アプリ連動感電デバイス「UNAGI」
SDKドキュメント
Godot版

シンフォニア株式会社

目次

目次 2
注意事項 3
概要 4
セットアップ・システム要件 5
　自分のプロジェクトへSDKを導入する 5
スクリプト解説 7
　接続 7
　UNAGIへのコマンド送信の基本形 9
　UNAGIテストアプリの諸機能 10
Windows（BLE通信） 11
　サンプルアプリケーションのビルドと動作 11
Android 13
　サンプルアプリケーションのビルドと動作 13
iOS 16
　サンプルアプリケーションのビルドと動作 16
MetaQuest コントローラ版 18
　サンプルアプリケーションのビルドと動作 18
　MetaQuest ハンドトラッキング版 19
Windows PCVR版 21
　サンプルアプリケーションのビルドと動作 21
コマンドリスト 23
　電流値設定 23
　時間設定 24
　周波数設定 25
　制御コマンド 26
改訂履歴 27

注意事項

- 「UNAGI」SDK（以下本SDK）はアプリケーション開発のサポートを行うものですが、各アプリケーションの動作を保証するものではありません。
- 本SDKを使用し発生した、いかなる損害に対して、当社は一切の責任を負わないものとします。
- 開発環境のアップデート等により本SDKの動作に不具合が出る可能性があります。
- 本SDKの問題解決や不具合の発生について、当社にご連絡頂くことは可能ですが、問題の修正やアップデートの提供が必ずしも保証されるものではありません。
- 本SDKは、貴社開発の、もしくは第三者のコンテンツやサービスとの連携を可能にするものですが、本SDKの正常な動作について保証するものではありません。
- 本SDKの不特定多数に対する再配布は利用規約により禁止されています。利用者は、本SDKそのものを直接、他のユーザーに提供することはできません。本SDKの複製物を再配布することは、ライセンスの違反と見なされます。

概要

本ドキュメントは、UNAGI感電体験装置を動作させるためのGodot用SDKのドキュメントです。本SDKでは、Windows・Android・iOS・MetaQuest・Windows PCVR向けのアプリケーションを作成できます。

SDKには、BLE通信を行うアドオンと、各プラットフォームのテストアプリのシーンを含みます。Windows PCVR版、MetaQuest コントローラ版、MetaQuest ハンドトラッキング版には、UI操作とオブジェクトへの接触でUNAGIへコマンドを送信するサンプルを用意しています。

バージョン：1.0.1

セットアップ・システム要件

動作テストにおいては以下の環境にて確認を行っています。実際には開発するプラットフォームに応じた開発環境のみセットアップすれば問題ないのでご注意ください。

- Godot 4.7.2
  - 同じバージョンのエクスポートテンプレート
- Android・MetaQuest向けビルド環境
  - JDK 17（18～23でも可、24以降は不可）
  - Android SDK Platform 36、Build Tools 36.1.0、Platform-Tools
- iOS向けビルド環境
  - MacとXcode
- Windows実行環境
  - Windows 11（x86_64）
  - BLE対応のBluetoothアダプター
- Android実行環境
  - Pixel 7
  - システムバージョン：Android 15
- MetaQuest実行環境
  - MetaQuest 3S
- Windows PCVR実行環境
  - Meta Horizon Link

MetaQuest版またはWindows PCVR版を使用する場合は、以下のアドオンを導入します。Windows BLE版、Android版、iOS版のみを使用する場合、以下は不要です。

- Godot OpenXR Vendors 5.1.0（本SDKには含みません）

本書のサンプルはGDScriptで作成しています。Standard版を使用でき、.NET版でもGDScriptを読み込めます。.NET版のGodotを使用する場合は、C#を使わなくても.NET用のエクスポートテンプレートが必要です。C#を併用する場合は、.NET SDKも用意してください。

自分のプロジェクトへSDKを導入する

本SDKは、addons/unagi フォルダーを収めたZIP（UNAGI_SDK_Godot_Addon_1.0.1.zip）で提供します。ZIPには、SDK本体とテストアプリのシーン・素材を含む addons/unagi と、第三者ライブラリ・素材の表記（licenses）が入っています。本書はZIPとは別に提供します。

1. Godotのプロジェクトマネージャーで新しいプロジェクトを作成します。レンダラーは、Windows・Android・iOS向けはCompatibility（互換性）、PCVR・MetaQuest向けはMobile（モバイ

ル）を選びます。既存のプロジェクトへ導入する場合は、そのプロジェクトを使います。

2. ZIPを展開し、addons/unagi フォルダーをプロジェクトの addons フォルダーへコピーします。addons フォルダーが無い場合は作成します。
3. Godotでプロジェクトを開き、リソースのインポートが完了するまで待ちます。
4. 「プロジェクト → プロジェクト設定 → プラグイン」で UNAGI SDK を有効にします。書き出し時に各OSの通信ライブラリを組み込むため、必ず有効にしてください。
5. 「エディター → エクスポートテンプレートの管理」を開き、書き出すプラットフォームのテンプレートを選んでインストールします。Windows版・PCVR版は Desktop の Windows x86_64、Android版・MetaQuest版は Android、iOS版はMacのGodotで iOS を選びます（Install All Templates ですべてを入れることもできます）。テンプレートはPCごとに1回入れれば、どのプロジェクトでも使えます。Androidのプロジェクトごとに行う「Androidビルドテンプレートのインストール」とは別のものです。
6. 自分のアプリで通信する場合は、シーンに UnagiClient ノードを追加します。シーンドックで親にするノードを右クリックし、「子ノードを追加」で UnagiClient を検索して作成します。このノードを使い、「スクリプト解説」の手順で通信を行います。テストアプリのシーンには UnagiClient ノード（名前は UNAGI）が入っているので、テストアプリを書き出すだけなら不要です。

テストアプリの各版は、各プラットフォームの章の「サンプルアプリケーションのビルドと動作」の手順で、この新しいプロジェクトから書き出せます。

Androidへ書き出す場合に必要な設定は、「Android」の章を参照してください。

テストアプリのシーンは addons/unagi/scenes_testapp に含まれています。PCVR・MetaQuest（XR）のシーンを使う場合に必要な設定は、各版の章を参照してください。XRを使わない場合は、XR用のシーン・素材は自動で書き出しから外れるので、設定は不要です。

スクリプト解説

接続

Windows・Android・iOS・MetaQuestはいずれもBLE通信により接続を行います。各プラットフォームで共通して使用するスクリプトは、addons/unagi/scripts/unagi_client.gd の UnagiClient です。Windowsは同梱のGDExtensionを介したWindows標準BLE、Android・MetaQuestはAndroid標準BLE、iOSはCore Bluetoothを使用します。

start_client() により初期化とスキャンを開始します。Android・MetaQuestでは初回にBluetoothの権限を要求し、許可後にスキャンが始まります。検出したUNAGIは devices_changed シグナルで通知されます。Windows・Android・MetaQuestではMACアドレス、iOSではCore BluetoothのDevice UUIDを使用します。

一覧から接続先を選び、connect_device(address) を呼び出します。接続、通知の購読、UNAGIの応答確認が完了すると connected シグナルが発生し、コマンドを送信できます。start_client() と connect_device() の戻り値は要求を受け付けたかどうかを示します。接続完了は connected または is_device_connected() で確認してください。

次の例では、スクリプトを付けたノードの子として UnagiClient ノードを追加し、名前を UNAGI にしておきます。

```
extends Node

@onready var unagi: UnagiClient = $UNAGI

func _ready() -> void:
    unagi.devices_changed.connect(_on_devices_changed)
    unagi.connected.connect(_on_connected)
    unagi.message_received.connect(_on_message_received)
    unagi.start_client()

func _on_devices_changed(addresses: PackedStringArray) -> void:
    print(addresses) # 一覧をUIへ表示します。

func select_device(address: String) -> void:
    unagi.connect_device(address)

func _on_connected(address: String) -> void:
    print("Connected: ", address)

func _on_message_received(message: String) -> void:
    print(message)
```

Inspector の Auto Start をオンにすると、ノードがシーンに入ったときに start_client() が自動で呼ばれます。接続に失敗した場合や通信が切れた場合は、自動でスキャンと再接続を行います。この動作は

Inspector の Reconnect Automatically で設定できます。オフにした場合、接続に失敗すると ERROR 状態で停止するので、start_client() でやり直します。再接続中（is_recovering() が true の間）は connect_device() で別の端末を選べません（false を返します）。別の端末に切り替えるときは disconnect_device() で再接続を止めてから選びます。処理を終了するときは stop_client() を呼び出します。接続だけを切って一覧の選択からやり直すときは disconnect_device() を呼び出します。send_stimulus() で開始した刺激（SDK が自動停止を管理しているもの）の途中で切断・停止・背景化した場合は stimulus_finished シグナルが発生します。write() で直接送った刺激は対象外です。

Android・iOS・MetaQuestでは、アプリがバックグラウンドへ移ると接続を解放し、復帰すると選択済みのUNAGIへ自動で再接続します。

BLE診断ログ

UnagiClient の Inspector にある Diagnostics File へ user:// で始まる保存先を設定すると、接続、切断、再接続、Service取得、Notify設定、Writeなどの処理結果をファイルへ保存できます。テストアプリでは、診断内容をGodotの出力またはアプリケーションのログへ出力します。

diagnostic_emitted シグナルと get_diagnostics() から、診断の内容を取得できます。診断は severity（Info / Warning / Error）、stage、code、code_name、message、address、state、phase を持ちます。すべての重大度が届くので、必要に応じて severity で絞ってください。get_diagnostics() は直近256件を返します。

Windows、Android、iOSで共通の形式を使用し、処理段階、OSのエラーコード、直前の状態、再スキャンや再接続など次に行う処理を確認できます。

UNAGIへのコマンド送信の基本形

文字列のコマンドを write() へ渡して送信します。コマンドの内容により、電流値、時間、周波数、感電、バッテリーチェックなどを指定できます。各コマンドは「コマンドリスト」を参照してください。

電流：1.0mA、時間：0.3s、周波数：5.4Hzの場合

```
unagi.write("#A10;#T03;#F01;#PEN;!")
```

バッテリーチェック

```
unagi.request_battery()
# 同じコマンドを文字列で送る場合
unagi.write("#BAT;")
```

残量値（0～99）は battery_received シグナルで受け取り、最後の値は battery_percent で参照できます（未受信は -1）。応答文字列 #Bnn; は message_received シグナルにも渡されます。バッテリーの残量値は大まかな値になりますのでご注意ください。

高電圧電源OFF

```
unagi.write("#PDS;")
```

動作上必ず必要なものではありませんが、ショック後にこのコマンドを送ると高電圧電源への電源供給が切断されるので、より長く電池を持たせることが可能です。

電流・時間・周波数を数値で指定する場合は、send_stimulus() も使用できます。

```
unagi.send_stimulus(1.0, 0.3, 5.4)
```

send_stimulus() は設定値を検査し、指定時間の0.5秒後に #PDS; を送信します。時間は0.0～3.0秒を0.1秒刻み、周波数はコマンドリストの13種類から指定します。電流は0.0～5.0mAで、コマンドリストに合わせて0.5mA刻みで指定してください。刺激が終わると stimulus_finished シグナルが発生

し、途中で止めるときは stop_stimulus() を呼び出します。write() はコマンドの設定値を検査せず、自動で #PDS; を送りません。テストアプリの刺激操作には send_stimulus() を使用しています。

UNAGIテストアプリの諸機能

テストアプリの基本機能は、addons/unagi/scripts/sample_session.gd の UnagiSampleSession にまとめています。UIと接触オブジェクトはこのNodeを通して、選択したUNAGIへコマンドを送信します。UI周りの参考として、必要に応じて改変してご利用ください。

シーンの構成

プラットフォーム	シーン
Windows	addons/unagi/scenes_testapp/windows_ble.tscn
Android・iOS	addons/unagi/scenes_testapp/mobile_ble.tscn
Windows PCVR	addons/unagi/scenes_testapp/pcvr_ble.tscn
MetaQuest コントローラ	addons/unagi/scenes_testapp/quest_controller.tscn
MetaQuest ハンドトラッキング	addons/unagi/scenes_testapp/quest_hand_tracking.tscn

各シーンの UNAGI が通信を行い、SampleSession が接続先と刺激設定を扱います。電流は0.0～5.0mAを0.5mA刻み、時間は0.0～3.0sを0.1s刻み、周波数は1.4～200Hzの13項目から選択します。

BatteryCheck は電池確認、Shock! は選択した設定での刺激、English／日本語は表示言語の切り替えです。スキャン中は Scanning...、接続完了時は Connected と表示します。

Windows（BLE通信）

サンプルアプリケーションのビルドと動作

1. 「自分のプロジェクトへSDKを導入する」の手順で、レンダラーをCompatibility（互換性）にした新しいプロジェクトを作成し、addons/unagi をコピーしてUNAGI SDKを有効にします。
2. 「プロジェクト → プロジェクト設定」を開き、一般タブの右上にある「高度な設定」を有効にします。手順5まではこの画面で設定します。
3. 一般タブ左側の一覧で「アプリ」の下にある「実行」を選び、右側の「メインシーン」に res://addons/unagi/scenes_testapp/windows_ble.tscn を指定します。欄にパスを入力するか、欄の右端にあるフォルダーのボタンからファイルを選びます。
4. 一般タブの「表示 → ウィンドウ」にある「サイズ」で、「ビューポートの幅」を800、「ビューポートの高さ」を450、「ウィンドウ広さのオーバーライド」を1280、「ウィンドウ高さのオーバーライド」を720にします。「ストレッチ」の「モード」を canvas_items、「アスペクト」を expand にします。
5. 必要に応じて、一般タブの「アプリ → 構成」にある「名前」、「アイコン」に res://addons/unagi/icon/unagi_icon.png、「Windowsアイコン」に res://addons/unagi/icon/unagi.ico を設定します。
6. 「プロジェクト → エクスポート」で「追加」から Windows Desktop を選び、プリセットを作成します。「バイナリフォーマット」の「アーキテクチャ」が x86_64 であることを確認します。
7. 必要に応じて、「アプリ」の「リソースの変更」を有効にし、「アイコン」（res://addons/unagi/icon/unagi.ico）、「ファイルバージョン」・「製品バージョン」（1.0.0.0）、「会社名」、「製品名」、「ファイルの説明」を設定します。
8. 「プロジェクトのエクスポート」を押し、保存先と任意のファイル名（.exe）を指定します。「デバッグ付きエクスポート」を外して保存します。
9. 出力されたEXEを起動します。同時に出力されるPCKとDLL（unagi_windows.x86_64.dll）も同じフォルダーに置いて使用してください。
10. UNAGIの電源を入れ、表示されたMACアドレスを選択します。ログメッセージに Connected と表示されれば接続完了です。
11. 電流・時間・周波数を選択して Shock! を押すと、UNAGIへコマンドを送信します。

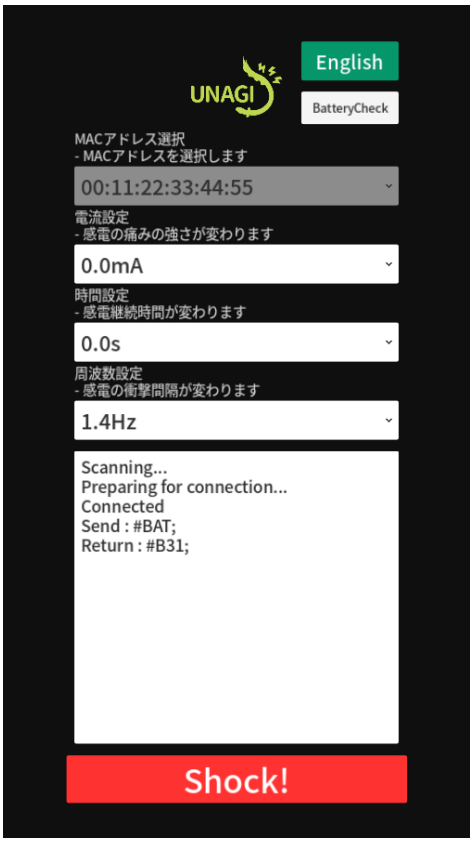
エクスポートの「リソース」は初期値の「プロジェクト内のリソースをすべてエクスポート」のままにします。addons/godotopenxrvendors が無いプロジェクトでは、XR用のファイルはUNAGI SDKのプラグインが自動で書き出し対象から外します。addons/godotopenxrvendors があるプロジェクトでは自動では外れないので、XR用のファイルを含めたくない場合は、プリセットの「リソース」→「Filters to exclude files/folders from project」に addons/unagi/editor/xr_only_files.json の各行と addons/godotopenxrvendors/* を追加してください。

Android

サンプルアプリケーションのビルドと動作

1. Android StudioのSDK Managerで、Android SDK Platform 36、Build Tools 36.1.0（Godot 4.7.2のAndroidビルドがこの版を使うため、どちらもこの版が必要です）と、最新のPlatform-Toolsをインストールします。JDKは17を用意します（18～23でも動作しますが、24以降はGodot 4.7.2のAndroidビルドが対応していないため使えません）。
2. Godotの「エディター → エディター設定」の「エクスポート → Android」で、「Java SDKパス」にJDK、「Android SDKパス」にAndroid SDKのフォルダーを指定します。
3. 「自分のプロジェクトへSDKを導入する」の手順で、レンダラーをCompatibility（互換性）にした新しいプロジェクトを作成し、addons/unagi をコピーしてUNAGI SDKを有効にします。
4. 「プロジェクト → Androidビルドテンプレートのインストール」を実行します。これはプロジェクトごとに初回の設定時に行います。
5. 「プロジェクト → プロジェクト設定」を開き、一般タブの右上にある「高度な設定」を有効にします。一般タブ左側の一覧で「アプリ」の下にある「実行」を選び、右側の「メインシーン」にres://addons/unagi/scenes_testapp/mobile_ble.tscn を指定します。欄にパスを入力するか、欄の右端にあるフォルダーのボタンからファイルを選びます。
6. 一般タブの「表示 → ウィンドウ」にある「サイズ」で、「ビューポートの幅」を720、「ビューポートの高さ」を1280、「ウィンドウ広さのオーバーライド」を504、「ウィンドウ高さのオーバーライド」を896にします。「ストレッチ」の「モード」を canvas_items、「アスペクト」を keep_height、「ポータブル」の「方向」をPortrait にします。
7. 一般タブの「レンダリング → テクスチャ」にある「VRAM圧縮」の「ETC2 ASTC をインポート」を有効にします。
8. 必要に応じて、一般タブの「アプリ → 構成」にある「名前」と「アイコン」（res://addons/unagi/icon/unagi_icon.png）を設定します。
9. 「プロジェクト → エクスポート」で「追加」から Android を選び、プリセットを作成します。「Gradleビルド」の「Gradleビルドを使用」を有効にし、「最小SDK」を24、「ターゲットSDK」を36にします。「アーキテクチャ」は arm64-v8a だけを有効にします。SDKのAndroidプラグインはGradleビルドでだけ組み込まれるため、自分のアプリをAndroidへ書き出す場合も、「Gradleビルドを使用」・「最小SDK」24以上・arm64-v8a は必要です。
10. 同じ画面の「パッケージ」で「固有名」と「名前」を、「バージョン」で「コード」（1）と「名前」（1.0.0）を設定します。必要に応じて、「ランチャーアイコン」の「Main 192x192」と「Adaptive Foreground 432x432」に res://addons/unagi/icon/unagi_icon.png などのアイコンを指定します。「権限」の追加は不要です。Bluetoothの権限（Android 12以降は「付近のデバイス」、Android 11以前はBluetoothと位置情報）とBLE対応端末の指定は、SDKのAndroidプラグインから自動で追加されます。

11. 「プロジェクトのエクスポート」を押し、「デバッグ付きエクスポート」を有効にしたまま、任意のファイル名（.apk）で保存します。「デバッグ付きエクスポート」を外すとリリース用の署名鍵（Keystore）が必要になります。
12. APKをAndroid端末へ転送して開き、インストールします。アプリを起動し、初回のBluetooth使用許可要求を許可します。検出したMACアドレスを選び、Connected と表示されれば接続完了です。



ドロップダウンの一覧は、項目上を上下にドラッグしてスクロールできます。配布用の署名を使用する場合は、エクスポート画面の「Keystore」にある「リリース」・「Release User」・「Release Password」を設定し、「デバッグ付きエクスポート」を外します。

addons/godotopenxrvendors が無いプロジェクトでは、XR用のファイルはUNAGI SDKのプラグインが自動で書き出し対象から外します。addons/godotopenxrvendors があるプロジェクトでは自動では外れないので、XR用のファイルを含めたくない場合は、プリセットの「リソース」→「Filters to exclude files/folders from project」に addons/unagi/editor/xr_only_files.json の各行と addons/godotopenxrvendors/* を追加してください。

iOS

iOSアプリケーションのビルドには、Mac環境とXcodeが必要です。

サンプルアプリケーションのビルドと動作

1. MacにGodot 4.7.2を用意し、エクスポートテンプレートマネージャーで iOS のテンプレートをインストールします。「自分のプロジェクトへSDKを導入する」の手順で、レンダラーをCompatibility（互換性）にした新しいプロジェクトを作成し、addons/unagi をコピーしてUNAGI SDKを有効にします。
2. addons/unagi/plugins/ios に iOS用通信ライブラリ UnagiIOS.xcframework が配置されていることを確認します。
3. 「プロジェクト → プロジェクト設定」を開き、一般タブの右上にある「高度な設定」を有効にします。一般タブ左側の一覧で「アプリ」の下にある「実行」を選び、右側の「メインシーン」に res://addons/unagi/scenes_testapp/mobile_ble.tscn を指定します。欄にパスを入力するか、欄の右端にあるフォルダーのボタンからファイルを選びます。Android版と共通のシーンです。
4. 一般タブの「表示 → ウィンドウ」にある「サイズ」で、「ビューポートの幅」を720、「ビューポートの高さ」を1280、「ウィンドウ広さのオーバーライド」を504、「ウィンドウ高さのオーバーライド」を896にします。「ストレッチ」の「モード」を canvas_items、「アスペクト」を keep_height、「ポータブル」の「方向」を Portrait にします。一般タブの「レンダリング → テクスチャ」にある「VRAM圧縮」の「ETC2 ASTC をインポート」を有効にします。必要に応じて、一般タブの「アプリ → 構成」にある「名前」と「アイコン」（res://addons/unagi/icon/unagi_icon.png）を設定します。
5. 「プロジェクト → エクスポート」で「追加」から iOS を選び、プリセットを作成します。「アプリ」の「App StoreチームID」にApple DeveloperのTeam ID、「Bundle Identifier」、「Short Version」（1.0.0）、「バージョン」（1）を設定し、「Min iOS Version」を15.0、「Targeted Device Family」を iPhone & iPad にします。「Export Project Only」を有効にします。
6. 「アーキテクチャ」の arm64 が有効であることを確認します。必要に応じて、「アイコン」の「Icon 1024x1024」に res://addons/unagi/icon/unagi_icon.png などのアイコンを指定します。Bluetoothの使用目的の表示（NSBluetoothAlwaysUsageDescription）は、SDKのプラグインが自動で追加します。
7. 「プロジェクトのエクスポート」から、任意の名前のXcodeプロジェクト（.xcodeproj）として保存します。「デバッグ付きエクスポート」を外すとRelease構成で書き出します。
8. 生成されたプロジェクトをXcodeで開きます。Signing & CapabilitiesでTeamとプロビジョニング等の設定を行います。
9. 接続したiPhoneまたはiPadを実行先に選び、Xcodeの実行ボタンでビルドしてインストールします。
10. 初回のBluetooth使用許可要求を許可します。表示されたDevice UUIDを選択し、Connectedと表示されれば接続完了です。

エクスポートの基本操作は、Godot公式ドキュメント「Exporting for iOS」も参照してください。

https://docs.godotengine.org/en/4.7/tutorials/export/exporting_for_ios.html

addons/godotopenxrvendors が無いプロジェクトでは、XR用のファイルはUNAGI SDKのプラグインが自動で書き出し対象から外します。addons/godotopenxrvendors があるプロジェクトでは自動では外れないので、XR用のファイルを含めたくない場合は、プリセットの「リソース」→「Filters to exclude files/folders from project」に addons/unagi/editor/xr_only_files.json の各行と addons/godotopenxrvendors/* を追加してください。

MetaQuest コントローラ版

事前にMetaQuestを開発者モードにし、USBデバッグを許可していることを前提とします。

サンプルアプリケーションのビルドと動作

1. Android StudioのSDK Managerで、Android SDK Platform 36、Build Tools 36.1.0（Godot 4.7.2のAndroidビルドがこの版を使うため、どちらもこの版が必要です）と、最新のPlatform-Toolsをインストールします。JDKは17を用意します（18～23でも動作しますが、24以降はGodot 4.7.2のAndroidビルドが対応していないため使えません）。Godotの「エディター → エディター設定」の「エクスポート → Android」で、「Java SDKパス」にJDK、「Android SDKパス」にAndroid SDKのフォルダーを指定します。すでに設定済みの場合は再設定不要です。
2. 「自分のプロジェクトへSDKを導入する」の手順で、レンダラーをMobile（モバイル）にした新しいプロジェクトを作成し、addons/unagi をコピーしてUNAGI SDKを有効にします。
3. Godot OpenXR Vendors 5.1.0 を、Godotのアセットライブラリまたは公式のリリースページ（GitHubの GodotVR/godot_openxr_vendors）から入手し、プロジェクトの addons/godotopenxrvendors に入れ、Godotを再起動します。
4. 「プロジェクト → Androidビルドテンプレートのインストール」を実行します。
5. 「プロジェクト → プロジェクト設定」を開き、一般タブの右上にある「高度な設定」を有効にします。一般タブ左側の一覧で「アプリ」の下にある「実行」を選び、右側の「メインシーン」に res://addons/unagi/scenes_testapp/quest_controller.tscn を指定します。欄にパスを入力するか、欄の右端にあるフォルダーのボタンからファイルを選びます。
6. 一般タブの「表示 → ウィンドウ」にある「サイズ」で、「ビューポートの幅」・「ウィンドウ広さのオーバーライド」を1280、「ビューポートの高さ」・「ウィンドウ高さのオーバーライド」を720にします。「ストレッチ」の「モード」を canvas_items、「アスペクト」を expand にします。
7. 一般タブの「レンダリング → レンダラー」にある「レンダリングメソッド」が mobile であることを確認します（「レンダリングメソッド.mobile」がある場合は、それも mobile にします）。「レンダリング → アンチエイリアス」にある「品質」の「MSAA 3D」を 2x に、「レンダリング → テクスチャ」にある「VRAM圧縮」の「ETC2 ASTC をインポート」を有効にします。
8. 一般タブの「XR → OpenXR」にある「有効」をオンにし、「デフォルトのアクションマップ」に res://addons/unagi/scripts/xr/controller_action_map.tres、「Reference Space」に Local を指定します。「XR → シェーダー」の「有効」もオンにします。
9. 一般タブの「XR → OpenXR」にある「Extensions」で、「User Presence」と、Meta の「Passthrough」・「Render Model」を有効にします（「Passthrough」はMRボタンのパススルー表示、「Render Model」はコントローラーの表示に使います）。「ハンドトラッキング」と、Meta の「Hand Tracking Mesh」・「Hand Tracking Aim」は無効にします。設定後、Godotを再起動します。
10. 「プロジェクト → エクスポート」で「追加」から Android を選び、プリセットを作成します。「Gradleビルド」の「Gradleビルドを使用」を有効にし、「最小SDK」を29、「ターゲットSDK」を36にします。「アーキ

テクチャ」は arm64-v8a だけを有効にします。

11. 同じ画面の「パッケージ」で「固有名」と「名前」を、「バージョン」で「コード」（1）と「名前」（1.0.0）を設定します。必要に応じて、「ランチャーアイコン」の「Main 192x192」と「Adaptive Foreground 432x432」に res://addons/unagi/icon/unagi_icon.png などのアイコンを指定します。「権限」の追加は不要です。Bluetoothの権限は、SDKのAndroidプラグインから自動で追加されます。
12. 「XR機能」の「XRモード」を OpenXR にし、「Enable Meta Plugin」を有効にします。「Meta XR Features」で、「Passthrough」と「Render Model」を Optional にし、「Quest 2 Support」・「Quest 3 Support」・「Quest Pro Support」を有効、「Quest 1 Support」を無効にします。
13. 「リソース」の「Filters to exclude files/folders from project」に、ハンドトラッキング版だけで使う addons/unagi/scenes_testapp/quest_hand_tracking.tscn, addons/unagi/prefabs/hand.tscn を追加します。
14. 「プロジェクトのエクスポート」を押し、「デバッグ付きエクスポート」を有効にして、任意のファイル名（.apk）で保存します。Meta Quest Developer HubなどでMetaQuestへインストールします。
15. アプリを起動し、初回のBluetooth使用許可要求を許可します。UNAGIを選択し、Connected と表示されれば接続完了です。
16. ポインターをUIへ向け、トリガーで選択します。一覧の項目上でトリガーを押したまま上下に動かすとスクロールできます。
17. 右手のドライバーを電気オブジェクトへ接触させると、Shock!を押した場合と同じコマンドをUNAGIへ送ります。MRボタンでパススルー表示を切り替えます。

quest_controller.tscn を開くと、UI、電気オブジェクト、XROriginとコントローラの配置を確認できます。配布用の署名方法はAndroid版と同じです。

MetaQuest ハンドトラッキング版

MetaQuest本体の設定でハンドトラッキングを有効にします。開発環境とBluetoothの使用許可は、コントローラ版と同じです。

サンプルアプリケーションのビルドと動作

1. コントローラ版の手順1～4と同じ準備をします。コントローラ版のプロジェクトを使う場合は、以下の設定に変更します。
2. 「プロジェクト → プロジェクト設定」を開き、一般タブの右上にある「高度な設定」を有効にします。一般タブ左側の一覧で「アプリ」の下にある「実行」を選び、右側の「メインシーン」に res://addons/unagi/scenes_testapp/quest_hand_tracking.tscn を指定します。欄にパスを入力するか、欄の右端にあるフォルダーのボタンからファイルを選びます。「表示」・「レンダリング」・「XR → OpenXR」・「XR → シェーダー」の設定は、コントローラ版の手順6～8と同じにします。
3. 一般タブの「XR → OpenXR」にある「Extensions」で、「User Presence」、「ハンドトラッキング」、「Hand Tracking Unobstructed Data Source」と、Meta の「Passthrough」・「Render Model」・「Hand Tracking Mesh」・「Hand Tracking Aim」を有効にします。「Hand

Interaction Profile」と「Hand Tracking Controller Data Source」は無効にします。「Hand Tracking Mesh」が無効のまま quest_hand_tracking.tscn を読み込むと、手の骨が見つからないというエラーが表示されます。設定後、Godotを再起動します。

4. 「プロジェクト → エクスポート」で、コントローラ版の手順10～12と同じ設定のAndroidプリセットを用意し、「Meta XR Features」の「ハンドトラッキング」を Required、「Hand Tracking Frequency」を High にします（手順3で「ハンドトラッキング」を有効にすると表示されます）。「パッケージ」の「固有名」と「名前」は、コントローラ版と別にします。除外フィルターには、手のシーン（quest_hand_tracking.tscn、hand.tscn）を追加しません。コントローラ版のプリセットを流用する場合は、この2項目を除外フィルターから削除します。
5. 「デバッグ付きエクスポート」を有効にし、任意のファイル名（.apk）で保存します。Meta Quest Developer HubなどでMetaQuestへインストールします。
6. コントローラを置き、MetaQuestが両手を認識した状態でアプリを使用します。
7. 手のポインターをUIへ向け、親指と人差し指をつまむ操作（ピンチ）で選択します。ピンチしたまま一覧の項目上を上下に動かすとスクロールできます。
8. 右手に付いたドライバーを電気オブジェクトへ接触させると、選択した設定でUNAGIへコマンドを送信します。

コントローラ版とハンドトラッキング版の「固有名」を別にすると、同じ端末へ両方をインストールできます。

Windows PCVR版

PCVR版はOpenXRを使用します。HMDとコントローラをPCへ接続し、使用するOpenXRランタイムを有効にしてから起動してください。

サンプルアプリケーションのビルドと動作

1. 「自分のプロジェクトへSDKを導入する」の手順で、レンダラーをMobile（モバイル）にした新しいプロジェクトを作成し、addons/unagi をコピーしてUNAGI SDKを有効にします。
2. Godot OpenXR Vendors 5.1.0 を、Godotのアセットライブラリまたは公式のリリースページ（GitHubの GodotVR/godot_openxr_vendors）から入手し、プロジェクトの addons/godotopenxrvendors に入れ、Godotを再起動します。
3. 「プロジェクト → プロジェクト設定」を開き、一般タブの右上にある「高度な設定」を有効にします。一般タブ左側の一覧で「アプリ」の下にある「実行」を選び、右側の「メインシーン」に res://addons/unagi/scenes_testapp/pcvr_ble.tscn を指定します。欄にパスを入力するか、欄の右端にあるフォルダーのボタンからファイルを選びます。
4. 一般タブの「表示 → ウィンドウ」にある「サイズ」で、「ビューポートの幅」を800、「ビューポートの高さ」を450、「ウィンドウ広さのオーバーライド」を1280、「ウィンドウ高さのオーバーライド」を720にします。「ストレッチ」の「モード」を canvas_items、「アスペクト」を expand にします。
5. 一般タブの「レンダリング → レンダラー」にある「レンダリングメソッド」が mobile であることを確認し、「レンダリング → レンダリングデバイス」にある「ドライバー.windows」を vulkan にします。
6. 一般タブの「XR → OpenXR」にある「有効」をオンにし、「デフォルトのアクションマップ」に res://addons/unagi/scripts/xr/controller_action_map.tres、「Reference Space」に Local を指定します。「XR → シェーダー」の「有効」もオンにします。
7. 一般タブの「XR → OpenXR」にある「Extensions」で、「User Presence」と Meta の「Passthrough」を有効にします（「Passthrough」はMRボタンのパススルー表示に使います）。「ハンドトラッキング」と、Meta の「Render Model」・「Hand Tracking Mesh」・「Hand Tracking Aim」は無効にします。設定後、Godotを再起動します。MetaQuest用に設定したプロジェクトを流用する場合も、「Hand Tracking Aim」は必ず無効にしてください。有効のまま Windows向けに書き出すと、HMDを接続せずに起動したときにアプリが異常終了します（Godot OpenXR Vendors 5.1.0 の不具合）。
8. 「プロジェクト → エクスポート」で「追加」から Windows Desktop を選び、プリセットを作成します。「バイナリフォーマット」の「アーキテクチャ」が x86_64 であることを確認します。必要に応じて、「アプリ」の「リソースの変更」を有効にし、「アイコン」（res://addons/unagi/icon/unagi.ico）、「ファイルバージョン」・「製品バージョン」（1.0.0.0）、「会社名」、「製品名」、「ファイルの説明」を設定します。
9. 「リソース」の「Filters to exclude files/folders from project」に、addons/unagi/scenes_testapp/quest_hand_tracking.tscn, addons/unagi/prefabs/hand.tscn を追加します。

10. 「プロジェクトのエクスポート」を押し、任意のファイル名（.exe）で、「デバッグ付きエクスポート」を外して保存します。出力されたPCKとDLL（unagi_windows.x86_64.dll、libgodotopenxrvendors.dll）も同じフォルダーに置いて使用してください。
11. Meta Horizon Linkで使用する場合は、Linkを現在のOpenXRランタイムに設定し、HMDとコントローラが認識されてからEXEを起動します。
12. UNAGIを選択し、Connected と表示されれば接続完了です。操作方法はMetaQuest コントローラ版と同じです。

PCVR版の入力はコントローラを使用します。右手のドライバーを電気オブジェクトへ接触させると、選択した設定でコマンドを送ります。MR表示は、パススルーに対応するランタイムで使用できます。

SteamVRを使用する場合は、SteamVRを現在のOpenXRランタイムに設定して、同じEXEを起動します。

この設定のプロジェクトでは、Godot Editorから pcvr_ble.tscn を実行してHMDで確認することもできます。

コマンドリスト

電流値設定

CURRENT	電流	
#A00;	0.0mA	
#A01;～#A04;	0.0mA	中間の値は直前の値と同じ
#A05;	0.5mA	0.5mA刻みで任意に設定可能
#A06;～#A09;	0.5mA	
#A10;	1.0mA	
#A11;～#A14;	1.0mA	
#A15;	1.5mA	
#A16;～#A19;	1.5mA	
#A20;	2.0mA	
#A21;～#A24;	2.0mA	
#A25;	2.5mA	
#A26;～#A29;	2.5mA	
#A30;	3.0mA	
#A31;～#A34;	3.0mA	
#A35;	3.5mA	
#A36;～#A39;	3.5mA	
#A40;	4.0mA	
#A41;～#A44;	4.0mA	
#A45;	4.5mA	
#A46;～#A49;	4.5mA	
#A50;	5.0mA	
#A51;～#A54;	5.0mA	中間の値は直前の値と同じ
#A55;以上	0.0mA	リミッタ（これ以上の値はすべて0.0mA）

※電流の値はあくまで目安であり、正確に表記の電流値が流れるものではありません。

※電流は人体の抵抗が10kΩ程度までを想定した値となっており、皮膚の乾燥状態などによって設定の電流値が流れない場合がございます。

時間設定

DURATION	持続時間	
#T00;	0.0s	0.1秒刻みで任意に設定可能
#T01;	0.1s	
#T02;	0.2s	
#T03;	0.3s	
～		中間省略
#T29;	2.9s	
#T30;	3.0s	
#T31;以上	3.0s	リミッタ（これ以上の値はすべて3.0s）

周波数設定

FREQUENCY	出力電流の周波数
#F00;	1.4Hz
#F01;	5.4Hz
#F02;	10Hz
#F03;	15.4Hz
#F04;	20Hz
#F05;	25Hz
#F06;	28.6Hz
#F07;	33.3Hz
#F08;	40Hz
#F09;	40Hz（F08と同じ）
#F10;	50Hz
#F11;～#F13;	50Hz（F10と同じ）
#F14;	66.7Hz
#F15;～#F19;	66.7Hz（F14と同じ）
#F20;	100Hz
#F21;～#F39;	100Hz（F20と同じ）
#F40;	200Hz
#F41;以上	200Hz（F40と同じ）

制御コマンド

CONTROL	制御内容	
!	電撃ON	成功時は@、電源OFF・失敗時は?が返却される
#PEN;	高電圧電源ON	
#PDS;	高電圧電源OFF	
#KIL;	動作終了コマンド	
#BAT;	バッテリ電圧確認	#B00;～#B99;が返却される

改訂履歴

版数	日付	改訂履歴
V1.0.0	2026/09/16	初版発行
V1.0.1	2026/09/30	iOSで接続の切り替え時に接続が切れることがある不具合を修正 テストアプリのiOSで、端末一覧の見出しと案内文をMACアドレス表記からUUID表記に修正

Sinfonia

シンフォニア株式会社
〒183-0023 東京都府中市宮町2-15-13第15三ツ木ビル３階