

MxE チュートリアル #1 SDK操作ガイド  
シングルボタンアプリ

2023.07 MxEサービスドキュメント

# 目次

- [本書の目的](#)
- [チュートリアルの内容](#)
- [対象となる読者と事前準備](#)
- [Step1 サンプルアプリの確認](#)
- [Step2 プロジェクト作成](#)
- [Step3 PSDファイルのインポート](#)
- [Step4 スクリプトの実装](#)
  - [Step4（前編）ボタンの有効化と状態別の表示変更](#)
  - [Step4（後編）イベント受信と対応動作のコーディング](#)
- [おわりに](#)

# 本書の目的

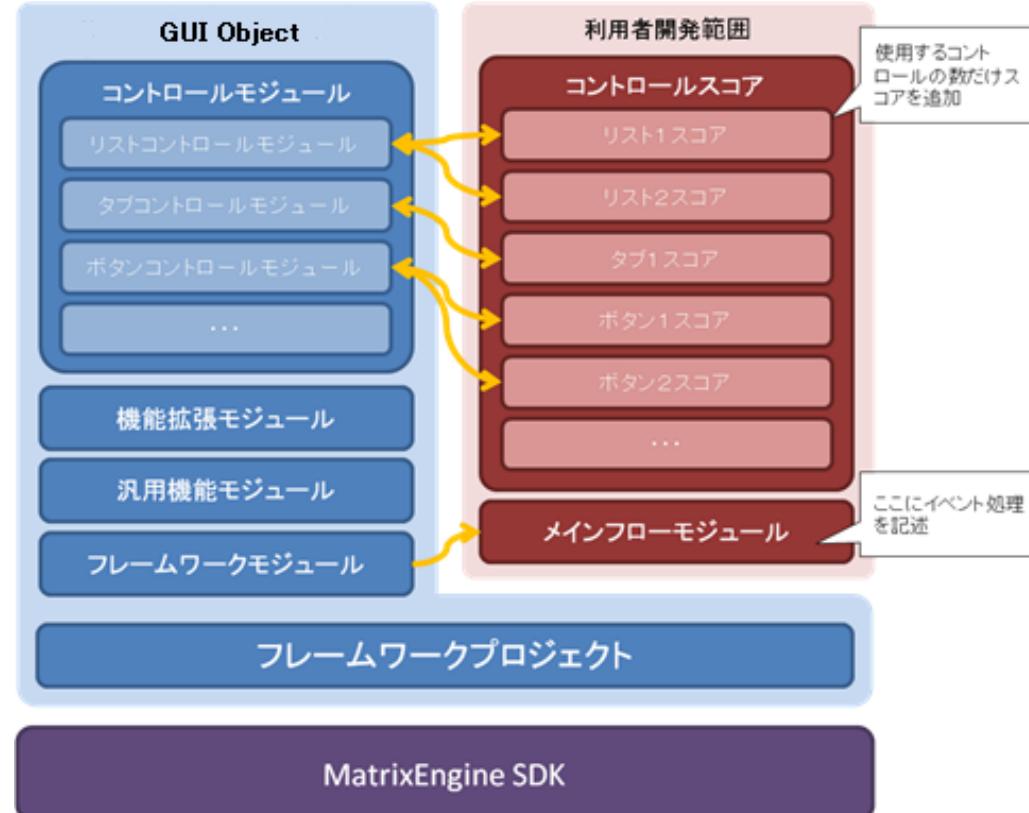
---

- この文書は、初めてMatrixEngine SDK (以降「MxE SDK」と略します)に触れる方に、動作や作りやすさを実際に見てもらうための手引きです。
- 本書は、MatrixEngine専用ライブラリ「GUI Object (※)」を利用したGUIアプリケーション開発の基本的な流れを習得していただくことを目的としています。
- サンプルの「シングルボタンアプリ（画面に一つのボタンを表示するアプリ）」を対象に、MxE SDKの操作を学びながら、同じアプリを実際に作成します。

## ※ GUI Objectについて

- MxE SDKは、専用ライブラリを利用した開発からスクラッチ開発まで、GUIアプリケーション作成に向けたいいくつかの方法を提供しています。
- 本書では、専用ライブラリ群「GUI Object」を利用したアプリケーション作成をご紹介します。
- 「GUI Object」を利用したアプリケーション開発は、[MxE SDKが最も推奨する開発スタイル](#)です。
- 詳細について知りたい場合は、GUI Object付属ドキュメントもご参照ください。

# GUI Objectによる開発の効率化



- 「GUI Object」とは、MatrixEngineスクリプトで記述された専用ライブラリ群です。コントロールモジュール（UI部品）とフレームワークを含む様々なモジュール群から成り立ちます。
- コントロールモジュールは、フレームワークモジュールと連携し、GUIの動作を自動制御します。
- アプリケーション開発者は、UIイベントをハンドリングするコードを記述するだけでイベントに応じた機能を実装することができます。

「GUI Object」によるアプリケーション開発は、  
MxE SDKが最も推奨する開発スタイルです。

\* 詳しい仕組みについては、GUI Object付属ドキュメントの「GUI Object共通マニュアル(One版).docx」もご参照ください\*

# チュートリアルの内容

- このチュートリアルは次のように構成されています。



- 本書で作成するアプリの完成イメージを確認します。



- GUI Object用のプロジェクトを新規に作成します。



- アプリで利用する画像をプロジェクトに取り込みます。



- 前編では表示処理、後編はイベントに応じたログ出力を実装します。

- STEP1では当社より提供したサンプルの動作確認を行います。STEP2以降では、実際にプロジェクトを作り、サンプルと同じアプリを作成していきます。
- 各ステップで利用するファイルについては、それぞれのステップ冒頭でご案内します。必要ファイルが揃っているか確認しながら進めてください。

# 対象の読者と事前準備

---

- 本書は次の方を読者に想定します。
  - 別資料「MatrixEngine概要」を読み、MatrixEngineの基本的な用語を理解している方。
  - 別資料「インストールおよび環境構築ガイド」に従って、MatrixEngine SDKのインストールと環境構築が完了している方。
  - C言語の基礎とイベント駆動型アプリケーションの概念を理解している方。
- 本チュートリアル開始には次の準備が必要です。ご確認ください。
  - MxE SDKのインストールが完了している
  - MxE SDKに拡張機能「PSD Importer」を設定済みである
  - GUI Objectのアーカイブファイル（GUIObject\_One\_yyyymmdd.zip）を取得済みである

# Step1 サンプルアプリの確認

---

MxE SDKでサンプルアプリの動作を確認する

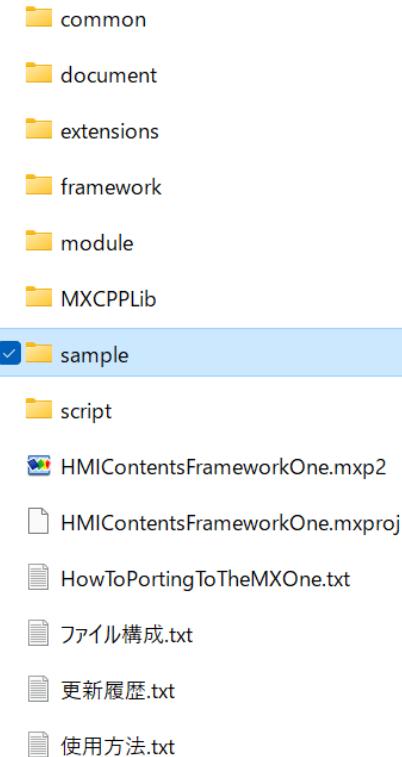
# 1-1. サンプルファイルの確認とSDKの起動

- 本チュートリアルで利用するサンプルは、当社よりお渡ししたアーカイブファイル（**GUIObject\_One\_yyyymmdd.zip**）に含まれています

## 操作

1. アーカイブファイル「**GUIObject\_One\_yyyymmdd.zip**」を任意の場所に展開してください。
2. 展開したフォルダ内に「sample」フォルダがあることを確認します。
3. 「sample」フォルダを開き、「**HMIContentsOneButtonSample.mxp2**」ファイルをダブルクリックしてください。
4. MxE SDKが起動し、シングルボタンアプリのプロジェクトが開きます。

- ✓ 本書では以降、アーカイブ展開フォルダを<GUIObject\_One フォルダ>と呼称します。
- ✓ 「sample」フォルダには、GUI Objectのサンプルが複数格納されています。
- ✓ このチュートリアルでご紹介するシングルボタンアプリのプロジェクトファイル名は「**HMIContentsOneButtonSample.mxp2**」です。



# 1-2. サンプルアプリの動作確認

- サンプルの動作を確認しましょう。
- MxE SDK上でプロジェクトを動作させるには、"Tool Box"の"START"ボタンを利用します。

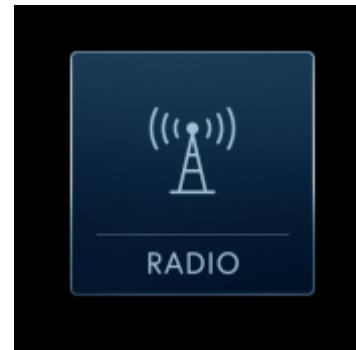
## 操作

1. "Tool Box" ウィンドウの"Control"タブを開きます。
  2. "START"ボタンを押下してください。右のような画面が表示されます。
  3. 表示された画面で、ボタンにマウスカーソルを合わせ、クリックしてみてください。
- ✓ マウスオーバー時、クリック時にボタンの色が変わることを確認してください。
- ✓ 確認後は、"Option"メニューから終了、もしくはウィンドウ右上の「×」ボタンを押下してアプリのウィンドウを閉じてください。



# 補足：状態別のボタン表示一覧

- サンプルアプリのボタン表示を状態別に一覧します。
- 通常→ホバリング状態→クリック時と明るさが変化します。



通常表示



マウスオーバー  
(ホバリング状態)



クリック時

# Step2 プロジェクト作成

---

GUI Object用のプロジェクトを新規に作成する

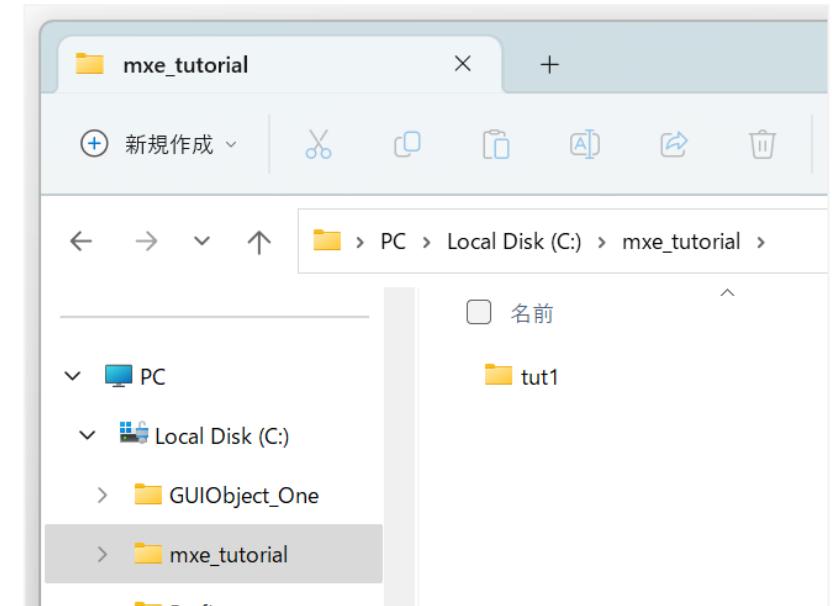
# 2-1. プロジェクトフォルダの新規作成

- アプリを作成するにあたって、任意の場所にプロジェクト用のフォルダを新規作成します。

## 操作

1. エクスプローラ上で、任意の場所にフォルダを作成してください。

- ✓ 本書では以降、プロジェクト用にフォルダ「C:\mxeTutorial\tut1」を作成したとして例示します。
- ✓ また、ここで作成したフォルダを以降「プロジェクトフォルダ」と呼称します。

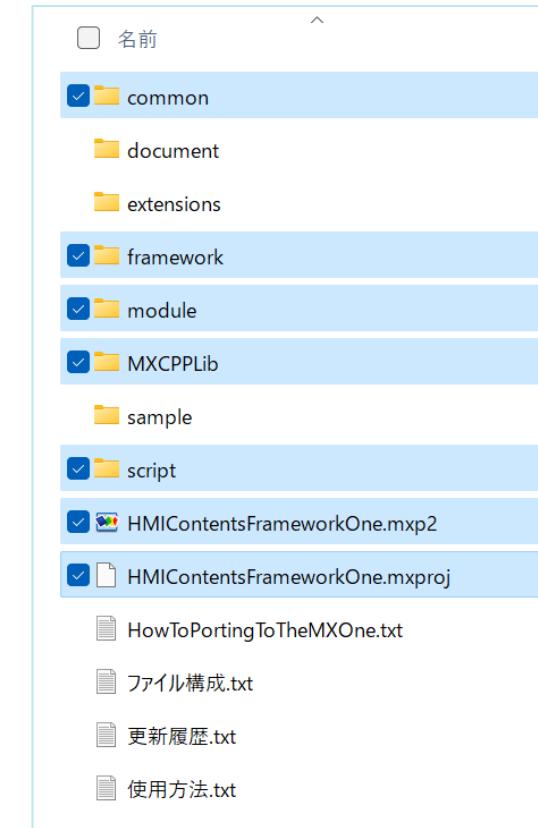


## 2-2. テンプレートの複製

- 2D GUIを作る環境のテンプレートとして、<GUIObject\_Oneフォルダ>直下に 「HMIContentsFrameworkOne.mxp2」 が用意されています。

### 操作

1. <GUIObject\_Oneフォルダ> から、次のファイルとフォルダを選択し、  
コピー (Ctrl+C) します。
  - HMIContentsFrameworkOne.mxp2
  - HMIContentsFrameworkOne.mxproj
  - common フォルダ（と中身）
  - framework フォルダ（と中身）
  - module フォルダ（と中身）
  - MXCPPLib フォルダ（と中身）
  - script フォルダ（と中身）
2. 2-1で作成したプロジェクトフォルダ内に上記すべてを貼り付け  
(Ctrl+V) してください。



## 2-3. プロジェクトファイルのリネーム

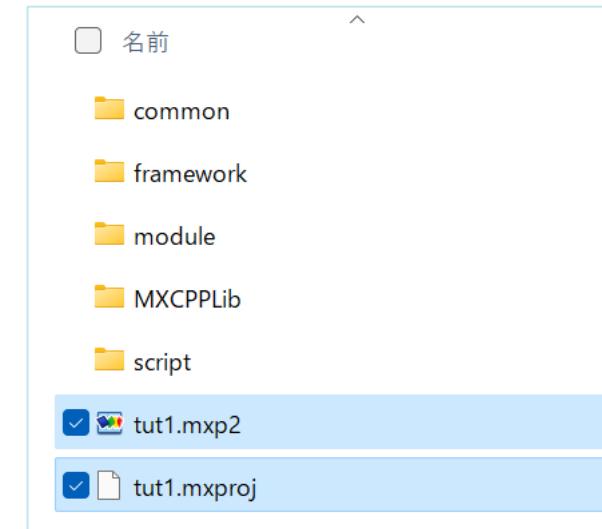
- プロジェクトファイル名をわかりやすいように変更します。「.mxp2」「.mxproj」の二つのファイルを同じ名称でリネームします。

### 操作

1. エクスプローラ上で、次のようにファイル名を変更してください。

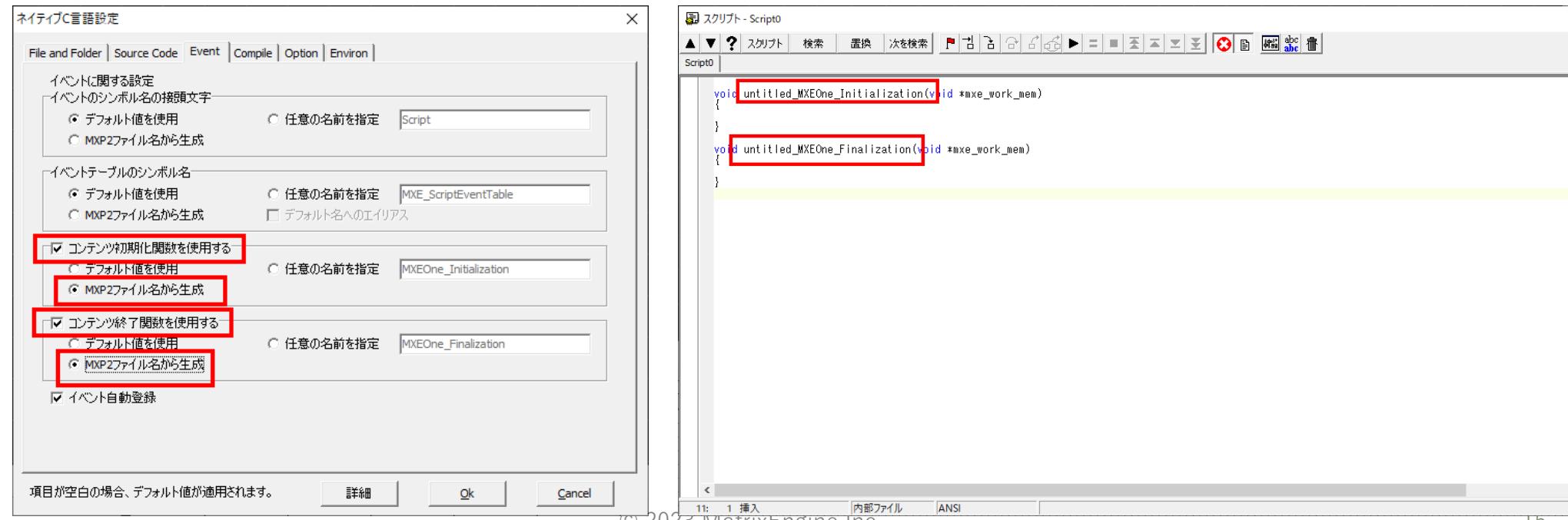
- HMIContentsFrameworkOne.mxp2 → tut1.mxp2
- HMIContentsFrameworkOne.mxproj → tut1.mxproj

- ✓ MxE SDKで「名前を付けて保存」を実行してもリネームが可能です。この場合、元ファイルとは別に、指定した名称で「.mxp2」「.mxproj」が保存されます。
- ✓ 一部のプロジェクトは[リネームに注意が必要](#)です。



# プロジェクトリネームの注意点

- 下記すべての条件を満たす場合、MXP2ファイル名と関数名が紐づいているためコンテンツのソースコードの当該箇所を修正する必要があります。
  - C言語スクリプトを使用している。
  - C言語スクリプトの「コンテンツ初期化関数」もしくは「終了関数」を使用している。
  - 上記関数名に「MXP2ファイル名から生成」を選択している。



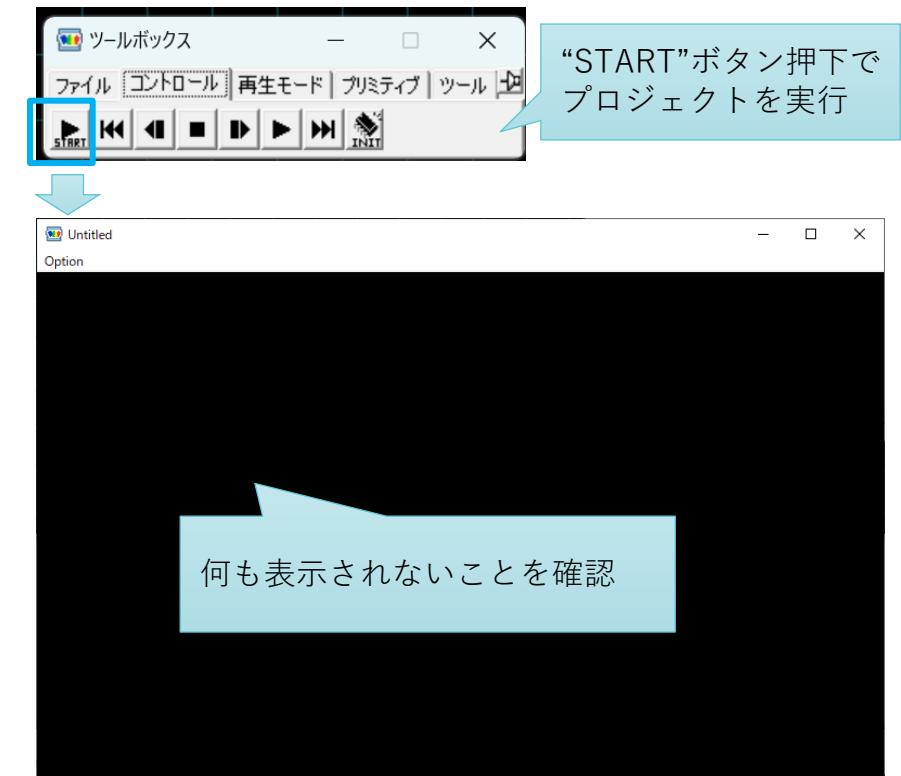
## 2-4. プロジェクトの動作確認

- 複製したプロジェクトの動作を確認します。
- MxE SDK上でプロジェクトを動作させるには、"Tool Box"の"START"ボタンを利用します。

### 操作

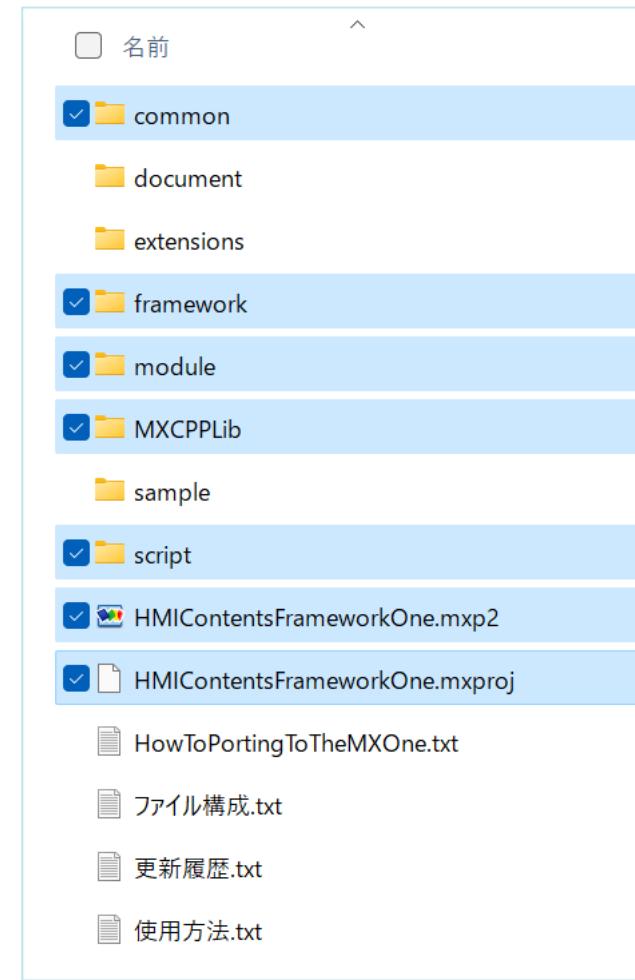
1. <プロジェクトフォルダ>の「tut1.mxp2」ファイルをダブルクリックして、MxE SDKを起動してください。
2. "Tool Box" ウィンドウの"Control"タブを開きます。
3. "START"ボタンを押下してください。
4. 何も動作しない（真っ黒な）アプリが立ち上がることを確認してください。

✓ 確認後は、"Option"メニューから終了、もしくはウィンドウ右上の「×」ボタンを押下してアプリのウィンドウを閉じてください。



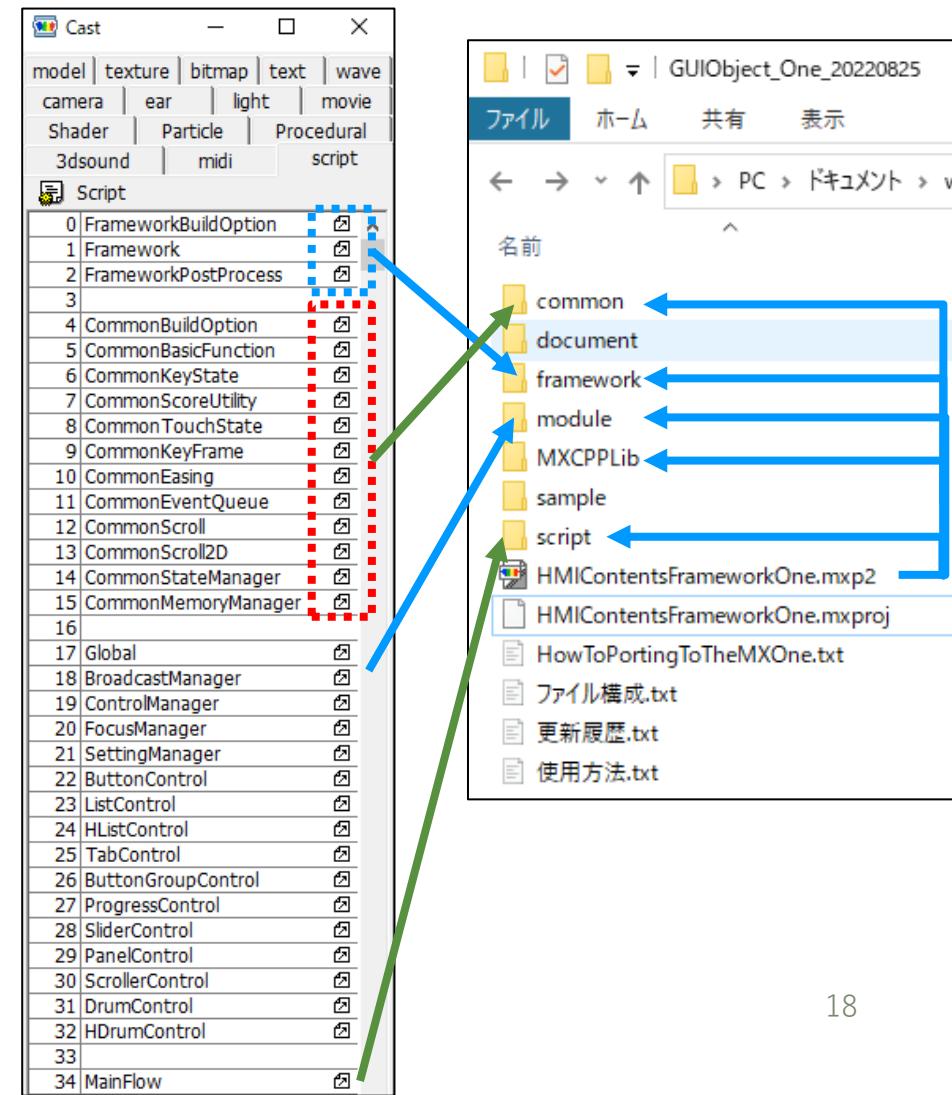
# テンプレートのファイル構成

- 次のファイル一式が、2DのHMI制作に特化したUIライブラリ (GUIObject) を持つプロジェクトのテンプレートです。
  - HMIContentsFrameworkOne.mxp2
  - HMIContentsFrameworkOne.mxproj
  - common フォルダ（と中身）
  - framework フォルダ（と中身）
  - module フォルダ（と中身）
  - MXCPPLib フォルダ（と中身）
  - script フォルダ（と中身）
- HMI制作の際には、テンプレートに様々なUI部品（画像やそれを制御するコントロール、テキストやスクリプト等）を加えていき、ゴールとなるUIコンテンツを組み上げていきます。



# テンプレートファイルの依存関係

- 2D GUIを作る環境のテンプレートとして、「**HMIContentsFrameworkOne.mxp2**」が用意されています。
  - このテンプレートプロジェクト（の中にあるスクリプト）は、同じ層にある**common**、**framework**、**module**、**script**フォルダの中のスクリプトを参照しています。
  - また、テンプレートプロジェクトは、ビルド時に「**MXCPPLib**」フォルダの中のC言語コードを必要とします。
- ✓ このため、テンプレートを複製する際には単一ファイルではなく、依存関係のあるファイルすべてをコピーする必要があります。



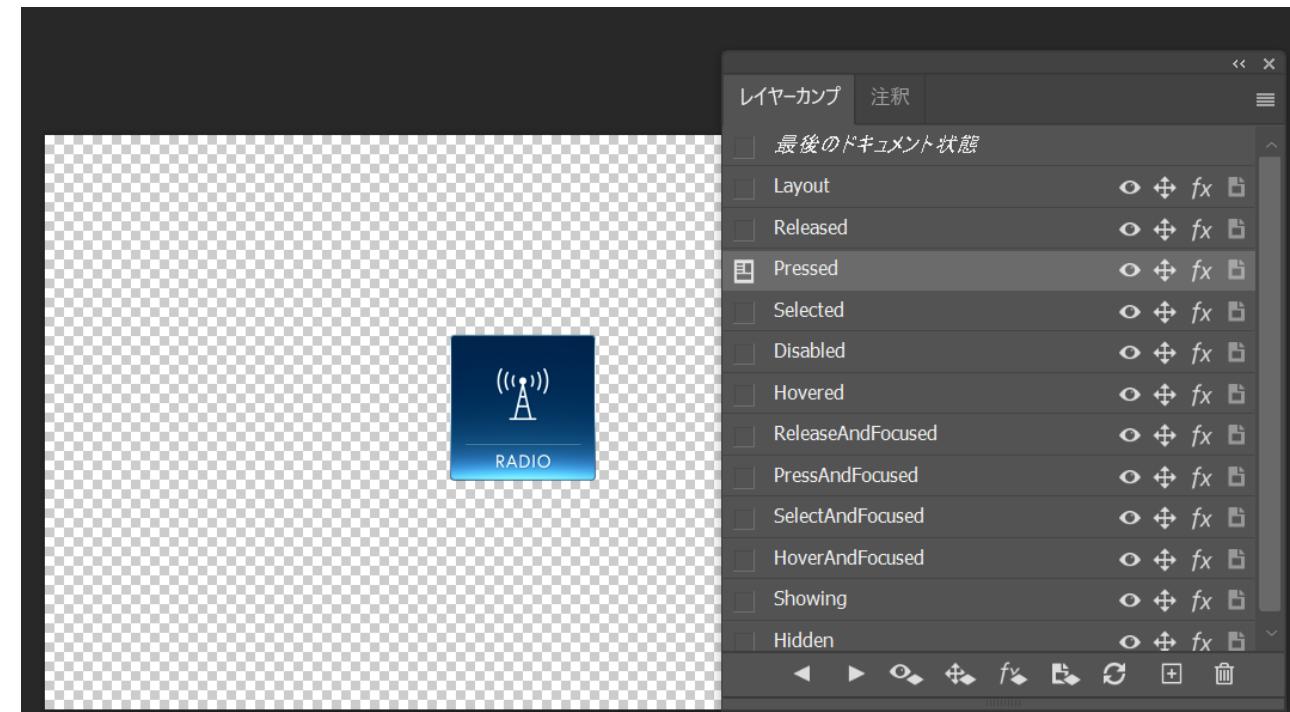
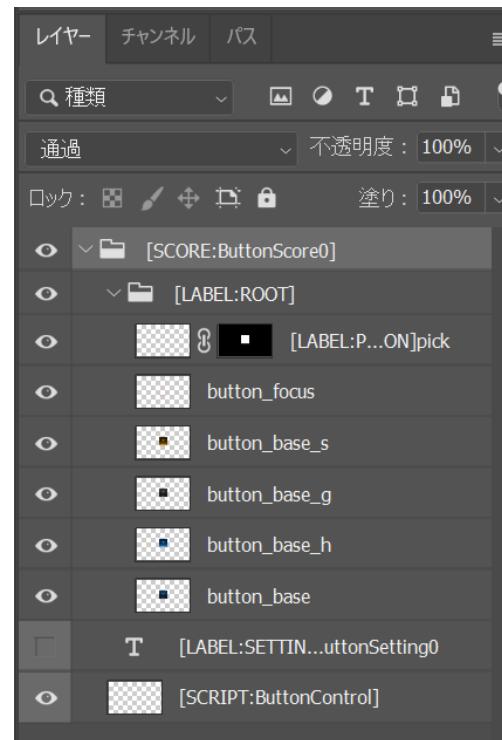
# Step3 PSDファイルのインポート

---

「PSD Importer」を利用してプロジェクトに画像を取り込む

# Step3でインポートするPSDファイルについて

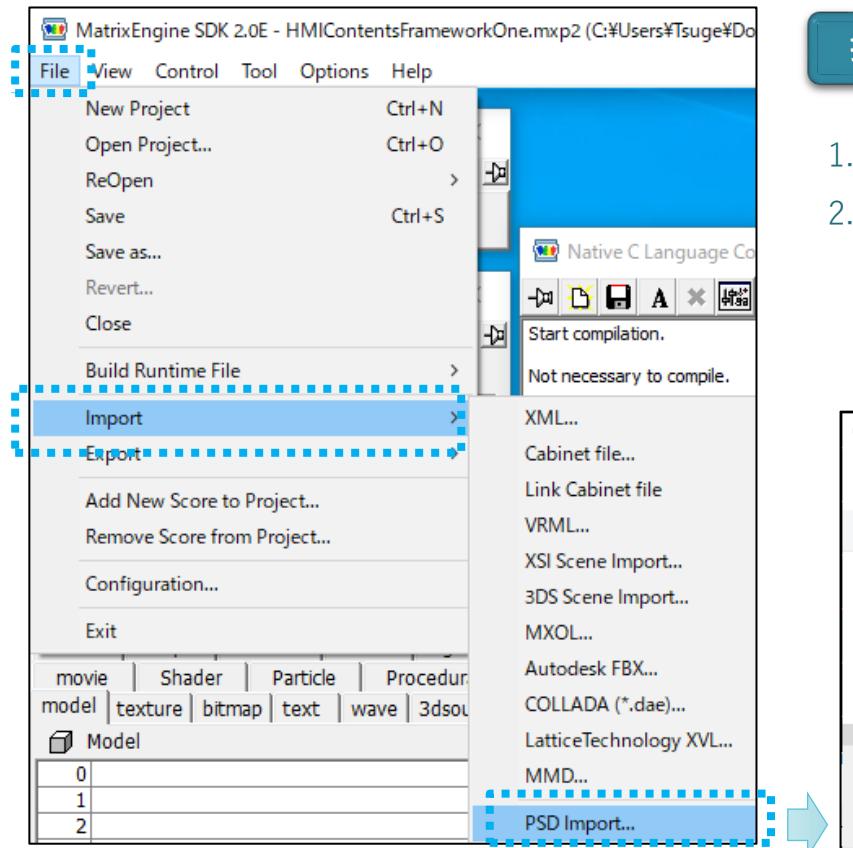
- このステップでインポートするPSDファイルは、MxE SDKへGUI Object ボタンコントロールとして取り込むために必要なレイヤー構成と設定情報、レイヤーカンプを予め保持しています。



「[MxE SDKに適したレイヤー構成でデザインするには](#)」のページも合わせてご参照ください。

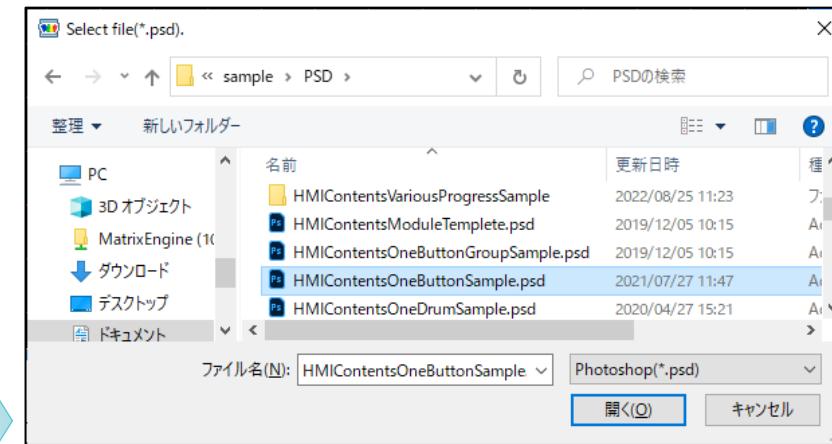
# 3-1. PSDファイルのインポート

- Step2で作成したプロジェクトに画像を取り込みます。PSDファイルのインポートには、"File"メニューの"Import"を利用します。
- インポートするPSDファイルは、「<GUI\_Objectフォルダ>\sample\PSD\HMIContentsOneButtonSample.psd」です。



## 操作

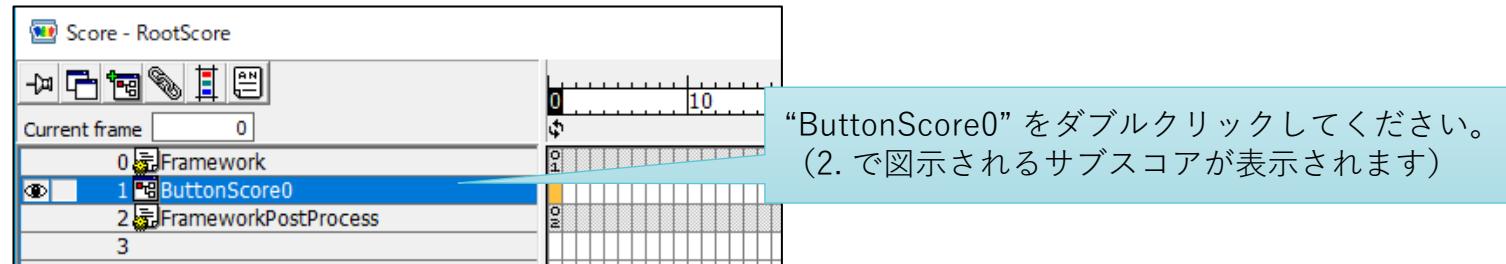
1. MxE SDK上で、File > Import > PSD Import...とメニューを選択してください。
2. ファイル選択ダイアログでは、次のファイルを選択します。  
ファイル：<GUI\_Objectフォルダ>\sample\PSD\HMIContentsOneButtonSample.psd  
※表示される質問ダイアログには、全てYesと回答してください。



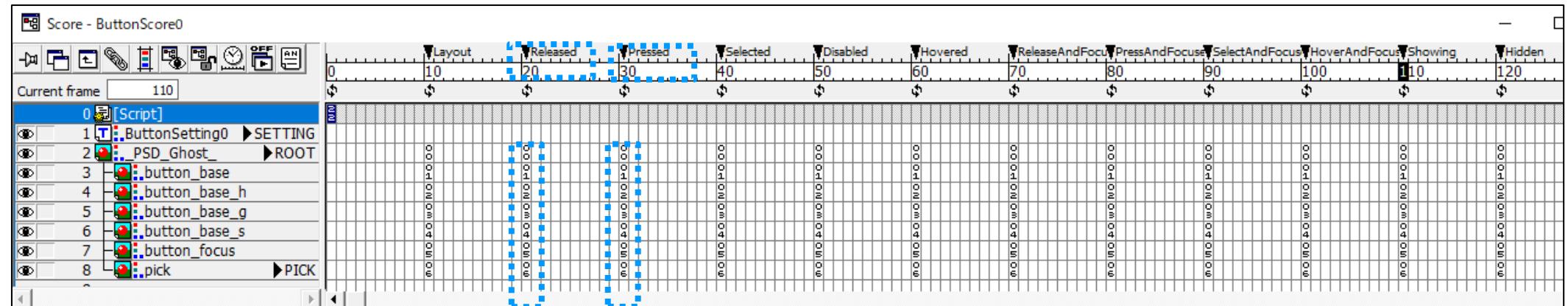
## 3-2. インポート後の確認 (1/2)

### 操作

- インポート後、RootScoreにButtonScore0というトラックが表示されます。トラック名をダブルクリックしてください。



- 下図のサブスコアが表示されます。PSDファイルからGUIObjectの仕様にそって、各状態（Released、Pressedなど）の画像が取り込まれたことが確認できます。

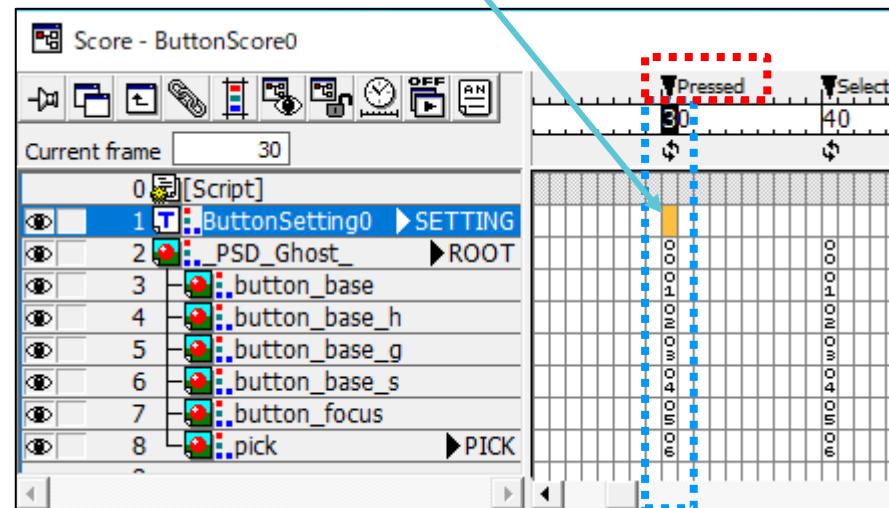


# 3-2. インポート後の確認 (2/2)

## 操作

3. サブスコアの状態の名前として**タグが付けられた列** (n0番フレーム) をクリックしてください。その状態の画像を「ワークビュー」で確認できます。

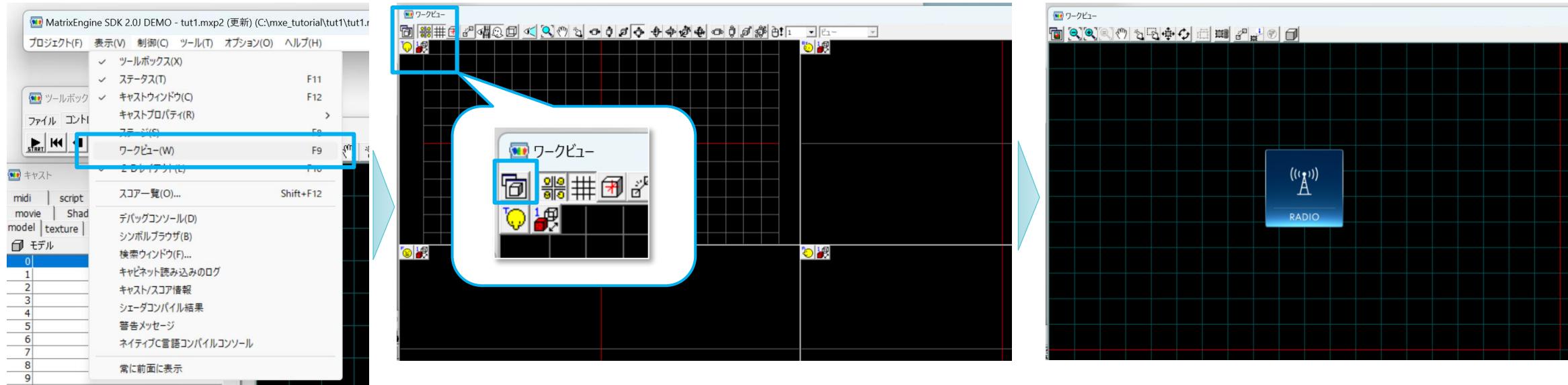
例えば：  
“Pressed”列の“ButtonSetting0”行（黄色マス）をクリック



# ワークビューの表示とレイアウトモードの切替

**操作**

1. ワークビューが表示されない場合は、メニューバーから”表示”-”ワークビュー”を選択してください。
2. 表示されたワークビュー上部の”2D/3D切り替え”でレイアウトモードを2Dに切り替えます。

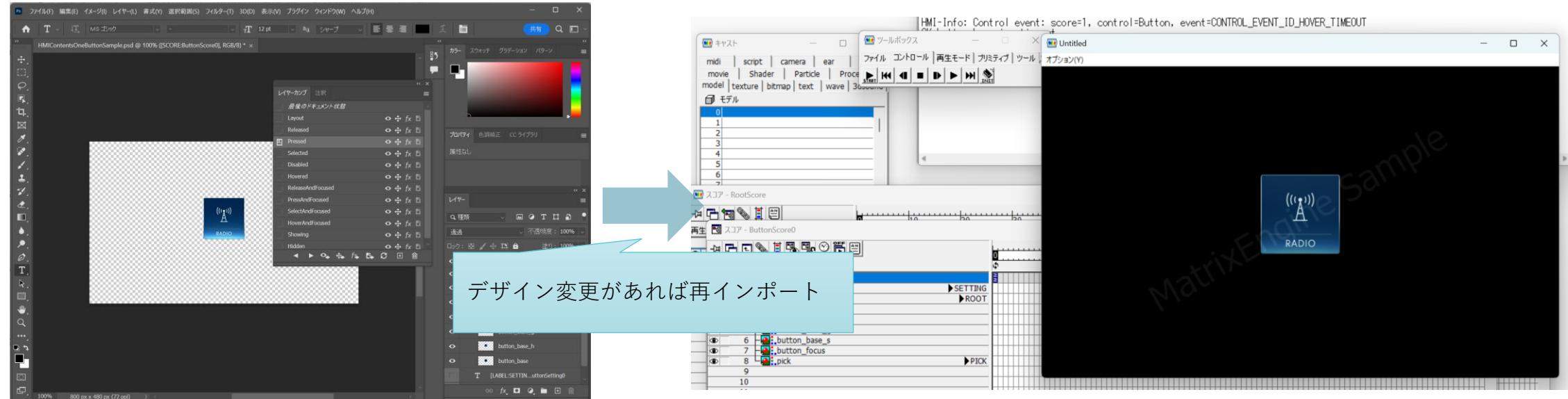


- “ワークビュー”では、「スコア」に登録された「キャスト」の再生時のレイアウトを視覚的に表示し確認できます。

# MxE SDKに適したレイヤー構成でデザインするには

- このステップでご紹介したように、MxE SDKではPhotoshopデザインデータを「[GUIObject](#)」として取り込むことができます。
- Photoshop用プラグイン「GUIObject Constructor」を使うと、MxEへの取り込みに適した構成のPSDファイルを簡単に作成できます。
- 最初からMxEに適したレイヤー構成でデザイン制作を行うことで、MxE SDKでの開発効率は飛躍的に向上します。（デザイン修正時のアプリへの反映も容易になります）
- 「GUIObject Constructor」の利用手順とPSDファイルの作成手順については、下記資料をご参照ください。本チュートリアルで取り扱っているButtonコントロールを例に詳細にご説明しています。
  - 「MxE SDK向けPSDファイル作成ガイド（MxE チュートリアル #1 シングルボタン）」
  - （ガイドに従って付属と同等のPSDファイルが作成できます）
- このステップでインポートしたPSDファイルについても、上記ドキュメントを参照しながらPhotoshopでファイルの内容を確認いただけするとより理解が深まります。

# デザインとエンジニアリングの分界点 = MatrixEngine



- デザイナーによるデザイン制作とエンジニアによる動作実装、MxE SDKは両者の分界点であり協働ツールです。
- Photoshopで制作したデータをそのままSDKにオブジェクトとして取り込むため、デザイナー-エンジニア間の認識齟齬も、情報の劣化も起こりません。
- デザインから実装へ、主要プレイヤーが切り替わる工程間のスムーズな情報連携をMxEは可能にします。

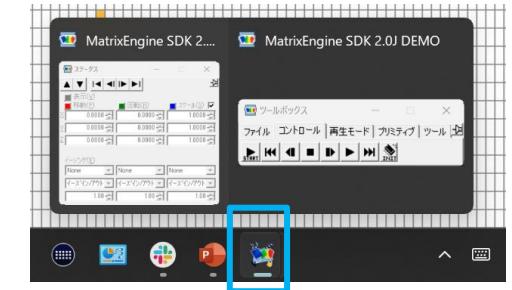
# Step4 スクリプトの実装

---

コンテンツを動作させる表示処理スクリプトを実装する

# このステップの内容と使用するファイル (1/2)

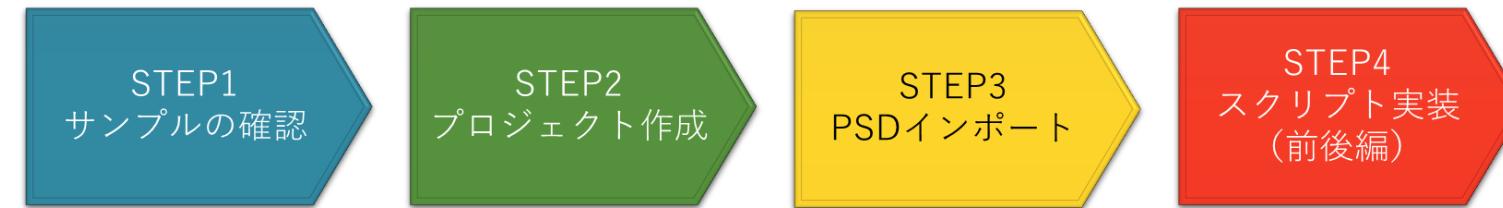
- このステップでは、[Step1で確認したサンプル](#) 「HMIContentsOneButtonSample.mxp2」 の「MainFlow」 (※) スクリプトの内容を解説していきます。
- 解説を参考に、Step2／Step3で新規作成・編集（画像取り込み）したプロジェクトファイル「tut1.mxp2」へ実際に実装してみてください。
- このステップでは、次のファイルを交互に取り扱います。
  - サンプルファイル : <GUI\_Objectファイル>¥sample¥ HMIContentsOneButtonSample.mxp2
  - 実装プロジェクト : <プロジェクトフォルダ>¥tut1.mxp2
- SDKで両ファイルを開いていることを事前に確認してください。
  - Windowsタスクバーをクリックするとファイルを切り替えられます。
  - SDKのメニューバーにも編集中のファイル名が表示されますので、確認しながら作業をして下さい。



(※) MainFlowスクリプトについては[こちら](#)をご参照ください。

# このステップの内容と使用するファイル (2/2)

- Step4は前後編に分かれています。
- 前編では、[Step1のサンプルで確認](#)した通りにボタン表示を変更するところまでを解説します。
- 後編では、クリック等のイベント種類に応じた動作をプログラムコードで実装するための手順をより詳細に解説します。（よりエンジニア向けの内容となります）



<STEP4 前編>  
ボタンの有効化と  
状態別の表示変更

<STEP4 後編>  
イベント受信と  
対応動作のコーディング

# 表示処理スクリプトの実装箇所について

- GUI Object のテンプレートでは、コンテンツを動作させるために実装が必要な箇所を「MainFlow」スクリプトに集約しています。
- MainFlowスクリプトには、制作者による実装箇所として、3つの関数が用意されています。
- 3つの関数はそれぞれ実行タイミングが異なり、必要に応じた動作を実装できます。

関数名	実行タイミング
MainFlowInitialize	コンテンツの初期化時に一度だけ呼び出されます。
MainFlowUpdate	各コントロールモジュール更新前に、毎フレーム呼び出されます。 基本的にはここにイベント処理を実装します。
MainFlowPostProcess	各コントロールモジュール更新後、描画の前に毎フレーム呼び出されます。 主に、描画に関連する処理を実装します。

テンプレートのMainFlowにはAndroid向けに上記に加えて「GUIObjectInitialize」「GUIObjectFinalize」関数が定義されています。  
本チュートリアルでは実装不要のため無視してください。

# Step4 スクリプトの実装（前編）

---

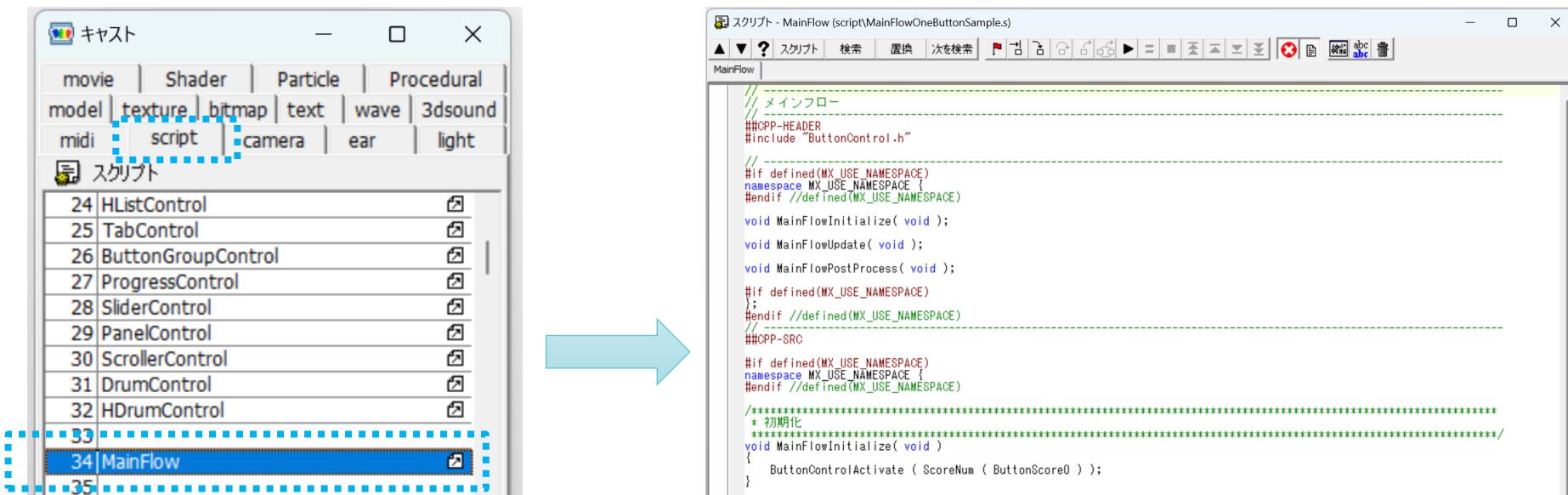
ボタンの有効化と状態別の表示変更

# 4-1. 対象のスクリプトを開く

- スクリプトの内容を確認するには、“Cast”ウィンドウの”Script”タブで、該当スクリプト名をダブルクリックします。
- MainFlowスクリプトの内容を確認するには、“MainFlow”をダブルクリックしてください。

## 操作

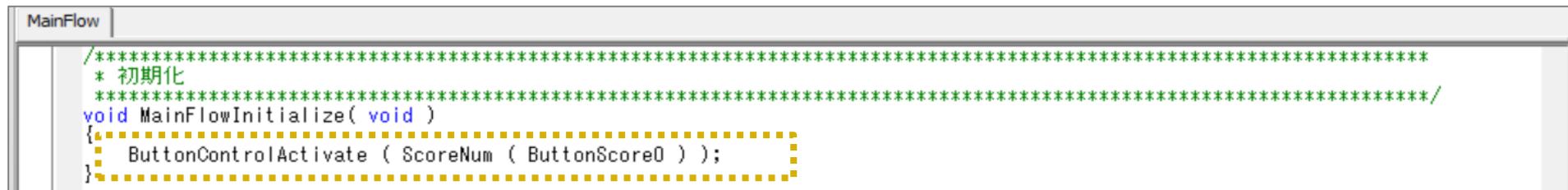
1. サンプルファイルの”Cast”ウィンドウで”Script”タブを選択します。
2. “MainFlow”をダブルクリックしてください。“MainFlow”スクリプトの内容が表示されます。



## 4-2. 初期化処理の実装

- アプリ起動時の初期化処理は、MainFlowスクリプトの「MainFlowInitialize」関数で実装します。
- サンプルファイルの「MainFlow」スクリプト内「MainFlowInitialize」関数の実装内容を理解し、実装プロジェクトに記述しましょう。

### 解説



```
MainFlow
=====
/*
 * 初期化
 */
void MainFlowInitialize( void )
{
    ButtonControlActivate ( ScoreNum ( ButtonScore0 ) );
}
```

- サンプルファイルの初期化処理「MainFlowInitialize」関数では、Step3でPSDファイルからインポートしたスコア「ButtonScore0」に対して、コントロールの有効化を行っています。
  - ✓ **ButtonControlActivate関数**：指定したButtonコントロールの有効化します（※詳細はリファレンスマニュアルをご参照ください）
  - ✓ **ScoreNumマクロ**：スコアの名前から番号に変換するためのマクロ。

### 操作

1. 実装プロジェクトのMainFlowスクリプトを開き、サンプルファイルと同様の内容を「MainFlowInitialize」関数に記述してください。

## 4-3. 初期化処理の動作確認

- 初期化処理を記述した実装プロジェクト (tut1.mxp2) の動作を確認しましょう。
- MxE SDK上でプロジェクトを動作させるには、"Tool Box"の"START"ボタンを利用します。

### 操作

1. 実装プロジェクトの"Tool Box"ウィンドウで"Control"タブを開きます。
2. "START"ボタンを押下してください。右のような画面が表示されます。
3. 表示された画面で、ボタンをクリックしてみてください。

- ✓ ボタン有効化以外に特別なコードを追加しなくても、[定義されたスコア設定](#)に従ってクリック時のボタン表示が変わることを確認してください。
- ✓ 確認後は、"Option"メニューから終了、もしくはウィンドウ右上の「×」ボタンを押下してアプリのウィンドウを閉じてください。



# 4-4. 状態更新処理のアウトライン

- 状態更新処理は、MainFlowスクリプトの「MainFlowUpdate」関数で実装します。
- サンプルファイルの「MainFlow」スクリプト内「MainFlowUpdate」関数の実装内容を解説します。

## 解説

- サンプルファイルの状態更新処理「MainFlowUpdate」関数では、大きく次の内容を実装しています（右図参照）。

- ✓ 必要な変数の宣言 . . . ①
- ✓ 毎フレームで実施する動作 . . . ②
- ✓ コントロールイベントの受信 . . . ③
- ✓ イベントの振り分けと対応 . . . ④

①②の各パートについて、次頁から詳しく見ていきます。

（③④については、ステップ後編で解説します。）

```
/*
 * 状態更新処理
 */
void MainFlowUpdate( void )
{
    RectangleI hoverArea;
    int eventIndex = 0;
    int eventId = 0;
    int score = -1;
    int paramStart = 0;
    int paramCount = 0;

    // マウス座標をホバリング判定領域に設定
    hoverArea.x = gSingleTouchState.plainPosition.x;
    hoverArea.y = gSingleTouchState.plainPosition.y;
    hoverArea.w = 1;
    hoverArea.h = 1;

    ButtonControlSetHoverStatus ( ScoreNum ( ButtonScore0 ), &hoverArea, true );

    // コントロールイベントを受信
    while ( ControlCatchEvents ( &eventIndex, &eventId, &score, &paramStart, &paramCount ) )
    {
        switch ( score )
        {
            // スコア ButtonScore0 に対するイベントを処理
            case ScoreNum ( ButtonScore0 ):
                switch ( eventId )
                {
                    // ボタン押下イベントを処理
                    case CONTROL_EVENT_ID_TAPPED:
                        // ボタンが押下されたらログを出力
                        #if LOG_LEVEL >= NORMAL
                            char str[128];
                            sprintf(str, "OK button tapped."); LOG(str);
                        #endif
                        break;

                    // ホバリング時間タイムアウトイベントを処理
                    case CONTROL_EVENT_ID_HOVER_TIMEOUT:
                        // ボタン上でのホバリング状態が一定時間を越えたらログを出力
                        #if LOG_LEVEL >= NORMAL
                            char str[128];
                            sprintf(str, "OK button hovering timeout."); LOG(str);
                        #endif
                        break;
                }
            }
        }
    }
}
```



## 4-5. 状態更新処理 ①変数の宣言

### 解説

```
/*
 * 状態更新処理
 ****
void MainFlowUpdate( void )
{
    RectangleI hoverArea;
    int eventIndex = 0;
    int eventId = 0;
    int score = -1;
    int paramStart = 0;
    int paramCount = 0;
```

①

- サンプルファイルの状態更新処理「MainFlowUpdate」関数では、毎フレームの処理に必要な変数を最初に宣言しています。
- 動作確認した通り、今回のチュートリアルでは次の場合に表示色が変わるボタンを実現します。
  - ✓ ボタンをクリックしたとき
  - ✓ マウスカーソルがボタン上にあるとき
- ここでは、上記の表示処理に必要な変数を宣言し、初期化しています。

### 操作

1. 実装プロジェクトのMainFlowスクリプトを開き、サンプルファイルの①の内容を「MainFlowUpdate」関数に記述してください。

## 4-6. 状態更新処理 ②毎フレームの動作

### 解説

```
// マウス座標をホーリング判定領域に設定  
hoverArea.x = gSingleTouchState.plainPosition.x;  
hoverArea.y = gSingleTouchState.plainPosition.y;  
hoverArea.w = 1;  
hoverArea.h = 1;  
  
ButtonControlSetHoverStatus ( ScoreNum ( ButtonScore0 ), &hoverArea, true );
```

②

- 続いて、毎フレームの動作（②）を記述します。
- gSingleTouchStateという大域変数から、アプリ画面上のマウス座標を取得、保持します。
- ButtonControlSetHoverStatus関数は、指定のボタンに、指定の座標でホーリング判定（マウスカーソルのように非タッチ状態ながら座標が重なった状態）を行うよう指示しています。

### 操作

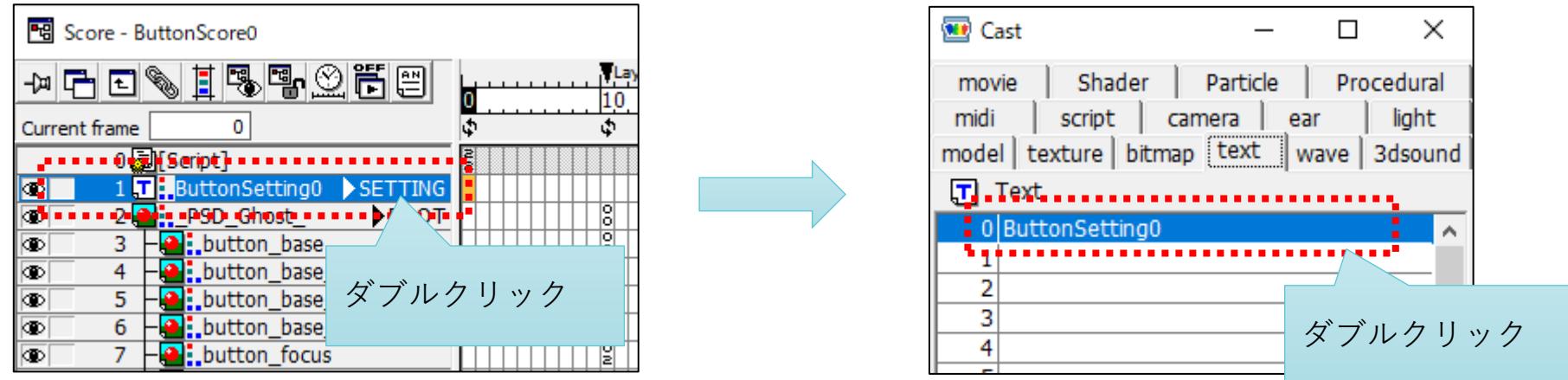
1. 実装プロジェクトのMainFlowスクリプトを開き、サンプルファイルの②の内容を「MainFlowUpdate」関数に記述してください。

# 4-7. ホバリング判定の有効化 (1/2)

- 「状態更新処理②毎フレームの動作」でスクリプトに記述したホバリング（マウスオーバー）判定指示については、Buttonコントロール側でホバリング判定を可能に設定しておく必要があります。

## 操作

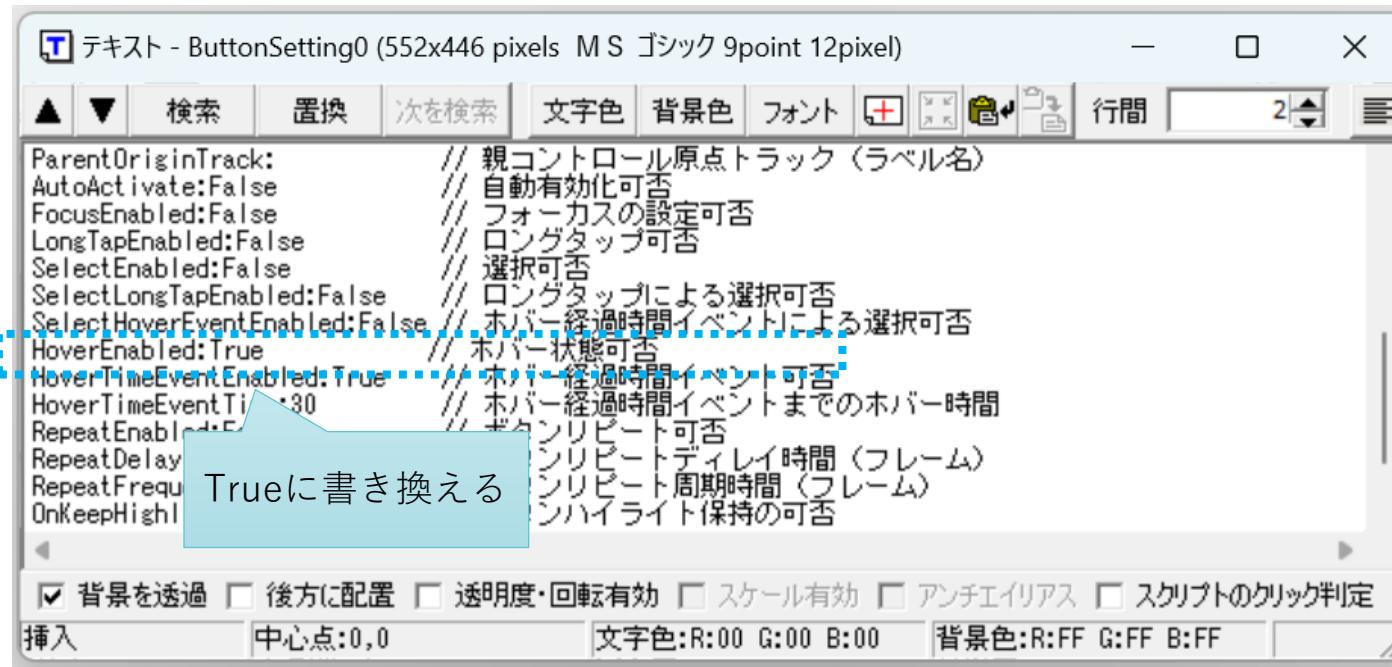
- 実装プロジェクトで、サブスコアButtonScore0のトラック1、ButtonSetting0をダブルクリックしてください。
- "cast"ウィンドウの"text"タブの中の同名のキャストがフォーカスされます。
- フォーカスされた"text"タブの"ButtonSetting0"をダブルクリックしてください。



## 4-7. ホバリング判定の有効化 (2/2)

### 操作

4. エディタが開きます。これは機能名と設定値が組となったコントロール設定用テキストキャストで、"HoverEnabled"を「True」に書き換えることでホバリング判定が可能になります。



✓ 以上でボタンの表示変更に関する実装は完了です。

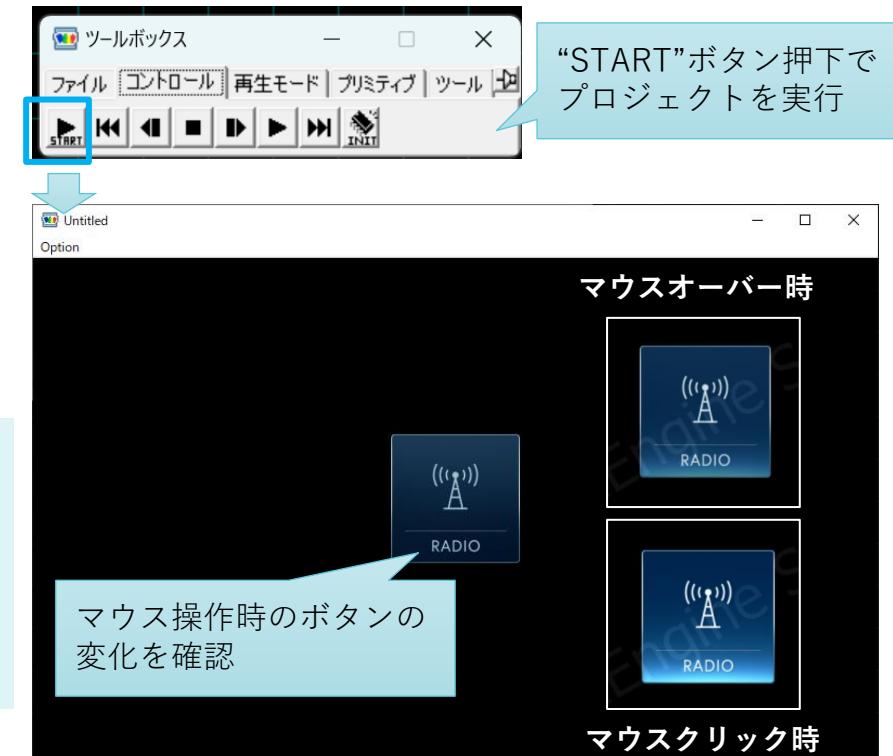
# 4-8. 状態更新処理の動作確認

- ボタンの表示変更の実装が完了しました。実装プロジェクト (tut1.mxp2) の動作を確認します。
- MxE SDK上でプロジェクトを動作させるには、"Tool Box"の"START"ボタンを利用します。

## 操作

1. 実装プロジェクトの"Tool Box"ウィンドウで"Control"タブを開きます。
2. "START"ボタンを押下してください。右のような画面が表示されます。
3. 表示された画面で、ボタンをクリックしてみてください。

- ✓ 初期化処理でのクリック時に加えて、マウスオーバー（ホバリング）時のボタン表示が変わることを確認してください。
- ✓ 確認後は、"Option"メニューから終了、もしくはウィンドウ右上の「×



# コントロールの設定項目について

- 設定項目は、コントロールによって異なります。
- 各コントロールの内容は、次のフォルダのドキュメントをご参照ください。
  - <GUIObject\_One フォルダ>/document/JP/GUIObject Control Manual
- 上記フォルダに、コントロールごとのマニュアルが保存されています。
  - 例えば：ボタンコントロールであれば「ボタンコントロールマニュアル.docx」を参照。
- マニュアルの「動作設定」に利用可能な設定項目を一覧しています。

✓ 「リファレンスマニュアル(One版)」フォルダにはさらに詳細なマニュアルをご用意しています。  
合わせてご参考ください。

1 更新履歴
2 目次
▲ 3 ボタンコントロールモジュールについて
▲ 3.1 機能概要
3.1.1 ホバリング状態対応
3.1.2 ハイライト保持機能
3.2 状態
3.3 フォーカス
3.4 動作設定
▲ 4 スクリプトによる制御
4.1 基本的な使い方
4.2 コントロールイベント
▲ 5 スコアの構造
5.1 概要
5.2 トランジションアニメーション種別
5.3 トック構造とトックラベル

# Step4 スクリプトの実装（後編）

---

イベント受信と対応動作のコーディング

## 4-9. 状態更新処理③コントロールイベントの受信

- ここからは、一歩進んでボタンコントロールから発せられたイベントを受信し、種類に応じて異なる処理を行う（本サンプルではログを出力します）パートです。
- サンプルファイルの「MainFlow」スクリプト内「MainFlowUpdate」関数の続き（③④）に沿って解説します。

解説

```
// コントロールイベントを受信
while (ControlCatchEvents ( &eventIndex, &eventId, &score, &paramStart, &paramCount ))
{
    switch (score)
    {
        // スコア ButtonScore0 に対するイベントを処理
        case ScoreNum ( ButtonScore0 ):
            switch (eventId)
            {
                // ボタン押下イベントを処理
                case CONTROL_EVENT_ID_TAPPED:
                    // ボタンが押されたときの処理
                }
            }
        }
    }
```

どのスコアからイベントが来たのか

どんな種類のイベントが来たのか

- while文の条件式で**ControlCatchEvents**関数を実行することで、1フレーム中に発生してスタックされているイベントを全て処理するまで、繰返し部のコードを実行します。
- **ControlCatchEvents**関数は、ポインタ引数で受信したイベントの情報を取得します。「**&score**」でイベントを通知したスコアの情報を（サンプルのボタンは”**ButtonScore0**”）、「**&eventId**」でイベント種類を取得できます。

# コントロールが通知するイベントについて

- 通知するイベントは、コントロールによって異なります。
    - 例えば、Buttonコントロールは次のようなコントロールイベントを通知します。
      - クリック時 : CONTROL\_EVENT\_ID\_TAPPED
      - ホバリングタイムアウト : CONTROL\_EVENT\_ID\_HOVER\_TIMEOUT
      - クリック長押し : CONTROL\_EVENT\_ID\_LONG\_TAPPED
  - 各コントロールの内容は次のフォルダのドキュメントをご参照ください。
    - <GUIObject\_Oneフォルダ>/document/JP/GUIObject Control Manual
  - 上記フォルダに、コントロールごとのマニュアルが保存されています。
    - 例えば：ボタンコントロールであれば「ボタンコントロールマニュアル.docx」を参照。
  - マニュアルの「コントロールイベント」に通知イベントを一覧しています。
- ✓ 「リファレンスマニュアル(One版)」フォルダにはさらに詳細なマニュアルをご用意しています。  
合わせてご参考ください。

1 更新履歴
2 目次
▲ 3 ボタンコントロールモジュールについて
▲ 3.1 機能概要
3.1.1 ホバリング状態対応
3.1.2 ハイライト保持機能
3.2 状態
3.3 フォーカス
3.4 動作設定
▲ 4 スクリプトによる制御
4.1 基本的な使い方
4.2 コントロールイベント
▲ 5 スコアの構造
5.1 概要
5.2 トランジションアニメーション種別
5.3 トراك構造とトラックラベル

## 4-10. 状態更新処理 ④ イベント振り分け（スコア）

解説

- ControlCatchEvents関数で取得したイベント情報を使って、どのスコアのイベントか判定し、振り分けます。
- アプリで使っているコントロールはボタン1つだけですが、複数コントロールへの拡張の参考として分岐処理を記述しています。
- case文に定数としてあてがうため、スコア名 ButtonScore0にScoreNumマクロを適用します。

```
// コントロールイベントを受信
while (ControlCatchEvents ( &eventIndex, &eventId, &score,
{
    switch (score)
    {
        // スコア ButtonScore0 に対するイベントを処理
        case ScoreNum ( ButtonScore0 ):
            switch (eventId)
            {
                // ボタン押下イベントを処理
                case CONTROL_EVENT_ID TAPPED:
                    // ボタンが押下されたらログを出力
                    #if LOG_LEVEL >= NORMAL
                    {
                        char str[128];
                        sprintf(str, "OK button tapped.");
                    }
                    #endif
                    break;

                // ホバリング時間タイムアウトイベントを処理
                case CONTROL_EVENT_ID HOVER_TIMEOUT:
                    // ボタン上でのホバリング状態が一定時間を越えた
                    #if LOG_LEVEL >= NORMAL
                    {
                        char str[128];
                        sprintf(str, "OK button hovering time out.");
                    }
                    #endif
                    break;

                // ロングタップイベントを処理
                case CONTROL_EVENT_ID LONG_TAPPED:
                    // ボタンが長時間押されたらログを出力
                    #if LOG_LEVEL >= NORMAL
                    {
                        char str[128];
                        sprintf(str, "OK button long tapped.");
                    }
                    #endif
                    break;
            }
    }
}
```

## 4-10. 状態更新処理 ④ イベント振り分け（イベントID）

### 解説

- スコア振り分け後は、Buttonコントロールから来るイベントIDで処理を振り分けます。
- サンプルでは、「タップ（クリック）」「ホバリングタイムアウト」「タップ長押し」それぞれのcaseでログを出力しています。

💡 ホバリングや長押しの振り分けには、判定を有効化しておく必要があります。  
[「ホバリング判定の有効化」](#) 「[長押し判定の有効化](#)」をご参考ください。

### 操作

1. 実装プロジェクトのMainFlowスクリプトを開き、サンプルファイル③④の内容を「MainFlowUpdate」関数に記述してください。

```

case ScoreNum ( ButtonScore0 ):
    switch (eventId)
    {
        // ボタン押下イベントを処理
        case CONTROL_EVENT_ID TAPPED:
            // ボタンが押下されたらログを出力
            #if LOG_LEVEL >= NORMAL
            {
                char str[128];
                int longTouched = 0;
                if (paramCount >= 1) {
                    longTouched = ControlEvent...
                }
                sprintf(str, "OK button tappe...
            }
            #endif
            break;

        // ホバリング時間タイムアウトイベントを処理
        case CONTROL_EVENT_ID HOVER TIMEOUT:
            // ボタン上でのホバリング状態が一定時間超えた場合
            #if LOG_LEVEL >= NORMAL
            {
                char str[128];
                sprintf(str, "OK button hover...
            }
            #endif
            break;

        // ロングタッチイベントを処理
        case CONTROL_EVENT_ID LONG TOUCHED:
            // ボタン押下中にロングタップ時間を越えたら
            #if LOG_LEVEL >= NORMAL
            {
                char str[128];
                sprintf(str, "OK button long...
            }
            #endif
            break;
    }
}

```

タップ時

ホバリング  
タイムアウト

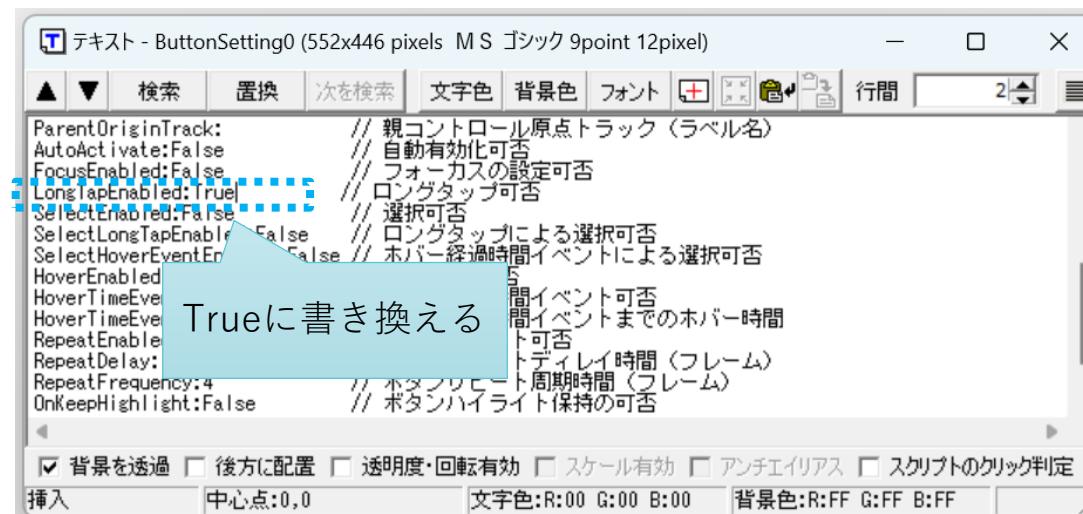
長押し

# 4-11. 長押し判定の有効化

- 「[ホバリング判定の有効化](#)」と同様に、ボタンを長押しした際にボタンコントロールから「長押しイベント」が通知されるようにするには、コントロール設定用テキストキャストで、"LongTapEnabled"を「True」に書き換えます。

## 操作

- 実装プロジェクトで、サブスコアButtonScore0のトラック1、ButtonSetting0をダブルクリックしてください。
- "cast"ウィンドウの"text"タブの中の同名のキャストがフォーカスされます。
- フォーカスされた"text"タブの"ButtonSetting0"をダブルクリックしてください。
- 表示されたエディタで"LongTapEnabled"を「True」に書き換えてください。



✓ 以上ですべての実装が完了です。

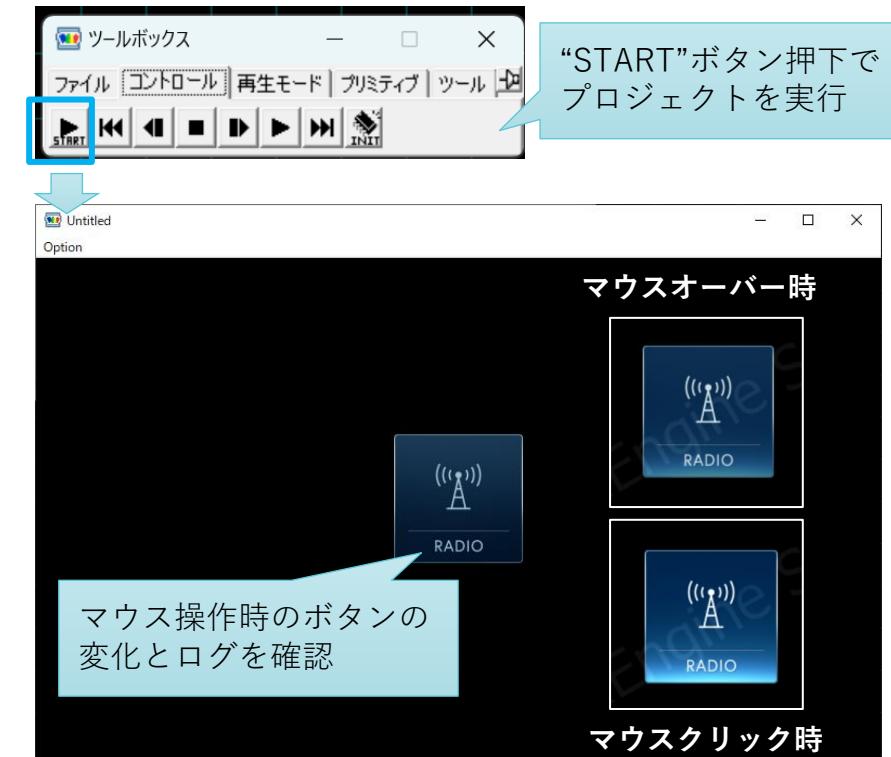
# 4-12. 最終動作確認

- 実装プロジェクト (tut1.mxp2) の動作を確認しましょう。
- MxE SDK上でプロジェクトを動作させるには、"Tool Box"の"START"ボタンを利用します。

## 操作

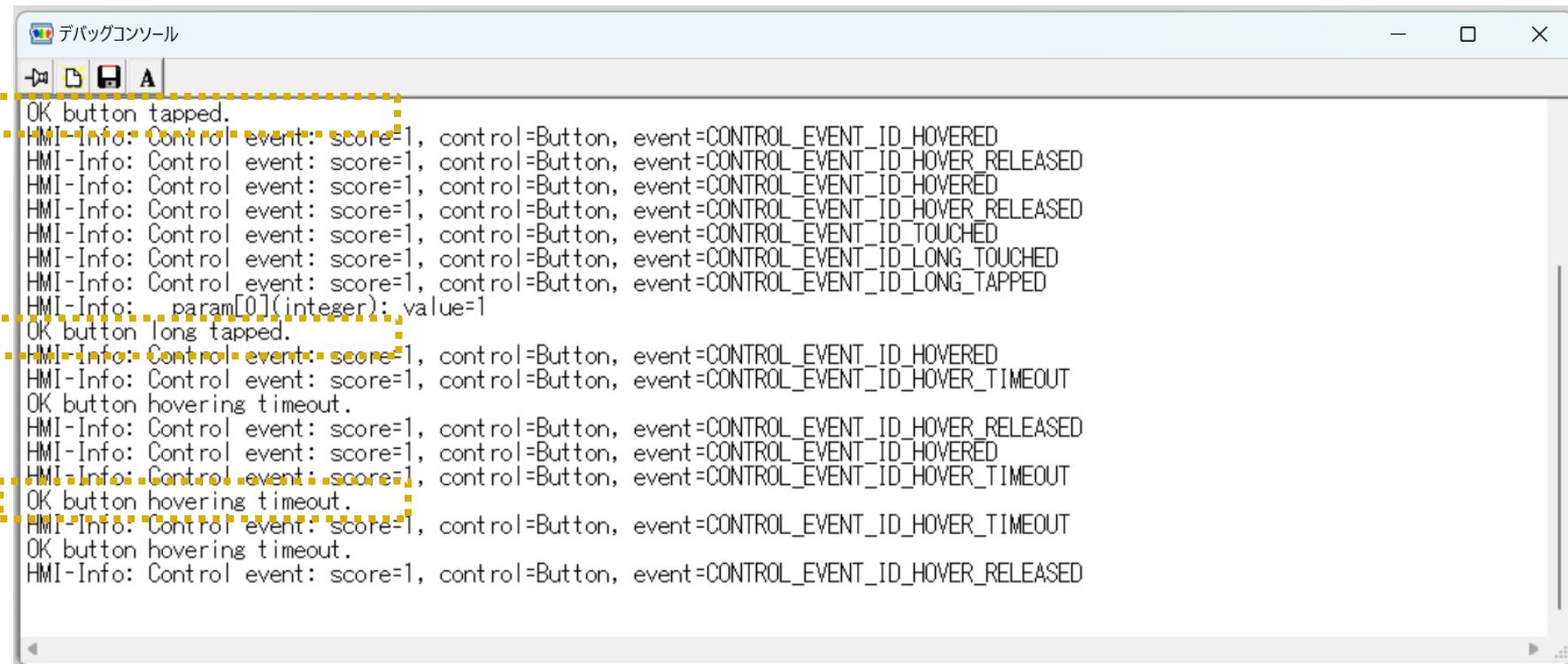
1. 実装プロジェクトの"Tool Box"ウィンドウで"Control"タブを開きます。
2. "START"ボタンを押下してください。画面が表示されます。
3. ボタンに対してマウスオーバー、クリック、長押しの操作を実行してみてください。

- ✓ マウスオーバー（ホバリング）時のボタン表示が変わることを確認してください。
- ✓ 次頁を参考に、"デバッグコンソール"にログが表示されることを確認してください。
- ✓ 確認後は、"Option"メニューから終了、もしくはウィンドウ右上の「×」ボタンを押下してアプリのウィンドウを閉じてください。



# 最終動作確認) イベント別のログ表示

- デバッグコンソールには、[イベントIDごとに実装](#)したログが表示されます。下図を参考に確認してください。



The screenshot shows the MatrixEngine Debug Console window. On the left, three yellow boxes categorize the log entries: "タップ時" (Top), "長押し" (Middle), and "ホバリングタイムアウト" (Bottom). The console displays the following log entries:

- タップ時 (Top):**
  - OK button tapped.
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVERED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_RELEASED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVERED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_RELEASED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_TOUCHED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_LONG\_TOUCHED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_LONG\_TAPPED
  - HMI-Info: param[0](integer): value=1
- 長押し (Middle):**
  - OK button long tapped.
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVERED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_TIMEOUT
  - OK button hovering timeout.
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_RELEASED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVERED
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_TIMEOUT
  - OK button hovering timeout.
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_TIMEOUT
  - OK button hovering timeout.
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_RELEASED
- ホバリングタイムアウト (Bottom):**
  - HMI-Info: Control event: score=1, control=Button, event=CONTROL\_EVENT\_ID\_HOVER\_RELEASED

- ✓ 以上ですべての作業が完了です。実装プロジェクトを保存し、MxE SDKを終了してください。

# 終わりに

---

スクリプトの実装によるアプリの発展と拡張

# 拡張・発展について

---

- 以上で本チュートリアルは終了です。
  - スクリプトの実装はC/C++言語によるプログラミングです。自由に拡張・発展が可能ですので、以下を参考に進めてください。
- 
- 1つのスクリプトにつき、##CPP-HEADERタグ以降がC/C++での.hファイルに相当し、##CPP-SRCタグ以降が.c/.cppファイルに相当します。（ビルド時に[スクリプト名].hファイルと[スクリプト名].c/.cppファイルに分割されます）
  - 実際の開発における大きい処理は、C/C++言語同様、別スクリプトに実装し処理を分割します（C/C++の関数呼出）。
  - ##CPP-HEADERタグ部分に関数宣言、マクロ定義など。##CPP-SRCタグ部分に関数定義、大域変数など。（C/C++同様）

# 参考資料一覧

- 本書でご紹介したドキュメント/ファイルを一覧します。合わせてご参照ください。

呼称	ファイル／フォルダ
MatrixEngine概要	MatrixEngine概要.pdf
インストール及び環境構築ガイド	環境構築ガイド.pdf
GUI Object サンプル	<GUIObject_One フォルダ>\sample
GUI Object 共通マニュアル	<GUIObject_One フォルダ>\document\JP\GUI Object共通マニュアル(One版).docx
GUI Object コントロールマニュアル	<GUIObject_One フォルダ>\document\JP\GUIObject Control Manual
GUI Object リファレンスマニュアル	<GUIObject_One フォルダ>\document\JP\GUIObject Control Manual\リファレンスマニュアル(One版)
MxE SDK向け PSDファイル作成ガイド (MxE チュートリアル #1 シングルボタン)	(本チュートリアルと同フォルダ) tut1_シングルボタン_PSD作成ガイド.pdf