Input Profile

A Sub-module for Embedded Input Module providing classes for Rebindable Input Mappings.

Optional features and classes require Embedded Input Module.

Input Profile is a C# file providing several abstract clases for defining actions and a base class for the creation of rebindable action maps for projects. Additional classes are provided for use with Embedded Input Module so developers can immediately integrate Input Profiles into their Unity Projects, however, core classes are written to be entirely independent. Input Profile can be decoupled and used as a standalone module for preferred input handlers like InputSystem and Rewired.

Scroll to the end of this document to see how to create custom actions, profiles and a manager.

Current Release: Version 1.1

Overview

Full C# XML Documentation

All classes and members are documented within C#, using inheritance to trickle down to included and new sub-classes.

InputProfile

The base class containing core members and methods for managing and retrieving rebindable actions.

This class is used for creating and managing collections of actions in arrays and lists.

AbstractContextualAction

An abstract class containing the core members for any virtual input and properties to determine the action's input type.

Suitable for defining a collection of actions in an array or list.

BaseContextualAction<TEnum>

An abstract class defining the input source of all actions.

Not suitable for collections of actions in an array or list.

BaseContextualButton<TEnum>

An abstract class defining the Pressed, Held and Released properties of an action bound to a button.

Not suitable for collections of actions in an array or list.

BaseContextualAxis

An abstract class defining the Vector2 value of any of the available input source axes.

Suitable for defining a collection of actions in an array or list.

Embedded Input Module Classes (Optional)

ContextualKey

A Virtual Button representing a key on a Keyboard, Mouse or Gamepad.

ContextualGamepad

A Virtual Button representing a control on a Gamepad.

ContextualAxis

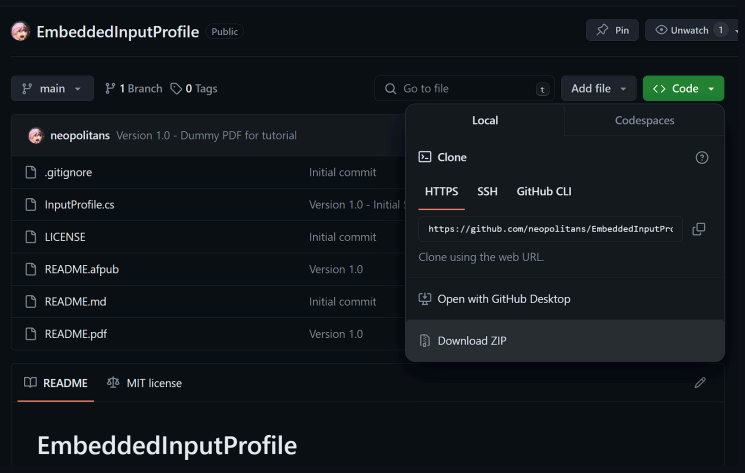
A virtual axis that represents a multi-directional control through a Joystick, Mouse or set of 4 buttons.

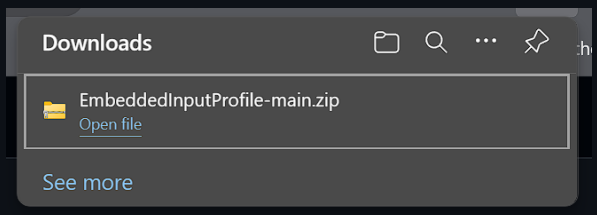
PlatformInputProfile

A class containing a list of virtual inputs for player actions with an identifier for a specific platform.

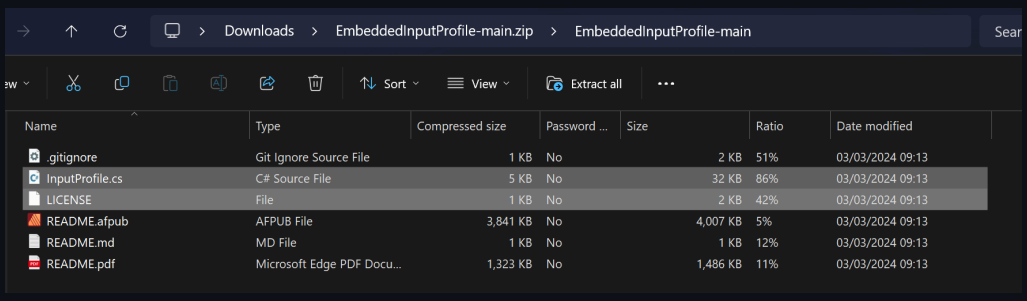
Installation

[**Step 1**] Download the GitHub repository, which will be a .zip file of all files and folders contained within.



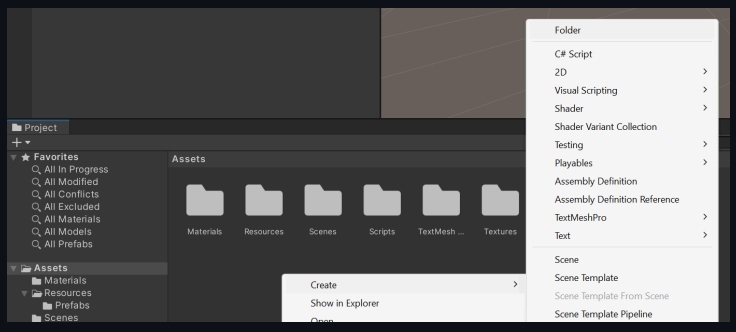


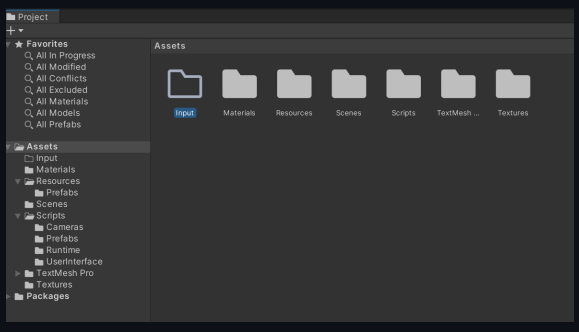
[**Step 2**] Open (or Extract) the .zip file, go into the folder then copy InputProfile.cs and the LICENSE file.



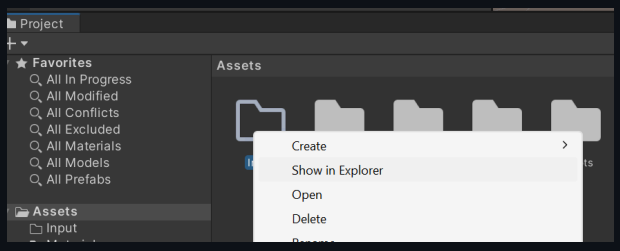
[**Step 3**] Go to the "Project" tab in Unity and create a new folder called "**Input**"

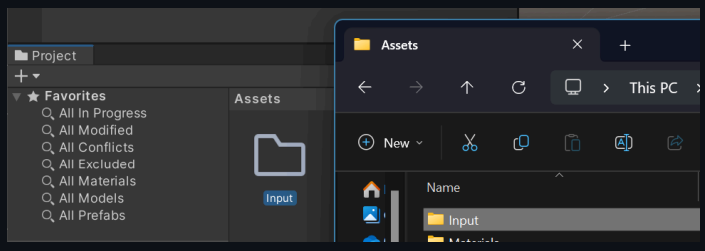
Both files **can** be copied directly into Assets or any folder with a preferred name, however some folders can be created with names which will not get included when you build your game. A detailed list can be found here.

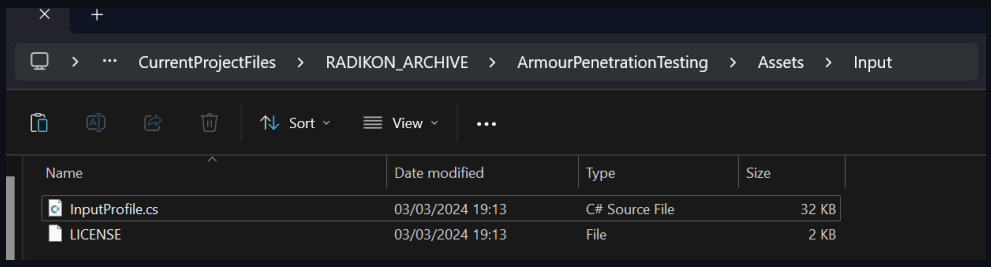




[**Step 4**] Open the new folder by highlighting it, clicking Show in Explorer and paste the copied files into the folder.







Implementing Actions and Profiles

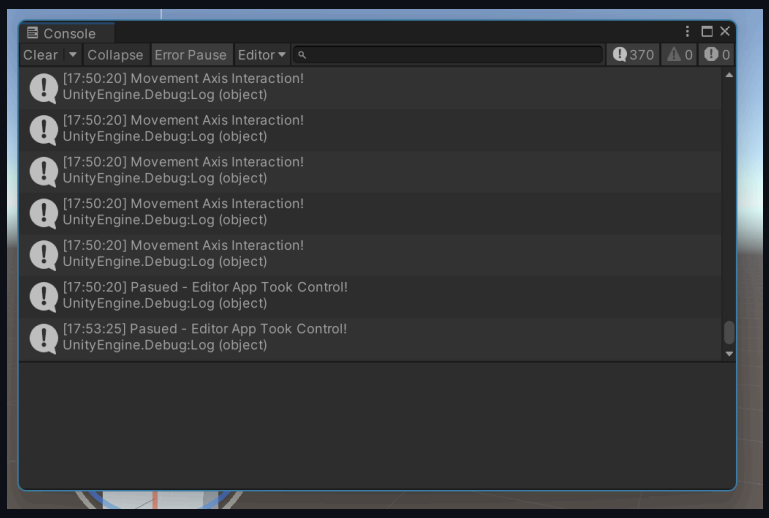
Actions and Input Profiles only require two steps for implementation. An object with the action mappings accessible to scripts and the scripting in relavent areas to retrieve action values. Most action classes are defined by the developer. This quick guide assumes that Embedded Input Module is being used, or new classes have been created/found for a preferred input handler.

[**Step 1**] Define an Input Profile at the top of a Script or within an

```
Unity Script | 0 references
public class TestingInputProfile : MonoBehaviour
{
    static InputProfile controls = new InputProfile(
        // ContextualAxis supports a 4-button combiantion, the mouse axis or Left/Right stick for gamepads.
        new ContextualAxis("Movement", KeyCode.D, KeyCode.A, KeyCode.W, KeyCode.S),
        new ContextualAxis("GamepadMovement", AvailableInputAxes.LeftStick),

        // Contextual Key supports KeyCodes, even Gamepad and Mouse 0-4 KeyCodes.
        new ContextualKey("Exit", KeyCode.Escape),

        // Contextual Gamepad Button Supports most standard gamepad buttons.
        new ContextualGamepadButton("GamepadExit", EmbeddedInputModule.GamepadControl.ButtonSouth)
        );
}
```

[**Step 2**] GetButton, GetButtonDown, GetButtonUp and GetAxis can be called from the Input Profile to read values of actions.
GetButton methods return a boolean, while GetAxis returns a Vector2.

```
Unity Script | 0 references
public class TestingInputProfile : MonoBehaviour
{
    static InputProfile controls =...

    Unity Message | 0 references
    public void Update()
    {
        if (controls.GetAxis("Movement").magnitude > 0.05f)
            Debug.Log("Movement Axis Interaction!");

        if (controls.GetAxis("GamepadMovement").magnitude > 0.05f)
            Debug.Log("Left Stick Axis Interaction!");

        if (controls.GetButtonDown("Exit") || controls.GetButtonDown("GamepadExit"))
        {
            #if UNITY_EDITOR
            EditorApplication.isPaused = !EditorApplication.isPaused;
            Debug.Log("Pasued - Editor App Took Control!");
            #else
            Application.Quit();
            #endif
        }
    }
}
```



It may be prefereable to have a separate, static class as the Input Manager in place of Unity Engine's original Input Manager. A separate static class may also provide fixed method calls to a changing reference to a "current" Input Profile dependent on the identified device platform.

Developers may also prefer to expand Actions further for unique input devices, such as racing wheels and joysticks for any simulation project. However, extra steps will be required for the action to be identified and to return it's input state.

Classes

New Abstract Class - AbstractContextualAction

As the core class containing members, properties and methods for any action, this is provided to facilitate collections of any Inheritng type. No input-reading logic is present, however, this is suitable for inheritance if developers are only concerned with collecting custom actions in a base InputProfile.

Members

	Description
AbstractContextualAction.label	The name that identifies the Action. Used to query the action during runtime.
AbstractContextualAction.onRebind	The delegate called when the Action is rebound during runtime.
AbstractContextualAction.isAxis	Does the type of the Action match that of a Virtual Axis?
AbstractContextualAction.isButton	Does the type of the action match that of a Virtual Button?

Methods

	Description
AbstractContextualAction.Clone	Creates a duplicate of the action. This will provide the data of the current instance to the class' constructor.

New Abstract Class - BaseContextualAction<TEnum> (TEnum is any type of enum class)

Inheritance: **AbstractContextualAction**

The first step of inheritance for most classes in this module, BaseContextualAction provides the ability to define any type of input source with an enumerator. This could store the direct source of an input or be a category defining input types.

Members

	Description
BaseContextualAction.inputSource	The source of an action's input values. An example being a KeyCode value, returning the respective button's state.

New Abstract Class - BaseContextualButton<TEnum> (TEnum is any type of enum class)

Inheritance: **BaseContextualAction<TEnum>, IDeviceInputButton**

Specifically for Keys, Gamepad Buttons or other inputs which return their state in a boolean form, BaseContextualButton contains the three common states of any virtual button. This leverages the generic typing of BaseContextualAction so that inputSource will share the type of the enumerator provided such as a KeyCode if UnityEngine.KeyCode is used.

Members

	Description
BaseContextualButton.Pressed	Whether the virtual button is pressed.
BaseContextualButton.Released	Whether the virtual button is released.
BaseContextualButton.Held	Whether the virtual button is held.

New Abstract Class - BaseContextualAxis

Inheritance: **BaseContextualAction<AvailableInputAxes>**

A type of Action requiring developers to provide logic for getting a Vector2 from a joystick, mouse or virtual axis of buttons.

Members

	Description
BaseContextualAxis.Value	A Vector2 directly provided from a joystick, mouse or virtual axis of buttons.

New Class - InputProfile

A class that contains a list of virtual inputs for player actions. This is suitable for collections in arrays and lists, if desired.

Methods

	Description
InputProfile.actions	The actions contained in the profile which can be queried.

Methods

	Description
InputProfile.GetButton	Returns true while the virtual button identified by the given name is held down.
InputProfile.GetButtonUp	Returns true during the first frame the user releases the virtual button identified by the given name.
InputProfile.GetButtonDown	Returns true during the first frame the user pressed down the virtual button identified by the given name.
InputProfile.GetAxis	Returns the value of the virtual axis identified by the given name. The values will be in the range of -1 to 1 for keyboard, joystick and mouse input devices.
InputProfile.Clone	Create a duplicate of the Input Profile by creating duplicates of each action within and assigning them to the duplicate instance.

Constructors

	Description
InputProfile(AbstractContextualAction[])	A constructor for InputProfile that takes any number of AbstractContextualActions. (uses **params** to make it possible to have any amount of values)

This class can be useful for creating a default class and duplicating it to make a second instance for the runtime of a game or appliaction. Such behaviour would provide a default that can be safely duplicated again and a profile which can incorporate rebindable controls safely. However, excessive copying without properly disposing an old instance may cause a memory leak.

New Interface - IDeviceInputButton

An interface which requires implementing classes to provide properties for getting the Held, Pressed and Released states of a virtual button. This interface is also used to identify if an Action is a type of button.

This is a globally accessible Interface. It is marked as internal to denote it's intended use as an internal type-casting utility.

Members

	Description
IDeviceInputButton.Pressed	Whether the virtual button is pressed.
IDeviceInputButton.Released	Whether the virtual button is released.
IDeviceInputButton.Held	Whether the virtual button is held.

New Enums

	Description
AvailableInputAxes	An enum as a list of basic supported input sources for a virtual axis. This list is incomplete and can be expanded if you do not wish to create more enumerator types. • LeftStick • RightStick • MouseAxis • ButtonAxis (4-Button Virtual Axis)

Optional Subclasses (Requires Embedded Input Module)

These classes inherit from BaseContextualButton, BaseContextualAxis and InputProfile, using a handful of features from Embedded Input Module. These can also be modified to work with other input handlers if desired.

New Class - ContextualKey

Inheritance: **BaseContextualButton<UnityEngine.KeyCode>**

A virtual button representing a key on a Keyboard, Mouse or Gamepad.

Members

	Description
ContextualKey.altInputSource	An optional alternate input source used for the virtual button's input state.

Methods

	Description
ContextualKey.Clone	Constructs a duplicate virtual button using the most detailed constructor available.

Constructors

	Description
ContextualKey(string, KeyCode)	A constructor for a virtual button that takes a label and a KeyCode as the primary input source.
ContextualKey(string, KeyCode, System.Action)	A constructor for a virtual button that takes a label, a KeyCode as the primary input source and a delegate method to bind to the callback invoked when a key is rebound.
ContextualKey(string, KeyCode, KeyCode)	A constructor for a virtual button that takes a label and two KeyCodes. The first as the primary input source and the second as the alternate input source.
ContextualKey(string, KeyCode, KeyCode, System.Action)	A constructor for a virtual button that takes a label, two KeyCodes and a delegate method to bind to the callback invoked when a key is rebound. The first KeyCode is the primary input source and the second is the alternate input source.

New Class - PlatformInputProfile

Inheritance: **InputProfile**

A class containing a list of virtual inputs for player actions with an identifier for a specific platform.

Members

	Description
PlatformInputProfile.platform	The platform that this Input Profile is created for.

Methods

	Description
PlatformInputProfile.Clone	Create a duplicate of the Input Profile by creating duplicates of each action within and assigning them to the duplicate instance.

Constructors

	Description
PlatformInputProfile(InputIconDisplayType, AbstractContextualAction[])	A constructor for PlatformInputProfile that takes a target platform and any number of AbstractContextualActions. (uses **params** to make it possible to have any amount of values)

This class does have limited use-cases and exists mostly as an example, however, developers may find some utility in associating a specific platform to a profile in a collection of profiles. Developers may also find utility in having a specific platform associated with a profile to prevent a mis-matched input profile and input device issue.

New Class - ContextualGamepadButton

Inheritance: **BaseContextualButton<EmbeddedInputModule.GamepadControl>**

A virtual button representing a button on a Gamepad. As gamepads have limited buttons, there is only one input source.

Methods

	Description
ContextualGamepadButton.Clone	Constructs a duplicate virtual button using the most detailed constructor available.

Constructors

	Description
ContextualGamepadButton(string, GamepadControl)	A constructor for a virtual button that takes a label and a GamepadControl as the input source.
ContextualGamepadButton(string, GamepadControl, System.Action)	A constructor for a virtual button that takes a label,a GamepadControl as the input source and a delegate method to bind to the callback invoked when a key is rebound.

New Class - ContextualAxis

Inheritance: **BaseContextualAxis**

A virtual axis that rerpesents a multi-directional control from a joystick, Mouse or set of 4 buttons.

Methods

	Description
ContextualAxis.positiveX	The KeyCode for positive X.
ContextualAxis.negativeX	The KeyCode for negative X.
ContextualAxis.positiveY	The KeyCode for positive Y.
ContextualAxis.negativeY	The KeyCode for negative Y.
ContextualAxis.altPositiveX	The optional alternate KeyCode for positive X.
ContextualAxis.altNegativeX	The optional alternate KeyCode for negative X.
ContextualAxis.altPositiveY	The optional alternate KeyCode for positive Y.
ContextualAxis.altNegativeY	The optional alternate KeyCode for negative Y.

Constructors

	Description
ContextualAxis(string, AvailableInputAxes)	A constructor for a virtual axis that reads from an input axis or a blank button axis.
ContextualAxis(string, KeyCode, KeyCode, KeyCode, KeyCode)	A constructor for a virtual axis that reads from a completed 4-button axis.
ContextualAxis(string, AvailableInputAxes, System.Action)	A constructor for a virtual axis that reads from an input axis or a blank button axis with a callback triggered when the action is rebound.
ContextualAxis(string, KeyCode, KeyCode, KeyCode, KeyCode, System.Action)	A constructor for a virtual axis that reads from a completed 4-button axis with a callback triggered when the action is rebound.
ContextualAxis(string, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode)	A constructor for a virtual axis that reads from a completed 4-button axis, with 4 alternate buttons for each direction.
ContextualAxis(string, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode, KeyCode, System.Action)	A constructor for a virtual axis that reads from a completed 4-button axis, with 4 alternate buttons for each direction.

For the brevity of this document 4 constructors have been omitted from the list. These use GamepadControl, however, they convert that value into a KeyCode value. Such constructors may not be of use, especially as EmbeddedInputModule.Dpad exists, however they are provided for convenience over the numerous KeyCode values for joystick buttons.

Alternate Axes are not advised for virtual axes constructed with Gamepad Controls.

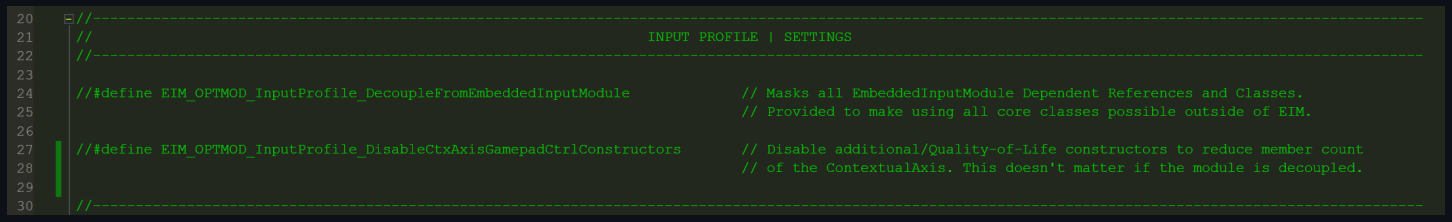
Configurable Settings

A few configurable settings exist inside Input Sequence. Each setting uses preprocessor directives within C# in order to switch between desired features and, where applicable, relevant restricted substitutes for features. Compared to the more in-depth settings in Embedded Input Module, there are only three settings for Input Sequence.

Enabling or Disabling Settings

To enable or disable each setting, open up the InputProfile.cs script from where you have it stored in your Unity Project and navigate to Line 21, which is the start of the header for the Settings section.

Within the section is a list of settings, each with their own description. The difference between enabled and disabled settings are in the colour of the text at the start of the line.



```
20  //------------------------------------------------------------------------------------------------------------------------
21  //                                                INPUT PROFILE | SETTINGS
22  //------------------------------------------------------------------------------------------------------------------------
23
24  //#define EIM_OPTMOD_InputProfile_DecoupleFromEmbeddedInputModule            // Masks all EmbeddedInputModule Dependent References and Classes.
25                                                                                 // Provided to make using all core classes possible outside of EIM.
26
27  //#define EIM_OPTMOD_InputProfile_DisableCtxAxisGamepadCtrlConstructors      // Disable additional/Quality-of-Life constructors to reduce member count
28                                                                                 // of the ContextualAxis. This doesn't matter if the module is decoupled.
29
30  //------------------------------------------------------------------------------------------------------------------------
```

To **Enable** a disabled setting, remove the two forward-slashes preceding the corresponding #define.
To **Disable** an enabled setting, add two forward-slashes preceding the corresponding #define.

List of Settings

EIM_OPTMOD_InputProfile_DecoupleFromEmbeddedInputModule

Masks **ContextualKey**, **ContextualGamepadButton**, **ContextualAxis** and **PlatformInputProfile** from inside the script.

If developers seek to use their own input handler or otherwise do not seek to use Embedded Input Module over Unity's solutions, it is advised this setting is Enabled. Developers will not have to remove any extra code with this setting enabled and may move over to Embedded Input Module later if desired.

Default: Disabled

Location: Line 24

EIM_OPTMOD_InputProfile_DisableCtxAxisGamepadCtrlConstructors.

Disables the additional constructors provided for the optional class ContextualAxis, which take GamepadControls instead of KeyCodes. This is recommended to be enabled if the amount of constructor variations are causing issues with intellisense or if the additional constructors go unused.

Disabling these constructors may have little impact outside of compilation time as a slight micro-optimization.

Default: Disabled (Don't bother changing this if DecoupleFromEmbeddedInputModule is enabled.)

Location: Line 27

A Quick Note on Class Naming

Due to the presence of InputActions and InputActionMaps in InputSystem, the naming of Actions and Action Mappings as "ContextualActions" and "InputProfiles" came as a means to prevent any errors and overlapping classes. These are designed to work for **any** input handler, while InputSystem may not work with other input handlers.

The behaviour of these classes perform similarly, however, they are designed to work similarly to the way UnityEngine.Input does. Input Profiles are also meant to provide the members missing from Embedded Input Module, related to Unity Engine's Input Manager found in [**Edit** > **Project Settings**].

Creating a Custom Input Sequence

For Developers using custom input handlers, using legacy UnityEngine.Input as an example

Developers may have preferred input handlers they are familiar with. Modules designed for Embedded Input Module seek to, where possible, be fully compatible with more contemporary and off-the-shelf solutions to handling game input. However, to demonstrate the flexibility of Input Sequence, this section demonstrates the requirements of a custom input handler and how to create a custom sequence class that uses the legacy **UnityEngine.Input** class.

A Custom Input Handler must provide the following for the Controller, Keyboard, Mouse or Peripheral intended for a class:

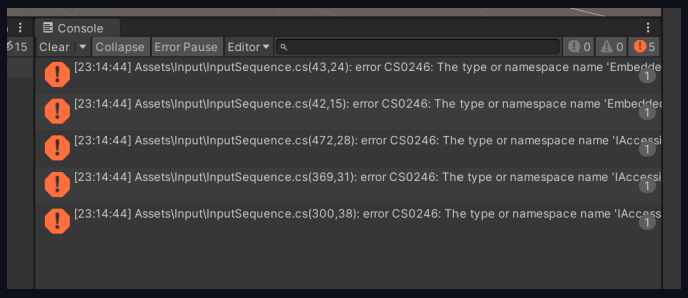
- An enumerator (that inherits from or directly is a System.Enum type)
- A method or property to access whether an input is pressed this frame (or held)

This comprehensive guide contains all core step so beginners can follow along.

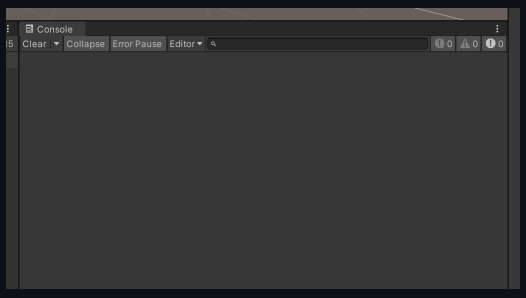
Sorry for any inconvenience in advance.

[**Step 0**] (Optional) Enable the Decoupling from Embedded Input Module setting.

If there are errors like this, then decoupling is necessary.



There should only be messages related to your project.

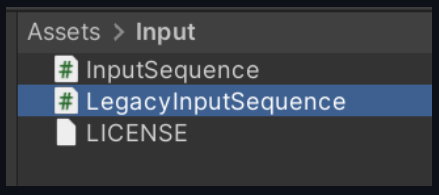



```
//-------------------------------------------------------------------------------------------------------------------------------
//                                                  INPUT SEQUENCE | SETTINGS
//-------------------------------------------------------------------------------------------------------------------------------

#define EIM_OPTMOD_InputSequence_DisableDebugMessages                   // Disables debug messages for Successful Inputs, Unsuccessful Inputs,
                                                                        // Sequenece completion, current indexes and next inputs for Dev Debugging.


#define EIM_OPTMOD_InputSequence_DecoupleFromEmbeddedInputModule        // Masks all EmbeddedInputModule Dependent References and Classes.
                                                                        // Provided to make using AbstractInputSequence<T> with any preferred input
                                                                        // handler easier.

#define EIM_OPTMOD_InputSequence_CheckGamepadInputsForKeyboardSequence  // Check gamepad inputs for each GamepadControl value in KeyboardSequence when
                                                                        // a non-sequential input is detected during UpdateSequence().

//-------------------------------------------------------------------------------------------------------------------------------
```

Settings Used For Step 1 Example

[**Step 1**] Create a new script in the installation folder (or other preferred location).

LegacyInputSequence is used here, however any name that isn't InputSequence should work fine.




```
Unity Script | 0 references
public class LegacyInputSequence : MonoBehaviour
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()...

    // Update is called once per frame
    Unity Message | 0 references
    void Update()...
}
```

[**Step 2**] Change **MonoBehaviour** to **InputSequence<KeyCode>** and remove the pre-generated methods.

```
0 references
public class LegacyInputSequence : InputSequence<KeyCode>
{

}
```

[**Step 3**] Create a new method called **UpdateSequence()** with the override keyword, this will contain the only custom logic required to maintain the input sequence.

```
0 references
public class LegacyInputSequence : InputSequence<KeyCode>
{
    5 references
    public override void UpdateSequence()
    {
        base.UpdateSequence();
    }
}
```

[**Step 4**] Add a check to see if the sequence is already complete before returning, followed by a variable that contains the state of the current key in the sequence.

As sequence is an array of KeyCode values, sequence[current] will return the KeyCode at the current index.

```
0 references
public class LegacyInputSequence : InputSequence<KeyCode>
{
    4 references
    public override void UpdateSequence()
    {
        // Has this sequence been completed? If so, exit.
        if (sequenceComplete) { return; }

        // Store whether the current key being checked in the sequence is pressed down or not.
        bool CurrentKey = Input.GetKeyDown(sequence[current]);
    }
}
```

[**Step 5**] Add an if statement based on the result of that current key.

Inside, set the value at successfulInputs[**current**] to true.

Then check if adding 1 to **current** will take the value over the length of the sequence.

If it doesn't, the value of **current** should increment by 1.

Otherwise, call **SetAsComplete()** to complete the sequence.

```
// Store whether the current key being checked in the sequence is pressed down or not.
bool CurrentKey = Input.GetKeyDown(sequence[current]);

// If an input is successful, set the current index to true.
//      If that input doesn't complete the sequence, go to the next index.
//      Otherwise, mark the sequence as complete.
if (CurrentKey)
{
    successfulInputs[current] = true;

    if (current + 1 < sequence.Length) { current++; }
    else SetAsComplete();
}
```

[**Step 6**] Append an else statement which checks if Input.anyKeyDown is true.

If it is, call **ResetSequence()** to reset the sequence.

```
// If an input is successful, set the current index to true.
//      If that input doesn't complete the sequence, go to the next index.
//      Otherwise, mark the sequence as complete.

// If an input isn't successful, check if there have been any other inputs.
//      If there have been, reset the sequence.

if (CurrentKey)
{
    successfulInputs[current] = true;

    if (current + 1 < sequence.Length) { current++; }
    else SetAsComplete();
}
else
{
    if (Input.anyKeyDown) { ResetSequence(); }
}
```

[**Step 7**] Create a new constructor for the class that takes an array of KeyCode values.

Set **sequence** to be that array.

Set **successfulInputs** to be a new array of boolean values equal to the length of **sequence**.

This example uses the 'params' keyword, which avoids the need to explicitly create an array by hand.

```
3 references
public class LegacyInputSequence : InputSequence<KeyCode>
{
    1 reference
    public LegacyInputSequence(params KeyCode[] keys)
    {
        // Set sequence to be the array of keys.
        sequence = keys;

        // Create a new array of bools of the same size as sequence.
        successfulInputs = new bool[sequence.Length];
    }

    4 references
    public override void UpdateSequence()...
}
```

Developers may wish to add various comments throughout the script to indicate the current status of sequences, some example debug messages can be found in the provided base classes and sub-classes.

Testing the New Class

[**Step 1**] In an existing or new script, define a variable at the top that is of the Sequence Class just created.

Use any preferred set of inputs. The directional inputs of the Konami Code are used here as it's well known.

This example used 'LegacyInputSequence', so that name is used here.

```
Unity Script | 0 references
public class TestingLIS : MonoBehaviour
{
    LegacyInputSequence legacy =
        new LegacyInputSequence(
            KeyCode.UpArrow, KeyCode.UpArrow,
            KeyCode.DownArrow, KeyCode.DownArrow,
            KeyCode.LeftArrow, KeyCode.RightArrow,
            KeyCode.LeftArrow, KeyCode.RightArrow
        );

    Unity Message | 0 references
    void Start() { }

    Unity Message | 0 references
    void Update()
    {

    }
}
```

[**Step 2**] Call Update Sequence in the Update() method of the testing script.

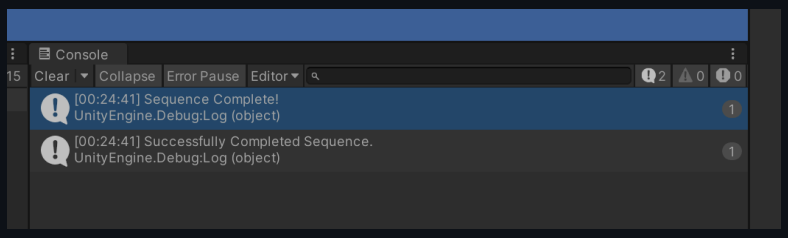
Then check if the variable is equal to true. This works because InputSequence has a custom true/false operator.

Lastly, if true, display a debug message and reset the sequence.

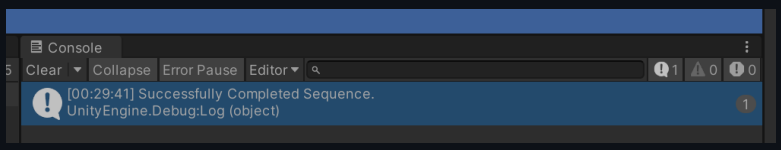
```
Unity Message | 0 references
void Update()
{
    // Keep updating the sequence.
    legacy.UpdateSequence();

    // If the sequence is complete, display a success message
    // then reset the message.
    if (legacy) {

        Debug.Log("Successfully Completed Sequence.");
        legacy.ResetSequence();
    }
}
```

[**Step 3**] Check to see if both scripts are saved, then run your project and perform the sequence.

If Debug Messages are enabled, two or more debug messages should appear:



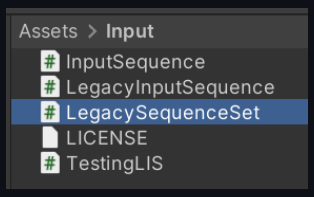
Otherwise only one debug message will appear:



Creating a Custom Sequence Set

For Developers using built-in or custom sequences.

[**Step 1**] Create a new script in the installation folder (or other preferred location) and remove the pre-generated methods.




```
using UnityEngine;

Unity Script | 0 references
public class LegacySequenceSet : MonoBehaviour
{

}
```

[**Step 2**] Change **MonoBehaviour** to **AbstractSequenceSet**.

```
0 references
public class LegacySequenceSet : AbstractSequenceSet
{

}
```

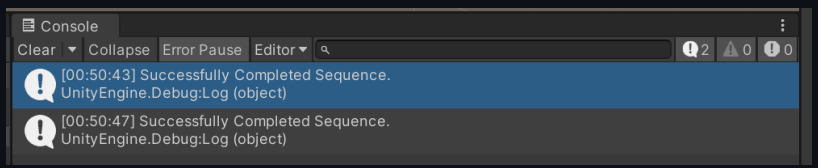
[**Step 3**] Create a new constructor for the class that takes an array of **AbstractInputSequence** objects.

Set **sequences** to be that array.

```
1 reference
public class LegacySequenceSet : AbstractSequenceSet
{
    0 references
    public LegacySequenceSet(params AbstractInputSequence[] inputSequences)
    {
        sequences = inputSequences;
    }
}
```

[**Step 4**] (Optional) If using the testing script from the previous section or a new script, define a variable at the top that is of the Sequence Set just created and test both sequences using the prior steps. The results should be identical.

```
LegacySequenceSet legacy = new LegacySequenceSet(
    new LegacyInputSequence(
        KeyCode.UpArrow, KeyCode.UpArrow,
        KeyCode.DownArrow, KeyCode.DownArrow,
        KeyCode.LeftArrow, KeyCode.RightArrow,
        KeyCode.LeftArrow, KeyCode.RightArrow
    ),
    new LegacyInputSequence(
        KeyCode.W, KeyCode.W,
        KeyCode.S, KeyCode.S,
        KeyCode.A, KeyCode.D,
        KeyCode.A, KeyCode.D
    ));
```



As **AbstractSequenceSet** is near identical to **AbstractInputSequence**, yet doesn't require logic specific to any input handler, it was able to be contained entirely within the abstract class. This allows any developer to use, modify and extend sets easily.