

# aar Documentation

v3.0.0

Version v3.0.0 of the Android SDK is only compatible with MindRove devices sold after November 2023, as these devices are using the new firmware version. If you plan to use this Android SDK with an older device, please contact MindRove support to request a firmware update before proceeding.

## Structure

*mylibrary*

↳ *mindrove*

↳ *ServerManager*

↳ *ServerThread*

↳ *SensorData*

↳ *Instruction*

The *SensorData* class in the *mylibrary.mindrove* package is a data class that represents sensor data.

- *SensorData.channel1*
  - Type: Double
  - Voltage measured on each (1-8) EEG channel (in microvolts)
- *SensorData.accelerationX*
  - Type: Int
  - Accelerometer data corresponding to the three axes (X, Y, Z)
- *SensorData.angularRateX*
  - Type: Int
  - Gyroscope data corresponding to the three axes (X, Y, Z)
- *SensorData.voltage*
  - Type: UInt
  - Battery voltage measured [%]
- *SensorData.user\_trigger*
  - Type: UInt
  - Trigger events; 0 — None, 1 — Beep trigger, 2 — Boop trigger
- *SensorData.numberOfMeasurement*
  - Type: UInt
  - Packet identifier
- *SensorData.impedance1ToDRL*
  - Type: Int
  - Magnitudes of impedance measured between pairs of electrodes [ $\Omega$ ]. Only available in impedance mode.
  - (1ToDRL, 3ToDRL, RefToDRL, RefTo4, 1To2, 2To3, 3To4, 5To4, 5To6, 6ToRef)

- *SensorData.pulse*
  - Type: Int
  - Heart rate [bpm].
- *SensorData.spo2*
  - Type: Int
  - Oxygen saturation [%].

The *ServerManager* class is responsible for managing a server thread and its interactions.

- *ServerManager.sendInstruction*
  - Sending instructions to the client
  - Expecting Instruction
- *ServerManager.start/stop*
  - Starting and stopping the server thread
- *ServerManager.isMessageReceived*
  - Check if a message has been received
- *ServerManager.ipAddress*
  - IP address of the server

The *Instruction* is an enum class for different types of instructions

- *Instruction.BEEP* for Beep trigger
- *Instruction.BOOP* for Boop trigger
- *Instruction.EEG* for EEG mode
- *Instruction.IMP* for impedance mode
- *Instruction.TEST* for generating test signals

The *ServerThread* class is a thread for the server, the whole class is managed by the *ServerManager*.

## Importing .aar file to new android studio project

- Add .aar file to projects libs folder (project\app\libs)  
<https://developer.android.com/studio/projects/android-library>

- build.gradle

```
implementation(files("libs/mindRove-debug.aar"))
implementation(fileTree(mapOf("dir" to "libs", "include" to
listOf("*.jar", "*.aar"))))
```

- Import classes

```
import mylibrary.mindrove.Instruction
import mylibrary.mindrove.SensorData
import mylibrary.mindrove.ServerManager
```

- Make sure that you have the necessary network permissions in your *AndroidManifest.xml* file. Add the following permission:

```
<uses-permission android:name="android.permission.INTERNET" />
```

- To write data to external storage:

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"
```

- For live data

```
implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.7.0")
implementation("androidx.compose.runtime:runtime:1.6.1")
```

The INTERNET permission is needed for network communication with the MindRove device, and the WRITE\_EXTERNAL\_STORAGE permission is needed to write sensor data to external storage.

## Getting started with code

The Android device needs to be connected to the MindRove device via Wi-Fi before launching the app!

1. Import the necessary classes from the library:

```
import mylibrary.mindrove.SensorData
import mylibrary.mindrove.ServerManager
```

2. Create an instance of ServerManager and provide a callback function that will be called when new data is received. The callback function takes a SensorData object as a parameter:

```
private val serverManager = ServerManager { sensorData: SensorData ->
    // Handle the received data here
}
```

3. Start the ServerManager when a network connection is available:

```
serverManager.start()
```

4. Stop the ServerManager when the activity is destroyed to clean up resources:

```
serverManager.stop()
```

## Example code in Kotlin:

```
import mylibrary.mindrove.SensorData
import mylibrary.mindrove.ServerManager
```

```
class MainActivity : ComponentActivity() {
    private val serverManager = ServerManager { sensorData: SensorData ->
        // Update the sensor data text
        sensorDataText.postValue(sensorData.accelerationX.toString())
    }
    private val sensorDataText = MutableLiveData("No data yet")
    private val networkStatus = MutableLiveData("Checking network status...")
    private lateinit var handler: Handler
    private lateinit var runnable: Runnable
    private var isServerManagerStarted = false
    private var isWifiSettingsOpen = false
```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    handler = Handler(Looper.getMainLooper())
    runnable = Runnable {
        val isNetworkAvailable = isNetworkAvailable()
        if (!isNetworkAvailable) {
            // If no network, update the network status and open Wi-Fi settings
            networkStatus.value = "No network connection. Please enable Wi-Fi."
            if (!isWifiSettingsOpen) {
                openWifiSettings()
                isWifiSettingsOpen = true
            }
        } else {
            networkStatus.value = "Connected to the network."
            isWifiSettingsOpen = false

            // Start the ServerManager here, when a network connection is available
            if (!isServerManagerStarted) {
                serverManager.start()
                isServerManagerStarted = true
            }
        }
        handler.postDelayed(runnable, 3000)
    }

    handler.post(runnable)

    setContent {
        Try2_0Theme {
            Surface(
                modifier = Modifier.fillMaxSize(),
                color = MaterialTheme.colorScheme.background
            ) {
                val networkStatusValue by networkStatus.asFlow()
                    .collectAsState(initial = "Checking network status...")
                val sensorDataTextValue by sensorDataText.asFlow()
                    .collectAsState(initial = "No data yet")

                Column {
                    // Display the network status
                    Text(text = networkStatusValue)
                    // Display the sensor data text
                    Text(text = sensorDataTextValue)
                }
            }
        }
    }
}

```

```

}

override fun onDestroy() {
    super.onDestroy()
    handler.removeCallbacks(runnable)

    // Stop the server when the activity is destroyed
    serverManager.stop()
}

// Function to check network connectivity
private fun isNetworkAvailable(): Boolean {
    val connectivityManager =
        getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager

    val network = connectivityManager.activeNetwork
    val capabilities = connectivityManager.getNetworkCapabilities(network)
    return capabilities != null &&
        (capabilities.hasTransport(NetworkCapabilities.TRANSPORT_WIFI) ||
            capabilities.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR))
}

private val wifiSettingsLauncher =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
        // This block is executed when the Wi-Fi settings activity is finished
        isWifiSettingsOpen = false
    }
// Function to open Wi-Fi settings
private fun openWifiSettings() {
    val intent = Intent(Settings.ACTION_WIFI_SETTINGS)
    wifiSettingsLauncher.launch(intent)
}
}

```