

MxE チュートリアル #2 SDK操作ガイド
ボタングループアプリ

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）ボタングループの有効化と初期表示](#)
 - [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](#)

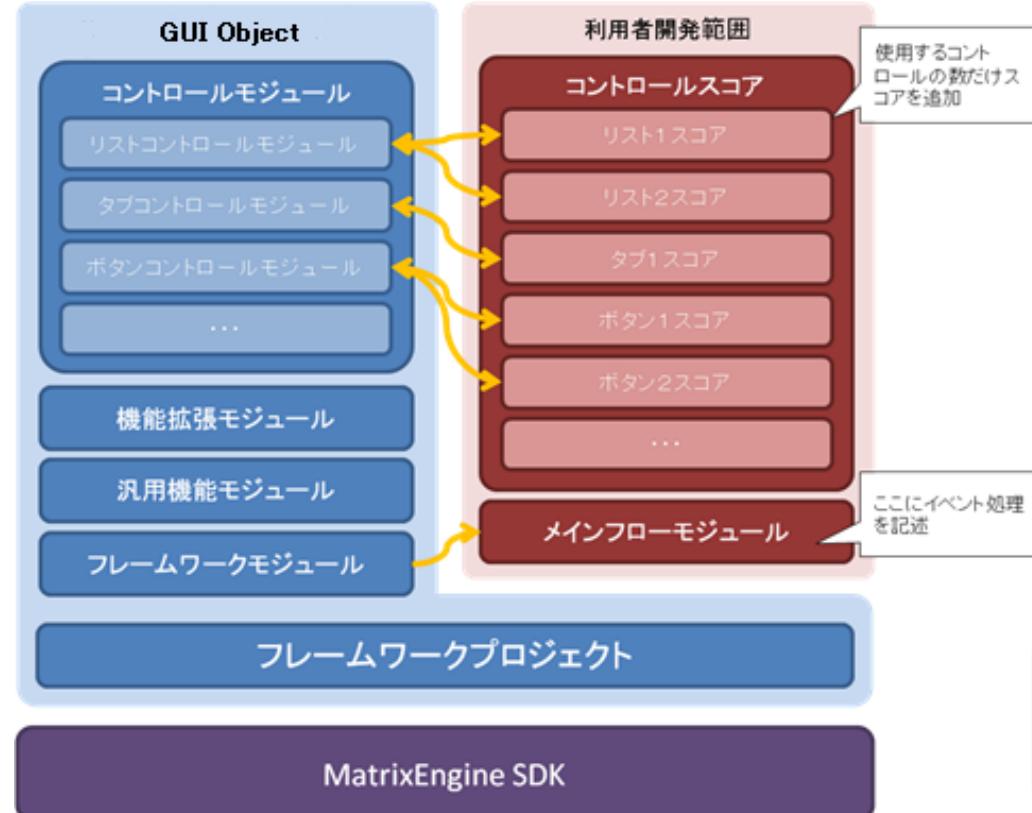
本書の目的

-
- 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。
 - 本書で扱う「ボタングループアプリ」は、複数のボタンをまとめて1つのコントロール（部品）として表示するサンプルアプリです。（ボタン一つを扱いたい場合は、別途「MxEチュートリアル#1 シングルボタンアプリ」をご参照ください。）
 - サンプルアプリを対象に、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。
 - 「チュートリアル#1 シングルボタンアプリ」を実施済みの方。
- 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_yyyyymmdd.zip) を取得済みである

※ チュートリアルシリーズについて

- 本書は、MxE SDK & GUIObject チュートリアルシリーズ#2です。
- チュートリアル#1 の内容を前提にしている箇所もありますので、順番に取り組んでいただくと理解がよりスムーズです。

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

ボタングループサンプルの動作仕様

- コンテンツを起動すると、エアコンのスイッチ風の5つのボタンが表示されます。
- 各ボタンは一つのボタングループコントロールで制御されていますが、それぞれ独立して動作し、押下するとログを出力します。
- フォーカス制御が有効になっており、キー操作でフォーカスが移動します。
- マウス等のホバリング判定可能なデバイスでは、ボタン上でホバリングする事で、ホバリング用のハイライト表示となります。

操作	割り当て動作
ボタンの押下	ボタンの押下処理（押下されたボタンのボタン番号をログ出力）
ボタン上でホバリング	一定時間ホバリングでタイムアウト処理（タイムアウトしたボタンのボタン番号をログ出力）
左右キー	フォーカスの左右移動
PageUp、PageDownキー	フォーカスの左右（前後）移動

次頁から、動作を実際に確認していきます。

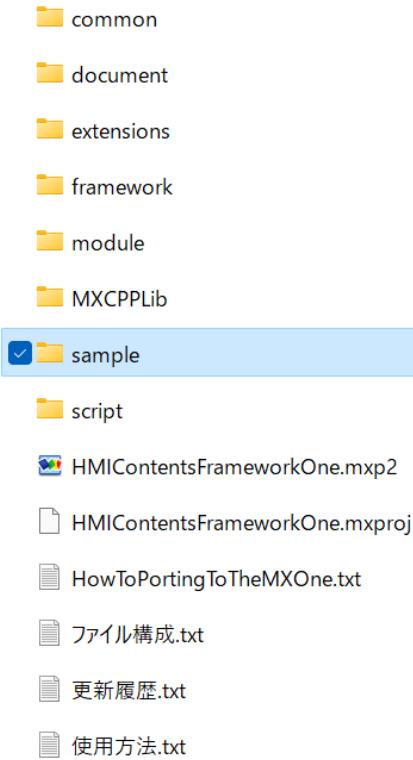
1-1. サンプルファイルの確認とSDKの起動

- 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、「**HMIContentsOneButtonGroupSample.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、ボタングループ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ボタングループアプリのプロジェクトファイル名は「**HMIContentsOneButtonGroupSample.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、ボタン上で順番にマウスを動かし、任意のボタンでクリックしてみてください。

- ✓ [動作仕様](#)を参考に操作してみてください。
- ✓ 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：ボタングループの表示色切替

- ボタングループ内のボタンの色は、マウスポインタがボタン上にあるかどうか、ボタンをクリックしたかで変化します。



- 本書では、ユーザーが操作を認識しやすいよう色を変更するサンプルを提示しています。
- ユーザーに複数ボタンから一つを選択して実行させる部品を「ボタングループコントロール」と呼びます。
- マウスポインタを動かすと、ポインタ下にあるボタンの色が変化します。 (左図)
- クリック時は、ボタンがより明るく光ります。 (下図)

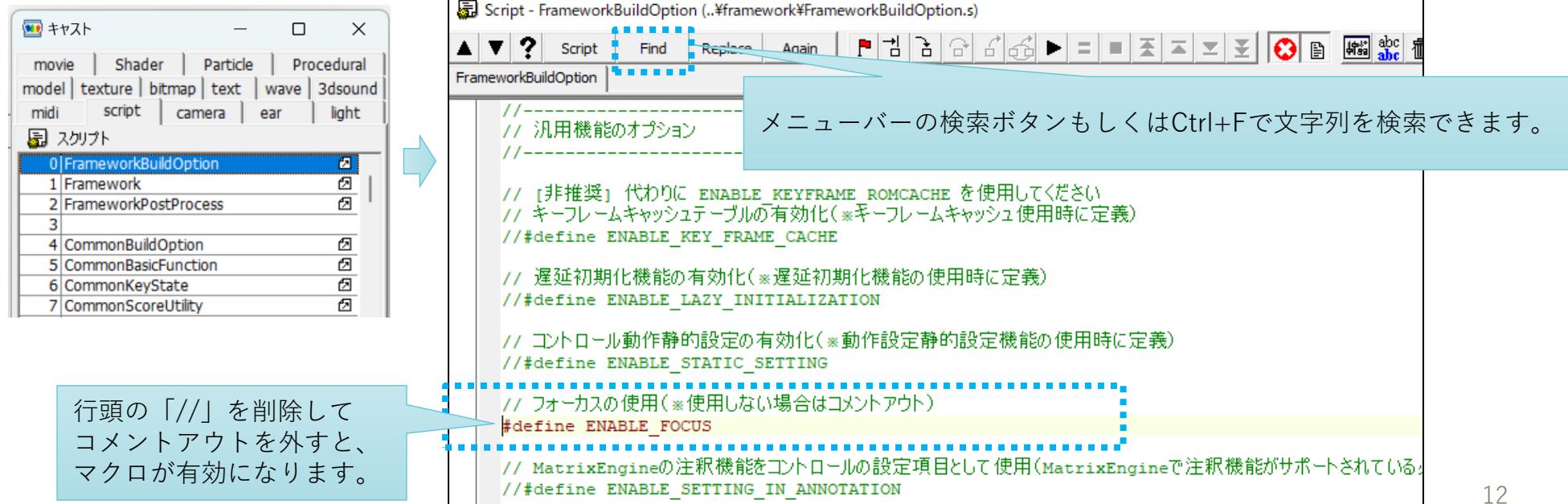


1-3. キー操作によるフォーカス移動の設定

- このサンプルでは、キー操作でフォーカスを移動する方法も紹介しています。
- 動作を確認するには、FrameworkBuildOptionスクリプトキャストを編集する必要があります。

操作

1. "cast" ウィンドウの"script" タブで「FrameworkBuildOption」をダブルクリックします。
2. ENABLE_FOCUSマクロの記述がコメントアウトされているので、これを有効化し、プロジェクトのフォーカス機能を一律で有効にします。



1-4. フォーカス移動の動作確認

- フォーカス移動に関して、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、任意のボタンをマウスクリック後、左右キーを押してみてください。

- ✓ 次頁を参考に、左右キーでフォーカスが移動す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：フォーカス移動時のボタン表示

- 初期表示後、左右キーまたはPgUp/PgDnキー押下でフォーカス（赤枠で表現）が移動します。ボタンを押下にはEnterキーを利用します。



- 本書では、ユーザーがフォーカス移動を認識しやすいよう赤枠を表示するサンプルを提示しています。
- 左右キーでフォーカスが移動します（左図）
- フォーカス状態でEnterキーを押下すると、フォーカス表示を保ったままボタン色が変化します。（下図）



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

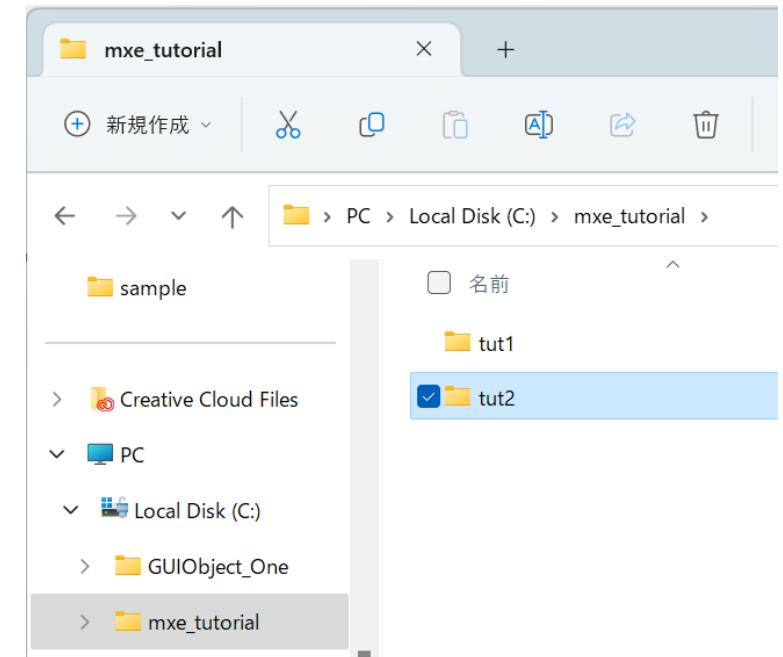
2-1. プロジェクトフォルダの新規作成

- 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:\mxe_tutorial\tut2」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。

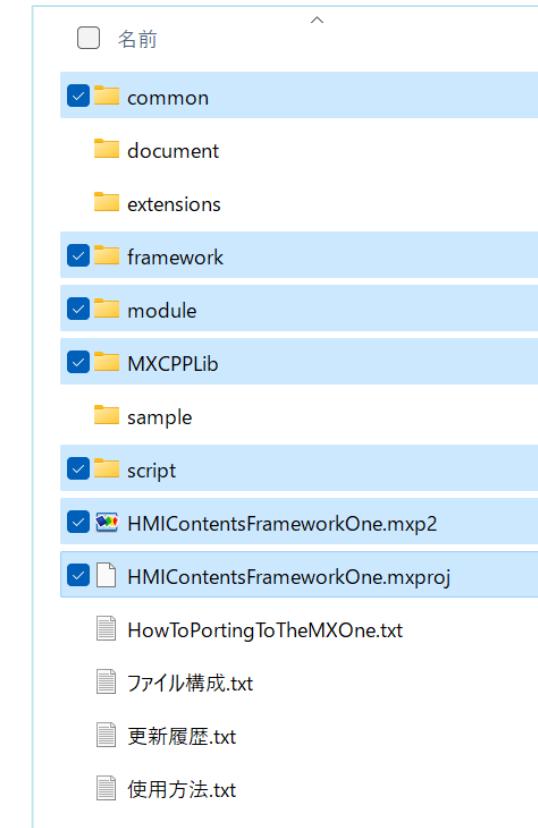


2-2. テンプレートの複製

- 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

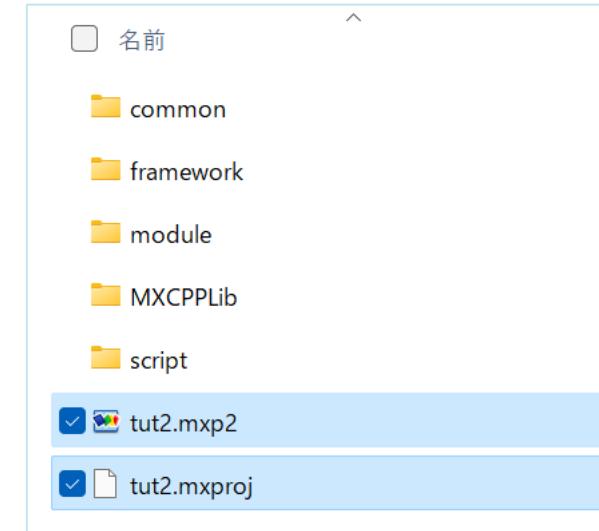
- プロジェクトファイル名をわかりやすいように変更します。「.mxp2」「.mxproj」の二つのファイルを同じ名称でリネームします。

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。

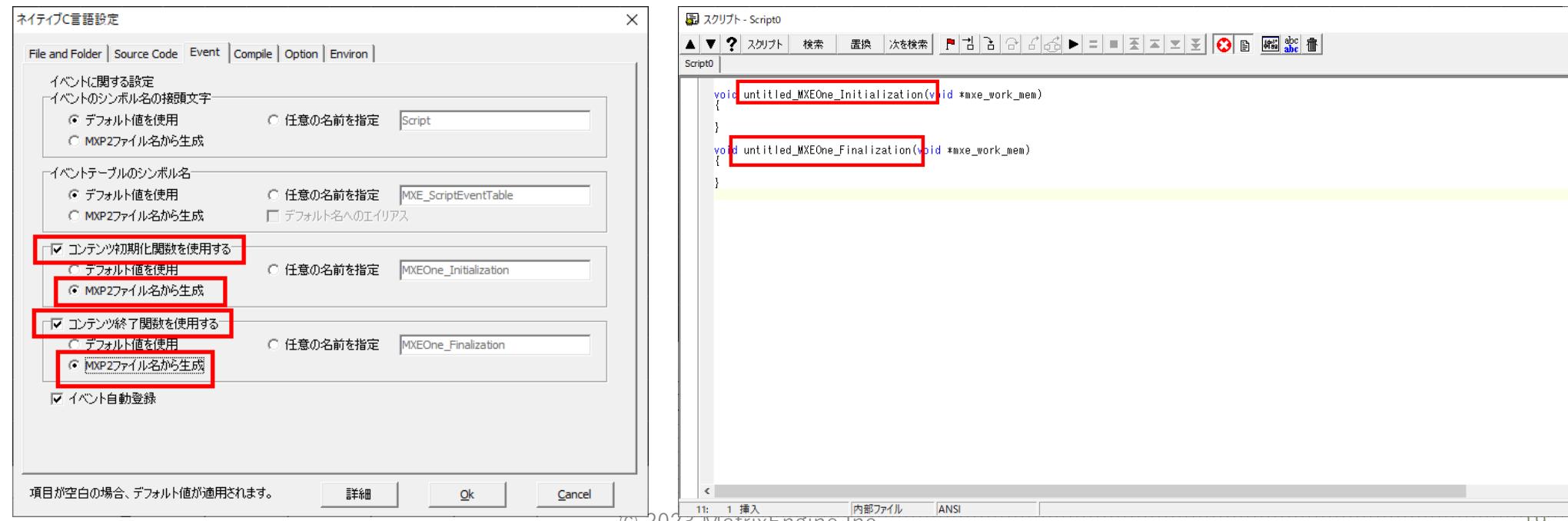
- HMIContentsFrameworkOne.mxp2 → tut2.mxp2
- HMIContentsFrameworkOne.mxproj → tut2.mxproj

- ✓ 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。



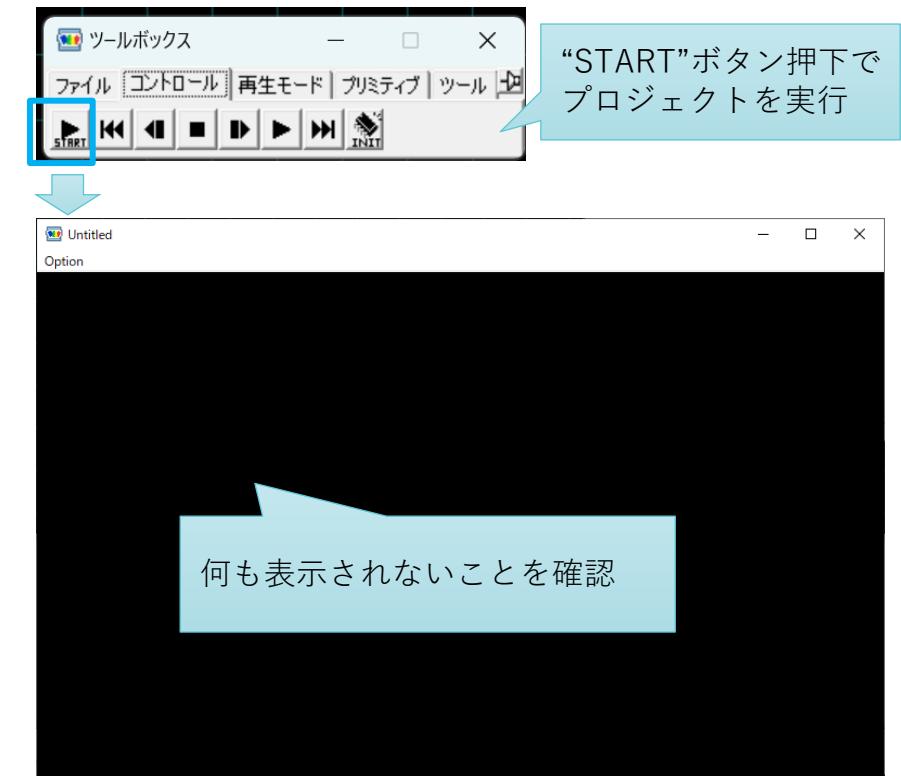
2-4. プロジェクトの動作確認

- 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

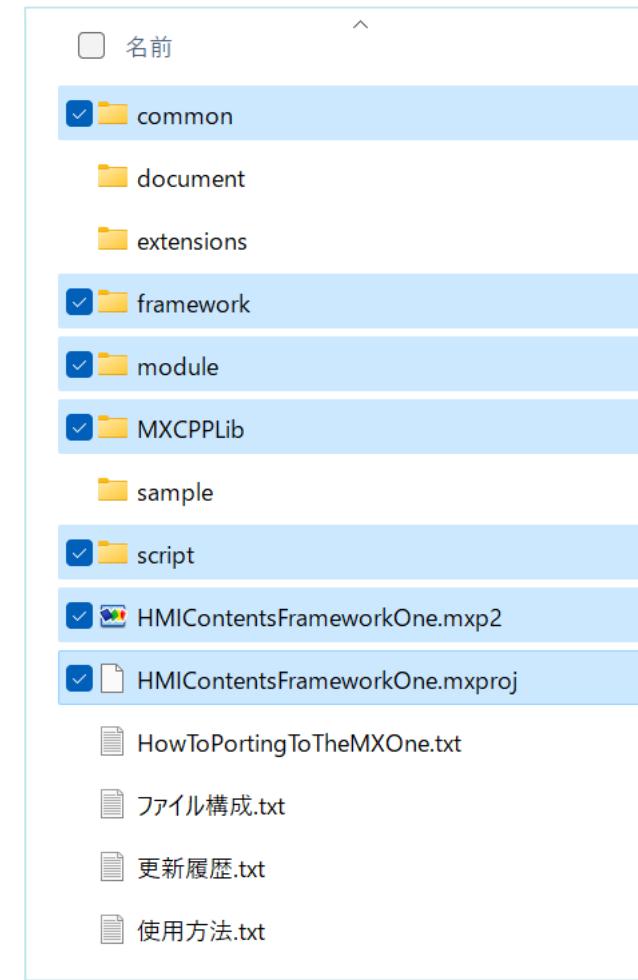
1. <プロジェクトフォルダ>の「tut2.mxp2」ファイルをダブルクリックして、MxE SDKを起動してください。
2. "Tool Box" ウィンドウの"Control"タブを開きます。
3. "START"ボタンを押下してください。
4. 何も動作しない（真っ黒な）アプリが立ち上がることを確認してください。

✓ 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。



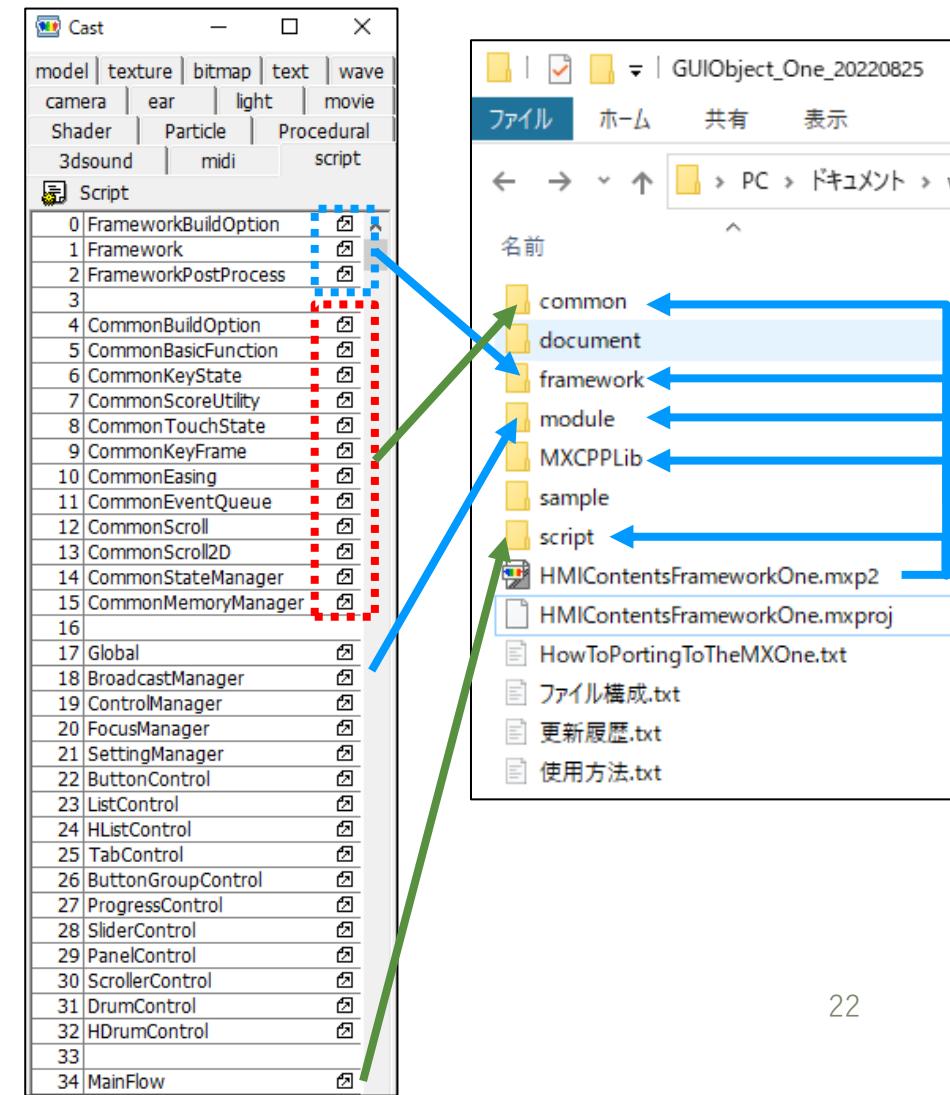
テンプレートのファイル構成

- 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。

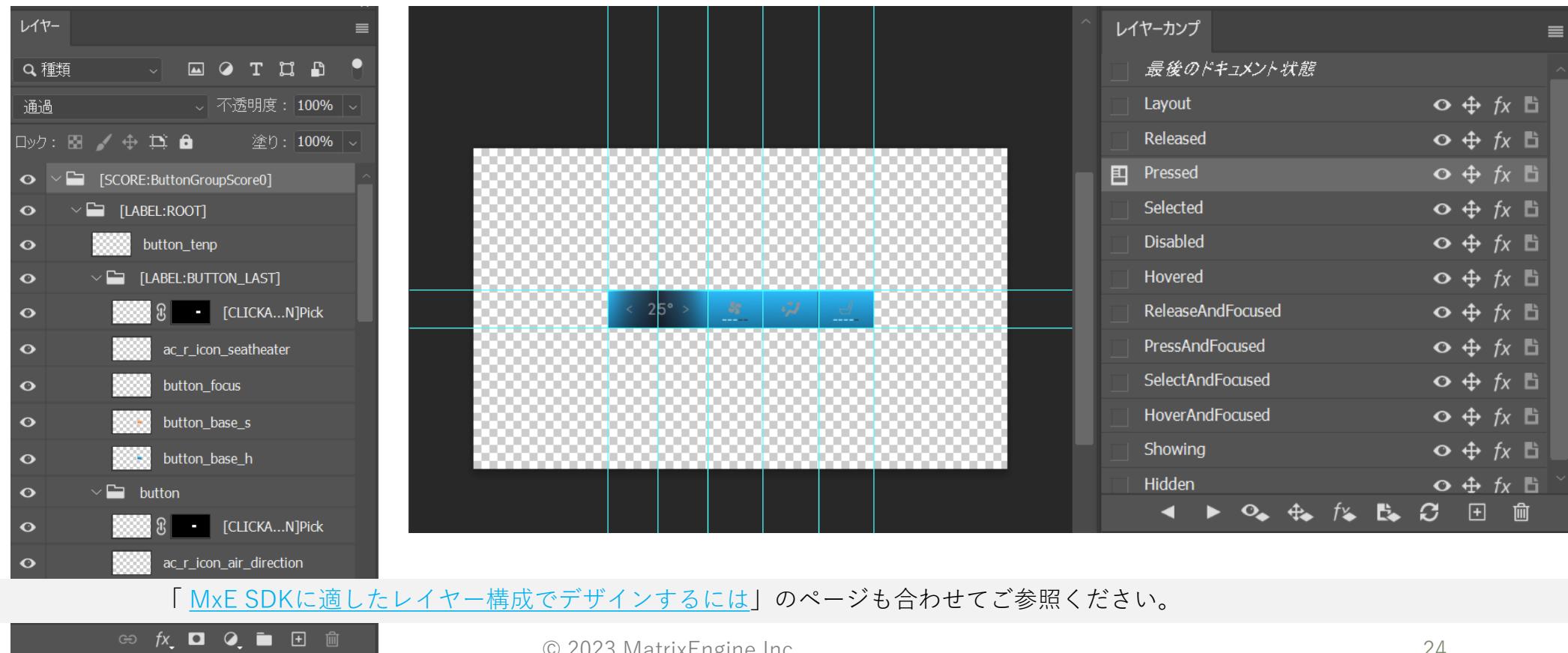


Step3 PSDファイルのインポート

「PSD Importer」を利用してプロジェクトに画像を取り込む

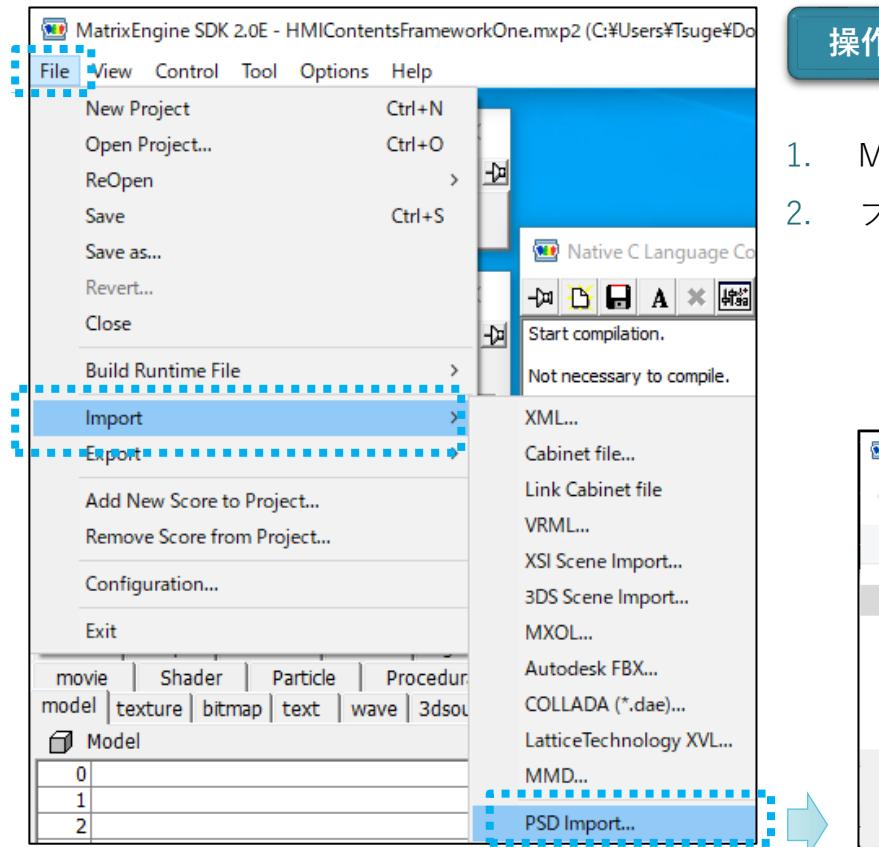
Step3でインポートするPSDファイルについて

- 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 ボタングループ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。



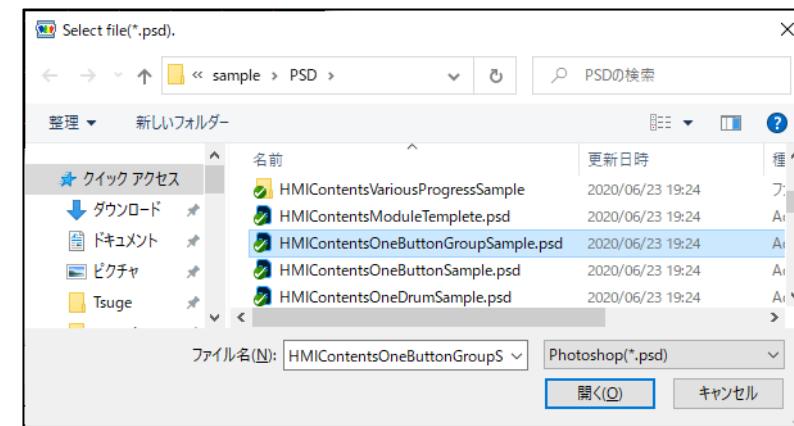
3-1. PSDファイルのインポート

- 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\HMIContentsOneButtonGroupSample.psd」です。



操作

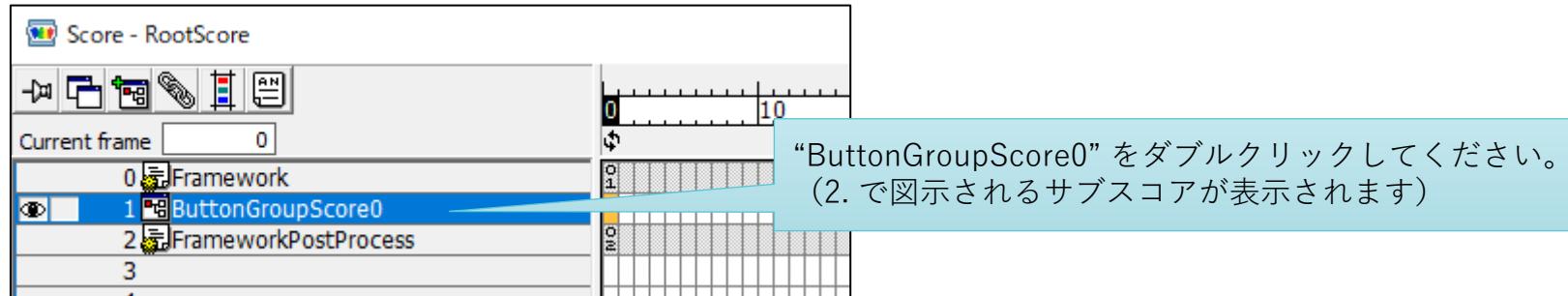
1. MxE SDK上で、File > Import > PSD Import...とメニューを選択してください。
2. ファイル選択ダイアログでは、次のファイルを選択します。
ファイル : <GUI_Objectフォルダ>\sample\PSD\HMIContentsOneButtonGroupSample.psd
※表示される質問ダイアログには、全て「Yes」または「上書き」と回答してください。



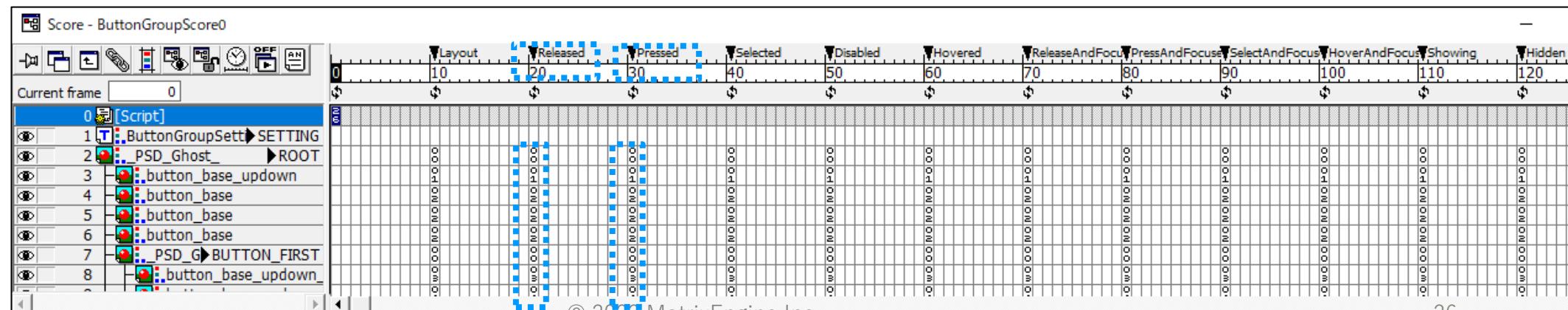
3-2. インポート後の確認 (1/2)

操作

- インポート後、RootScoreにButtonGroupScore0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。



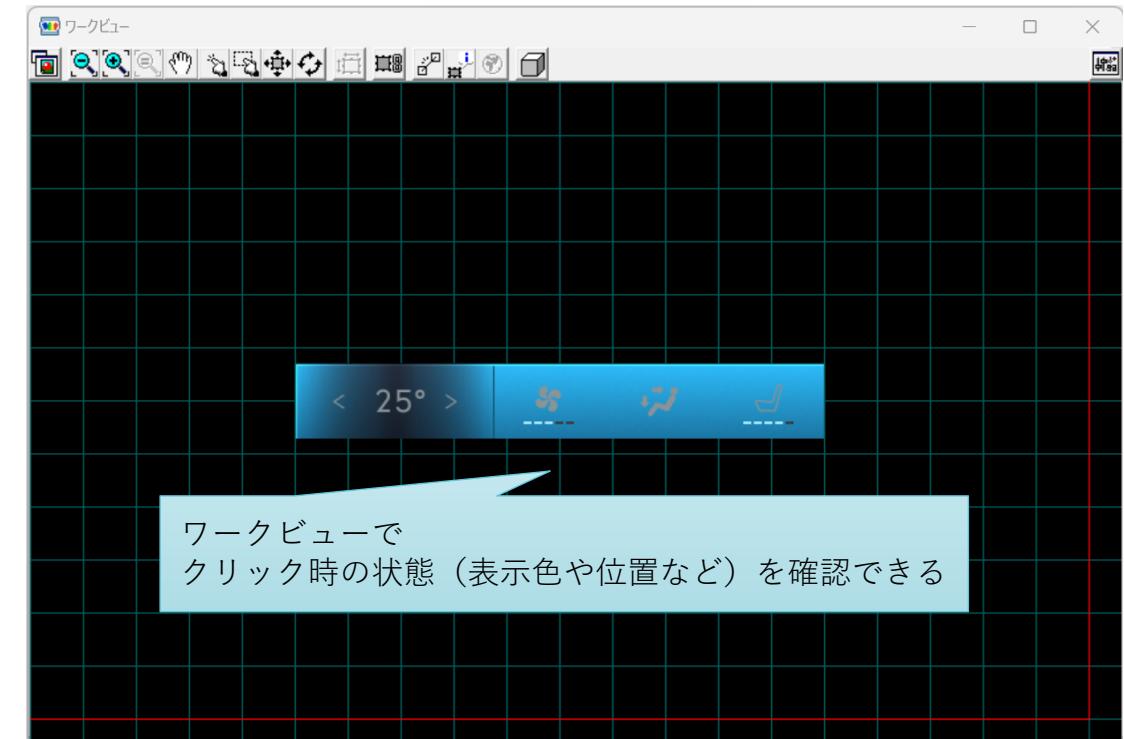
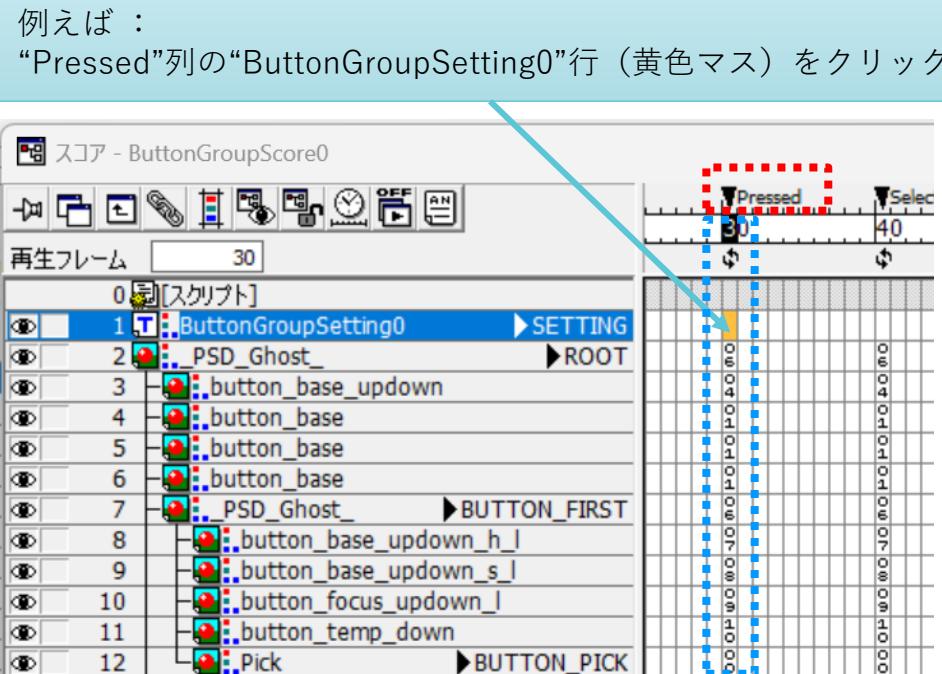
- 下図のサブスコアが表示されます。PSDファイルからGUIObjectの仕様にそって、各状態（Released、Pressedなど）の画像が取り込まれたことが確認できます。



3-2. インポート後の確認 (2/2)

操作

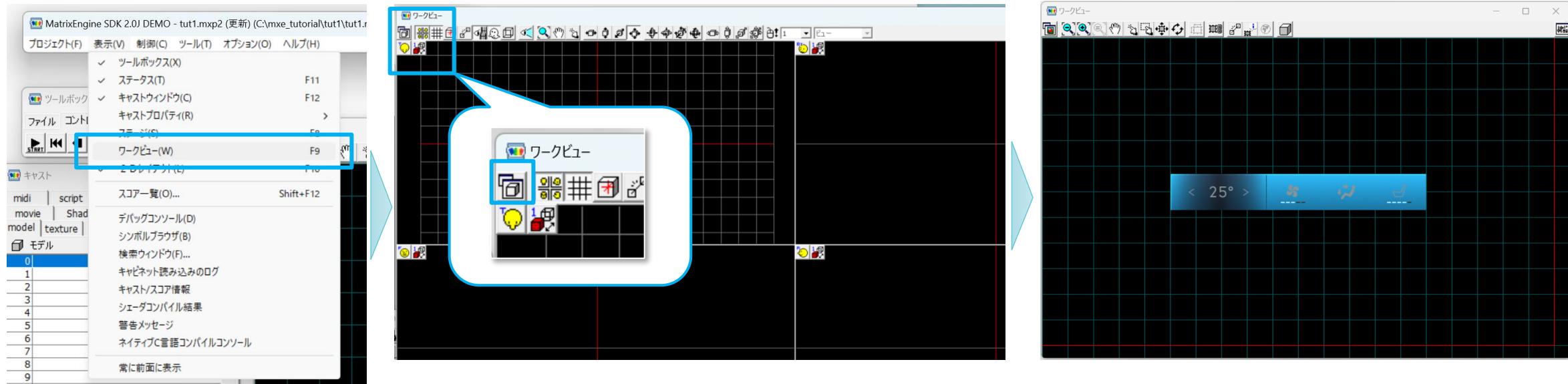
3. サブスコアの状態の名前として**タグが付けられた列** (n0番フレーム) をクリックしてください。その状態の画像を「ワークビュー」で確認できます。



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。

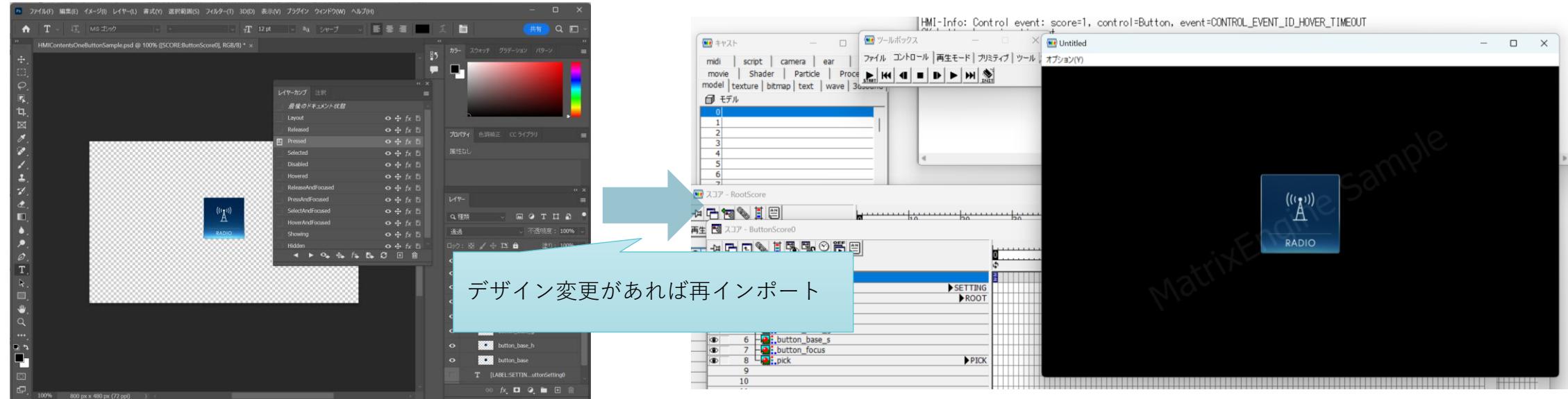


□ “ワークビュー”では、「スコア」に登録された「キャスト」の再生時のレイアウトを視覚的に表示し確認できます。

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ButtonGroup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 #2 ボタングループ）」
 - （ガイドに従って付属と同等の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ます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



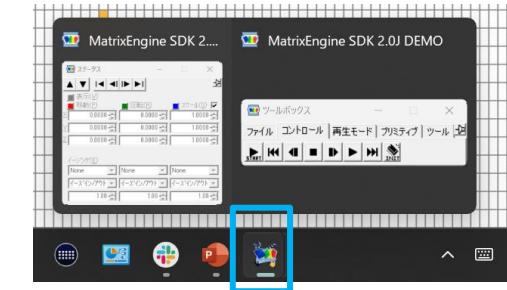
- デザイナーによるデザイン制作とエンジニアによる動作実装、MxE SDKは両者の分界点であり協働ツールです。
- Photoshopで制作したデータをそのままSDKにオブジェクトとして取り込むため、デザイナー-エンジニア間の認識齟齬も、情報の劣化も起こりません。
- デザインから実装へ、主要プレイヤーが切り替わる工程間のスムーズな情報連携をMxEは可能にします。

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)

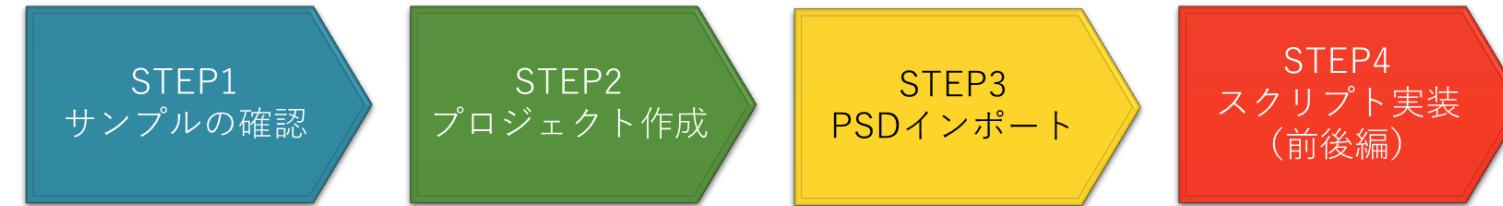
- このステップでは、[Step1で確認したサンプル](#) 「HMIContentsOneButtonGroupSample.mxp2」 の「MainFlow」 (※)スクリプトの内容を解説していきます。
- 解説を参考に、Step2／Step3で新規作成・編集（画像取り込み）したプロジェクトファイル「tut2.mxp2」へ実際に実装してみてください。
- このステップでは、次のファイルを交互に取り扱います。
 - サンプルファイル : <GUI_Objectファイル>¥sample¥ HMIContentsOneButtonGroupSample.mxp2
 - 実装プロジェクト : <プロジェクトフォルダ>¥tut2.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)

- チュートリアル#1と同様に、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](#)したボタングループを初期状態で表示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>
ボタングループの
有効化と初期表示

<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）

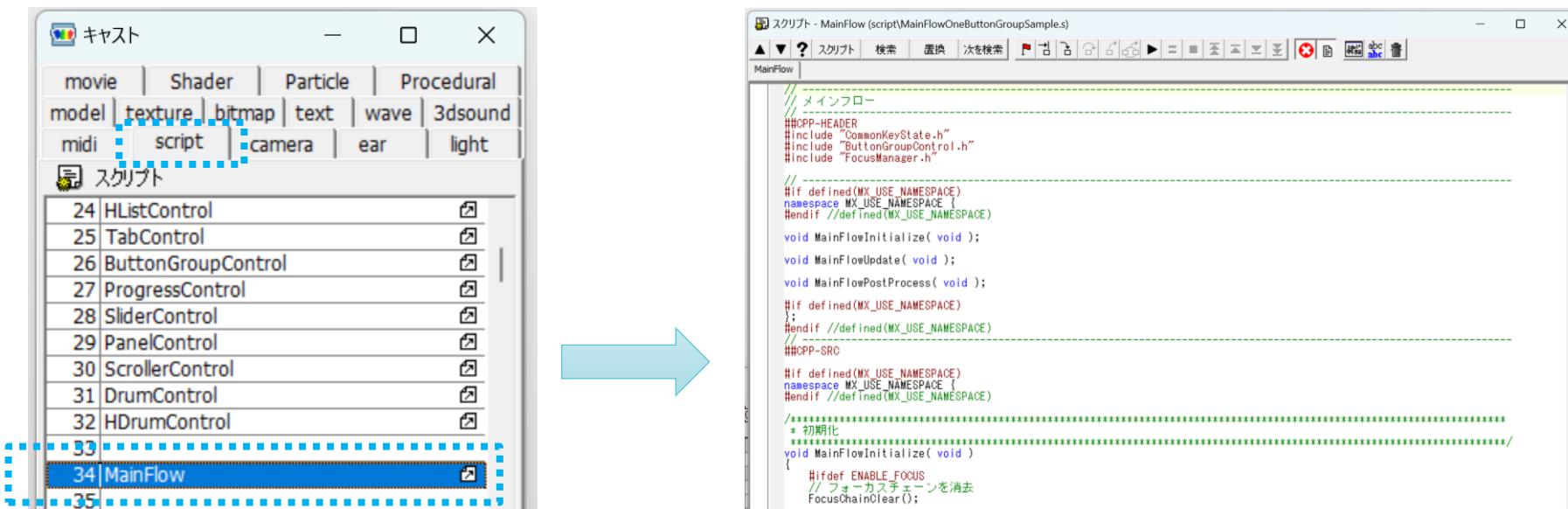
ボタングループの有効化と初期表示

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 (1/2)

- 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」関数の内容を解説します。

解説

```

MainFlow | MainFlowInitialize
MainFlowInitialize
void MainFlowInitialize( void )
{
    #ifdef ENABLE_FOCUS
        // フォーカスチェーンを消去
        FocusChainClear();
        // ボタングループをフォーカスチェーンに追加
        FocusChainAddControl( ScoreNum( ButtonGroupScore0 ), -1, -1, -1, -1, -1, -1 );
    #endif /* ENABLE_FOCUS */
    // ボタンコントロールの有効化
    ButtonGroupControlActivate( ScoreNum( ButtonGroupScore0 ) );
}

```

- 初期化処理は起動時に一度だけ実行されます。サンプルでは、次の処理を行っています。
- ① フォーカスチェーンの初期化と設定
 - ② コントロールの有効化

4-2. 初期化処理の実装 (2/2)

解説

① フォーカスチェーンの初期化と設定

- GUIObjectのコントロールモジュールには、カーソルキー等の入力デバイスによる操作に対応するため、フォーカスを受け取る機能が備わっています。
- 各コントロールモジュール間の繋がりを表す「フォーカスチェーン」を画面毎に作成することで、フォーカス移動が自動的に処理されます。
- ここでは最初に**FocusChainClear関数**で繋がりをリセットし、**FocusChainAddControl 関数**でButtonGroupをフォーカスチェーンに追加しています。この結果、フォーカス制御（キー操作等の対象化）が可能になります。

② コントロールの有効化

- [Step3](#)でPSDファイルからインポートしたスコア「ButtonGroupScore0」に対して**ButtonGroupControlActivate 関数**でコントロールの有効化を行っています。

操作

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

初期化処理で利用している関数とマクロ

関数名	内容
FocusChainClear	コントロール同士のフォーカスの繋がり（キー入力での移動など）をリセットします。
FocusChainAddControl	指定のコントロールをフォーカスチェーンに登録します。 第一引数で登録対象のコントロールを、以降の引数で前後上下左右キーでの移動先となるコントロールを指定します（フォーカス移動の際、非活性化されたコントロールは自動でスキップされます）。
ButtonGroupControlActivate	前頁の例では、他コントロールへの移動は全て「-1」 = 「移動先なし」と指定されています。
マクロ名	内容
ScoreNum	スコアの名前から番号に変換するマクロです。

- フォーカスについての詳細は[別資料](#)「GUI Object リファレンスマニュアル」の「機能拡張・フォーカス管理」をご参照ください。

4-4. 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 (tut2.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。

- ✓ ボタングループが初期状態で表示されることを確認してください。
- ✓ マウスオーバー時／クリック時の動作は、この時点ではサンプルと異なります。
- ✓ 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「×



4-4. 状態更新処理のアウトライン (1/2)

- [状態更新処理](#)は、MainFlowスクリプトの「MainFlowUpdate」関数で実装します。（毎フレーム実行されます）
- 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（次頁参照）。
 - ✓ [必要な変数の宣言](#) . . . ①
 - ✓ [キー入力処理](#) . . . ②
 - ✓ [ホバリング判定処理](#) . . . ③
 - ✓ コントロールイベントの受信と対応 . . . ④

各パートについて、次頁から詳しく見ていきます。

(④については、[ステップ後編](#)で解説します。)

4-4. 状態更新処理のアウトライン (2/2)

```


//*****状態更新処理*****:
* 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;
    int number = 0;

    #ifdef WIN32
    #ifdef ENABLE_FOCUS
    // フォーカスの制御 (キーでフォーカス移動、Enterキーで選択)
    if (KeyStateIsUp ( VK_PRIOR )) FocusRequestMove ( FOCUS_MOVE_PREV );
    if (KeyStateIsUp ( VK_NEXT )) FocusRequestMove ( FOCUS_MOVE_NEXT );
    if (KeyStateIsUp ( VK_UP )) FocusRequestMove ( FOCUS_MOVE_UP );
    if (KeyStateIsUp ( VK_DOWN )) FocusRequestMove ( FOCUS_MOVE_DOWN );
    if (KeyStateIsUp ( VK_LEFT )) FocusRequestMove ( FOCUS_MOVE_LEFT );
    if (KeyStateIsUp ( VK_RIGHT )) FocusRequestMove ( FOCUS_MOVE_RIGHT );

    if (KeyStateIsDown ( VK_RETURN )) FocusRequestSelectDown();
    if (KeyStateIsUp ( VK_RETURN )) FocusRequestSelectUp();
    #endif /* ENABLE_FOCUS */
    #endif // WIN32

    // 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;

    ButtonGroupControlSetHoverStatus ( ScoreNum ( ButtonGroupScore0 ), &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 ButtonGroupScore0 に対するイベントを処理
        case ScoreNum ( ButtonGroupScore0 ):
            switch (eventId)
            {
                // ボタン押下イベントを処理
                case CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_TAPPED:
                    // ボタンが押下されたらログを出力
                    number = ControlEventGetParamInteger ( paramStart );
                    #if LOG_LEVEL >= NORMAL
                    {
                        char str[128];
                        sprintf(str, "button %d tapped.", number); 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_BUTTON_GROUP_ITEM_HOVER_TIMEOUT:
                    // ボタン上のホバリング状態が一定時間を越えたらログを出力
                    number = ControlEventGetParamInteger ( paramStart );
                    #if LOG_LEVEL >= NORMAL
                    {
                        char str[128];
                        sprintf(str, "button %d hovering timeout.", number); LOG(str);
                    }
                    #endif
                    break;

                // ロングタッチイベントを処理
                case CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_LONG_TAPPED:
                    // ボタン押下中にロングタップ時間を越えたらログを出力
                    number = ControlEventGetParamInteger ( paramStart );
                    #if LOG_LEVEL >= NORMAL
                    {
                        char str[128];
                        sprintf(str, "button %d long tapped.", number); 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;
    int number = 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グループを実現します。
 - ✓ マウスカーソルがボタン上にあるとき
 - ✓ ボタンにフォーカスがあるとき、フォーカスが移動したとき、ボタンを押下したとき
- 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キー入力処理

解説

```
#ifdef WIN32
#ifndef ENABLE_FOCUS
// フォーカスの制御（キーでフォーカス移動、Enterキーで選択）
if (KeyStateIsUp ( VK_PRIOR )) FocusRequestMove ( FOCUS_MOVE_PREV );
if (KeyStateIsUp ( VK_NEXT )) FocusRequestMove ( FOCUS_MOVE_NEXT );
if (KeyStateIsUp ( VK_UP )) FocusRequestMove ( FOCUS_MOVE_UP );
if (KeyStateIsUp ( VK_DOWN )) FocusRequestMove ( FOCUS_MOVE_DOWN );
if (KeyStateIsUp ( VK_LEFT )) FocusRequestMove ( FOCUS_MOVE_LEFT );
if (KeyStateIsUp ( VK_RIGHT )) FocusRequestMove ( FOCUS_MOVE_RIGHT );

if (KeyStateIsDown ( VK_RETURN )) FocusRequestSelectDown();
if (KeyStateIsUp ( VK_RETURN )) FocusRequestSelectUp();
#endif /* ENABLE_FOCUS */
#endif // WIN32
```

②

- 特定のキーが押された際にボタングループコントロールのフォーカスを移動する処理が記述されています。
 - KeyStatelsUp関数を利用しているため、正確には「キーが押された際に」ではなく「離された際に」チェックしています。
- 押されたキー毎に FocusRequestMove 関数で、フォーカスに対する動作を実装しています。
 - PRIOR/PREVとNEXTはPageUpとPageDownに、他は上下左右キーに対応しています。

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。

キー入力処理で利用しているフォーカス操作関数

関数名	内容
FocusRequestMove	フォーカスを指定方向へ移動します。
FocusRequestSelectDown	現在フォーカスのあるコントロールに選択開始を要求します。 コントロールが選択可能な状態にあれば、押下（ハイライト）状態に変化します。 ※本関数の呼び出しは必須ではなく、FocusRequestSelectUp 呼び出しのみでも動作します。
FocusRequestSelectUp	現在フォーカスのあるコントロールに選択完了を要求します。 コントロールが選択可能な状態にあれば、選択されます。

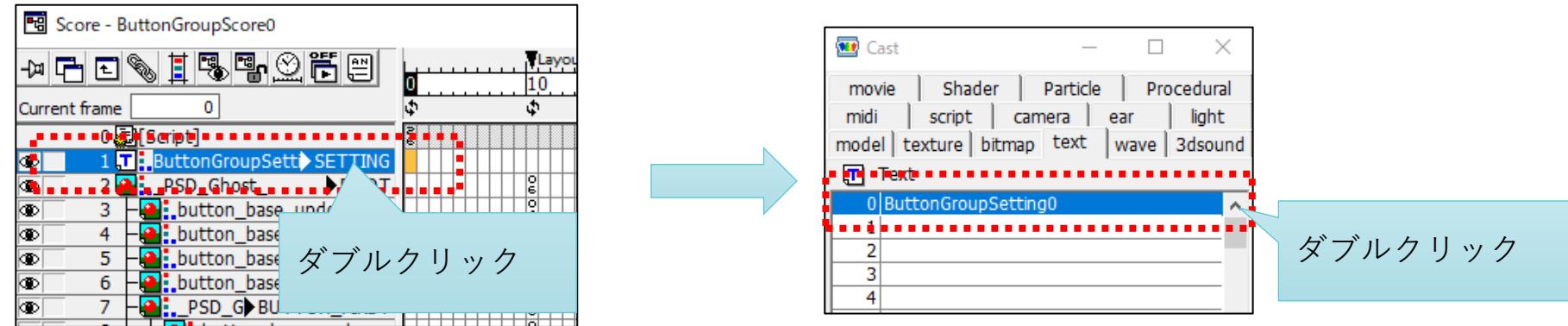
- フォーカスについての詳細は[別資料「GUI Object リファレンスマニュアル」](#)の「機能拡張・フォーカス管理」をご参照ください。

4-7. フォーカス移動の有効化（1/2）

- 「[状態更新処理②キー入力処理](#)」でスクリプトに記述したフォーカス操作については、ButtonGroupコントロール側の設定項目を編集し、キーによるフォーカス移動可否を「可」に設定しておく必要があります。

操作

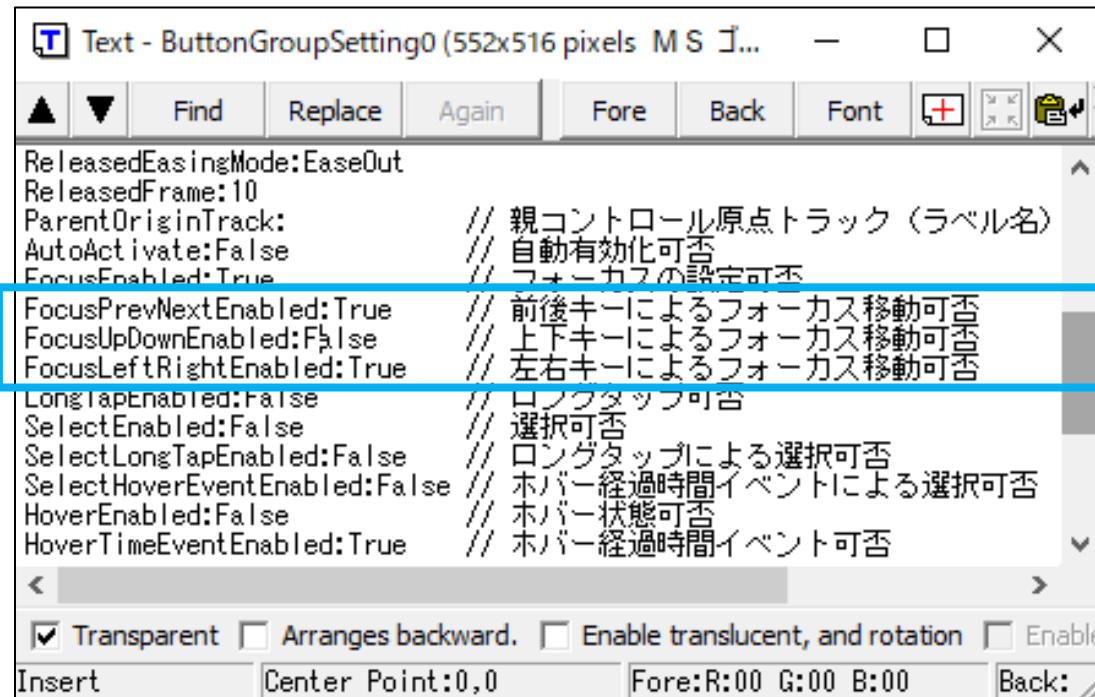
- 実装プロジェクトで、サブスコアButtonGroupScore0のトラック1、ButtonGroupSetting0をダブルクリックしてください。
- "cast"ウィンドウの"text"タブの中の同名のキャストがフォーカスされます。
- フォーカスされた"text"タブの"ButtonGroupSetting0"をダブルクリックしてください。



4-7. フォーカス移動の有効化 (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、"Focus<キー>Enabled"を「True」にすることで該当キーによるフォーカス移動が可能になります。



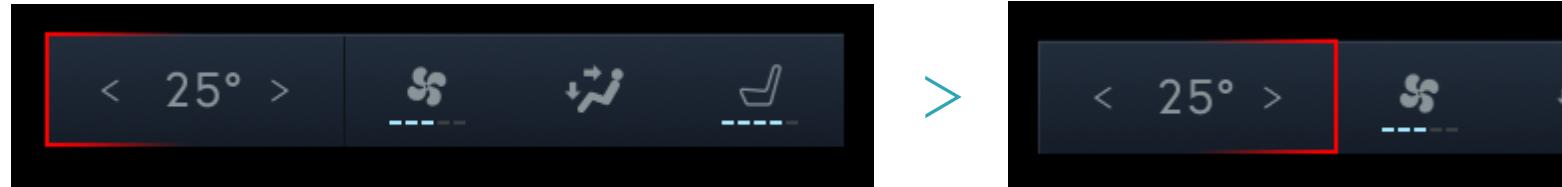
上下キーによるフォーカス移動がFalseに設定されていますので、Trueに書き換えてください。

- ✓ 以上でキーによるフォーカス移動の実装は完了です。プロジェクトを実行すると、キーによるフォーカス移動を確認できます。

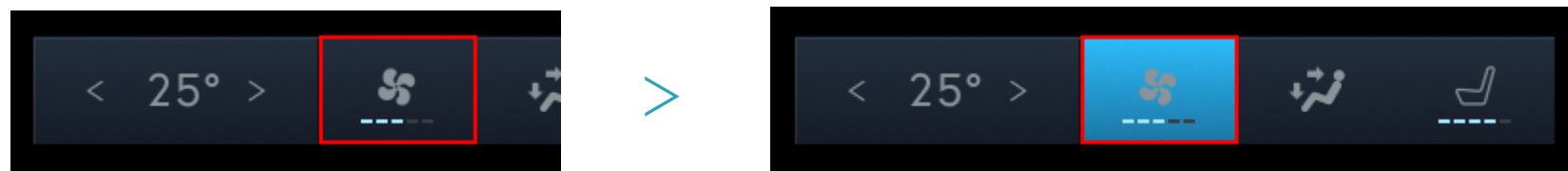
補足：キー入力処理とフォーカス移動有効化後の動作

- ここまで実装で、キーによるフォーカス移動とEnterキーでのボタン選択（押下）を確認できます。
- 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。

- ✓ 上下左右／PgUp PgDnキーにてフォーカスが移動することを確認できます。



- ✓ フォーカスがある状態でEnterキーを押下するとボタンが明るく光ります。



4-8. 状態更新処理 ③ホバリング判定処理

解説

```
// 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;  
  
ButtonGroupControlSetHoverStatus ( ScoreNum ( ButtonGroupScore0 ), &hoverArea, true );
```

③

- 続いて、マウスのホバー時にボタン表示を変更するため、ホバリング判定処理を記述します。この処理はチュートリアル#1「シングルボタンアプリ」と同じ内容です。
- 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。
- ButtonGroupControlSetHoverStatus関数は、指定のボタンに、指定の座標でホバリング判定（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

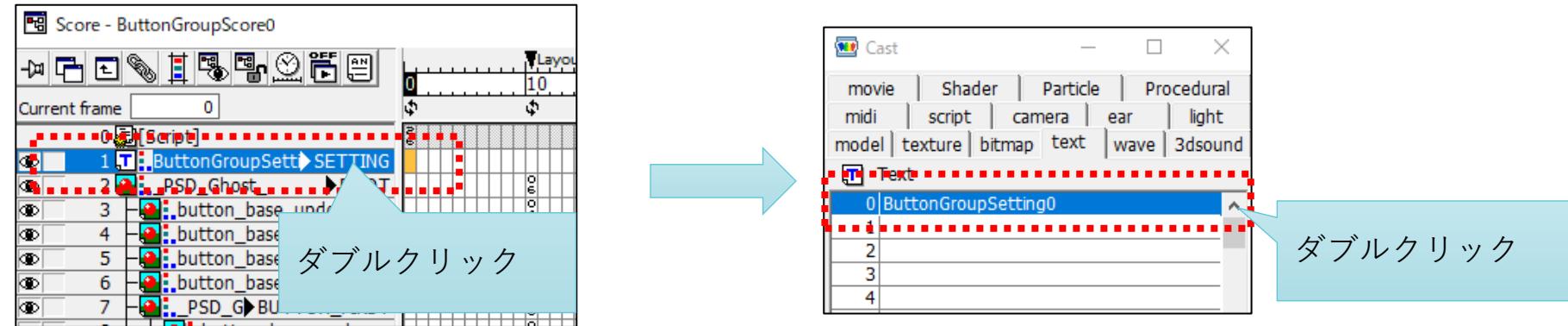
1. 実装プロジェクトのMainFlowスクリプトを開き、サンプルファイルの③の内容を「MainFlowUpdate」関数に記述してください。

4-9. ホバリング判定の有効化 (1/2)

- チュートリアル#1と同様に、「状態更新処理③ホバリング判定」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、ButtonGroupコントロール側の設定項目を編集し、ホバリング状態可否を「可」と設定しておく必要があります。

操作

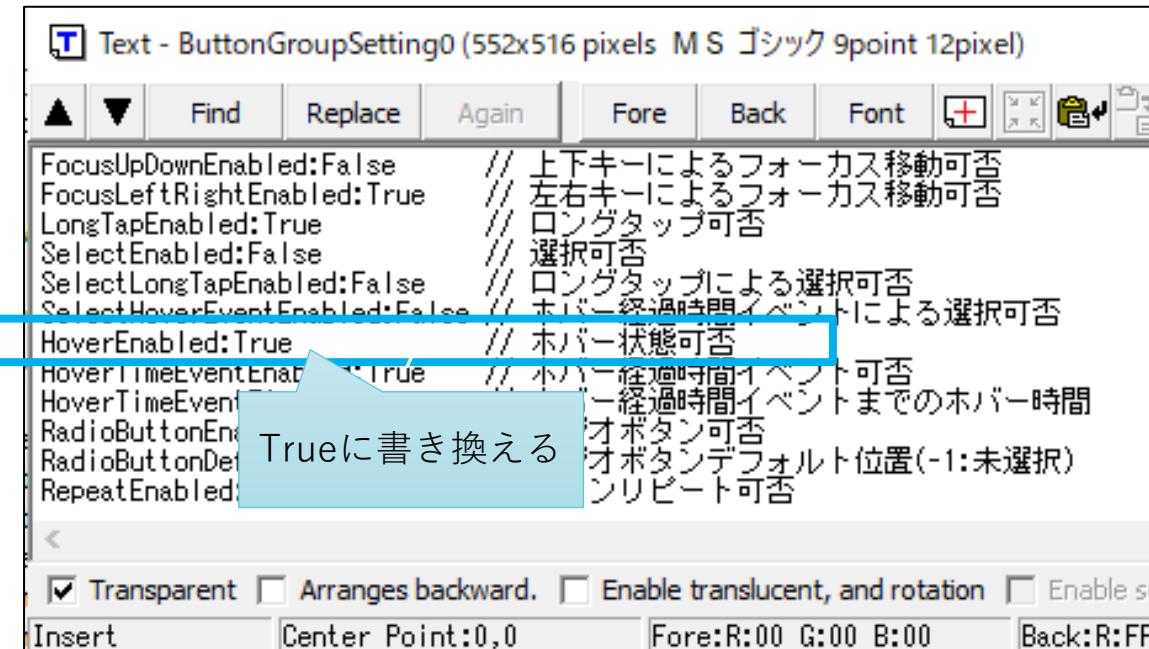
- 実装プロジェクトで、サブスコアButtonGroupScore0のトラック1、ButtonGroupSetting0をダブルクリックしてください。
- "cast"ウィンドウの"text"タブの中の同名のキャストがフォーカスされます。
- フォーカスされた"text"タブの"ButtonGroupSetting0"をダブルクリックしてください。



4-9. ホバリング判定の有効化 (2/2)

操作

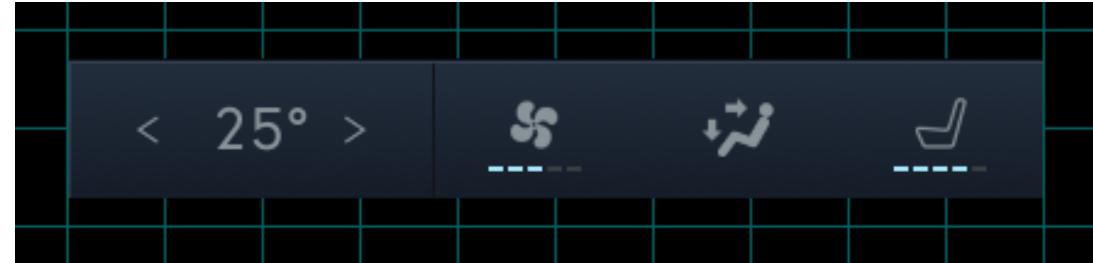
4. エディタが開きます。これは機能名と設定値が組となったコントロール設定用テキストキャストで、"HoverEnabled"を「True」に書き換えることでホバリング判定が可能になります。



- ✓ 以上でボタングループの表示変更に関する実装は完了です。動作を確認してみましょう。

補足：ホバリング状態を有効化の結果

通常表示：



マウスカーソルOn・ホバリング状態：
PSDファイルの時点でホバリング用のレイヤーで
透明度を施しているため下が透けています。

クリック時：



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：ボタングループコントロールであれば「ボタングループ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 ボタングループ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 要素固有の状態
3.4 フォーカス
3.5 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 ボタンの表示数と素材について
5.3 トランジションアニメーション種別
5.4 トック構造とトックラベル

4-10. 状態更新処理の動作確認

- ボタングループの表示に関する実装が完了しました。実装プロジェクト (tut2.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. 表示された画面で動作を確認してください。
 - [キーによるフォーカス移動](#)
 - [マウスオーバー時／クリック時のボタン](#)

✓ 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。



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

4-11. 状態更新処理④イベントの受信と分岐

- 続いて、ボタングループ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パート（④）です。
- 本サンプルでは「何番目のボタンが押下されたか」というログを出力します。

解説

- チュートリアル#1の時と同様、while文の条件式の中のControlCatchEvents関数でイベントを受信し、スコア（コントロール）別の分岐、イベントID別の分岐で、異なる処理を行います。

```

// 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 ButtonGroupScore0 に対するイベントを処理
        case ScoreNum ( ButtonGroupScore0 ):
            switch (eventId)
            {
                // ボタン押下イベントを処理
                case CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_TAPPED:
                    // ボタンが押下されたらログを出力
                    number = ControlEventGetParamInteger ( paramStart );
                    #if LOG_LEVEL >= NORMAL
                    {
                        char str[128];
                        sprintf(str, "button %d tapped.", number); LOG(str);
                    }
            }
    }
}

```

case文に定数としてあてがうため、スコア名ButtonGroupScore0にScoreNumマクロを適用します。

スコアで分岐：どのスコアのイベントか

イベントIDで分岐：イベント種類は何か

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、「&eventId」でイベント種類を取得できます。

4-12. 状態更新処理 ④ イベントIDの分岐

解説

- スコア振り分け後は、チュートリアル#1と同様に、ButtonGroup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。後述の「[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。

タップ時

ホバリング
タイムアウト

長押し

```
// スコア ButtonGroupScore0 に対するイベントを処理
case ScoreNum ( ButtonGroupScore0 ):
    switch (eventId)
    {
        // ボタン押下イベントを処理
        case CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_TAPPED:
            // ボタンが押下されたらログを出力
            number = ControlEventGetParamInteger ( paramStart );
            #if LOG_LEVEL >= NORMAL
            {
                char str[128];
                sprintf(str, "button %d tapped.", number); 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_BUTTON_GROUP_ITEM_HOVER_TIMEOUT:
            // ボタン上のホバリング状態が一定時間を越えたらログを出力
            number = ControlEventGetParamInteger ( paramStart );
            #if LOG_LEVEL >= NORMAL
            {
                char str[128];
                sprintf(str, "button %d hovering timeout.", number); LOG(str);
            }
            #endif
            break;

        // ロングタッチイベントを処理
        case CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_LONG_TAPPED:
            // ボタン押下中にロングタップ時間を越えたらログを出力
            number = ControlEventGetParamInteger ( paramStart );
            #if LOG_LEVEL >= NORMAL
            {
                char str[128];
                sprintf(str, "button %d long tapped.", number); LOG(str);
            }
            #endif
            break;
    }
}
```

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、ButtonGroup 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_BUTTON_GROUP_ITEM_TAPPED
 - ホバリングタイムアウト : CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVER_TIMEOUT
 - クリック長押し : CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_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：ボタングループコントロールであれば「ボタングループ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。合わせてご参照ください。

1 更新履歴
2 目次
▲ 3 ボタングループ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 要素固有の状態
3.4 フーカス
3.5 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 ボタンの表示数と素材について
5.3 トランジションアニメーション種別
5.4 トランジションアニメーション種別

4-13. 状態更新処理 ④ イベントパラメータ取得とログ出力

解説

- 受信したイベントにはIDの他に任意の数の「パラメータ」データが含まれています。ButtonGroupコントロールでは、この「パラメータ」で何番目のボタンによるイベントかを判別できます。
- サンプルコードではボタンのインデクスを示す、1番目（インデクス[+0]）のパラメータデータを抽出して、何番目のボタンが押されたかを判別しています。

```
// ボタン押下イベントを処理
case CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_TAPPED:
    // ボタンが押下されたらログを出力
    number = ControlEventGetParamInteger ( paramStart );
    #if LOG_LEVEL >= NORMAL
    {
        char str[128];
        sprintf(str, "button %d tapped.", number); LOG(str);
    }
    #endif
    break;
```

ButtonGroupの中の何番目のボタンであるかを示す、
1番目（インデクス+0）のパラメータを抽出

ログに出力

※パラメータの詳細は、[別資料](#)「GUI Objrct リファレンスマニュアル」の「機能拡張・コントロール管理」内「コントロールイベント定数」をご参照ください。

操作

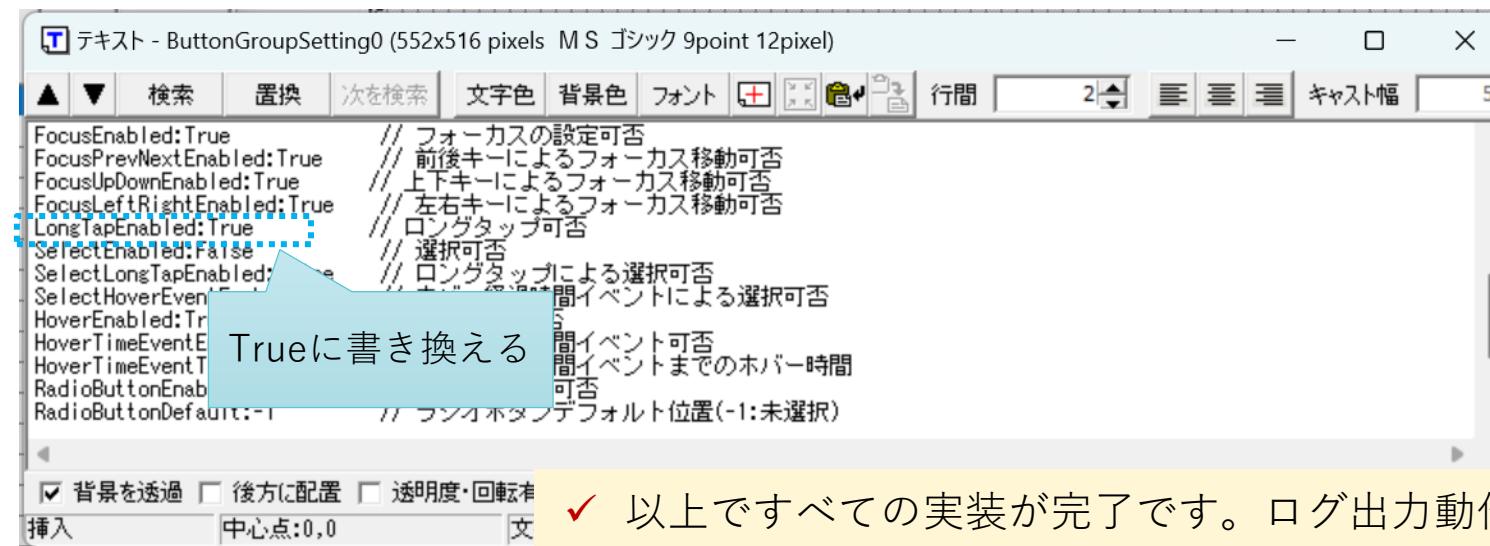
1. 実装プロジェクトのMainFlowスクリプトを開き、サンプルファイル④の内容を「MainFlowUpdate」関数に記述してください。

4-14. 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「長押しつイベント」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ButtonGroupScore0のトラック1、ButtonGroupSetting0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"ButtonGroupSetting0"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。ログ出力動作を確認しましょう。

4-15. 最終動作確認

- 実装プロジェクト (tut2.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

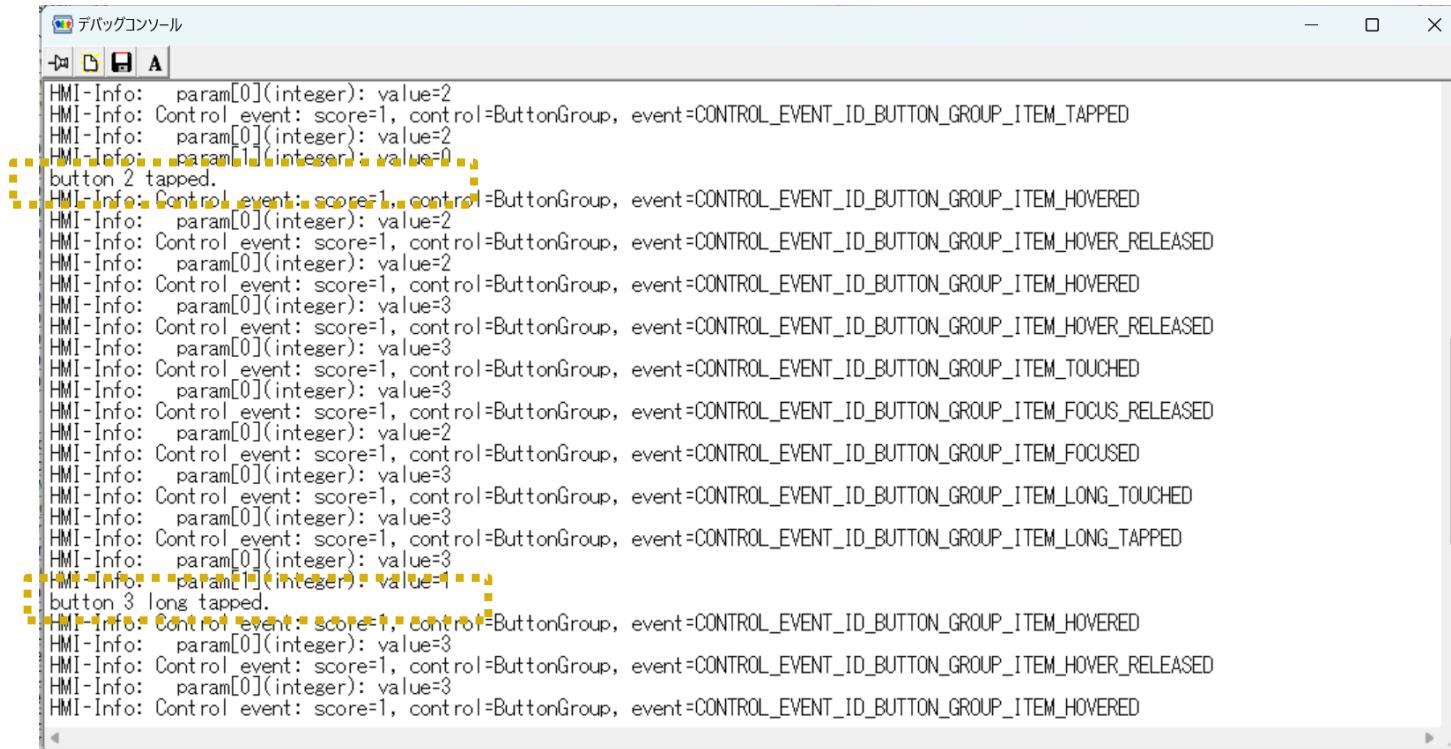
- 実装プロジェクトの"Tool Box"ウィンドウで"Control"タブを開きます。
- "START"ボタンを押下してください。画面が表示されます。
- ボタンに対して左右キーやマウスを操作、クリック、長押しの操作を実行してみてください。

- 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。



補足：イベントごとのログ表示例

- 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 a Windows-style application window titled "デバッグコンソール" (Debug Console). The window contains a text area displaying log messages from the HMI module. The log entries are as follows:

```
HMI-Info: param[0](integer): value=2
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_TAPPED
HMI-Info: param[0](integer): value=2
HMI-Info: param[1](integer): value=0
button 2 tapped.
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVERED
HMI-Info: param[0](integer): value=2
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVER_RELEASED
HMI-Info: param[0](integer): value=2
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVERED
HMI-Info: param[0](integer): value=3
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVER_RELEASED
HMI-Info: param[0](integer): value=3
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_TOUCHED
HMI-Info: param[0](integer): value=3
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_FOCUS_RELEASED
HMI-Info: param[0](integer): value=2
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_FOCUSED
HMI-Info: param[0](integer): value=3
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_LONG_TOUCHED
HMI-Info: param[0](integer): value=3
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_LONG_TAPPED
HMI-Info: param[0](integer): value=3
HMI-Info: param[1](integer): value=1
button 3 long tapped.
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVERED
HMI-Info: param[0](integer): value=3
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVER_RELEASED
HMI-Info: param[0](integer): value=3
HMI-Info: Control.event: score=1, control=ButtonGroup, event=CONTROL_EVENT_ID_BUTTON_GROUP_ITEM_HOVERED
```

- ✓ 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 このフォルダには、各コントロールのマニュアルが含まれます。 本チュートリアル実施後に「ボタングループコントロールマニュアル.docx」をお読みになると、より深い理解が得られます。
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) このフォルダには、以下が含まれます。 • 各コントロールごとの設定や関数リファレンス • 本チュートリアルで一部ご紹介したフォーカス管理・コントロール管理等の機能拡張関数リファレンス

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
MxE SDK向け PSDファイル作成ガイド (MxE チュートリアル #2 ボタングループ)	(本チュートリアルと同梱) tut2_ボタングループ_PSD作成ガイド.pdf