Manual SDK Linphone

Import SDK Linphone

ไปที่ไฟล์ `build.gradle(:app)` แล้วทำการเพิ่ม `url` ที่ `repositories` ดังภาพที่ 1

```
// We need to declare this repository to be able to use Liblinphone SDK
repositories {
    maven {
        url "https://linphone.org/maven_repository"
    }
}
```

ภาพที่ 1 respositories section

และเพิ่ม `implementation` ที่ `dependencies` ดังภาพที่ 2

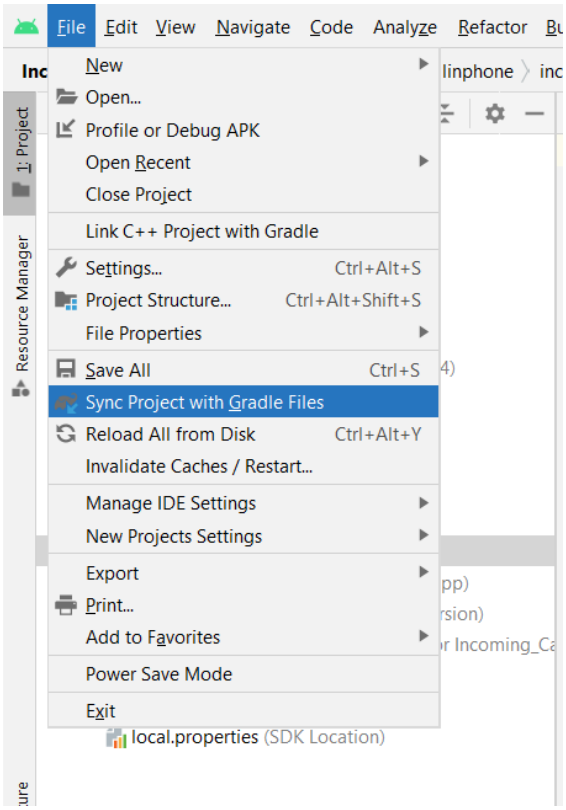
```
debugImplementation "org.linphone:linphone-sdk-android-debug:5.0+"
releaseImplementation "org.linphone:linphone-sdk-android:5.0+"

dependencies {

    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    // Latest version is 5.0.x, using + to get the latest available
    debugImplementation "org.linphone:linphone-sdk-android-debug:5.0+"
    releaseImplementation "org.linphone:linphone-sdk-android:5.0+"
    // Adding this dependency allows the linphone-sdk to automatically handle audio focus
    implementation 'androidx.media:media:1.2.0'
}
```

ภาพที่ 2 dependencies section

หลังจากนั้นต้องทำการกด Sync Project with Gradle Files โดยเข้าไปที่ File -> Sync Project with Gradle Files ดังภาพที่ 3



ภาพที่ 3 Sync Project with Gradle Files

สามารถดูเพิ่มเติม :

https://wiki.linphone.org/xwiki/wiki/public/view/Lib/Getting%20started/Android/

หลังจากนั้นสามารถ import ได้ตามที่ต้องการ เช่น `import org.linphone.core.*`

Funtion ใน SDK Linphone

- Initial Core Linphone

```
val authInfo = Factory.instance().createAuthInfo(username, userid: null, password, ha1: null, realm: null, domain, algorithm: null)
val params = core.createAccountParams()
val identity = Factory.instance().createAddress( addr: "sip:$username@$domain")
params.identityAddress = identity

val address = Factory.instance().createAddress( addr: "sip:$domain")

address?.transport = transportType
params.serverAddress = address
params.registerEnabled = true
val account = core.createAccount(params)

core.addAuthInfo(authInfo)
core.addAccount(account)

core.defaultAccount = account
core.addListener(coreListener)
core.start()
```

ภาพที่ 4 initial core linephone

```
// For video to work, we need two TextureViews:
// one for the remote video and one for the local preview
core.nativeVideoWindowId = findViewById(R.id.remote_video_surface)
// The local preview is a org.linphone.mediastream.video.capture.CaptureTextureView
// which inherits from TextureView and contains code to keep the ratio of the capture video
core.nativePreviewWindowId = findViewById(R.id.local_preview_video_surface)

// Here we enable the video capture & display at Core level
// It doesn't mean calls will be made with video automatically,
// But it allows to use it later
core.enableVideoCapture( enable: false)
core.enableVideoDisplay( enable: true)

// When enabling the video, the remote will either automatically answer the update request
// or it will ask it's user depending on it's policy.
// Here we have configured the policy to always automatically accept video requests
core.videoActivationPolicy.automaticallyAccept = true
```

ภาพที่ 5 Initial Element Video Call

- Incoming Call Activity

  ฟังก์ชันการรับสายจะไม่มีการเรียกฟังก์ชัน แต่เป็นการวนลูปไปเรื่อยๆเพื่อเช็คสถานะหรือ State ของ Linphone เมื่อมีผู้ใช้โทรเข้ามาจะเข้าสู่ State IncommingReceived

```
// When a call is received
when (state) {
    Call.State.IncomingReceived -> {
        findViewById<Button>(R.id.hang_up).isEnabled = true
        findViewById<Button>(R.id.answer).isEnabled = true
        findViewById<EditText>(R.id.remote_address).setText(call.remoteAddress.asStringUriOnly())

    }
    Call.State.Connected -> {

        findViewById<Button>(R.id.mute_mic).isEnabled = true
        findViewById<Button>(R.id.toggle_speaker).isEnabled = true
        findViewById<Button>(R.id.toggle_video).isEnabled = true
    }
    Call.State.StreamsRunning -> {
        // This state indicates the call is active.
        // You may reach this state multiple times, for example after a pause/resume
        // or after the ICE negotiation completes
        // Wait for the call to be connected before allowing a call update
    }
```

เมื่อผู้ใช้กดรับสายจะไปสู่ State Connected และ StreamsRunning ตามลำดับ

ภาพที่ 6 ตัวอย่างสถานการณ์เมื่อมีคนโทรเข้ามา

ดังภาพที่ 6 กรอบสีแดงเป็น field ที่บอกว่าผู้ใช้คนใดโทรเข้ามา ส่วนกรอบสีเหลืองเป็นการแสดงสถานะของ Linphone

```
if (core.callsNb == 0) return
val call = if (core.currentCall != null) core.currentCall else core.calls[0]
call ?: return

// We will need the CAMERA permission for video call
if (packageManager.checkPermission(Manifest.permission.CAMERA, packageName) != PackageManager.PERMISSION_GRANTED) {
    requestPermissions(arrayOf(Manifest.permission.CAMERA), requestCode: 0)
    return
}

// To update the call, we need to create a new call params, from the call object this time
val params = core.createCallParams(call)
// Here we toggle the video state (disable it if enabled, enable it if disabled)
// Note that we are using currentParams and not params or remoteParams
// params is the object you configured when the call was started
// remote params is the same but for the remote
// current params is the real params of the call, resulting of the mix of local & remote params
params?.enableVideo(!call.currentParams.videoEnabled())


// Finally we request the call update
call.update(params)
```

ภาพที่ 7 การส่ง Request Video Call ไปยังผู้ใช้

- Outgoing Call Activity

```
private fun outgoingCall() {
    // As for everything we need to get the SIP URI of the remote and convert it to an Address
    val remoteSipUri = findViewById<EditText>(R.id.remote_address).text.toString()
    val domain = "49.0.87.17:5060"
    val remoteAddress = Factory.instance().createAddress( addr: "sip:$remoteSipUri@$domain")
    remoteAddress ?: return // If address parsing fails, we can't continue with outgoing call process

    // We also need a CallParams object
    // Create call params expects a Call object for incoming calls, but for outgoing we must use null safely
    val params = core.createCallParams( call: null)
    params ?: return // Same for params

    // We can now configure it
    // Here we ask for no encryption but we could ask for ZRTP/SRTP/DTLS
    params.mediaEncryption = MediaEncryption.None
    // If we wanted to start the call with video directly
    //params.enableVideo(true)

    // Finally we start the call
    core.inviteAddressWithParams(remoteAddress, params)
    // Call process can be followed in onCallStateChanged callback from core listener
}
```

ภาพที่ 8 Outgoing Call Function

ภาพที่ 8 เป็นการเรียกใช้ฟังก์ชันโทรออก นั่นคือ

“`core.inviteAddressWithParams(remoteAddress, params)`”

แต่ก่อนที่จะเรียกใช้ได้นั้นต้องทำการสร้าง Address และ CallParams ก่อน โดยฟังก์ชัน createAddress จะใส่พารามิเตอร์เป็นดังนี้ “sip:user@domain” เท่านั้นจึงจะสามารถสร้าง address ได้

```
when (state) {
    Call.State.OutgoingInit -> {
        // First state an outgoing call will go through

    }
    Call.State.OutgoingProgress -> {
        // Right after outgoing init

    }
    Call.State.OutgoingRinging -> {
        // This state will be reached upon reception of the 180 RINGING

    }
    Call.State.Connected -> {
        // When the 200 OK has been received

    }
    Call.State.StreamsRunning -> {
        // This state indicates the call is active.
        // You may reach this state multiple times, for example after a pause/resume
        // or after the ICE negotiation completes
        // Wait for the call to be connected before allowing a call update
        android.util.Log.i( tag: "Sequence",  msg: "fun onCallStateChanged() Call.State.StreamsRunning")

        findViewById<Button>(R.id.pause).isEnabled = true
        findViewById<Button>(R.id.pause).text = "Pause"
        findViewById<Button>(R.id.toggle_video).isEnabled = true

        // Only enable toggle camera button if there is more than 1 camera and the video is enabled
        // We check if core.videoDevicesList.size > 2 because of the fake camera with static image created by our SDK (see below)
        findViewById<Button>(R.id.toggle_camera).isEnabled = core.videoDevicesList.size > 2 && call.currentParams.videoEnabled()
    }
```

ภาพที่ 9 ลำดับ State Outgoing Call Activity

- Pause Or Resume Activity

```
private fun pauseOrResume() {
    if (core.callsNb == 0) return
    val call = if (core.currentCall != null) core.currentCall else core.calls[0]
    call ?: return

    if (call.state != Call.State.Paused && call.state != Call.State.Pausing) {
        // If our call isn't paused, let's pause it
        call.pause()
    } else if (call.state != Call.State.Resuming) {
        // Otherwise let's resume it
        call.resume()
    }
}
```

ภาพที่ 10 ตัวอย่างการเรียกใช้ฟังก์ชัน Pause และ Resume

ฟังก์ชัน Pause สามารถเรียกได้โดยตรง call.pause( ) ดังภาพที่ 10

ฟังก์ชัน Resume สามารถเรียกได้โดยตรง call.resume( ) ดังภาพที่ 10

- Hangup Activity

```
private fun hangUp() {
    if (core.callsNb == 0) return

    // If the call state isn't paused, we can get it using core.currentCall
    val call = if (core.currentCall != null) core.currentCall else core.calls[0]
    call ?: return

    // Terminating a call is quite simple
    call.terminate()


}
```

ภาพที่ 11 ตัวอย่างการเรียกใช้ฟังก์ชัน Hangup

การเรียกใช้ฟังก์ชันวางสาย สามารถเรียกใช้ได้โดยตรง คือ call.terminate() ดังภาพที่ 11