ClickerAce プラグイン 開発資料

2020 年 9 月 8 日版

目次

1　ClickerAce プラグインについて

1.1 プラグインで出来る事

ClickerAce にご自分でプログラムされた独自の操作を行わせる事が出来るようになります。例えば操作の実行後にクリップボードの中身をデータベースに書き込んだり、操作の途中に続行するかどうかの確認ダイアログを表示する等、基本プラグインでは提供しにくい細かなところに手が届くような機能を追加する事が出来ます。

現在の機能では登録と再生を行う際に動作するプラグイン機能のみの利用が可能ですが、今後のバージョンアップによって行える操作を増やす予定です。

1.2 プラグイン推奨開発環境

OS

Microsoft Windows 10

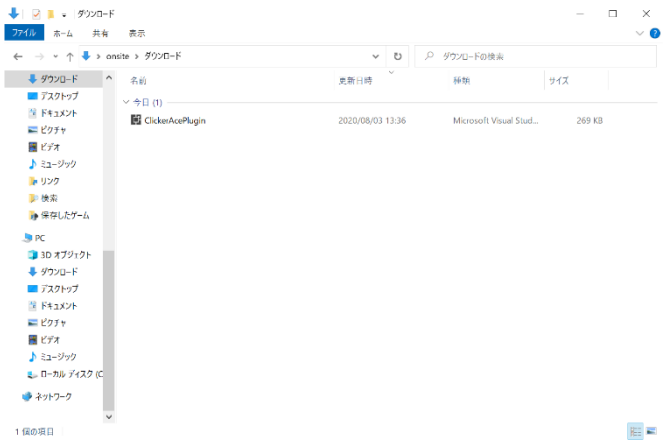
IDE

Microsoft Visual Studio 2019 Community Edition 以上

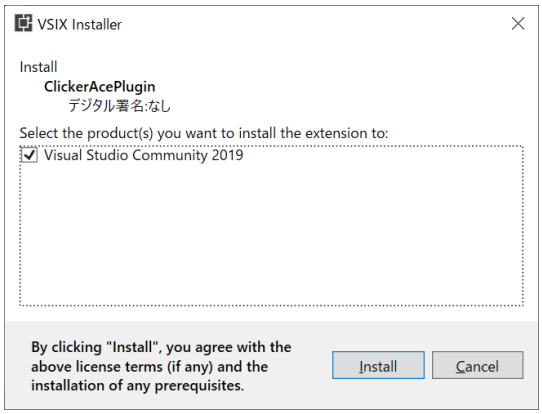
2　プラグインプロジェクト作成

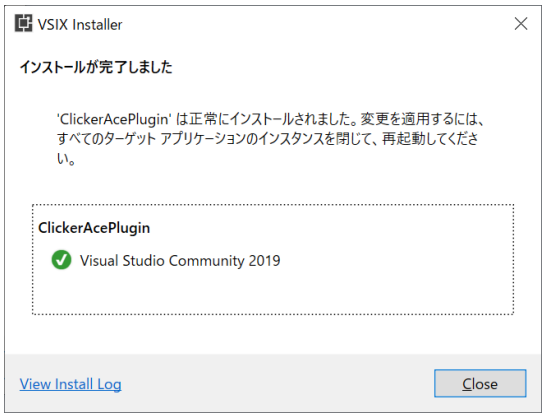
2.1 ClickerAce プラグイン開発用拡張機能のインストール

ClickerAce Web サイト(http://clickerace.com/)より、プラグイン開発用拡張機能をダウンロードします。



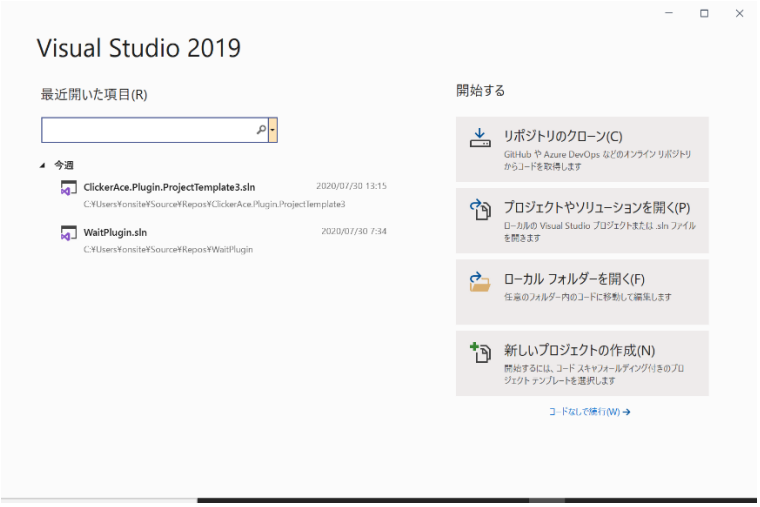
ダウンロードされた ClickerAce プラグイン開発用拡張機能(ClickerAcePlugin.vsix)を開き、インストールします。





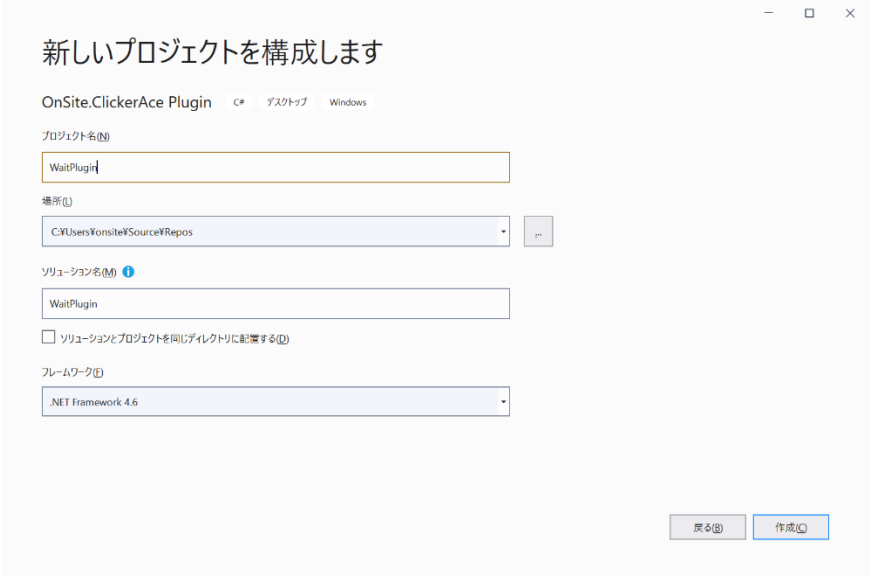
2.2 プラグインプロジェクトの作成

Visual Studio 2019 を起動し、新しいプロジェクトの作成を選択。

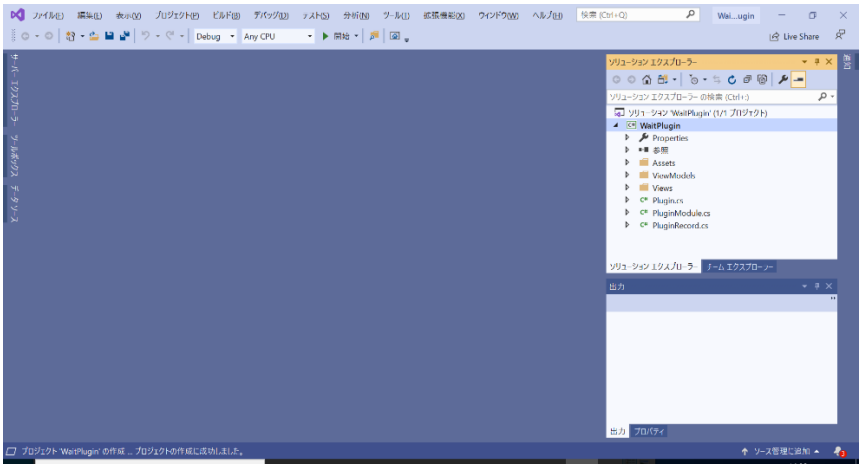


拡張機能からインストールされた「OnSite ClickerAce Plugin」プロジェクトテンプレートを選択。

[プロジェクト名]に作成するプラグインの名前を入力しプロジェクトを作成します。

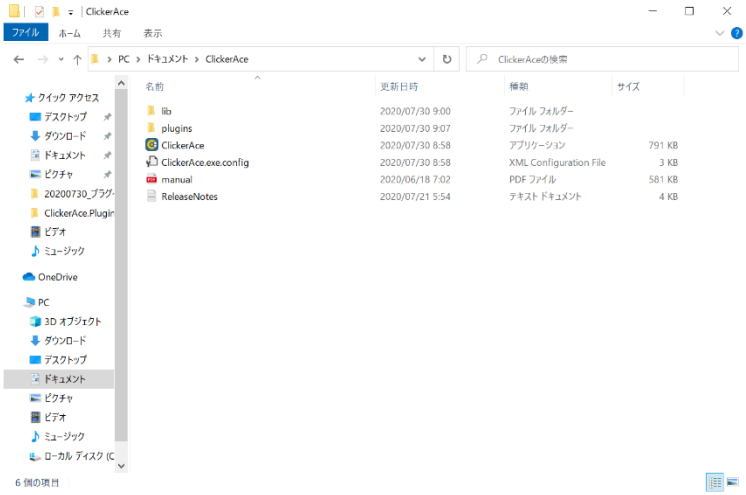


以上の手順でプラグインプロジェクトが作成されます。

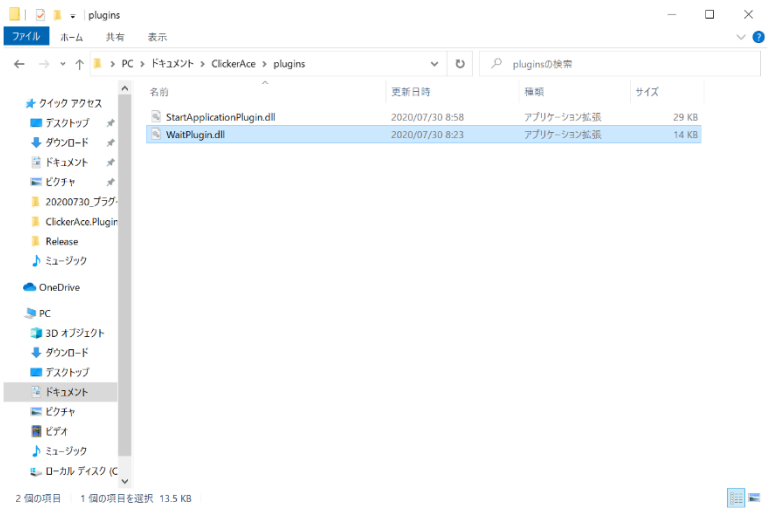


3　作成したプラグインの配置

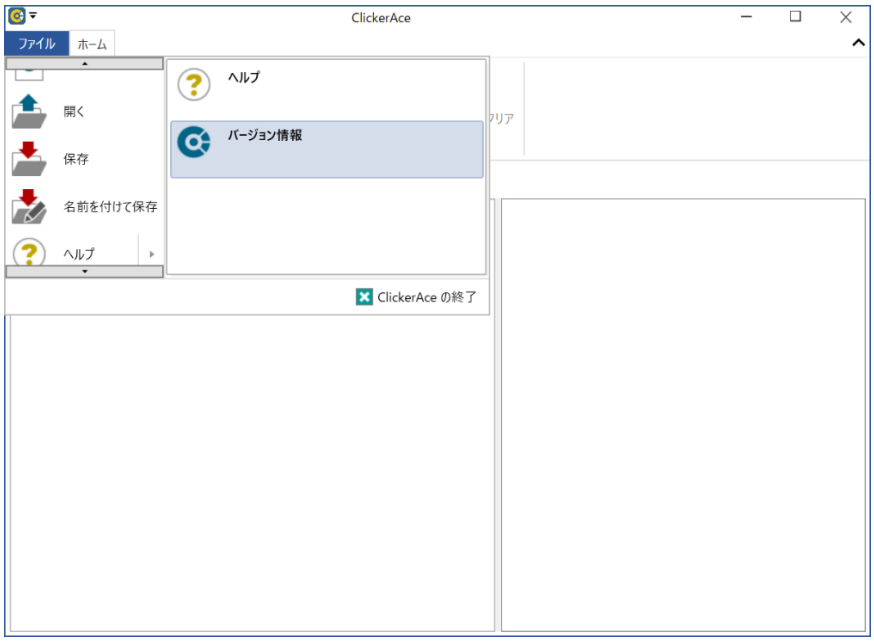
ClickerAce Web サイト(http://clickerace.com/)より、プログラム本体をダウンロードし展開します。



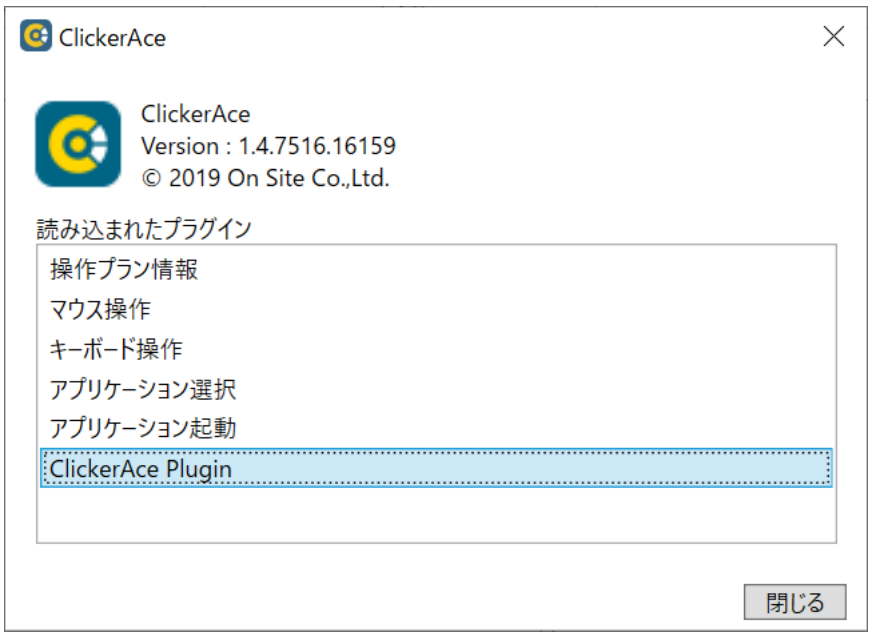
[plugins]フォルダ内に作成したプラグインファイルを配置します。



[ClickerAce]を起動し、[ファイル]メニューからバージョン情報を開きます。



プラグインが正常に読み込まれた場合はプラグイン一覧に作成したプラグインが表示されます。



※ 画面はプロジェクトテンプレートから作成したプラグインの初期タイトルである「ClickerAce Plugin」となっているもの

4　サンプルプログラム

「OnSite ClickerAce Plugin」プロジェクトテンプレートから作成したプロジェクトは、そのままビルドするだけで ClickerAce のプラグインとして利用可能なサンプルプログラムとなっています。
実際の動作と対応するプログラムについて説明します。

4.1 テンプレートプロジェクト構成

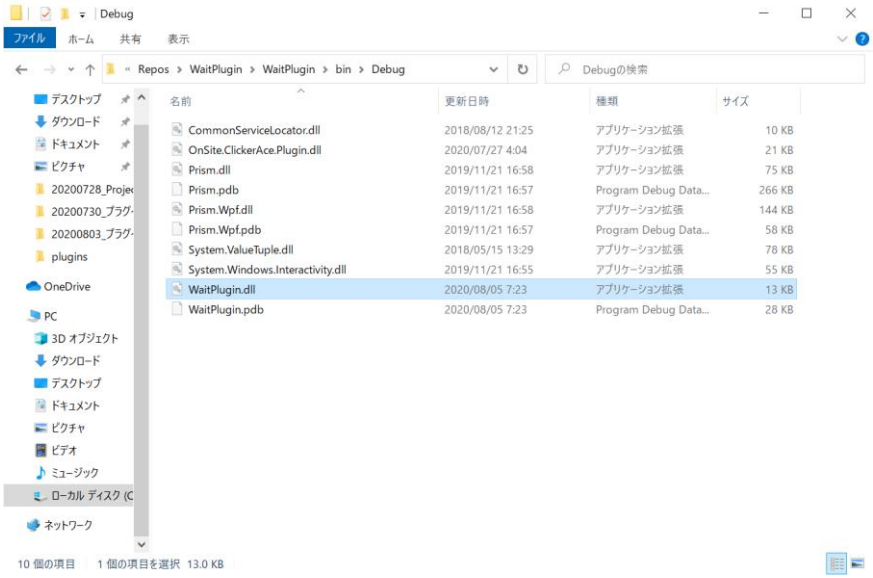
[プロジェクト]
┣ PluginModule.cs
┣ Plugin.cs
┣ PluginRecord.cs
┣ [Assets]
┃ ┗ Icon.png
┣[Views]
┃ ┗ EditView.xaml
┗[ViewModels]
　┗ EditViewModel.cs

4.1.1　PluginModule.cs

ClickerAce が使用する WPF フレームワーク「Prism」のモジュール宣言を行います。
設定画面が 1 つしかないプラグインを作成する場合には基本的に変更は不要です。

4.2 サンプルプラグインの配置

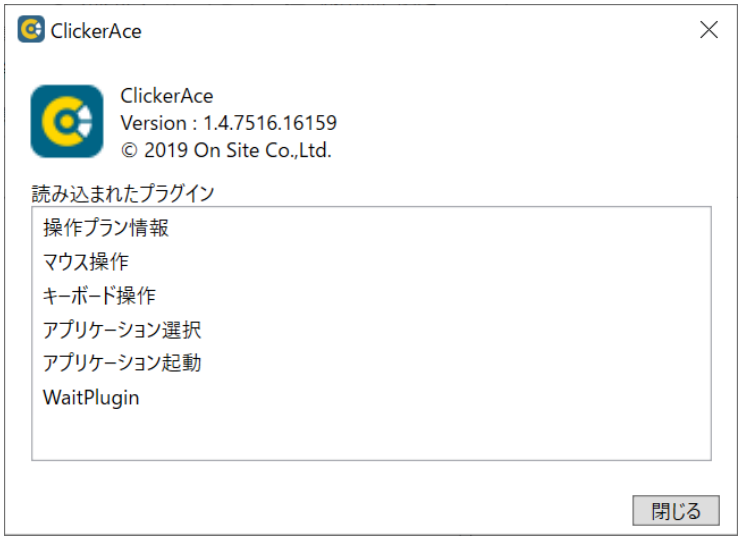
作成したプロジェクト(ここでは WaitPlugin)をビルドし、「WaitPlugin.dll」を ClickerAce の「plugins」フォルダにコピーします。



※ 「OnSite.ClickerAce.Plugin.dll」や、「Prism.dll」等の DLL はビルド先フォルダに出力されますが、ClickerAce 本体に同梱されていますのでコピーの必要はありません。

4.3 プラグインの読込確認

ClickerAce を起動し、[ファイル]→[ヘルプ]→[バージョン情報]からプラグインが読み込まれているか確認出来ます。
ここでは「WaitPlugin」を追加しています。



ここで表示されるプラグイン名はプロジェクト内の「Plugin.cs」で設定します。

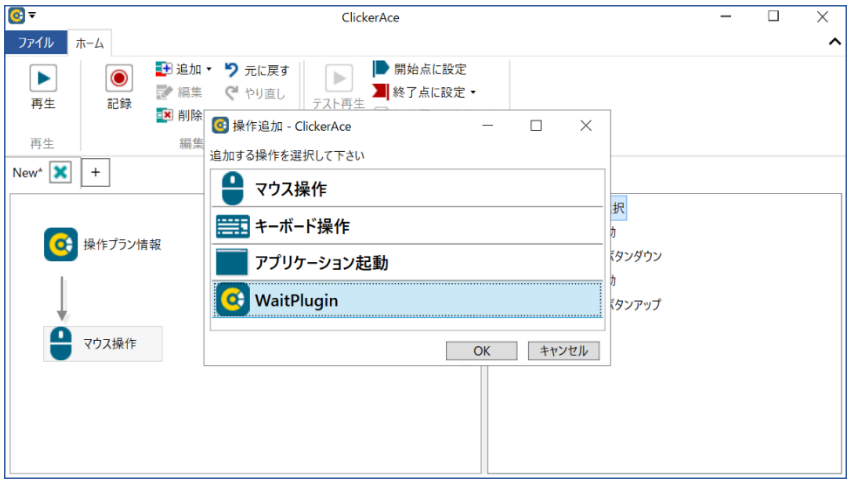
4.3.1 Plugin.cs

```
public class Plugin : PluginBase<PluginRecord>, IAddablePlugin {
    public override string Title => "WaitPlugin";
}
```

「Plugin.cs」ではそのほか、プラグインアイコンや操作の処理等が定義されます。

4.4 操作の追加

ClickerAce から操作追加の実行で追加したプラグインが選択出来るようになります。



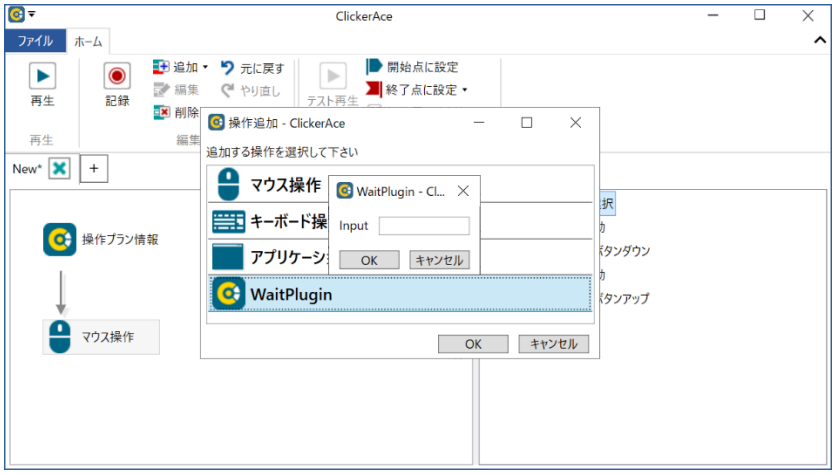
ここで表示されるプラグイン名、アイコンはプロジェクト内の「Plugin.cs」で設定します。
アイコンについてはプロジェクト内の「Assets/Icon.png」を差し替える事で変更出来ます。
※ プラグイン表示名は前項と同様の設定箇所になります。

4.4.1 Plugin.cs

```
public class Plugin : PluginBase<PluginRecord>, IAddablePlugin {
    public override ImageSource SmallIcon { get; } = new BitmapImage(new
Uri($"pack://application:,,,/{Assembly.GetExecutingAssembly().GetName()};component/Asse
ts/Icon.png"));
    public override ImageSource Icon { get; } = new BitmapImage(new
Uri($"pack://application:,,,/{Assembly.GetExecutingAssembly().GetName()};component/Asse
ts/Icon.png"));
    public override ImageSource LargeIcon { get; } = new BitmapImage(new
Uri($"pack://application:,,,/{Assembly.GetExecutingAssembly().GetName()};component/Asse
ts/Icon.png"));
}
```

4.5 プラグイン操作追加画面

操作追加で「WaitPlugin」を選択すると編集画面が開きます。



ここで表示される編集画面は「Views/EditView.xaml」で設定し、ボタン等の制御は「ViewModels/EditViewModel.cs」で設定します。

ClickerAce では MVVM（Model-View-ViewModel）フレームワーク「Prism」を採用しておりプラグイン開発時でもこちらを使用します。
詳しくは「Prism Library Documentation」（https://prismlibrary.com/docs/）をご参照下さい。

4.5.1 Views/EditView.xaml

```
<UserControl
    x:Class="WaitPlugin.Views.EditView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:plugin="clr-
namespace:OnSite.ClickerAce.Plugin.Views;assembly=OnSite.ClickerAce.Plugin"
    xmlns:prism="http://prismlibrary.com/"
    d:DesignHeight="200"
    d:DesignWidth="300"
    prism:ViewModelLocator.AutoWireViewModel="True"
```

```
    mc:Ignorable="d">
    <prism:Dialog.WindowStyle>
        <Style TargetType="Window">
            <Setter Property="prism:Dialog.WindowStartupLocation" Value="CenterOwner"
/>
            <Setter Property="SizeToContent" Value="WidthAndHeight" />
            <Setter Property="ShowInTaskbar" Value="False" />
            <Setter Property="ResizeMode" Value="NoResize" />
        </Style>
    </prism:Dialog.WindowStyle>
    <DockPanel>
        <plugin:PluginButtons DockPanel.Dock="Bottom" />
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <TextBlock
                Grid.Column="0"
                Margin="5"
                Text="Input" />
            <TextBox
                Grid.Column="1"
                Margin="5"
                Text="{Binding InputValue}" />
        </Grid>
    </DockPanel>
</UserControl>
```

「plugin:PluginButtons」は「OK」、「キャンセル」ボタンを表示するコントロールです。
入力項目等を変更される場合は「plugin:PluginButtons」下の「Grid」内を編集して下さい。

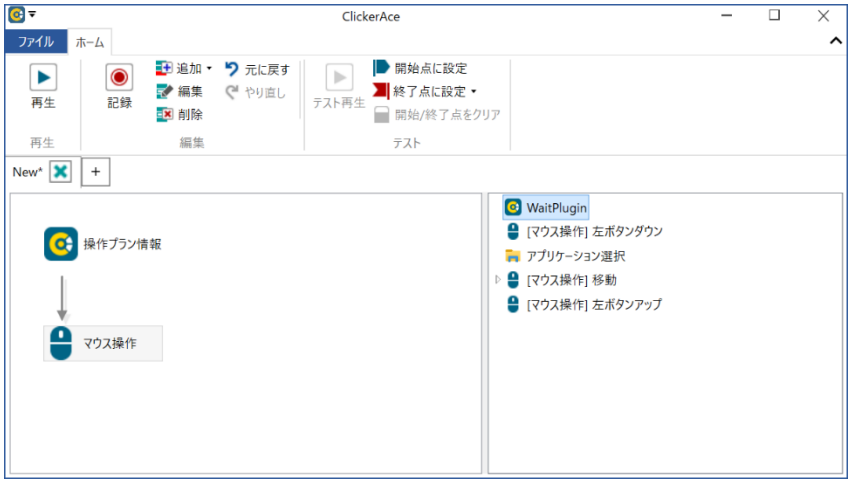
4.5.2 ViewModels/EditViewModel.cs

```
public class EditViewModel : EditDialogBaseViewModel<PluginRecord> {
    public EditViewModel() {
    }
    private string _inputValue;
    public string InputValue {
        get => _inputValue;
        set => SetProperty(ref _inputValue, value);
    }
    public override void OnDialogOpened(IDialogParameters parameters) {
        base.OnDialogOpened(parameters);
        if (!IsNew) {
            if (Value == null) throw new ArgumentNullException();
            this.InputValue = Value.SampleValue;
        }
    }
    protected override PluginRecord Apply() {
        var rec = new PluginRecord {
            SampleValue = InputValue
        };
        return rec;
    }
}
```

「OK」ボタンで入力値から操作時の値を示す PluginRecord を生成する「Apply 関数」が呼び出され、null を返した場合は「OK」ボタンイベントがキャンセルされます。

4.6 操作ビュー

操作追加を行うと、操作ビューに追加した操作が表示されます。



ここで表示されるタイトル及びアイコンは初期状態では「Plugin.cs」で指定されたものが使用されます。
入力値によって表示内容を変更したい場合は「PluginRecord.cs」で設定します。

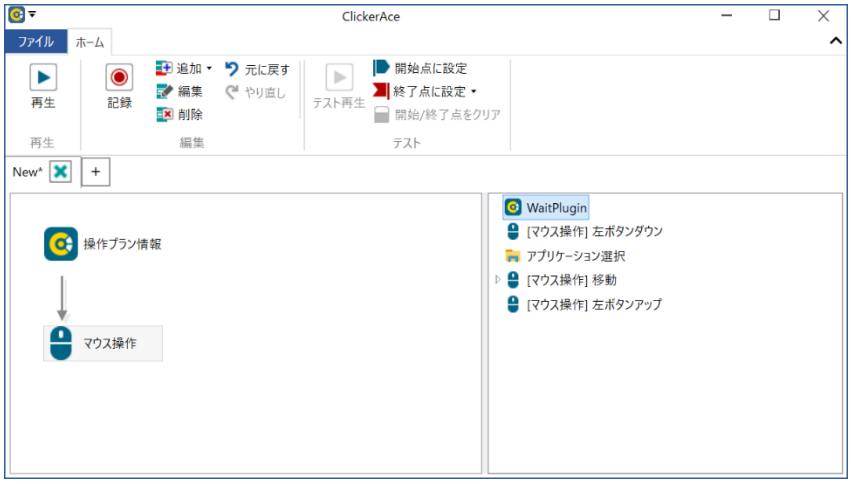
4.6.1 PluginRecord.cs

```
[DataContract]
public class PluginRecord : PluginRecordBase<Plugin> {
    private string _sampleValue;
    [DataMember]
    public string SampleValue {
        get => _sampleValue;
        set => SetProperty(ref _sampleValue, value);
    }
}
```

操作値は直列化して保持される為、保持すべきプロパティ／フィールドには[DataMember]宣言する必要があります。
表示タイトルは「ListTitle」をオーバーライドする事で変更でき、
同様に表示アイコンは「Icon／SmallIcon/LargeIcon」をオーバーライドする事で変更できます。

4.7 操作の編集

操作ビューから編集を行う事で、対応する各プラグインの編集画面が表示されます。



編集画面は「4.44.5 プラグイン操作追加画面」と同じ画面を使用します。

4.7.1 ViewModels/EditViewModel.cs

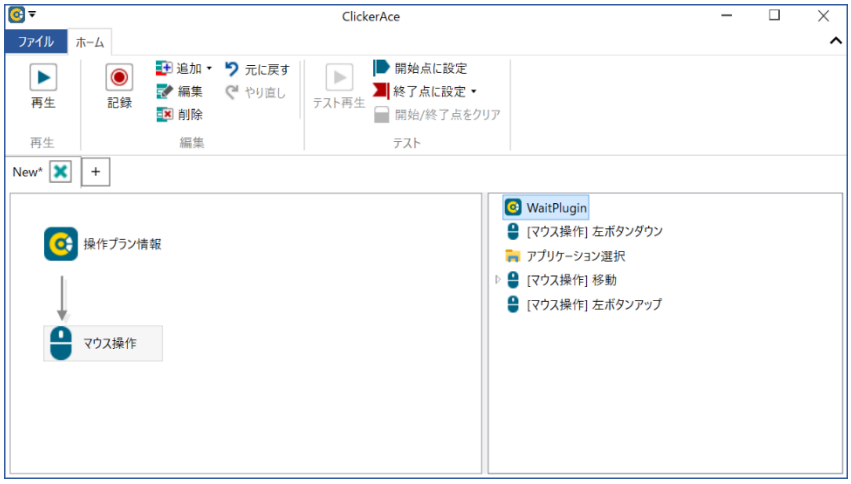
```
public class EditViewModel : EditDialogBaseViewModel<PluginRecord> {
    public EditViewModel() {
    }
    private string _inputValue;
    public string InputValue {
        get => _inputValue;
        set => SetProperty(ref _inputValue, value);
    }
    public override void OnDialogOpened(IDialogParameters parameters) {
        base.OnDialogOpened(parameters);
        if (!IsNew) {
            if (Value == null) throw new ArgumentNullException();
            this.InputValue = Value.SampleValue;
        }
    }
    protected override PluginRecord Apply() {
        var rec = new PluginRecord {
            SampleValue = InputValue
        };
        return rec;
    }
}
```

「OnDialogOpened 関数」ダイアログ表示時の初期値設定を行います。
「IsNew」プロパティで新規／編集モードが判定でき、編集モードだった場合は「Value」プロパティに継承元で指定したデータ型（サンプルでは「PluginRecord」）の値が登録されます。

「Apply 関数」については操作の追加時と同じく、操作時に使用される値を返してください。

4.8 操作の再生

「再生」ボタンを実行する事で、対応する各プラグインの操作が実行されます。



再生時は対応する各プラグインで「Plugin.cs」内の「Action」関数が実行されます。
サンプルプログラムでは入力した値がメッセージダイアログで表示されます。

4.8.1 Plugin.cs

```
public class Plugin : PluginBase<PluginRecord>, IAddablePlugin {
    public override bool Action(IOperateParameter parameter) {
        if (parameter.Record is PluginRecord rec) {
            MessageBox.Show(rec.SampleValue);
        }
        return true;
    }
}
```

「IOperateParameter」の「Record」プロパティに操作値である「PluginRecord」が格納されています。
この操作値を用いて必要な操作処理を行って下さい。
また、戻り値は操作の成否を示す真偽値を返し、false だった場合、再生が停止されます。
※ 今後のアップデートで失敗時操作に遷移機能を実装する予定です。