result)
                        { enableBtConnect(true)
                    }
                }

                else -> {

                }
            }
        }

        //Processing
        basic commands private fun doBaseCmdRsp(address: String, p1:
            CmdRsp<*>) { if (p1.status == CmdError.STATUS_SUCCESS) {
                when (p1.type)
                    { CmdType.CMD_GET_CONFIG
                        -> { val configRsp = p1 as CmdRspGetConfig
                        parserConfig(configRsp.value)
                    }

                    CmdType.CMD_GET_VERSION ->
                        { val versionRsp = p1 as CmdRspVersion
                        glassDeviceVersion.postValue(versionRsp.value)
                    }

                    CmdType.CMD_GET_BRIGHTNESS_VALUE
                        -> { val versionRsp = p1 as CmdRspGetBrightnessValue
                        parserLight(versionRsp)
                    }

                    CmdType.CMD_VOLUME_GET_VAL
                        -> { val volumeRsp = p1 as CmdRspGetVolumeValue
                        parserVolume(volumeRsp)
                    }

                    CmdType.CMD_GET_BATT_INFO ->
                        { parserBattery((p1 as CmdRspGetBattInfo).value)
                    }

                    else -> {

                    }
                }

            } doCmdRsp(address, p1)
```

```
}

//Parse
configuration information private fun parserConfig(mgConfig: MetaGlassConfig) {
        synchronized(mLock) {
                var config = glassDeviceConfig.value if (config
                == null) { config =
                        NewDeviceConfigBean()

                } config.enableDND = mgConfig.isEnableDND
                config.enableSMS = mgConfig.isEnableSMS
                config.enableCallNotification = mgConfig.isEnableCallNoti
                config.enableWeather = mgConfig.isEnableWeather
                glassDeviceConfig.postValue(config)
        }
}

//Parse
brightness data private fun parserLight(value: CmdRspGetBrightnessValue)
        { synchronized(mLock) {
                val lightBean = DeviceLightBean(value.value.toInt())
                glassDeviceLight.postValue(lightBean)
        }
}

//Parse volume
data private fun parserVolume(value: CmdRspGetVolumeValue)
        { synchronized(mLock) {
                val volumeBean = DeviceVolumeBean(value.value.toInt())
                glassDeviceVolume.postValue(volumeBean)
        }
}

//Parse power
data private fun parserBattery(mMetaGlassBattery: MetaGlassBattery)
        { glassDeviceBattery.postValue(mMetaGlassBattery)
}

//Process the
response command private fun doCmdRsp(address: String, p1: CmdRsp<*>) {
        log("doCmdRsp:${gson.toJson(p1)}")
        mCmdRspListeners.forEach
                { it.onGlassDeviceCmdRsp(address, p1)
        }
}
```

```
        //Process request
        command private fun doCmdReq(address: String, p1: CmdReq<*>) {
                log("doCmdReq:${gson.toJson(p1)}")
                mCmdReqListeners.forEach
                        { it.onGlassDeviceCmdReq(address, p1)
                }
        }

        //Set OTA
        upgrade fun setMetaGlassUpdateFwListener(file: String, listener:
MetaGlassUpdateFwListener?)
                { this.glassDeviceUpdateFwListener = listener
                metaGlassDeviceConnect?.startUpdateFw(file, this)
        }

        //Set the task of transmitting
        multimedia data fun setMetaGlassCommDataListener(commData: CommData, listener:
MetaGlassCommDataListener?)
                { metaGlassDeviceConnect?.startSendCommData(commData, listener)
        }

        //Set the task of transmitting
        multimedia data fun setMetaGlassReplaceCommDataListener(
                listener: MetaGlassReplaceCommDataListener
        ) {
                metaGlassDeviceConnect?.setSendReplaceCommDataListener(listener)

        } //Initialize large data
        transmission fun initSendReplaceCommDataTask()
                { metaGlassDeviceConnect?.initSendReplaceCommDataTask()

        } //Start large data
        transmission fun


        startSendReplaceCommData( p0: CommData ): StatusRet? { return metaGlassDeviceConnect?.startSendReplaceCo
        }

        //Stop OTA
        upgrade fun cancelUpdateFw() {
                metaGlassDeviceConnect?.stopUpdateFw()
                glassDeviceUpdateFwListener = null
        }
```

```
//Stop the large data
transmission task fun
        stopSendCommData() { metaGlassDeviceConnect?.stopSendCommData()
}

//Stop the large data
transmission task fun
        stopSendReplaceCommData() { metaGlassDeviceConnect?.stopSendReplaceCommData()
}

//Add cyclic
task fun addScheduledTask(task: CmdTask)
        { synchronized(mLock) { if
                (mScheduledThreadPoolExecutor != null) {
                        mScheduledThreadPoolExecutor?.shutdownNow()

                } mScheduledThreadPoolExecutor = ScheduledThreadPoolExecutor(1)
                mScheduledThreadPoolExecutor?.scheduleAtFixedRate( Runnable
                        { sendCmd(task.cmd) }, task.initialDelay,
                        task.period,

                        task.timeUnit
                )
        }
}


//Stop the loop
task fun stopScheduledTask()
        { synchronized(mLock) { if
                (mScheduledThreadPoolExecutor != null) {
                        mScheduledThreadPoolExecutor?.shutdownNow()
                        mScheduledThreadPoolExecutor = null
                }
        }
}


//Add device version
monitoring fun addGlassDeviceVersionObs(owner: LifecycleOwner, observer:
Observer<MetaGlassVersion>)
        { glassDeviceVersion.observe(owner, observer)
}
```

```
    //Add device
    configuration listener fun addGlassDeviceConfigObs(owner: LifecycleOwner, observer:
Observer<NewDeviceConfigBean>)
        { glassDeviceConfig.observe(owner, observer)
    }

    //Add device
    configuration listener fun addGlassDeviceConfigObs(observer: Observer<NewDeviceConfigBean>) {
        glassDeviceConfig.observeForever
            { observer.onChanged(it)
        }
    }

    //Add device battery
    monitoring fun addGlassDeviceBatteryObs(owner: LifecycleOwner, observer:
Observer<MetaGlassBattery>)
        { glassDeviceBattery.observe(owner, observer)
    }

    //Add brightness
    monitoring fun addGlassDeviceLightObs(owner: LifecycleOwner, observer:
Observer<DeviceLightBean>)
        { glassDeviceLight.observe(owner, observer)

   } //Add brightness
    monitoring fun getGlassDeviceLightObs(): DeviceLightBean?
        { return glassDeviceLight.value

    } //Add volume
    monitoring fun addGlassDeviceVolumeObs(owner: LifecycleOwner, observer:
Observer<DeviceVolumeBean>)
        { glassDeviceVolume.observe(owner, observer)

    } //Add volume
    monitoring fun getGlassDeviceVolumeObs(): DeviceVolumeBean?
        { return glassDeviceVolume.value
    }

    //Add device version
    monitoring fun getGlassDeviceVersionObs(): MetaGlassVersion?
        { return glassDeviceVersion.value
    }

    //Add device
    configuration listener fun getGlassDeviceConfigObs(): NewDeviceConfigBean? {
          return glassDeviceConfig.value
      }

      //Add device power
      monitoring fun getGlassDeviceBatteryObs(): MetaGlassBattery?
          { return glassDeviceBattery.value
      }

      //Get BT Bluetooth
      address fun getBTAddress(): String {
          return metaGlassDeviceConnect?.btAddress ?: ""
      }

      //Get BT Bluetooth
      address fun getBLEAddress(): String {
          return metaGlassDeviceConnect?.address ?: ""

      } //Judge whether to
      connect to the device fun
          isConnected(): Boolean { if (metaGlassDeviceConnect == null) {
              return false


          } return glassDeviceState.value?.isConnected == true
      }


      //Get the device boot
      mode fun getBootMode(): BootParam? {
          return metaGlassDeviceConnect?.bootParam

      } //Get BT
      status fun getBTState(): MetaGlassState {

          val temp = metaGlassDeviceConnect?.btState ?:
              MetaGlassState(MetaGlassState.State.CONNECT_FAILED)
          glassDeviceBTState.postValue(temp)

          return temp
      }

      //Add device state
      monitoring fun addGlassDeviceStateObs(owner: LifecycleOwner, observer:
  Observer<MetaGlassState>) {
          glassDeviceState.observe(owner, observer)
      }
```

```
//Add device status
monitoring fun getGlassDeviceState(): MutableLiveData<MetaGlassState>
        { return glassDeviceState
}

//Add Req
listener fun addGlassDeviceCmdReqListener(listener: GlassDeviceCmdReqListener) { if (!
        mCmdReqListeners.contains(listener))
                { mCmdReqListeners.add(listener)
        }

} //Remove Req
listener fun removeGlassDeviceCmdReqListener(listener: GlassDeviceCmdReqListener) {
        if (mCmdReqListeners.contains(listener))
                { mCmdReqListeners.remove(listener)
        }
}

//Add Rsp
listener fun addGlassDeviceCmdRspListener(listener: GlassDeviceCmdRspListener) { if (!
        mCmdRspListeners.contains(listener))
                { mCmdRspListeners.add(listener)
        }

} //Remove Rsp
listener fun removeGlassDeviceCmdRspListener(listener: GlassDeviceCmdRspListener) {
        if (mCmdRspListeners.contains(listener))
                { mCmdRspListeners.remove(listener)
        }

} //Add data
stream listening fun setMetaGlassStreamListener(listener: MetaGlassStreamListener?)
        { metaGlassDeviceConnect?.setMetaGlassStreamListener(listener)

} //Add SPP
connection listener fun connectSPP(listener: MetaGlassSPPConnectListener?)
        { metaGlassDeviceConnect?.connectSPP(listener)

} //Close the
SPP connection fun
        closeSPP() { metaGlassDeviceConnect?.closeSPP()

} //Judge whether SPP
is connected fun isSPPConnect(): Boolean {
        return metaGlassDeviceConnect?.sppState?.isConnected == true

    } //Judge whether BT
    is connected fun isBTConnect(): Boolean {
        return glassDeviceBTState.value?.isConnected ?: false
    }

    //Add BT status
    monitoring fun addGlassDeviceBTObs(owner: LifecycleOwner, observer:
Observer<MetaGlassState>) {
        glassDeviceBTState.observe(owner, observer)

    } //Get BT connection
    status fun getGlassDeviceBTState(): MutableLiveData<MetaGlassState> {
        return glassDeviceBTState
    }

    //Connect BT
    Bluetooth fun enableBtConnect(boolean: Boolean)
        { metaGlassDeviceConnect?.enableBtConnect(boolean)
    }

    //Get SDK version
    information fun getSDKVersion(): String {
        return metaGlassDeviceConnect?.sdkVersion ?: ""

    } //Get the SEKey of
    the glasses fun getSEKey(): Int {
        return metaGlassDeviceConnect?.seKey ?: 0

    } //Monitor the BT state
    change of the glasses end override fun onMetaGlassBtConnectStateChange(p0:
String?, p1:
        MetaGlassState) { glassDeviceBTState.postValue(p1)

    } //Start reading the
    glasses end log fun startReadLog(listener: MetaGlassReadLogListener, path: String)
        { metaGlassDeviceConnect?.startReadLog(listener, path)
    }

    //Stop reading the
    glasses end log fun
        stopReadLog() { metaGlassDeviceConnect?.stopReadLog()
    }
```

```
        //Start reading the glasses file
        fun startReadFile(listener: MetaGlassReadFileListener, path: String, p2:
MetaBinType) {
                metaGlassDeviceConnect?.startReadFile(listener, path, p2)

        } //Stop reading the glasses
        end file fun stopReadFile()
                { metaGlassDeviceConnect?.stopReadFile()
        }
}
```

The following is a detailed description of the main methods:

buildGlass builds glasses object

connect to the glasses, the parameters must be consistent with the bound glasses

sendCmd(cmd: MetaGlassCmd<*>) All commands are sent to the glasses through this method, including **CmdReq** command and **CmdRsp** instruction

onMetaGlassCmdReqMsg callback for all glasses to report commands to the APP

onMetaGlassCmdRspMsg callback of all glasses responding to instructions to APP

enableBtConnect Enable or disable BT connection

setMetaGlassReplaceCommDataListener adds a large data transmission class whose parameters are **CommData** subclasses.

setMetaGlassUpdateFwListener starts OTA upgrade, the parameter is the local file path

connectSPP connects to SPP, mainly used for audio stream acquisition, needs to be used with **setMetaGlassStreamListener**

Please make sure that you have connected to BT Bluetooth before use.

setMetaGlassStreamListener gets the audio stream callback

disConnect Disconnect from the device

unBindGlass cancels the pairing relationship

unpairDevice cancels the pairing relationship between the mobile phone and the glasses. If the reconnection or binding fails, please call this method.

Or you can use the following code to cancel the pairing relationship before scanning and then connect:

Copy code

```
//Unpair the Bluetooth pairing between
the phone and the glasses private fun unpair(address:
        String)
                { try { val mDevice: BluetoothDevice? =
                        (requireContext().getSystemService(Context.BLUETOOTH_SERVICE) as
BluetoothManager).adapter.getRemoteDevice(
                        address

            ) mDevice?.let { val
                removeBondMethod = it.javaClass.getMethod("removeBond") removeBondMethod.invoke(it)

            }
        } catch (e: Exception) { e.printStackTrace()

        }

}
```

7. Audio Acquisition

The premise of obtaining audio data is to maintain BT Bluetooth connection.

7.1 Connect to BT Bluetooth

Connect to BT by calling **the enableBtConnect** method of **MetaGlassDeviceGlobal .**

Note: Due to the Android system, some mobile phones may not respond to the BT connection after connecting to BLE. You need to wait 20 seconds to reconnect. If the BT Bluetooth is already connected, Step 1 can be omitted.

7.2 Connecting SPP

Call the **connectSPP** method through **MetaGlassDeviceGlobal** to connect to SPP.

7.3 Setting up audio stream monitoring

Call **the setMetaGlassStreamListener** method through **MetaGlassDeviceGlobal** to set the audio stream listening callback.

7.4 Control the MIC switch of the glasses

Send **CmdReqEnableMIC** command to turn on/off the MIC on the glasses

Note: If the audio stream has been obtained, Step 4 can be omitted.

8. Instruction Set

Note: All commands are bidirectional, that is, the APP can actively send commands to the glasses, and the glasses can also send commands to the APP. All commands can be added to listeners through **addGlassDeviceCmdReqListener** and **addGlassDeviceCmd RspListener** of **MetaGlassDeviceGlobal ,** and removed through **removeGlassDeviceCmdReqListener** and **removeGlassDeviceCmdRspListener .**

Instruction base class

The **CmdReq** class is the base class for all requests, and all request objects inherit from this class.

CmdReq

Copy code

```
/** *
  Construction
  method * @param type: instruction
  type * @param value: generic entity
  */
public CmdReq(CmdType type, T value){} /** * Constructor
*
  @param type:
  instruction type * @param value:
  generic entity * @param period:
  timeout duration */

public CmdReq(CmdType type, T value, long period) {}

/** *
  Abstract method *
  Subclass implementation, assemble data according to protocol
  specifications */
public abstract byte[] buildBytes();
```

The **CmdRsp** class is the base class for all responses, and all response objects inherit from this class.

Copy code

```
/** *
  Construction
  method * @param type: instruction
  type * @param status: response error
  code * @param period: response generic
  entity */
public CmdRsp(CmdType type, final CmdError status, T value) {} /** * Constructor * @param type:

instruction type
  * @param status: response data
  * @param period: response generic
  entity */

public CmdRsp(CmdType type, byte[] byes){}

/** *
  Abstract method
  * Subclass implementation, parse data according to
  protocol specification * @return T: generic entity
```

```
 */
public abstract T fromBytes(byte[] bytes);
```

The **CmdRspStatus** class is a response class with no return value.

CmdRspStatus   Copy code

```
/** *
  Construction
  method * @param type: instruction
  type * @param status: response error
  code */
public CmdRspStatus(CmdType type, CmdError status) {} /** * Constructor * @param
type:
  command type
  * @param status: response data */


public CmdRspStatus(CmdType type, byte[] byes) {}
```

Device Binding

Device binding request **CmdReqBind**

Device Binding Response: **CmdRspStatus**

Device unbinding request **CmdReqUnbind**

Device unbinding response: **CmdRspStatus**

Device Information

Get version number request **CmdReqGetVersion**

Get the version number response **CmdRspVersion ,** which contains a **MetaGlassVersion ,** which is the version number entity

MetaGlassVersion   Copy code

```
public class MetaGlassVersion { final private short
        major;//Major version number final private short minor;//
        Minor version number final private short revision;//Final
        version number final private short reserve;//Reserve font


}
```

Get device power request **CmdReqGetBattInfo**

Get the device power response **CmdRspGetBattInfo ,** the response value is **MetaGlassBattery ,** which is the power entity

The device actively generates power **CmdReqGlassBattInfo ,** and the response value is **MetaGlassBattery ,** which is the power entity

MetaGlassBattery Copy code

```
public class MetaGlassBattery implements Serializable { private final short
        value;//Battery power private final byte state;//
        Battery status
}
```

Get Bluetooth address **CmdReqGetBtAddress**

Get Bluetooth address response **CmdRspGetBtAddress ,** the response value **MacAddress** is the Bluetooth address entity

MacAddress Copy code

```
public class MacAddress { private
        final byte[] address; //Bluetooth address data
}
```

Query boot mode request **CmdReqGetBootMode**

Query boot mode response **CmdRspGetBootMode ,** response value **BootParam** , **BootParam** contains **BootMod e**

BootParam Copy code

```
public class BootParam { private
        final BootMode masterMode;//Loda private final BootMode
        slaveMode;//AP4
}
```

BootMode Copy code

```
public enum BootMode {
        BOOTNULL((byte)0x00), //No boot mode
        BOOTSYSTEM((byte)0x01), //Normal boot mode
        BOOTLOADER((byte)0x02),; //BootLoader startup mode
}
```

Get device brightness mode request **CmdReqGetBrightnessMode**

Get the device brightness mode response **CmdRspGetBrightnessMode ,** the response value **is BrightnessMode**

BrightnessMode Copy code

```
public enum BrightnessMode {
        BRIGHTNESS_AUTO((byte)0x00),
        BRIGHTNESS_MANUAL((byte)0x01),;
}
```

Get device brightness value request **CmdReqGetBrightnessValue**

Get the device brightness value response **CmdRspGetBrightnessValue ,** response value **Byte**

Query Bluetooth name request **CmdReqGetBtName**

Query Bluetooth name response **CmdRspGetBtName ,** response value **String**

Query version number list request **CmdReqGetVersionList**

Query version number list response **CmdRspGetVersionList ,** response value **Map<Integer, MetaGlassVersion>**

MetaGlassVersion Copy code

```
public class MetaGlassVersion { final private short
        major;//Major version number final private short minor;//
        Minor version number final private short revision;//Final
        version number final private short reserve;//Reserve font


}
```

Query device type request **CmdReqGetDevType**

Query device type response **CmdRspGetDevType ,** response value **MetaGlassType**

MetaGlassType Copy code

```
public enum MetaGlassType {
        MG_TYPE_UNKNOWN(0),
        MG_TYPE_1001 (0x10010000),
        MG_TYPE_1002 (0x10020000),
        MG_TYPE_1003 (0x10030000),
        MG_TYPE_1004 (0x10040000),
        MG_TYPE_1003_OPPO(0x10030001);
}
```

Query the device's automatic screen off time request **CmdReqGetScreenOffTime**

Query the device's automatic screen off time response **CmdRspGetScreenOffTime ,** the response value **is Integer**

Query low power handling mode request **CmdReqGetLowPowerOpt**

Query low power handling mode response **CmdRspGetLowPowerOpt ,** response value **LowPowerMode**

LowPowerMode   Copy code

```
public enum LowPowerMode {
        LOWPOWER_OPT_NULL((byte) 0x00), //No processing method
        LOWPOWER_OPT_SCREEN_LEFT_OFF((byte) 0x01), //Left screen off
        LOWPOWER_OPT_SCREEN_RIGHT_OFF((byte) 0x02);//right screen off
}
```

Get the list of supported font sizes (CmdReqGetFontList)

Get the list of supported font sizes. Response: **CmdRspGetFontList .** Response value: **List<MetaAutocueFont>**

MetaAutocueFont   Copy code

```
public class MetaAutocueFont { private int cw;//
        TODO private int ch;//TODO
        private int ew;//TODO private int
        eh;//TODO


}
```

Query device SN request **CmdReqGetSN**

Query device SN response **CmdRspGetSN ,** response value **String**

Query the device language list request **CmdReqGetLanguageList**

Query the device language list response **CmdRspGetLanguageList ,** the response value **is ArrayList<MetaLanguage>**

MetaLanguage Copy code

```
public enum MetaLanguage {
        META_LANGUAGE_CN ((byte) 0x00),
        META_LANGUAGE_EN ((byte) 0x01),
        META_LANGUAGE_JA ((byte) 0x02),
        META_LANGUAGE_UNKNOWN((byte)0xFF);
}
```

Query the current language of the device **CmdReqGetLanguageCurrent**

Query the current language of the device. Response: **CmdRspGetLanguageCurrent .** Response value: **MetaLanguage**

MetaLanguage Copy code

```
public enum MetaLanguage {
        META_LANGUAGE_CN ((byte) 0x00),
        META_LANGUAGE_EN ((byte) 0x01),
        META_LANGUAGE_JA ((byte) 0x02),
        META_LANGUAGE_UNKNOWN((byte)0xFF);
}
```

Get optical and mechanical parameters request **CmdReqGetLcdParam**

Get the optical and mechanical parameters response **CmdRspGetLcdParam ,** the response value **is MetaLcdParam**

MetaLcdParam Copy code

```
public class MetaLcdParam { private
        byte select; private byte
        channel; private byte type;
        private int value;


}
```

Device Setup

Enable SMS notification: **CmdReqEnableSMS**

Short message notification enable response: **CmdRspStatus**

Incoming call message enable **CmdReqEnableCall**

Incoming message enable response: **CmdRspStatus**

Set system time request **CmdReqSetSystemTime ,** request value **MetaGlassTimestamp**

MetaGlassTimestamp Copy code

```
public class MetaGlassTimestamp {
        private final long timestamp; // timestamp private final
        int timezone; // time zone
}
```

Set system time response: **CmdRspStatus**

Set Bluetooth name **CmdReqSetBtName ,** request value **String**

Set Bluetooth name response: **CmdRspStatus**

Device restart **CmdReqReboot ,** request value **BootParam** , **BootParam** contains **BootMode**

BootParam Copy code

```
public class BootParam { private
        final BootMode masterMode;//Loda private final BootMode
        slaveMode;//AP4
}
```

BootMode Copy code

```
public enum BootMode {
        BOOTNULL((byte)0x00), //No boot mode
        BOOTSYSTEM((byte)0x01), //Normal boot mode
        BOOTLOADER((byte)0x02),; //BootLoader startup mode
}
```

Device restart without **Rsp**

Set the device brightness mode **CmdReqSetBrightnessMode ,** request value **BrightnessMode**

BrightnessMode Copy code

```
public enum BrightnessMode {
        BRIGHTNESS_AUTO((byte)0x00),
        BRIGHTNESS_MANUAL((byte)0x01),;
}
```

Set device brightness value request **CmdReqSetBrightnessValue ,** request value **Byte**

Set the device brightness value in response to **CmdRspStatus**

Set the Do Not Disturb mode request **CmdReqEnableDND ,** request value **Boolean**

Please set the Do Not Disturb mode to respond to **CmdRspStatus**

Set weather enable request **CmdReqEnableWeather ,** request value **Boolean**

Set weather enable response **CmdRspStatus**

Set the device screen off time request **CmdReqSetScreenOffTime ,** request value **Integer**

Set the device screen off time response **CmdRspStatus**

Set the device low power handling mode request **CmdReqSetLowPowerOpt ,** request value **LowPowerMode**

LowPowerMode Copy code

```
public enum LowPowerMode {
        LOWPOWER_OPT_NULL((byte) 0x00), //No processing method
        LOWPOWER_OPT_SCREEN_LEFT_OFF((byte) 0x01), //Left screen off
        LOWPOWER_OPT_SCREEN_RIGHT_OFF((byte) 0x02);//right screen off
}
```

Set the device low battery handling method to respond to **CmdRspStatus**

Set the current language of the device **CmdReqSetLanguageCurrent ,** request value **MetaLanguage**

MetaLanguage Copy code

```
public enum MetaLanguage {
        META_LANGUAGE_CN ((byte) 0x00),
        META_LANGUAGE_EN ((byte) 0x01),
        META_LANGUAGE_JA ((byte) 0x02),
        META_LANGUAGE_UNKNOWN((byte)0xFF);
}
```

Captouch detection subscription request **CmdReqCaptouchSubscribe ,** request value **Byte**

Copy code

```
public static final byte SUBSCRIBE = 0x01; public static final byte
UNSUBSCRIBE = 0x00;
```

Wearing detection subscription response **CmdRspStatus**

Restore factory settings request **CmdReqResetFactory**

Restore factory settings without **Rsp**

Restore factory settings and reboot request **CmdReqFactoryReboot**

Restore factory settings and restart without **Rsp**

Set optical and mechanical parameters request **CmdReqSetLcdParam ,** request value **MetaLcdParam**

MetaLcdParam Copy code

```
public class MetaLcdParam { private byte
        select; private byte channel;
        private byte type; private int value;



}
```

Device Notifications

Incoming call information **CmdReqUpdateCallInfo ,** request value **MetaCallInfo**

MetaCallInfo Copy code

```
public class MetaCallInfo {
        private final String mNumber; //Caller number
        private final String mName; //Caller name
}
```

Incoming call information response: **CmdRspStatus**

Message notification **CmdReqMsgNoti ,** request value **MsgNotiReq**

MsgNotiReq Copy code

```
public class MsgNotiReq {
        private final MetaMessageID messageID;//Message ID
        private final boolean canReply; //Can I reply?
        private final long time; //Message time
        private final String title; //Message title
        private final String text; //Message content
}
```

Weather update request **CmdReqUpdateWeather ,** request value **MetaGlassWeather**

MetaGlassWeather Copy code

```
public class MetaGlassWeather {
        /**
         *          0x00--No wind direction
         *          0x01--Northeast
         *          0x02--East
         *          0x03--Southeast
         *          0x04--South
         *          0x05--Southwest
         *          0x06--West
         *          0x07--Northwest
```

```
    *           0x08--North
    *           0xFF--Uncertain rotation
    */
   private int winddirection;//wind
   directionprivate int windpower;//
   wind powerprivate int humidity;//
   humidityprivate short temperature;//
   temperatureprivate short temperature_min;//
   minimum temperatureprivate short
   temperature_max;//maximum
   temperatureprivate short air_quality;//air
   qualityprivate String weather;//weather type: sunny/rainy/ private String location;//area
}
```

Weather update response **CmdRspStatus**

QQ Music

Synchronize QQ Music authorization status request **CmdReqSyncQQMusicAuthStatus ,** request value **QQMusicAuthStatus**

QQMusicAuthStatus   Copy code

```
public enum QQMusicAuthStatus { /**
   Unauthorized*/
   UNAUTH((byte) 0x00), /
   ** Authorized */
   AUTHED((byte) 0x01);
}
```

Synchronize QQ Music authorization status response **CmdRspStatus**

Synchronize QQ Music authorization status request (Req) **CmdReqSyncQQMusicAuthStatusByGlass**

Synchronize QQ Music authorization status response (Req) **CmdRspSyncQQMusicAuthStatus ,** response value **QQMusicAuthStatus s**

QQMusicAuthStatus   Copy code

```
public enum QQMusicAuthStatus { /**
   Unauthorized*/
   UNAUTH((byte) 0x00),
        /** Authorized */
        AUTHED((byte) 0x01);
}
```

navigation

Enter navigation

CmdReqNavigationEnter

Response: **CmdRspStatus**

Exit navigation

CmdReqNavigationExit

Response: **CmdRspStatus**

Set to always on during navigation

CmdReqSetNavigationKeepBright

Copy code

```
/** *
  @param value: true means always on, false means follow the system screen
  off time */
public CmdReqSetNavigationKeepBright(boolean value) {
        super(CmdType.CMD_SET_NAVIGATION_KEEP_BRIGHT, value);
}
```

Response: **CmdRspStatus**

Get the always-on configuration during navigation

CmdReqGetNavigationKeepBright

Response: **CmdRspGetNavigationKeepBright** contains a Boolean. True means the screen is always bright, and false means the screen is off when the system is turned off.

between

Sending navigation data

CommNavigationData This object is sent through **the startSendReplaceData** method .

Copy code

```
public class CommNavigationData extends CommData {
        /**
```

```
    * Navigation
    information */
   private final String guideInfo; /** * Navigation

   icon */

   private Bitmap icon; /** * The

   width and height of the
    scaled image */
   private int iconW; private
   int iconH; /** * Remaining

   time */

   private String timeInfo; /** *

   Remaining
    distance */
   private String dstInfo; /** *

   Navigation
    type */
   private int bwalkFlag; /** *

   Highlight
    information */
   private String wordLight; /** * Maximum

   length of width and height, subject to the
    highest value */
   private int maxSize = 100;

}
```

Example:

```
val cmdMap = CommNavigationData(result.data ?: "", bitmap)
cmdMap.setTimeInfo(data.timeInfo)
cmdMap.setWordLight(data.wordLight)
cmdMap.setDstInfo(data.disInfo)
cmdMap.setBwalkFlag(data.bwalkFlag)
MetaGlassDeviceGlobal.startSendReplaceData(cmdMap)
```

Teleprompter

Set the teleprompter text **CmdReqAutocueText ,** request value **MetaAutocueText**

MetaAutocueText   Copy code

```
public class MetaAutocueText {
        private int curPage; //Current page number private
        int totalPage; //Total page number private String
        text; //Text content
}
```

Set the prompter text response **CmdRspAutocueText ,** the response value **is Short**

Teleprompter page turning request **CmdReqAutocuePage ,** request value **MetaPage**

MetaPage   Copy code

```
public enum MetaPage {
        PAGE_UP((byte)0x00), //Previous page
        PAGE_DOWN((byte)0x01);//Next page
}
```

Teleprompter page turning response **CmdRspAutocuePage ,** response value **Short**

Get the current font number of the teleprompter request **CmdReqGetAutocueFont**

Get the current font number of the teleprompter. Response: **CmdRspGetAutocueFont .** Response value: **Integer**

Set the current font number of the teleprompter. Request **CmdReqSetAutocueFont .** Request value **Integer**

Set the current font number of the teleprompter in response to **CmdRspStatus**

Query the teleprompter display area request **CmdReqAutocueArea**

Query the teleprompter display area response **CmdRspGetAutocueArea ,** response value **MetaAutocueArea**

MetaAutocueArea Copy code

```
public class MetaAutocueArea { private int w;//
        width private int h;//height


}
```

Teleprompter start speech request **CmdReqAutocueStartSpeech**

The teleprompter starts speaking in response to **CmdRspAutocueStartSpeech ,** with a response value **of Short**

Teleprompter end speech request **CmdReqAutocueStopSpeech**

Teleprompter ends speech response **CmdRspStatus**

translate

Set the speaking role

CmdReqVoiceRTTSpeak Copy code

```
/** *
  Construction
  method * @param value: speaking role
  enumeration class */
public CmdReqVoiceRTTSpeak(MetaRole value) {
        super(CmdType.CMD_VOICE_RTT_SPEAK, value);
}
```

MetaRole Copy code

```
public enum MetaRole { //Speaking
        on the glasses side
        META_ROLE_DEVICE((byte)0x00),
        //Speech on the APP
        META_ROLE_HOST((byte)0x10), //Third-
        party speech
        META_ROLE_UNKNOWN((byte)0xFF);
}
```

Response: **CmdRspStatus**

Set the content of the speech

CmdReqVoiceRTTText Copy code

```
/** *
  @param role: speaking role enumeration
  class * @param sessionId: message id, if the id is the same, it will be
  overwritten, if the id is different, it

will be added * @param text: message content */ public CmdReqVoiceRTTText(MetaRole role, byte sessionId, String text) {
        super(CmdType.CMD_VOICE_RTT_TEXT, text); mRole =
        role; mSessionId =
        sessionId;
}
```

Response: **CmdRspStatus**

Set up translation

CmdReqVoiceRTTEnter Copy code

```
/** *
  @param role: speaking role enumeration
  class * @param language: language

enumeration class */ public CmdReqVoiceRTTEnter(MetaRole role, MetaLanguage language)
        { super(CmdType.CMD_VOICE_RTT_ENTER, language); mRole =
        role;
}
```

MetaLanguage Copy code

```
public enum MetaLanguage { /** Chinese
        */
        META_LANGUAGE_CN((byte)0x00), /**
        English */
        META_LANGUAGE_EN((byte)0x01), /**
        Japanese */
        META_LANGUAGE_JA((byte)0x02), /**
        Unknown */
        META_LANGUAGE_UNKNOWN((byte)0xFF);
}
```

Response: **CmdRspStatus**

Exit Translation

CmdReqVoiceRTTExit

Response: **CmdRspStatus**

Set translation language

CmdReqVoiceRTTLang Copy code

```
/** *
  @param language: language enumeration

class */ public CmdReqVoiceRTTLang(MetaLanguage language)
        { super(CmdType.CMD_VOICE_RTT_LANG, language);
}
```

Response: **CmdRspStatus**

Schedule

Get all schedule keys

CmdReqAllScheduleKeys

response

CmdRspAllScheduleKeys contains an **ArrayList<Short> ,** which is a collection of all schedule keys on the glasses.

Get a single itinerary

CmdReqGetSchedule Copy code

```
/** *
  @param key: speaking role

enumeration class */ public CmdReqGetSchedule(short key) {
        super(CmdType.CMD_SCHEDULE_GET, key);
}
```

response

CmdRspGetSchedule contains a complete **MetaScheduleBean** schedule information

MetaScheduleBean Copy code

```
/**
```

```
  * @param key: schedule key -1 represents adding, other
  represents modifying *
  @param name: schedule name *
  @param startTime: start time *
  @param endTime: end time * @param

remindTime: reminder time in advance */ public MetaScheduleBean(short key, String name, long startTime, long
endTime, int remindTime)
        { this.key = key;
        this.name = name;
        this.startTime = startTime; this.endTime
        = endTime; this.remindTime =
        remindTime;
}
```

Add Schedule

CmdReqAddSchedule

CmdReqAddSchedule Copy code

```
/**
  * @param value: schedule entity
*/
public CmdReqAddSchedule(MetaScheduleBean value) {
        super(CmdType.CMD_SCHEDULE_ADD, value);
}
```

response

CmdRspAddSchedule contains the schedule key just added

Modify schedule

CmdReqEditSchedule

CmdReqEditSchedule Copy code

```
/**
  * @param value: schedule entity
*/
public CmdReqEditSchedule(MetaScheduleBean value)
        { super(CmdType.CMD_SCHEDULE_UPDATE_ONE, value);
}
```

Response: **CmdRspStatus**

Delete Schedule

CmdReqDeleteSchedule

Copy code

```
/** *
  @param value: schedule key */ public

CmdReqDeleteSchedule(short value) {
        super(CmdType.CMD_SCHEDULE_REMOVE_ONE, value);
}
```

Response: **CmdRspStatus**

Frequently used contacts

Get the number of frequently used contacts (it is recommended to store no more than 10)

CmdReqContactsCountGet

Response: **CmdRspGetContactsCount** contains a Short object, which represents the total number of contacts. To obtain frequently used contacts, you need to use a loop to obtain them.

Get Frequent Contacts

CmdReqGetContacts

response

CmdRspGetContacts contains a **Map<Short, MetaContacts>** object, where key (short) represents the current

Copy code

```
/** *
  @param name: contact name * @param
  mobile: contact phone number * @param
  isCut: whether the contact name exceeds 20 bytes */ public

MetaContacts(String name, String mobile, boolean isCut) { this.name = name; this.mobile = mobile; this.isCut =
        isCut;
}
```

Set up frequently used contacts (it is recommended to store no more than 10)

`CmdReqSetContacts`

```
/** *
  @param value: all common contacts */

public CmdReqSetContacts(List<MetaContacts> value)
        { super(CmdType.CMD_CONTACTS_SET, value);
}
```

Response: `CmdRspStatus`

Chat with Glass

Enter Chat with Glass

`CmdReqStartGPT`

Response: `CmdRspStatus`

Exit Chat with Glass

`CmdReqEndGPT`

Response: `CmdRspStatus`

The glasses initiate a speech (Req)

`CmdType.CMD_START_GPT_RECORDING`

APP response:

CmdRspStartGPTRecording

```
CmdRspStartGPTRecording(ByteArray(1).apply
        { CmdError.STATUS_SUCCESS.code
})
```

CmdRspStartGPTRecording

```
/**
  * @param state: response status , fixed value is ByteArray(1) */

public CmdRspStartGPTRecording(byte[] state) {
        super(CmdType.CMD_START_GPT_RECORDING, state); this.state
        = state;
}
```

Glasses end speech (Req)

CmdType.CMD_END_GPT_RECORDING

APP response:

CmdRspEndGPTRecording   Copy code

```
CmdRspEndGPTRecording((ByteArray(1).apply
        { CmdError.STATUS_SUCCESS.code
}))
```

Example:

Demo   Copy code

```
override fun onGlassDeviceCmdReq(address: String?, req: CmdReq<*>?) { when (req?.type)

        { CmdType.CMD_START_GPT_RECORDING -> {
                    mDeviceManager.sendCmd(CmdRspStartGPTRecording(ByteArray(1).apply {
                            CmdError.STATUS_SUCCESS.code
                    })) //
                    Start speech recognition
              }

              CmdType.CMD_END_GPT_RECORDING -> {
                    mDeviceManager.sendCmd(CmdRspEndGPTRecording((ByteArray(1).apply {
                            CmdError.STATUS_SUCCESS.code }))) //
                    Stop
                    speech recognition
              }
        }
}
```

Send Question

CmdReqSetGPTQuestionText

CmdReqSetGPTQuestionText Copy code

```
/**
* @param total: total number
of packets * @param index:
which packet * @param messageId: message id, if messageId is the same, it will be appended, if
messageId is different, it

will be overwritten * @param text: content */ public CmdReqSetGPTQuestionText(byte total, byte index, byte
messageId,
        byte[] text) { super(CmdType.CMD_SET_GPT_QUESTION_TEXT,
        null); this.total = total;
        this.index = index;
        this.messageId = messageId; if (text ==
        null) { this.text = new
                byte[0]; } else { this.text = text;



        }
}
```

Response: **CmdRspStatus**

Send Answer

CmdReqSetGPTAnswerText

Copy code

```
/**
* @param total: total number
of packets * @param index:
which packet * @param messageId: message id, if messageId is the same, it will be appended, if
messageId is different, it

will be overwritten * @param text: content */ public CmdReqSetGPTAnswerText(byte total, byte index, byte
messageId,
        byte[] text) { super(CmdType.CMD_SET_GPT_ANSWER_TEXT,
        null); this.total = total;
        this.index = index;
        this.messageId = messageId; if (text ==
        null) { this.text = new
                byte[0]; } else { this.text = text;



        }
}
```

Response: **CmdRspStatus**

Subcontracting example:

Copy code

```
//Maximum length
of each packetprivate val MAX_PACKAGE_SIZE
= 501 //Get the
total number of bytesval toByteArray =

result.toByteArray() //Calculate the total number of packetsval total = if
        (toByteArray.size % MAX_PACKAGE_SIZE == 0)

{ toByteArray.size / MAX_PACKAGE_SIZE } else { toByteArray.size / MAX_PACKAGE_SIZE + 1

} //Loop
sending packets for (index in
        1..total) { //Calculate the
        bytes sent in each packet val temp = if (index * MAX_PACKAGE_SIZE >
                toByteArray.size) { toByteArray.size - (index - 1) *

        MAX_PACKAGE_SIZE } else { MAX_PACKAGE_SIZE

        } val tempB = ByteArray(temp)


        System.arraycopy( toByteArray, (index - 1) *


                MAX_PACKAGE_SIZE, tempB, 0, tempB.size

        )




          mDeviceManager.sendCmd( CmdReqSetGPTQuestionText( total.toByte(), index.toByte(), messageId.toByte(), tempB
                )
        )
}
```

Voice Assistant

Long press the temple for 2 seconds to wake up the voice assistant switch request **CmdReqSetVoiceAssistantWakeSwitch ,** request value **VoiceAssistantStatus**

VoiceAssistantStatus Copy code

```
public enum VoiceAssistantStatus { /** Close*/

        CLOSE((byte) 0x00), /**
        Open */
        OPEN((byte) 0x01);
}
```

Long press the temple for 2 seconds to wake up the voice assistant switch response **CmdRspStatus**

Long press the temple for 2 seconds to wake up the voice assistant switch. Current status request **CmdReqGetVoiceAssistantWakeSwitch**

Long press the temple for 2 seconds to wake up the voice assistant switch. The current status response is **CmdRspGetVoiceAssistantWakeSwitch ,** and the response value **is VoiceAssistantStatus**

VoiceAssistantStatus Copy code

```
public enum VoiceAssistantStatus { /** Close*/

        CLOSE((byte) 0x00), /**
        Open */
        OPEN((byte) 0x01);
}
```

Set the voice assistant wake-up status request **CmdReqSetVoiceAssistantStatus ,** request value **VoiceAssistantStatus s**

Set the voice assistant wake-up status response **CmdRspStatus**

Set the voice assistant wake-up status request (Req) **CmdReqSetVoiceAssistantStatusByGlass ,** request value **VoiceAsagentStatus**

Set the voice assistant wake-up status response (Req) **CmdRspStatus**

Whether the glasses are in the voice assistant application (Req) **CmdReqVoiceAssistantInApp ,** request value **VoiceAssistantGlassStatus**

VoiceAssistantGlassStatus Copy code

```
public enum VoiceAssistantGlassStatus {
        STATUS_IN_ALL_PAGE((byte) 0x01),
        STATUS_IN_STATUS_BAR((byte) 0x00);
}
```

Voice assistant main switch setting request **CmdReqSetVoiceAssistantMainSwitch ,** request value **VoiceAssistantStatus s**

VoiceAssistantStatus Copy code

```
public enum VoiceAssistantStatus { /** Close*/

        CLOSE((byte) 0x00), /**
        Open */
        OPEN((byte) 0x01);
}
```

Voice assistant master switch setting response **CmdRspStatus**

The voice assistant obtains the current status of the main switch **CmdReqGetVoiceAssistantMainSwitch**

The voice assistant gets the current status of the main switch. Response: **CmdRspGetVoiceAssistantMainSwitch .** Response value: **VoiceAssistantStatus**

VoiceAssistantStatus Copy code

```
public enum VoiceAssistantStatus { /** Close*/

        CLOSE((byte) 0x00), /**
        Open */
        OPEN((byte) 0x01);
}
```