アプリ連動感電デバイス「UNAGI」
SDKドキュメント
Unreal Engine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 8
　接続 8
　UNAGIへのコマンド送信の基本形 11
　UNAGIテストアプリの諸機能 12
テストアプリのビルド準備 13
　Android・MetaQuestの環境設定 13
Windows（BLE通信） 14
　サンプルアプリケーションのビルドと動作 14
Android 15
　サンプルアプリケーションのビルドと動作 15
iOS 17
　サンプルアプリケーションのビルドと動作 17
MetaQuest コントローラ版 19
　サンプルアプリケーションのビルドと動作 19
　MetaQuest ハンドトラッキング版 20
Windows PCVR版 22
　サンプルアプリケーションのビルドと動作 22
コマンドリスト 24
　電流値設定 24
　時間設定 26
　周波数設定 27
　制御コマンド 28
改訂履歴 29
発行元 30

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Unreal Engine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は全対応プラットフォーム共通のプラグインZIP（`UNAGI_SDK_Unreal_Plugin_1.0.1.zip`）で提供します。ZIPには、通信プラグインとテストアプリのソース・レベル・UIを収めた`Plugins/UNAGI`フォルダーと第三者表記（`ThirdPartyNotices`）が含まれます。本書はZIPとは別に提供します。自分で作成したUEプロジェクトへ`Plugins/UNAGI`を入れて使用します。

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。本SDKはソースプラグインのため、Blueprint主体のプロジェクトでもC++開発環境が必要です。

- Unreal Engine 5.8.1
- Visual Studio 2022
  - C++によるゲーム開発、MSVC v143、Windows SDK
- Android・MetaQuest向けビルド環境（Windows上）
  - Android SDK（SDK Platform 34・35、Build-tools 35.0.1）
  - NDK r27c
  - JDK 21
- iOS向けビルド環境
  - Mac
  - UE 5.8.1に対応するXcodeとiOS SDK、Appleの署名設定
- Windows実行環境
  - Windows 11
  - BLE対応Bluetoothアダプター（PC内蔵またはUSB）

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

- Meta XR Plugin 207.0以降（UE 5.8用。導入手順は「テストアプリのVR機能を使う場合」を参照）

PCVRでは、HMD・コントローラーとOpenXRランタイムも必要です。Android端末とMetaQuestはUSBデバッグを有効にし、MetaQuestは開発者モードにしてください。

自分のプロジェクトへSDKを導入する

1. UE 5.8.1のプロジェクトブラウザーで「ゲーム」の「ブランク」テンプレートを選び、「プロジェクト デフォルト」でC++を選んで新しいプロジェクトを作成します。既存のプロジェクトへ導入する場合は、手順2から行います。
2. Editorを終了し、プラグインZIPを展開します。展開したフォルダー内の `Plugins/UNAGI` フォルダーを、プロジェクトの `Plugins` フォルダーへコピーします（`Plugins` フォルダーがなければ作成します）。プロジェクトのフォルダーに `Plugins/UNAGI/UNAGI.uplugin` がある状態になります。
3. Epic Games LauncherからUE 5.8を起動し、プロジェクトブラウザーの「参照」でプロジェクトの `.uproject` を選んで開きます。モジュールのリビルドを求められたら「はい」を選びます。

4. 編集 → プラグインを開き、左の一覧の「Input Devices」に「UNAGI SDK」があり、有効になっていることを確認します。

本SDKはソースプラグインのため、プロジェクトにC++のSourceモジュールが必要です。Blueprint主体の既存プロジェクトでは、ツール → 新規 C++ クラス...から空のC++クラスを一つ追加してSourceモジュールを作り、Editorをリビルドしてください。ゲーム処理はBlueprintで実装できます。Windows BLE DLLはプラグインのThirdParty/Win64からビルド時に配置されるため、このフォルダーも含めてコピーします。

テストアプリのVR機能を使う場合

同梱テストアプリのPCVR・MetaQuest用の処理（パススルー、手の表示、ハンドポインターとピンチ、コントローラーモデル）は、MetaのMeta XR Plugin（プラグイン名 `OculusXR`）を使用します。VR用の処理は、プロジェクトでOculusXRプラグインを有効にしたときだけ組み込まれます。OculusXRを有効にしないプロジェクトでは組み込まれず、Windows・Android・iOS向けのビルドには影響しません。Meta XR PluginはSDKのZIPに含まれないため、次の手順で用意します。

1. Meta XR Plugin（207.0以降、UE 5.8用）を、Metaの開発者サイトからZIPで入手します。FabのMeta XRのページは外部サイトへのリンクで、Fabからエンジンへはインストールできません。
2. Editorを終了し、ZIPを展開したプラグインのフォルダーを、エンジンの `Engine/Plugins/Marketplace` に置きます。例えば `C:\Program Files\Epic Games\UE_5.8\Engine\Plugins\Marketplace\MetaXR\OculusXR.uplugin` となる位置です。
3. 編集 → プラグインを開き、左の一覧の「Virtual Reality」にある「Meta XR」にチェックを入れて有効にし、Editorを再起動します。再起動すると、編集 → プロジェクト設定 → プラグインに「Meta XR」が表示されます。`.uproject` のPluginsにOculusXRが記載されます。Meta XRが使うOpenXRプラグインは自動で有効になります。WindowsではPCVR用、AndroidではMetaQuest用の処理が組み込まれます。
4. 編集 → プロジェクト設定 → プラグイン → Meta XRを、次の表のとおり設定します。

区分	設定
一般	XR APIをEpic Native OpenXR (Recommended)
モバイル	Passthrough Enabledを有効。Supported Meta Quest devicesにMeta Quest 2・Meta Quest Pro・Meta Quest 3・Meta Quest 3Sを追加
（編集 → プロジェクト設定 → プロジェクト → パッケージ化）	見出し「パッケージ化」の詳細設定にある「クックする追加アセットディレクトリ」に `/OculusXR/Meshes` を追加（Meta XR Pluginのコントローラーモデルをパッケージに含めるため）

5. そのほかのプロジェクト設定も、編集 → プロジェクト設定を開き、「MetaQuest コントローラ版」「MetaQuest ハンドトラッキング版」「Windows PCVR版」の表のとおり設定します。

OculusXRを有効にしたプロジェクトのビルドでは「Plugin 'UNAGI' does not list plugin 'OculusXR' as a dependency」という警告が表示されます。Meta XR Pluginを入れていないプロジェ

クトでもSDKを使えるよう、SDKの定義にはOculusXRを依存として記載していないためで、動作に問題はありません。警告を消す場合は、`Plugins/UNAGI/UNAGI.uplugin` の `Plugins` に次を追加します。`.uproject` でOculusXRの対象プラットフォーム・ターゲット・構成を限定している場合は、同じ指定（AllowList・DenyList）を付けてください。この記載はOculusXRの有効化にも作用するため、後で `.uproject` 側のOculusXRの設定を変えたときは、こちらも合わせて変えてください。

```
{
  "Name": "OculusXR",
  "Enabled": true
}
```

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のは、Game Instance Subsystem の `Unagi Client`（`Plugins/UNAGI/Source/UNAGI/Public/UnagiClient.h` の UUnagiClient）です。Windowsは同梱DLL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。

StartClient() 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 OnDevicesChanged で通知されます。一覧は GetDevices() でも取得できます。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、ConnectDevice(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 OnConnected が通知され、コマンドを送信できます。StartClient() と ConnectDevice() 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 OnConnected または IsDeviceConnected() で確認してください。

次の例では、自分のActorからUnagi Clientを取得して使用します。このように自分のC++コードからSDKを使う場合は、先にプロジェクトの `Source/<プロジェクト名>/<プロジェクト名>.Build.cs` を開き、`PublicDependencyModuleNames` の一覧に `"UNAGI"` を追加してください（例：`{ "Core", "CoreUObject", "Engine", "InputCore", "EnhancedInput", "UNAGI" }`）。Blueprintだけで使う場合と、テストアプリを動かすだけの場合は不要です。

```
// UnagiExample.h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "UnagiTypes.h"
#include "UnagiExample.generated.h"

class UUnagiClient;

UCLASS()
class AUnagiExample : public AActor
{
    GENERATED_BODY()

public:
    // UIで選んだ端末へ接続します。
    UFUNCTION(BlueprintCallable)
    void SelectDevice(const FString& Address);
```

```
protected:
    virtual void BeginPlay() override;
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override;

private:
    UFUNCTION()
    void OnDevicesChanged(const TArray<FUNAGIDeviceInfo>& Devices);

    UFUNCTION()
    void OnConnected(const FString& Address);

    UFUNCTION()
    void OnMessageReceived(const FString& Message);

    UPROPERTY()
    TObjectPtr<UUnagiClient> Unagi;
};
```

```
// UnagiExample.cpp
#include "UnagiExample.h"
#include "Engine/GameInstance.h"
#include "UnagiClient.h"

void AUnagiExample::BeginPlay()
{
    Super::BeginPlay();
    Unagi = GetGameInstance()->GetSubsystem<UUnagiClient>();
    Unagi->OnDevicesChanged.AddUniqueDynamic(this,
&AUnagiExample::OnDevicesChanged);
    Unagi->OnConnected.AddUniqueDynamic(this, &AUnagiExample::OnConnected);
    Unagi->OnMessageReceived.AddUniqueDynamic(this,
&AUnagiExample::OnMessageReceived);
    Unagi->StartClient();
}

void AUnagiExample::EndPlay(const EEndPlayReason::Type EndPlayReason)
{
    if (Unagi)
    {
        Unagi->OnDevicesChanged.RemoveAll(this);
        Unagi->OnConnected.RemoveAll(this);
        Unagi->OnMessageReceived.RemoveAll(this);
    }
    Super::EndPlay(EndPlayReason);
}
```

```
void AUnagiExample::OnDevicesChanged(const TArray<FUNAGIDeviceInfo>& Devices)
{
    for (const FUNAGIDeviceInfo& Device : Devices)
    {
        UE_LOG(LogTemp, Log, TEXT("%s"), *Device.Address); // 一覧をUIへ表示しま
す。
    }
}

void AUnagiExample::SelectDevice(const FString& Address)
{
    Unagi->ConnectDevice(Address);
}

void AUnagiExample::OnConnected(const FString& Address)
{
    UE_LOG(LogTemp, Log, TEXT("Connected: %s"), *Address);
}

void AUnagiExample::OnMessageReceived(const FString& Message)
{
    UE_LOG(LogTemp, Log, TEXT("%s"), *Message);
}
```

C++で通知を登録するときは、UFUNCTIONを持つ受信先にAddUniqueDynamicを使い、終了時のRemoveDynamicまたはRemoveAllと対にしてください。Blueprintでは、Get Game Instance SubsystemからUnagi Clientを取得して同じ順に呼び出します。Blueprintの最小サンプルは `Plugins/UNAGI/Content/Blueprints/Examples/BP_UNAGI_MinimalConnection` です。

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 SetReconnectAutomatically() 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、StartClient() でやり直します。再接続中（IsRecovering() が true の間）は ConnectDevice() 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 DisconnectDevice() で再接続を止めてから選びます。処理を終了するときは StopClient() 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 DisconnectDevice() を呼び出します。SendStimulus() 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 OnStimulusFinished が通知されます。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。Unagi Client は Game Instance Subsystem のため、StopClient() を呼ばなければレベルを切り替えても接続は保たれます。同梱テストアプリはレベルの終了時に StopClient() を呼びます。

BLE診断ログ

接続、切断、再接続、Service取得、Notify設定、Writeなどの処理結果は、Editorの出力ログまたはPlayerログへ [UNAGI BLE] から始まる形式で出力されます。診断ファイルは `UNAGIBLE.log` で、保存先は GetDiagnosticLogPath() で取得できます。

OnDiagnosticEmitted と GetDiagnostics() から、診断の内容を取得できます。診断は Severity（Info / Warning / Error）、Stage、Code、CodeName、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。GetDiagnostics() 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。Blueprintでは Write の Command に文字列を渡します。以下の例の SDK は、接続の例と同じく GetGameInstance()->GetSubsystem() で取得した Unagi Client です。

電流：1.0mA、時間：0.3s、周波数：5.4Hzの場合

```
SDK->Write(TEXT("#A10;#T03;#F01;#PEN;!"));
```

バッテリーチェック

```
SDK->RequestBattery();
// 同じコマンドを文字列で送る場合
SDK->Write(TEXT("#BAT;"));
```

残量値（0～99）は OnBatteryReceived で受け取り、最後の値は GetBatteryPercent() で参照できます（未受信は -1）。応答文字列 #Bnn; は OnMessageReceived 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

```
SDK->Write(TEXT("#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、SendStimulus() も使用できます。Blueprintでは Make UNAGIStimulationSettings の出力を Send Stimulus に接続します。

```
FUNAGIStimulationSettings Settings;
Settings.CurrentMilliAmps = 1.0f;
Settings.DurationSeconds = 0.3f;
Settings.Frequency = EUNAGIFrequency::Hz5_4;
SDK->SendStimulus(Settings);
```

SendStimulus() 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 EUNAGIFrequency の13種類（コマンドリストの周波数）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OnStimulusFinished が通知され、途中で止めるときは StopStimulus() 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SendStimulus() を使用しています。

SendStimulus() の自動停止は、通信断やアプリ停止時に物理的な停止到達を保証する仕組みではないため、UNAGI本体側の仕様とあわせて使用します。APIと型の定義は `Plugins/UNAGI/Source/UNAGI/Public` の UnagiClient.h と UnagiTypes.h を参照してください。

UNAGIテストアプリの諸機能

テストアプリは、UNAGI Sample ControlからSDKを呼び出して動作します。レベルは `Plugins/UNAGI/Content/Scenes_TestApp`、UIは `Plugins/UNAGI/Content/Prefabs` にあります（コンテンツブラウザで見るときは、設定の「プラグインコンテンツ」を有効にします。Editorのツール → UNAGI Samplesからも開けます）。UIや接触操作の参考として、必要に応じて改変してご利用ください。

起動するとスキャンが始まります。表示されたデバイスを選択し、ログに `Connected` と表示されれば接続完了です。電流・時間・周波数を選び、Shock!を押すと設定した値で刺激を要求します。設定の選択だけでは送信せず、刺激実行中の重複要求は受け付けません。使用方法と装着条件はUNAGI本体の提供資料に従ってください。

BatteryCheckはバッテリー残量の確認、Englishは日本語と英語の切り替えです。Windows・PCVR・MetaQuestでは画面の×で終了します。MR ON / MR OFFはパススルー表示を切り替えます。利用できない環境ではMR unavailableと表示されます。

未検出時は15秒間隔で `Scanning...` が表示されます。画面ログは表示範囲を超えると古い行から消えます。

テストアプリのビルド準備

テストアプリは、「自分のプロジェクトへSDKを導入する」の手順で作成したプロジェクトで、各章の手順どおりに設定して書き出します。Windows・PCVR・Android・MetaQuestはWindows上、iOSはMac上で書き出します。1つのプロジェクトで複数のアプリを書き出す場合は、書き出す前にそのアプリの設定へ切り替えてください。

テストアプリの確認では、次の描画設定も使用しています。見た目を合わせる場合の参考にしてください。

区分	設定
エンジン → レンダリング	「自動露出」「アンビエントオクルージョン」「モーションブラー」を無効、「動的グローバル イルミネーション メソッド」と「反射メソッド」をNone、「メッシュ距離フィールドの生成」「ハードウェア レイトレーシングをサポートする」「層状マテリアル」を無効
プラットフォーム → Windows	見出し「ターゲットのRHI」の「デフォルトのRHI」をDirectX 11（Windows・PCVR）

Android・MetaQuestの環境設定

1. Android Studioと、UE 5.8.1が要求するAndroid SDK・NDK r27c・JDK 21を用意します。手順はプラットフォーム → SDK 管理 → Android → インストールのヘルプ…で開くUEの資料に従ってください。
2. 編集 → プロジェクト設定 → プラットフォーム → Android SDKで、Location of Android SDK・Location of Android NDK・Location of JAVAを設定します。環境変数ANDROID_HOME・NDKROOT・JAVA_HOMEを設定している場合は空欄でもかまいません。設定したらEditorを再起動します（Android向けの書き出しの準備はEditorの起動時に行われるため、再起動するまで「プロジェクトをパッケージ化」を押せません）。
3. 編集 → プロジェクト設定 → プラットフォーム → Androidを開き、プロジェクトの構成を求められた場合は「今すぐ設定」を押します。続けて「SDKライセンスを承諾する」を押し、SDKのライセンスに同意します。
4. プラットフォーム → SDK 管理 → Androidを開き、「許可されたSDKバージョン」にインストール済みのSDKのバージョンが表示され、エラーが出ていないことを確認します。エラーが表示された場合は、手順1～2の不足しているツールとパスを修正してください。「許可されたAutoSDKバージョン」は、AutoSDK（SDKを自動で配布する仕組み）を使わない場合は確認不要です。 APKは、Distribution Signingを設定しない場合、UEのデバッグ用の鍵で署名されます。自分の署名鍵を使う場合は、keytoolで作成した鍵ストアをプロジェクトの `Build/Android` に置き、編集 → プロジェクト設定 → プラットフォーム → Androidの見出し「Distribution Signing」でKey Store、Key Alias、Key Store Password、Key Passwordを設定して、編集 → プロジェクト設定 → プロジェクト → パッケージ化の「配布用」を有効にします。同じアプリを更新する場合はパッケージ名と署名鍵を継続します。パスワードはプロジェクトの `Config/DefaultEngine.ini` に保存されるため、プロジェクトを他者へ渡すときは鍵ストアとパスワードを含めないでください。

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」の手順で新しいプロジェクトを作成し、`Plugins/UNAGI` を入れます。
2. ツール → UNAGI SamplesからWindowsサンプルを開きます。使用レベルは `L_UNAGI_Windows` です。UIは `Plugins/UNAGI/Content/Prefabs/WBP_UNAGI_Desktop` です（コンテンツブラウザで見るときは、右上の「設定」で「プラグインコンテンツ」を有効にし、プラグインのフォルダーの `Prefabs` を開きます）。プレイで画面を確認できます。
3. 編集 → プロジェクト設定を開き、次の表のとおり設定します。

区分	設定
プロジェクト → マップ＆モード	見出し「Default Maps」の「ゲームのデフォルトマップ」を `L_UNAGI_Windows`
プロジェクト → パッケージ化	見出し「パッケージ化」の詳細設定にある「パッケージ化されたビルドに含めるマップのリスト」に `L_UNAGI_Windows` を追加

4. Editor上部のツールバーのプラットフォーム → プロジェクトをパッケージ化を開き、「ビルド構成」で「開発」を選びます（配布用は「シッピング」）。同じメニューの「パッケージ」の下で「Windows」を選び、「プロジェクトをパッケージ化」を押して、空の出力先フォルダーを指定します。
5. 出力先の `Windows` フォルダーにある、プロジェクト名のEXEを起動します。EXEと同じフォルダーの内容は一緒に保持します。初回に前提コンポーネントが求められた場合は、同梱のセットアップを実行します。
6. WindowsのBluetoothを有効にし、UNAGIの電源を入れます。表示されたMACアドレスを選択し、ログにConnectedと表示されれば接続完了です。マウスで設定と各ボタンを操作します。

Android

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」の手順で新しいプロジェクトを作成し、`Plugins/UNAGI` を入れます。Meta XRプラグインは有効にしません。
2. 「Android・MetaQuestの環境設定」に従ってAndroid環境を設定します。
3. ツール → UNAGI SamplesからMobileサンプルを開きます。使用レベルは `L_UNAGI_Mobile` です。UIは `Plugins/UNAGI/Content/Prefabs/WBP_UNAGI_Mobile` です（コンテンツブラウザで見るときは、右上の「設定」で「プラグインコンテンツ」を有効にし、プラグインのフォルダーの `Prefabs` を開きます）。
4. 編集 → プロジェクト設定を開き、次の表のとおり設定します。

区分	設定
プロジェクト → マップ＆モード	見出し「Default Maps」の「ゲームのデフォルトマップ」を `L_UNAGI_Mobile`
プロジェクト → パッケージ化	見出し「パッケージ化」の詳細設定にある「パッケージ化されたビルドに含めるマップのリスト」に `L_UNAGI_Mobile` を追加
プラットフォーム → Android	APK PackagingのAndroid Package Nameを任意の名前（テストアプリは `biz.sinfonia.unagi_testandroid_unreal`）、Application Display Nameを任意の名前、OrientationをPortrait、Package game data inside .apk?を有効、Maximum supported aspect ratio.を2.4、Use display cutout region?を有効

5. Editor上部のツールバーのプラットフォーム → プロジェクトをパッケージ化を開き、「ビルド構成」で「開発」を選びます（配布用は「シッピング」）。同じメニューの「パッケージ」の下で「Android」を選び、「プロジェクトをパッケージ化」を押して、空の出力先フォルダーを指定します。
6. 出力先に作成されたフォルダーのAPKを、USBデバッグを有効にしたAndroid端末へインストールします。同じフォルダーのInstallで始まるバッチファイルでもインストールできます。
7. アプリを起動し、初回のBluetooth権限を許可します。表示されたMACアドレスを選択し、ログにConnectedと表示されれば接続完了です。タッチで操作します。

Android SDKのplatform-toolsへPATHを通した場合のインストール例です。SERIALは `adb devices` に表示された対象端末、APK_PATHは出力されたAPKのパスへ置き換えます。USB接続時のデバッグ許可は端末側で承認してください。

```
adb devices
```

```
adb -s SERIAL install -r "APK_PATH"
```

Android 12以降は「付近のデバイス」、Android 11以前は位置情報権限を使用します。権限を拒否した場合はOSのアプリ設定から許可し、アプリを再起動してください。

iOS

iOSアプリケーションのビルドにはMac環境が必要です。署名に使用するAppleのTeam、証明書、端末へのインストールを許可するプロビジョニングを用意してください。

サンプルアプリケーションのビルドと動作

1. Macで、「自分のプロジェクトへSDKを導入する」の手順で新しいプロジェクトを作成し、`Plugins/UNAGI` を入れます。Meta XRプラグインは有効にしません。
2. ツール → UNAGI SamplesからMobileサンプルを開きます。使用レベルは `L_UNAGI_Mobile` です。UIは `Plugins/UNAGI/Content/Prefabs/WBP_UNAGI_Mobile` です（コンテンツブラウザで見るときは、右上の「設定」で「プラグインコンテンツ」を有効にし、プラグインのフォルダーの `Prefabs` を開きます）。
3. 編集 → プロジェクト設定を開き、次の表のとおり設定します。Xcode側にも同じTeamの署名環境を用意してください。

区分	設定
プロジェクト → マップ＆モード	見出し「Default Maps」の「ゲームのデフォルトマップ」を `L_UNAGI_Mobile`
プロジェクト → パッケージ化	見出し「パッケージ化」の詳細設定にある「パッケージ化されたビルドに含めるマップのリスト」に `L_UNAGI_Mobile` を追加
プラットフォーム → iOS	Bundle InformationのBundle Identifierを任意の名前（テストアプリは `biz.sinfonia.unagi-unreal`）、Bundle Display Nameを任意の名前。見出し「オリエンテーション」では、Supports Portrait Orientationだけを有効にし、Supports Upside Down Orientation・Supports Landscape Left Orientation・Supports Landscape Right Orientationを無効
プラットフォーム → Xcodeプロジェクト	見出し「Xcode」の「バンドルID」にBundle Identifierと同じ値を設定。自動署名では、見出し「Xcode」の「Apple Developerアカウントのチーム ID」にTeam IDを入力し、見出し「コード署名」の「自動コード署名を使用」を有効。手動署名では、見出し「コード署名」の「自動コード署名を使用」を無効にし、「iOS / tvOS: 署名ID」と「iOS: プロビジョニングプロファイル」を設定

4. Editor上部のツールバーのプラットフォーム → プロジェクトをパッケージ化を開き、「ビルド構成」で「開発」を選びます（配布用は「シッピング」）。同じメニューの「パッケージ」の下で「IOS」を選び、「プロジェクトをパッケージ化」を押して、空の出力先フォルダーを指定します。
5. 出力されたIPA（Xcodeの設定によっては署名済みの.app）を、署名で許可されたiPhoneまたはiPadへXcodeのDevices and Simulatorsなどでインストールします。

6. アプリを起動し、初回のBluetooth使用許可を承認します。表示されたDevice UUIDを選択し、ログにConnectedと表示されれば接続完了です。縦画面でタッチ操作します。

Bluetoothの使用目的の表示（Info.plist）は、SDKがビルド時に追加します。署名エラーが出た場合は、Xcodeプロジェクトの署名設定とXcodeのアカウント設定を確認してください。

MetaQuest コントローラ版

事前にMetaQuestを開発者モードにし、USB接続時のデバッグ許可を承認してください。

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」の手順で新しいプロジェクトを作成し、`Plugins/UNAGI` を入れます。
2. 「Android・MetaQuestの環境設定」に従ってAndroid環境を設定します。
3. 「テストアプリのVR機能を使う場合」の手順で、Meta XR Pluginを入れて有効にし、編集 → プロジェクト設定 → プラグイン → Meta XRを設定します。
4. ツール → UNAGI SamplesからQuest Controllerサンプルを開きます。使用レベルは `L_UNAGI_QuestController` です。UIは `Plugins/UNAGI/Content/Prefabs/WBP_UNAGI_VR` です（コンテンツブラウザで見るときは、右上の「設定」で「プラグインコンテンツ」を有効にし、プラグインのフォルダーの `Prefabs` を開きます）。
5. 編集 → プロジェクト設定を開き、次の表のとおり設定します。

区分	設定
プロジェクト → マップ＆モード	見出し「Default Maps」の「ゲームのデフォルトマップ」を `L_UNAGI_QuestController`
プロジェクト → パッケージ化	見出し「パッケージ化」の詳細設定にある「パッケージ化されたビルドに含めるマップのリスト」に `L_UNAGI_QuestController` を追加
プロジェクト → 説明	見出し「設定」の「VRで開始」を有効
エンジン → 入力	見出し「入力」の「モーションコントロールを有効化」を有効
プラットフォーム → Android	Advanced APK PackagingのPackage for Meta Quest devices.とPackage for OpenXR Immersive devices in general.を有効。APK PackagingのAndroid Package Nameを任意の名前（テストアプリは `biz.sinfonia.unagi_testcontroller_unreal` ）、Application Display Nameを任意の名前、OrientationをLandscape、Package game data inside .apk?を有効

区分	設定
エンジン → レンダリング	モバイルの「モバイルシェーディング」をForward Shading、モバイルの「アンチエイリアス手法」をMultisample Anti-Aliasing (MSAA)、Default Settingsの「MSAAサンプルカウント」を4x MSAA、VRの「モバイルHDR」を無効、VRの「モバイルマルチビュー」を有効
プラグイン → Meta XR	モバイルのHand Tracking SupportをControllers Only
パススルー（MR ON / OFFを使う場合）	エンジン → レンダリング → モバイルの「モバイルアルファ出力」と、エンジン → レンダリング → Default Settingsの「アルファ出力」を有効

6. Editor上部のツールバーのプラットフォーム → プロジェクトをパッケージ化を開き、「ビルド構成」で「開発」を選びます（配布用は「シッピング」）。同じメニューの「パッケージ」の下で「Android」を選び、「プロジェクトをパッケージ化」を押して、空の出力先フォルダーを指定します。
7. 出力されたAPKを、Androidの章に記載したadbコマンドでMetaQuestへインストールします。
8. アプリを開き、初回のBluetooth権限を許可します。UNAGIを選択し、ログにConnectedと表示されれば接続完了です。コントローラーのポインターとトリガーでUIを操作します。長い選択リストはスティック上下で移動できます。

MetaQuest用のマニフェストの記載（ヘッドトラッキング、パススルー、対応機種）は、Package for Meta Quest devices.とMeta XRの設定から、Meta XR Pluginがビルド時に追加します。UE 5.8はこの設定のAPKへハンドトラッキングの権限も自動で追加しますが、コントローラ版の動作には影響しません。

右手のドライバーを電気オブジェクトElectricCubeへ接触させると、Shock!と同じ設定で刺激を要求します。PawnのContact Stimulation RequestedからUNAGI Sample Controlを呼び出す接触処理です。

接触範囲は `BP_UNAGIElectricCube` の `AuthoredTouchVolume` の位置とBox Extentで編集できます。コンポーネント名とOverlap設定は維持してください。ElectricCubeの位置・回転を変えると、表示と接触判定が一緒に移動します。Editor上の基準目線は `Contact Object Preview Eye Height` の160cmで、起動時に実際の目線へ合わせます。上下位置は、この基準からElectricCubeのZを変更して調整します。

MetaQuest ハンドトラッキング版

MetaQuest コントローラ版と同じビルド環境を使用します。ヘッドセット側のハンドトラッキングを有効にしてください。

サンプルアプリケーションのビルドと動作

1. MetaQuest コントローラ版の「サンプルアプリケーションのビルドと動作」の手順1～3を行います。
2. ツール → UNAGI SamplesからQuest HandTrackingサンプルを開きます。使用レベルは `L_UNAGI_QuestHandTracking` です。UIは `Plugins/UNAGI/Content/Prefabs/WBP_UNAGI_VR` です（コンテンツブラウザで見るときは、右上の「設定」で「プラグインコンテンツ」を有効にし、プラグインのフォルダーの `Prefabs` を開きます）。
3. 編集 → プロジェクト設定を開き、コントローラ版の表のとおり設定します。ただし、次の表の項目はこちらの値にします。

区分	設定
プロジェクト → マップ＆モード	見出し「Default Maps」の「ゲームのデフォルトマップ」を `L_UNAGI_QuestHandTracking`
プロジェクト → パッケージ化	見出し「パッケージ化」の詳細設定にある「パッケージ化されたビルドに含めるマップのリスト」に `L_UNAGI_QuestHandTracking` を追加
プラットフォーム → Android	APK PackagingのAndroid Package Nameを任意の名前（テストアプリは `biz.sinfonia.unagi_testhand_unreal`）
プラグイン → Meta XR	モバイルのHand Tracking SupportをControllers And Hands、Hand Tracking FrequencyをHIGH、Hand Tracking VersionをV2

4. Editor上部のツールバーのプラットフォーム → プロジェクトをパッケージ化を開き、「ビルド構成」で「開発」を選びます（配布用は「シッピング」）。同じメニューの「パッケージ」の下で「Android」を選び、「プロジェクトをパッケージ化」を押して、空の出力先フォルダーを指定します。
5. 出力されたAPKをadbコマンドでMetaQuestへインストールします。
6. アプリを開き、Bluetooth権限を許可してUNAGIを選択します。Connectedを確認し、ハンドポインターと親指・人差し指のピンチでUIを操作します。

右人差し指に追従するドライバーをElectricCubeへ接触させると、Shock!と同じ設定で刺激を要求します。接触対象の編集方法はコントローラ版と同じです。

ハンドトラッキングの権限・機能の宣言と頻度・版の指定（`com.oculus.handtracking.frequency`・`com.oculus.handtracking.version`）は、Meta XR Pluginが上の設定からマニフェストへ追加します。手のメッシュはMetaQuestのランタイムから取得して表示します。

Tips：コントローラ版とハンドトラッキング版を別アプリケーションとして使用したい場合、Android Package Nameを別の名前で設定します。1つのプロジェクトで両方を書き出す場合は、書き出す前に「ゲームのデフォルトマップ」「パッケージ化されたビルドに含めるマップのリスト」、Android Package Name、編集 → プロジェクト設定 → プラグイン → Meta XRのHand Tracking Supportを、書き出すアプリに合わせて切り替えます。

Windows PCVR版

Windows PCVR版は、Meta Horizon Linkで接続したMetaQuestで使用します。Meta Horizon LinkをWindowsのOpenXRランタイムに設定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」の手順で新しいプロジェクトを作成し、`Plugins/UNAGI` を入れます。
2. 「テストアプリのVR機能を使う場合」の手順で、Meta XR Pluginを入れて有効にし、編集 → プロジェクト設定 → プラグイン → Meta XRを設定します。
3. ツール → UNAGI SamplesからPCVRサンプルを開きます。使用レベルは `L_UNAGI_PCVR` です。UIは `Plugins/UNAGI/Content/Prefabs/WBP_UNAGI_VR` です（コンテンツブラウザで見るときは、右上の「設定」で「プラグインコンテンツ」を有効にし、プラグインのフォルダーの `Prefabs` を開きます）。VR プレビューで確認できます。
4. 編集 → プロジェクト設定を開き、次の表のとおり設定します。

区分	設定
プロジェクト → マップ＆モード	見出し「Default Maps」の「ゲームのデフォルトマップ」を `L_UNAGI_PCVR`
プロジェクト → パッケージ化	見出し「パッケージ化」の詳細設定にある「パッケージ化されたビルドに含めるマップのリスト」に `L_UNAGI_PCVR` を追加
プロジェクト → 説明	見出し「設定」の「VRで開始」を有効
エンジン → 入力	見出し「入力」の「モーションコントロールを有効化」を有効
エンジン → 拡張入力	見出し「Enhanced Input」の「デフォルトのマッピングコンテキスト」に `IMC_UNAGI_XR`（`Plugins/UNAGI/Content/Input/XR`）を追加し、「デフォルトのマッピングコンテキストを有効化」を有効。VR プレビューでもこの設定を使用します
エンジン → レンダリング	フォワードレンダラの「フォワードシェーディング」を有効（「アンチエイリアス手法」は自動でMSAAになります）、「MSAAサンプルカウント」を4x MSAA、Default Settingsの「ブルーム」「自動露出」「レンズフレア(画像ベース)」を無効

区分	設定
パススルー（MR ON / OFFを使う場合）	エンジン → レンダリング → Default Settingsの「アルファ出力」を有効

5. Editor上部のツールバーのプラットフォーム → プロジェクトをパッケージ化を開き、「ビルド構成」で「開発」を選びます（配布用は「シッピング」）。同じメニューの「パッケージ」の下で「Windows」を選び、「プロジェクトをパッケージ化」を押して、空の出力先フォルダーを指定します。
6. HMDとコントローラーを接続し、次の「Meta Horizon Linkで起動する場合」の手順で、出力先の `Windows` フォルダーにあるプロジェクト名のEXEを起動します。EXEと同じフォルダーの内容は一緒に保持します。UNAGIとの通信にはWindows PCのBLE対応Bluetoothアダプターを使用します。
7. UNAGIを選択し、ログにConnectedと表示されれば接続完了です。コントローラーのポインターとトリガーで操作します。

Meta Horizon Linkで起動する場合

LinkアプリのSettings → General → OpenXR RuntimeでMeta Horizon Linkを有効なランタイムに設定し、ヘッドセット側でQuest Linkを開始してからEXEを起動します。

パススルーを使用する場合は、Settings → DeveloperのDeveloper Runtime FeaturesとPassthrough over Meta Horizon Linkを有効にします。MR機能はランタイム側の対応にも依存します。

コマンドリスト

次の表はUNAGI本体へ送信するコマンドです。`#` で始まる設定・制御コマンドは末尾に `;` を付けます。刺激開始の `!` は単独の1文字です。複数のコマンドは `#A10;#T03;#F01;#PEN;!` のように連結できます。

表に示す中間値・範囲外の動作は本体の通信仕様です。Unrealの `SendStimulus` は「UNAGIへのコマンド送信の基本形」に記載した入力範囲に従います。Raw送信で本体のリミッタに依存した値を設定せず、通常は型付きAPIを使用してください。

電流値設定

CURRENT	電流	備考
#A00;	0.0mA	
#A01;～#A04;	0.0mA	中間の値は直前の設定段階と同じ。
#A05;	0.5mA	0.5mA刻みで設定可能。
#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	

CURRENT	電流	備考
#A50;	5.0mA	
#A51;～#A54;	5.0mA	#A50;と同じ。
#A55;以上	0.0mA	この範囲では電流値は0.0mAになります。

型付きAPIでは0.0～5.0mAを0.5mA刻みで指定します。高電圧電源をOFFにする場合は `#PDS;` を使用します。

電流値は目安であり、正確に表記の電流値が流れるものではありません。人体の抵抗が10kΩ程度までを想定した値のため、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あります。

時間設定

DURATION	持続時間	備考
#T00;	0.0s	0.1秒刻みで設定可能。
#T01;	0.1s	
#T02;	0.2s	
#T03;	0.3s	
#T04;～#T28;	0.4s～2.8s	中間の値も0.1秒刻み。
#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。

`#Txx;` のxxは持続時間を10倍した2桁の値です。例えば0.3秒は `#T03;` です。Unrealの型付きAPIでは `DurationSeconds` へ秒数をそのまま渡します。

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）

Unrealの `EUNAGIFrequency` は13段階の周波数を表します。列挙値の通し番号をそのまま `#Fxx;` に変換しないでください。例えば `Hz50` に対応するコマンドは `#F10;` です。変換は型付きAPIが行います。

制御コマンド

CONTROL	制御内容	応答・備考
!	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	刺激開始前に送信する。
#PDS;	高電圧電源OFF	高電圧電源への電源供給を切る。
#KIL;	動作終了コマンド	本体の動作を終了するコマンド。
#BAT;	バッテリー確認	#B00;～#B99;が返却される。値はおおまかな目安。

`#PEN;` に続けて `!` を送ると、設定した電流・時間・周波数で刺激を開始します。`SendStimulus` は設定コマンドから刺激開始までをまとめて送ります。高電圧電源をOFFにするときは `StopStimulus` を使用できます。

SDKの接続確認には `#VER;` を使用し、`#Vdd;` の応答を待ちます。`dd` は本体のバージョンを表す2桁です。`@`・`?` や `#Bdd;` などの応答文字列は `OnMessageReceived`、バッテリー値は `OnBatteryReceived` で受け取ります。

`StopClient`・`DisconnectDevice` はBLEセッションを管理するSDKのAPIであり、`#KIL;` と同じ操作ではありません。Rawコマンドを使用する場合は、必要な本体操作とアプリの接続状態をそれぞれ管理してください。

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 Mac・iOSでビルドできない不具合を修正 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