EmbeddedInputModule

A module built with Unity Engine's new InputSystem to mimic the legacy UnityEngine.Input module.

This REQUIRES Unity's new InputSystem package to be in your project.
Instructions on how to do this are in <u>Installation</u>.

EmbeddedInputModule is a single C# file which takes as much of the structure and names of members within the now legacy UnityEngine.Input class (henceforth Legacy Input Module), building functionality of each member within back up using Unity Technologies' new InputSystem. It includes all the documentation and is (hopefully) a direct drag-and-drop replacement for the old system in most cases. This module seeks to provide a quick, fast and well-documented means of still using the same members and methods that those developers are used to, while leveraging all the benefits of the new system where possible.

This module started life during my time at University, where I was working on one of my Advanced Games Programming tasks. It exists because the thought of **"*wouldn't it be cool if I could just switch controllers right in the middle of gameplay?*"** had occured during one of several tests of the camera system. During which, there were attempts to use the InputManager, to no avail. Upon seeing that InputSystem had controller support and some hotplugging features for at the very least switching controllers, that was attempted.

However, during attempts to use InputSystem, the InputActions, InputMaps and Editor Window for managing the InputSystem package felt dissatisfying compared to the C# approach which I had been much more accustomed to and longed to return to. For a week straight afterwards in November, this module saw an extremely fast development cycle to be what it is now with the goal of facilitating the ability to quickly identify and hot-swap controllers even when multiple are plugged in.

Features

EmbeddedInputModule's API is, almost entirely, that of the Legacy Input Module's API. Almost everything within Unity Engine's Scripting API should work as expected and the available members are listed in ***Compatibility with Legacy Input Module***. For any cases where the API isn't working as intended, please create an Issue on the repository.

Developers may find use in these members and methods:

EmbeddedInputModule.PlatformIcons

A value of the type EmbeddedInputModule.InputIconDisplayType which can be used by developers to describe the control scheme icons to be used based on what platform the player is currently on.

Current Device Support: [PlayStation | Xbox | Switch | PC]
Tested through a University Project - (Radikon War: Lone Warrior)

EmbeddedInputModule.CurrentGamepad

A value of the type EmbeddedInputModule.DeviceIdentifier which points to the current gamepad being used for input.

EmbeddedInputModule allows, and encourages, the incorporation of hotswappable gamepads during gameplay so that players can have the most choice possible.

EmbeddedInputModule.DeviceIdentifier

The wrapper class used to identify and categorize Gampads and other Input Devices. Can be further expanded with new functionality, however, this was scaled back for the initial release.

Full C# XML Documentation

All documentation found on Unity Technologies' Scripting API for UnityEngine.Input and within this readme is included in the C# script as summaries and documentation for each

For more information on what is included, scroll to the ***Compatibility with Legacy Input Module*** and ***Additions*** sections.

Installation

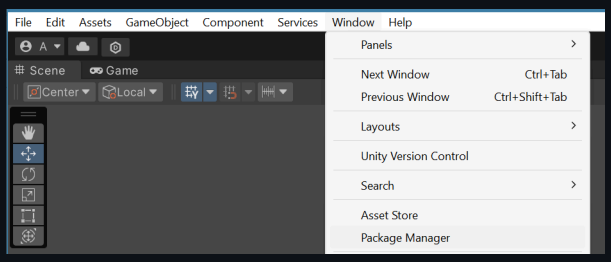
This installation guide is written to be completed from a ground-up perspective so that no step is missed and that any user of any experience with Game Engines, scripts and/or GitHub may be able to follow along at their own pace.

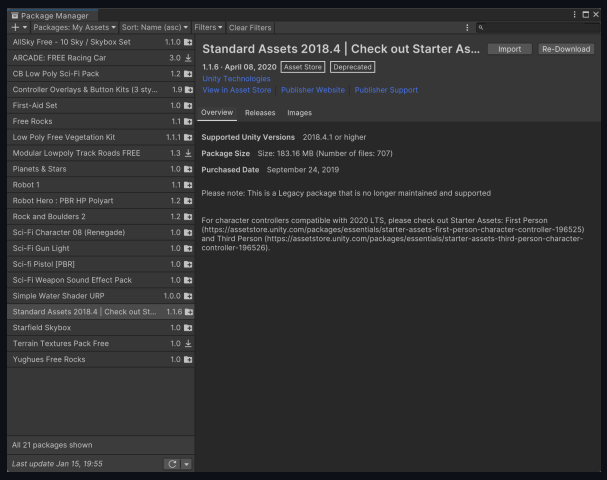
Install the new InputSystem Package

If you already have the InputSystem package, skip this subsection.

[**Step 1**] Open Window → Package Manager

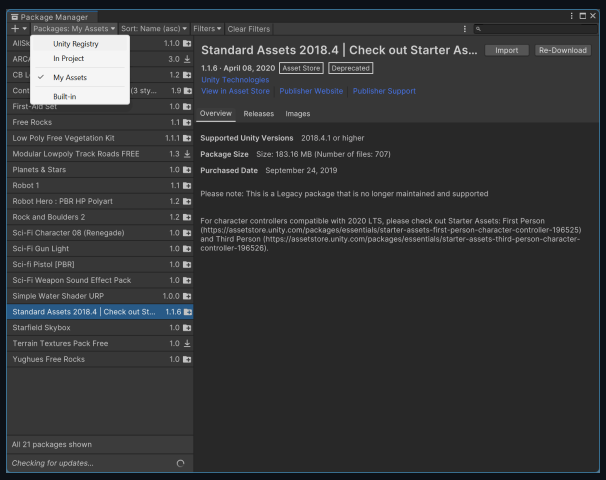
The window on the right will appear.

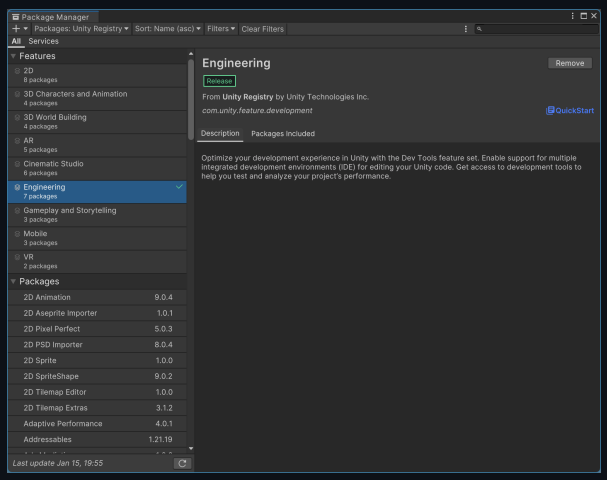




[**Step 2**] Change the Packages option from **My Assets** to **Unity Registry**

My Assets will show any assets that have been bought from Unity Asset Store.
Unity Registry will show any additional systems and packages available for the editor.

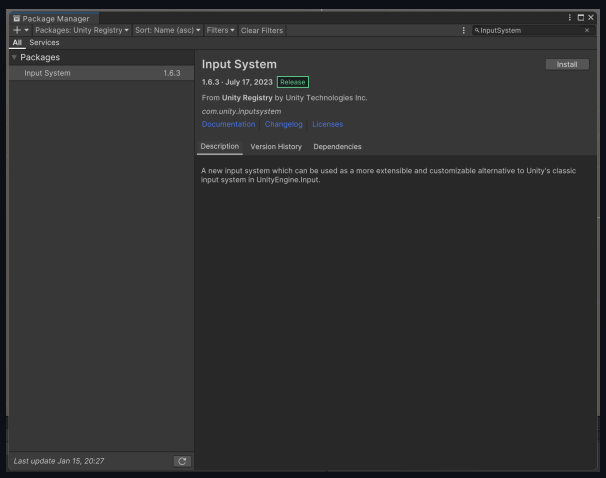


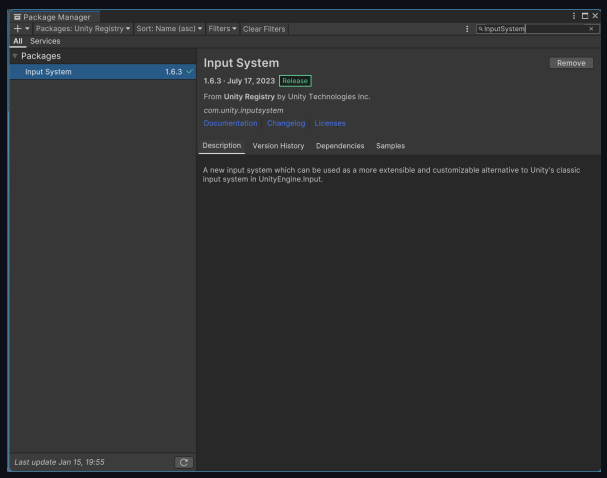


[**Step 3**] Search for **InputSystem** in the top-right search bar.

For users who haven't got the InputSystem package, the button on the right will say "Install".

Otherwise, the button on the right will say "Remove", in this case go to ***Installing the Embedded Input Module***

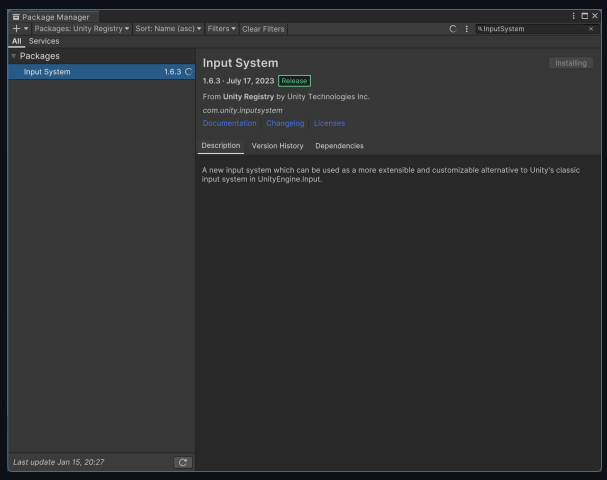


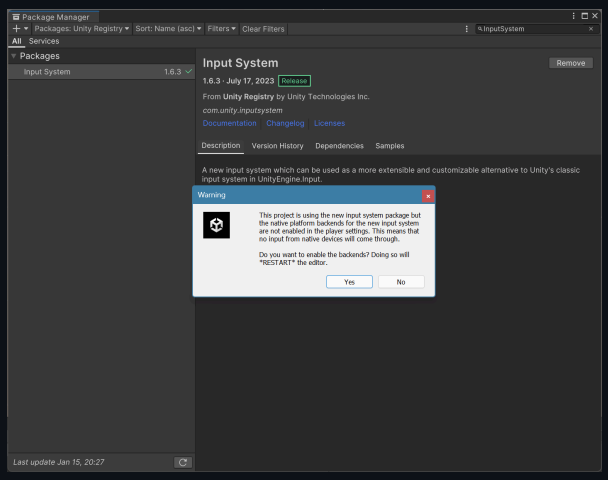


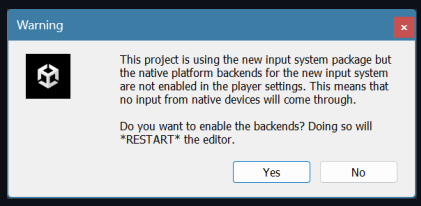
[**Step 4**] Install the **InputSystem** package.

Clicking Install will start downloading the package and a progress icon will appear on the package's title in the list and near the Search bar for a short while.

Once the package has finished downloading, a prompt will appear asking to restart the editor to enable the new InputSystem. Click **Yes**, which will re-compile the project and close the editor down afterwards. Once

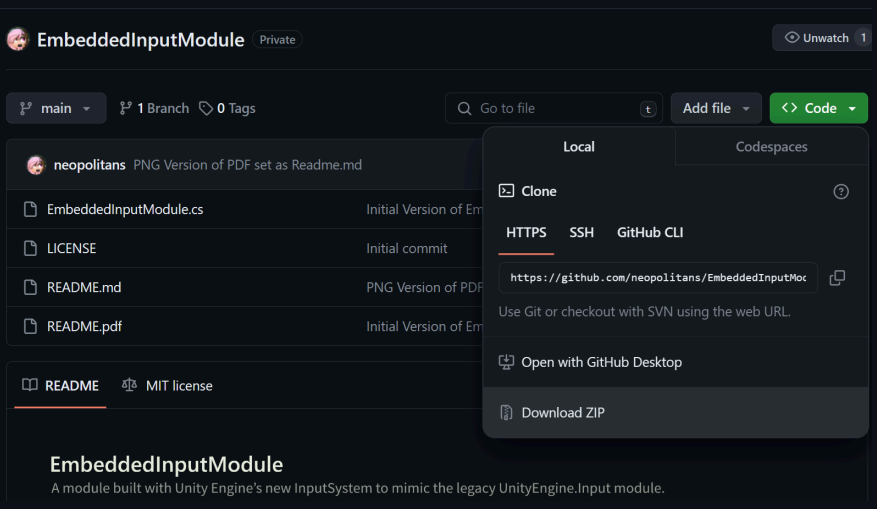


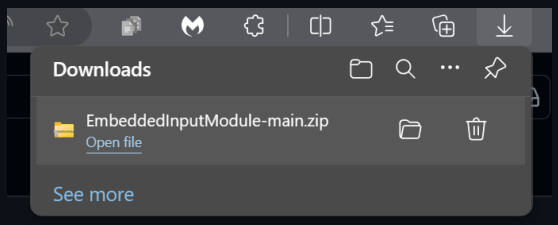




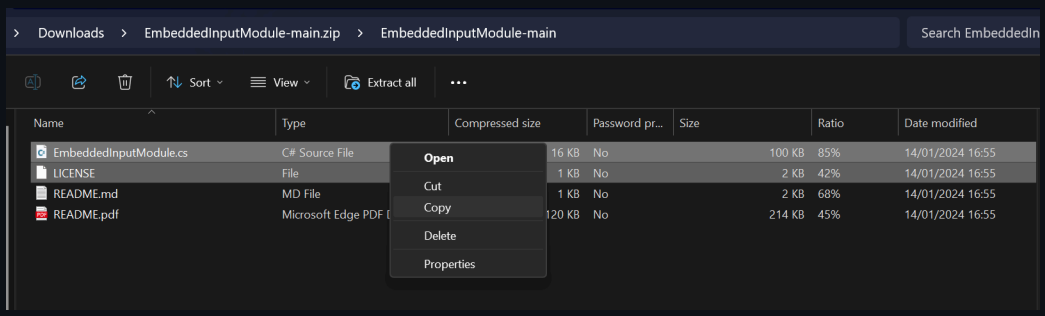
"Installing" the Embedded Input Module

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



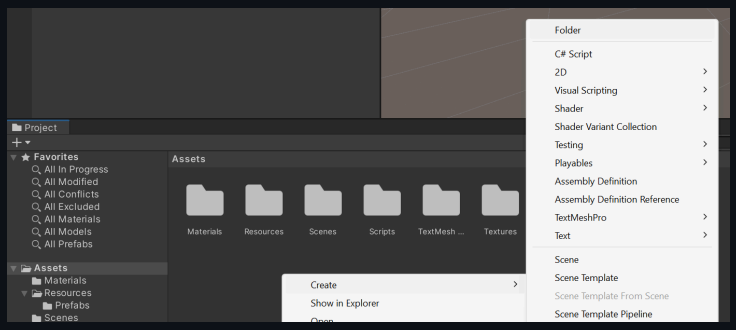


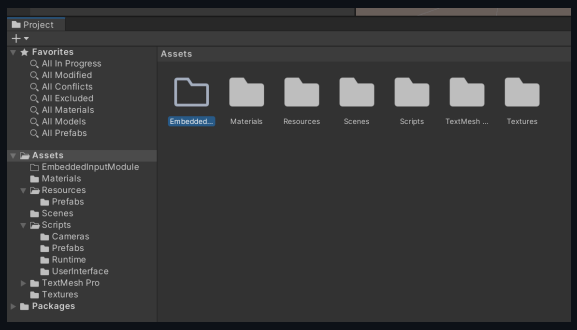
[**Step 2**] Open (or Extract) the .zip file, go into the folder then copy EmbeddedInputModule.cs and the LICENSE file.



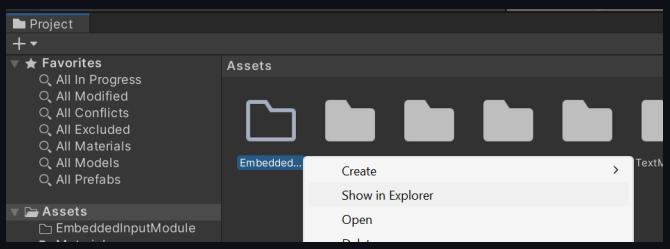
[**Step 3**] Go to the “Project” tab in Unity and create a new folder called “**EmbeddedInputModule**”

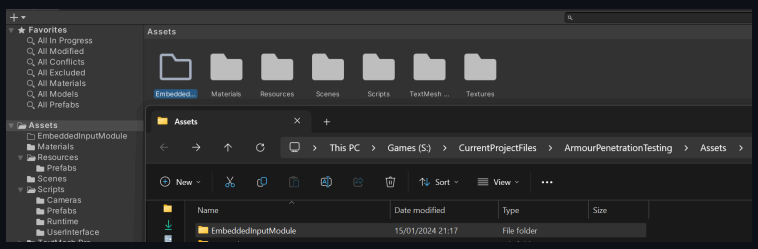
Both files **can** be copied directly into Assets or any folder with a preferred name, however some folders can be created with names which will not get included when you build your game. A detailed list can be found here.

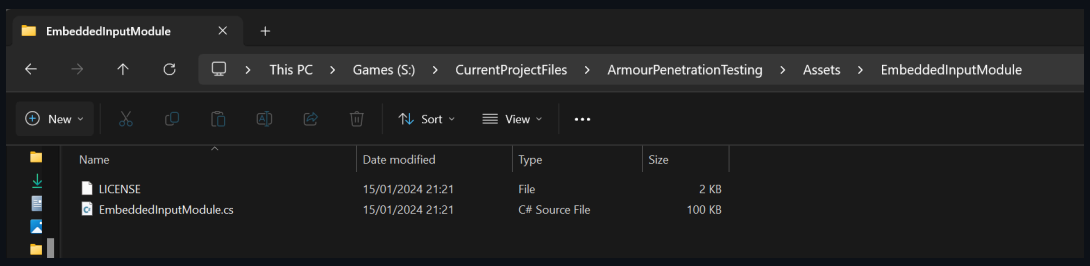




[**Step 4**] Open the new folder by highlighting it, clicking Show in Explorer and paste the copied files into the folder.







Embedded Input Module will now be set-up for your project and available to use. However, it doesn’t take the name of the Legacy Input Module (UnityEngine.Input) for compatibility concerns though this can be modified at your own risk. However, C# offers an ability in scripts to make use of a desired name for a specific class within a C# project as an “alias”.

How to call through “Input” rather than “EmbeddedInputModule”

If you want to use EmbeddedInputModule as if you were directly using the Input class itself, add the following line below the other “using” lines at topof the script you are looking to use this module within:

```
using Input = EmbeddedInputModule;
```

As long as you are not using the old *UnityEnigne.Input* class nor the new *InputSystem* within your script, then this will work fine. If not, check if your script uses either of those and amend according to your needs. Alternatively, give this module an alias of your preference.

Brief Explanation

This is the alias ability that C# has, which enables names that currently exist to be used within a project without overwriting the name of another existing class in the project. This only works so long as the other class(es) are not used within the same script. A common example of name conflicts that some developers may have run into is when using the **System** namespace alongside the **UnityEngine** namespace, causing the “Object” class to have issues.

So long as there is one, and only one, of that alias or named class being used within a script, this should not happen.

Compatibility with Legacy Input Module

An ongoing process for EmbeddedInputModule (**EIM**) is the compatibility with the Legacy Input Module (**LIM**). With the use of the new InputSystem (**IS**) package there are still API features not available yet, which **EIM** is affected by as well. If a member, method or class from the **LIM** is marked as available within **EIM** in the table below, it has been translated to be used exactly as the legacy UnityEngine.Input API describes in the exact same way.

Available under **InputSystem** means that Legacy Input Module behaviour exists but usually requires accesing InputDevice or a sub-class. Available with caveats means it has to be rewritten or manually added and handled (i.e. KeyCode support).

KEY: ✓ → Available ✓* → Available with caveats ✗ → Not Available ? → Absent in Documentation

Member Compatibility

	LIM	IS	EIM	NOTES
Input.acceleration	✓	✓	✓	
Input.accelerationEventCount	✓	✗	✗	No API.
Input.accelerationEvents	✓	✗	✗	No API.
Input.anyKey	✓	✓*	✓	***A setting can be enabled to include Gamepad and Mouse Inputs.***
Input.anyKeyDown	✓	✓*	✓	***A setting can be enabled to include Gamepad and Mouse Inputs.***
Input.backButtonLeavesApp	✓	✗	✗	No API.
Input.compass	✓	✗	✗	No API.
Input.compensateSensors	✓	✓	✓	
Input.compositionCursorPos	✓	✓*	✓*	InputSystem only has a method to set Cursor Position. ***EIM tries to track this. Report issues on GitHub with example projects.***
Input.compositionString	✓	✓*	✓*	***EIM*** *handles the event subscription and conversion noted in InputSystem.*
Input.deviceOrientation	✓	✗	✗	No API.
Input.gyro	✓	✓*	✓*	Input.rotationRateUnbiased has no API. *Gyro is now a public static class.*
Input.imeCompositionMode	✓	✗	✗	No API.
Input.imeIsSelected	✓	✓	✓*	A get-set property is created for this. Needs testing so **Asterisk'd for now.**
Input.inputString	✓	✓*	✓*	A get property is created for this. Needs testing so **Asterisk'd for now.**
Input.location	✓	✗	✗	No API.
Input.mousePosition	✓	✓	✓	
Input.mousePresent	✓	✓	✓	
Input.mouseScrollDelta	✓	?	✗	Not documented or referenced in any version of InputSystem.
Input.multiTouchEnabled	✓	✗	✗	No API.
Input.penEventCount	✓	?	✗	Not documented or referenced in any version of InputSystem.
Input.simulateMouseWithTouches	✓	✗	✗	No API.
Input.stylusTouchSupported	✓	✗	✗	No API.
Input.touchCount	✓	✓	✓	
Input.touches	✓	✓	✓	Touches are of the InputSystem Touch class, not UnityEngine.Touch class.
Input.touchPressureSupported	✓	✗	✗	No API.
Input.touchSupported	✓	✓	✓	

Every attempt has been made to provide, find alternatives to or replace missing API features.

Input.gyro does not work exactly as the original member does, being replaced by a static class which mimics the underlying members of the original **UnityEngine.Gyroscope** class. This class directly hooks onto any current Gysoscope, GravitySensVor, AttitudeSensor, Accelerometer and LinearAccelerationSensor in the device the application is running on and attempts to retrieve data from them. In the event the device is not found, the default for the object type to be returned is given.

Input.anyKey and **Input.anyKeyDown** makes use of the current keyboard device's synthetic **AnyKey** button. Instead of Unity's suggestion for wasUpdatedThisFrame, the use of Keyboard.isPressed and Keyboard.wasPressedThisFrame are used as they behave as expected of the original input system. **wasUpdatedThisFrame** may still work.

KEY: ✓ → Available ✓* → Available with caveats ✗ → Not Available ? → Absent in Documentation

Method Compatibility

	LIM	IS	EIM	NOTES
Input.ClearLastPenContactEvent	✓	?	✗	Not documented or referenced in any version of InputSystem.
Input.GetAccelerationEvent	✓	✗	✗	No API.
Input.GetAxis	✓	✗	✗	*In-Development as part of an add-on to EmbeddedInputModule.*
Input.GetAxisRaw	✓	✗	✗	No current use-case, so was skipped for the time-being.
Input.GetButton	✓	✗	✗	*In-Development as part of an add-on to EmbeddedInputModule.*
Input.GetButtonDown	✓	✗	✗	*In-Development as part of an add-on to EmbeddedInputModule.*
Input.GetButtonUp	✓	✗	✗	*In-Development as part of an add-on to EmbeddedInputModule.*
Input.GetJoystickNames	✓	✓*	✓*	Returns a list of discovered devices by InputSystem. ***A setting can be enabled to only return names of Gamepads.***
Input.GetKey	✓	✓*	✓	*Custom ported*. Some KeyCodes aren't supported.
Input.GetKeyDown	✓	✓*	✓	*Custom ported*. Some KeyCodes aren't supported.
Input.GetKeyUp	✓	✓*	✓	*Custom ported*. Some KeyCodes aren't supported.
Input.GetLastPenContactEvent	✓	?	✗	Not documented or referenced in any version of InputSystem.
Input.GetMouseButton	✓	✓*	✓	*Support for **Mouse4** and **Mouse5** as Ids [**3**] and [**4**] added.*
Input.GetMouseButtonDown	✓	✓*	✓	*Support for **Mouse4** and **Mouse5** as Ids [**3**] and [**4**] added.*
Input.GetMouseButtonUp	✓	✓*	✓	*Support for **Mouse4** and **Mouse5** as Ids [**3**] and [**4**] added.*
Input.GetPenEvent	✓	✗	✗	No API.
Input.GetTouch()	✓	✓	✓	Polymorphic method. GetTouch() retrieves the last touch. GetTouch(int) retrieves the touch at the given index, if it's in range.
Input.IsJoystickPreconfigured	✓	✗	✗	*Not needed due to Gamepad auto-mapping controls.*
Input.ResetInputAxes	✓	✗	✗	*In-Development as part of an add-on to EmbeddedInputModule.*
Input.ResetPenEvents	✓	?	✗	Not documented or referenced in any version of InputSystem.

Every attempt has been made to provide, find alternatives to or replace missing API features.

Input.GetKey, **Input.GetKeyDown** and **Input.GetKeyUp** were ported over with the ability to use UnityEngine's old KeyCode enum. Each value was, to the best ability possible, carefully compared and translated over where possible. However, there are still keycodes which at this time currently do not have an equivalent *Keyboard* key. Additionally some regions have keyboard keys (e.g. Tilde) in various locations. For missing KeyCodes falling into the latter case, a KeyboardRegionLayout setting is under consideration to provide an adequate means for developers to handle these.

Methods for Pen Events within the Legacy Input Module currently do not have any documentation or point of reference in InputSystem. Although they are present in UnityEngine.Input, it is possible that the InputSystem either already has included the intended behaviours within places, such as the PenDevice. Until more is known about these specific methods and how to best re-incorporate them into EmbeddedInputModule, these have been skipped.

On the topic of the Virtual Axes featured within LegacyInputModule's InputManager, it was originally considered to not be possible, however a 2015 forum post has since highlighted otherwise. Upon further investigation, it was deemed that while GetAxis is still of use (though GetAxisRaw may not be), any such work is best kept as an optional add-on as custom inspectors and extra classes to facilitate the re-implementation of Virtual Axes would result in extra files. In this future VirtualAxes addon module,the goal is to offer the support (as best as possible) for any axes-related methods which are currently not included.

As to why Virtual Axes will not appear in the main module, that in the goal of this module. EmbeddedInputModule is about providing a single file which contains everything necessary to make use of all of the benefits of Unity's new InputSystem while offering the documentation, members and methods of the Legacy Input Module. Virtual Axes creating additional classes and inspectors means there are extra requirements the end-user must perform to get the drag-and-drop replacement of that same functionality. It also means that additional documentation is required specifically to help developers set the add-on up.

Additions

EmbeddedInputModule also includes some extra members and methods as quality-of-life additions to remove some of the extra steps required to get some Inputs. On top of this, a class, a method used internally and some enums exist which are exposed so that developers can independently add in any additional input devices or extend to more device identifiers. Lastly, it was decided to separate the *Pressed*, *Released* and *Held* controller input members for the sake of legibility.

The following lists *assume* that there is a directive at the top of a developer's code giving the alias of **Input** to this module.

New Members (Unique)

	Description
Input.PlatformIcons	Which platform-specific icions should UI elements be displaying.
Input.CurrentGamepad	The DeviceIdentifier object for the current controller device being used for input. This changes based on the last controller used for input, if there are various connected.

New Members (Device *or* Sensor Present)

	Description
Input.gamepadPresent	Returns true if a gamepad has been detected and set as the current gamepad.
Input.keyboardPresent	Returns true if Keyboard.Current isn't null.
Input.accelerometerPresent	Returns true if there is an accelerometer on the device.
Input.gyroscopePresent	Returns true if there is a gyroscope on the device.
Input.altitudeSensorPresent	Returns true if there is an altitude sensor on the device.
Input.gravitySensorPresent	Returns true if there is a gravity sensor on the device.
Input.linearAccelerationSensorPresent	Returns true if there is a linear acceleration sensor on the device.

Some are used by the new static class mimicing **Input.gyro**. Others are available because **Input.mousePresent** is.

New Members (Controller Input Pressed)

	Description
Input.LeftStickButton	Returns true if the Left Joystick Button on the currnet Controller was pressed this frame.
Input.RightStickButton	Returns true if the Right Joystick Button on the currnet Controller was pressed this frame.
Input.LeftShoulder	Returns true if the Left Shoulder on the currnet Controller was pressed this frame.
Input.RightShoulder	Returns true if the Right Shoulder on the currnet Controller was pressed this frame.
Input.LeftTrigger	Returns true if the Left Trigger on the currnet Controller was pressed this frame.
Input.RightTrigger	Returns true if the Right Trigger on the currnet Controller was pressed this frame.
Input.ButtonNorth	Returns true if the North Facing button on the currnet Controller was pressed this frame. [**Y** - Xbox | **Triangle** - Playstation | **X** - Switch]
Input.ButtonSouth	Returns true if the South Facing button on the currnet Controller was pressed this frame. [**A** - Xbox | **Cross** - Playstation | **B** - Switch]
Input.ButtonEast	Returns true if the East Facing button on the currnet Controller was pressed this frame. [**B** - Xbox | **Circle** - Playstation | **A** - Switch]
Input.ButtonWest	Returns true if the West Facing button on the currnet Controller was pressed this frame. [**X** - Xbox | **Square** - Playstation | **Y** - Switch]
Input.DpadUp	Returns true if D-Pad Up on the currnet Controller was pressed this frame.
Input.DpadDown	Returns true if D-Pad Down on the currnet Controller was pressed this frame.
Input.DpadLeft	Returns true if D-Pad Left on the currnet Controller was pressed this frame.
Input.DpadRight	Returns true if D-Pad Right on the currnet Controller was pressed this frame.
Input.Select	Returns true if the Select button on the currnet Controller was pressed this frame. [**View** - Xbox | **Share** - Playstation | **Minus** - Switch]
Input.Start	Returns true if the Select button on the currnet Controller was pressed this frame. [**Menu** - Xbox | **Options** - Playstation | **Plus** - Switch]

New Members (Controller Input Released)

	Description
Input.LeftStickButtonReleased	Returns true if the Left Joystick Button on the currnet Controller was released this frame.
Input.RightStickButtonReleased	Returns true if the Right Joystick Button on the currnet Controller was released this frame.
Input.LeftShoulderReleased	Returns true if the Left Shoulder on the currnet Controller was released this frame.
Input.RightShoulderReleased	Returns true if the Right Shoulder on the currnet Controller was released this frame.
Input.LeftTriggerReleased	Returns true if the Left Trigger on the currnet Controller was released this frame.
Input.RightTriggerReleased	Returns true if the Right Trigger on the currnet Controller was released this frame.
Input.ButtonNorthReleased	Returns true if the North Facing button on the currnet Controller was released this frame. [**Y** - Xbox | **Triangle** - Playstation | **X** - Switch]
Input.ButtonSouthReleased	Returns true if the South Facing button on the currnet Controller was released this frame. [**A** - Xbox | **Cross** - Playstation | **B** - Switch]
Input.ButtonEastReleased	Returns true if the East Facing button on the currnet Controller was released this frame. [**B** - Xbox | **Circle** - Playstation | **A** - Switch]
Input.ButtonWestReleased	Returns true if the West Facing button on the currnet Controller was released this frame. [**X** - Xbox | **Square** - Playstation | **Y** - Switch]
Input.DpadUpReleased	Returns true if D-Pad Up on the currnet Controller was released this frame.
Input.DpadDownReleased	Returns true if D-Pad Down on the currnet Controller was released this frame.
Input.DpadLeftReleased	Returns true if D-Pad Left on the currnet Controller was released this frame.
Input.DpadRightReleased	Returns true if D-Pad Right on the currnet Controller was released this frame.
Input.SelectReleased	Returns true if the Select button on the currnet Controller was released this frame. [**View** - Xbox | **Share** - Playstation | **Minus** - Switch]
Input.StartReleased	Returns true if the Select button on the currnet Controller was released this frame. [**Menu** - Xbox | **Options** - Playstation | **Plus** - Switch]

New Members (Controller Input Held)

	Description
Input.LeftStickButtonHeld	Returns true if the Left Joystick Button on the currnet Controller is held down.
Input.RightStickButtonHeld	Returns true if the Right Joystick Button on the currnet Controller is held down.
Input.LeftShoulderHeld	Returns true if the Left Shoulder on the currnet Controller is held down.
Input.RightShoulderHeld	Returns true if the Right Shoulder on the currnet Controller is held down.
Input.LeftTriggerHeld	Returns true if the Left Trigger on the currnet Controller is held down.
Input.RightTriggerHeld	Returns true if the Right Trigger on the currnet Controller is held down.
Input.ButtonNorthHeld	Returns true if the North Facing button on the currnet Controller is held down. [**Y** - Xbox | **Triangle** - Playstation | **X** - Switch]
Input.ButtonSouthHeld	Returns true if the South Facing button on the currnet Controller is held down. [**A** - Xbox | **Cross** - Playstation | **B** - Switch]
Input.ButtonEastHeld	Returns true if the East Facing button on the currnet Controller is held down. [**B** - Xbox | **Circle** - Playstation | **A** - Switch]
Input.ButtonWestHeld	Returns true if the West Facing button on the currnet Controller is held down. [**X** - Xbox | **Square** - Playstation | **Y** - Switch]
Input.DpadUpHeld	Returns true if D-Pad Up on the currnet Controller is held down.
Input.DpadDownHeld	Returns true if D-Pad Down on the currnet Controller is held down.
Input.DpadLeftHeld	Returns true if D-Pad Left on the currnet Controller is held down.
Input.DpadRightHeld	Returns true if D-Pad Right on the currnet Controller is held down.
Input.SelectHeld	Returns true if the Select button on the currnet Controller is held down. [**View** - Xbox | **Share** - Playstation | **Minus** - Switch]
Input.StartHeld	Returns true if the Select button on the currnet Controller is held down. [**Menu** - Xbox | **Options** - Playstation | **Plus** - Switch]

New Members (Controller Input Axes - Vector2)

	Description
Input.LeftStick	Get the Vector2 axis representing the current controller's Left Joystick inputs.
Input.RightStick	Get the Vector2 axis representing the current controller's Right Joystick inputs.
Input.Dpad	Get the Vector2 axis representing the current controller's D-Pad inputs.

New Members (Controller Input - Other)

	Description
Input.LeftTriggerActuation	Get how far down the Left Trigger is pressed.
Input.RightTriggerActuation	Get how far down the Right Trigger is pressed.

New Methods

	Description
Input.ConvertKeyCode	Convert a **UnityEngine.KeyCode** enum value to it's corresponding ButtonControl value. Returns *true* or *false* depending on if a ButtonControl was found for the given KeyCode. [**Parameters**] **KeyCode key** → The desired key *out* **ButtonControl control** → The resulting control.
Input.EnableTouch	Enable InputSystem's EnhancedTouchSupport. Prevents multiple calls to enable touch, as multiple calls to enable requires an equivalent amount of calls to then disable the system.
Input.DisableTouch	Disable InputSystem's EnhancedTouchSupport. Prevents multiple calls to disable touch, as multiple calls to disable requires an equivalent amount of calls to then enable the system.

New Enums

	Description
Input.DeviceType	An enum as a list of **InputDevice** types that can be identified, if a device has been found. These are: • Keyboard • Mouse • Gamepad • Pen • Unknown
Input.GamepadType	An enum as a list of **Gamepad** types that can be identified, if a device has been identified as a gamepad. These are: • Xbox • Playstation • Switch • Generic
Input.InputIconDisplayType	An enum as a list of possible types that can be used to identify which icons should be appearing for input hints, button prompts or other context clues on user interfaces. • PC • Xbox • PlayStation • Switch • Generic It is at the developers discretion to interpret the Generic value however they deem fit. It is advised that for the other categories they display icons for those respective platforms.

DeviceType and GamepadType also include the value "None" as the default, first value for variables of these enum types.

New Class - DeviceIdentifier

Although there already existed InputDevice and it's inheriting classes, the detection of some controllers was found to be inconsistent in testing. There were also concerns that third-party controllers which present icons for a specific platform may not be correctly identified if the HID information was slightly off. DeviceIdentifer ***can't*** rectify this immediately, however it and the current values within the enumerators previously mentioned are a good enough starting point for an initial release.

In terms of what DeviceIdentifier does, it is most simply a wrapper around InputDevice with extra members that handle the identification of the the device through it's name (as of right now).

If you have a controller which has yet to be identified, please create an Issue and submit the controller's name and HID information. *Images of the controller may also help, especially in cases where the controller is not for a specific platform.*

Public Members

	Description
DeviceIdentifier.type	The current type of device. *(DeviceType enum value)*
DeviceIdentifier.gamepadType	The current type of gamepad, if the device is a gamepad. *(GamepadType enum value)*
DeviceIdentifier.device	The InputDevice object which this identifier wraps around.
DeviceIdentifier.IsKeyboard	Is this DeviceIdentifier a Keyboard?
DeviceIdentifier.IsMouse	Is this DeviceIdentifier a Mouse?
DeviceIdentifier.IsPen	Is this DeviceIdentifier a Pen?
DeviceIdentifier.IsGamepad	Is this DeviceIdentifier a Gamepad?
DeviceIdentifier.IsDualSense	Is this DeviceIdentifier a PS5 Gamepad?
DeviceIdentifier.IsXInput	Is this DeviceIdentifier an Xbox Gamepad?
DeviceIdentifier.IsSwitch	Is this DeviceIdentifier a Switch Gamepad?
DeviceIdentifier.name	Get the name of the underlying InputDevice.
DeviceIdentifier.deviceId	Get the device ID of the underlying InputDevice.
DeviceIdentifier.Null	Is the underlying InputDevice null?

Public Methods

	Description
DeviceIdentifier.ConvertKeycode	Check if the underlying InputDevice of this DeviceIdentifier matches the comparative InputDevice provided. [**Parameters**] **InputDevice comparative** → The device to compare this one to.
DeviceIdentifier.ToString	Override of the generic ToString method provided by C#. Converts the DeviceIdentifier into a readable format. The following will be returned (with line-break characters). *Device: [InputDevice.name]* *Type: [DeviceIdentifier.type]* *Gamepad Type: [DeviceIdentifier.gamepadType]*

Constructor Variants

	Description
DeviceIdentifier(InputDevice)	Create a DeviceIdentifier from the provided InputDevice and attempt to discover what type of device it is, including the GamepadType if it is found to be a gamepad.
DeviceIdentifier(InputDevice, DeviceType)	Create a DeviceIdentifier from the provided InputDevice and DeviceType. Desired if the type of device is already known to prevent unnecessary identification attmepts.

Only these are valid constructors for the DeviceIdentifier class. The default DeviceIdentifier() will not be able to identify a device as none would be passed as a parameter in the constructor.

Configurable Settings (Advanced)

Within the script of the EmbeddedInputModule is a list of configuration settings for users who may want or need specific features to be used. This uses preprocessor directives within C# in order to switch between desired features and, where applicable, relavent restricted substitutes for features. There are three “Codes” which define what a setting does.

Settings Code	Description
MSG	Debug Message
OPT	Code Optimization / Code Micro-optimization
CFT	Compatibility Feature - Features made for compatibility with code using the old input system.

For each code, a setting may exist which enables, disables or restricts the feature it relates to. These appear directly before the code as **ENABLE**, **DISABLE** or **RESTRICT**. The order of which these are the order of which they were created, generally.

These will appear like in this example setting:

```
#define EIM_DISABLEMSG_EnhancedTouchEnabledORDisabled            // DISABLE "Enhanced Touch Already Enabled/Disabled." MESSAGE
```

Enabling or Disabling Settings

To enable or disable each setting, open up the EmbeddedInputModule.cs script from where you have it stored in your Unity Project and navigate to Line 30, 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 color of the text at the start of the line.

```
37  //#define EIM_DISABLEMSG_ActiveGamepadSet                    // DISABLE "Active Gamepad Set as..." MESSAGE
38
39    #define EIM_DISABLEMSG_EnhancedTouchEnabledORDisabled      // DISABLE "Enhanced Touch Already Enabled/Disabled." MESSAGE
```

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

Although all settings are described briefly in the script already, it is felt that a full list of the settings and what each setting does most specifically is a necessity as these settings exist just to provide additional functionality to the module.

EIM_DISABLEMSG_ActiveGamepadSet

Disables the message which appears whenever the EmbeddedInputModule switches the DeviceIdentifier bound to EmbeddedInputModule.CurrentGamepad.

Default: Disabled

EIM_DISABLEMSG_EnhancedTouchEnabledOrDisabled

Disable the message which states that the EnhancedTouchSupport has already been enabled or disabled, if multiple calls to EmbeddedInputModule.EnableTouch() or EmbeddedInputModule.DisableTouch() have occurred.

Default: Disabled

EIM_DISABLEMSG_EnhancedTouchNeedsToBeEnabled

Disable the message that appears when attempting to call any Touch-related member or method while the Enhanced Touch system is disabled within InputSystem.

Default: Disabled

EIM_RESTRICTCFT_anyKeyANDAnyKeyDown

Restrict the way in which the values of EmbeddedInputModule.anyKey and EmbeddedInputModule.anyKeyDown are obtained. When this is compatibility feature restriction is applied, only the Keyboard.Current.Any synthetic key is read.

If this is disabled, the module will instead go through each and every individual KeyCode for each ButtonControl with the use of EmbeddedInputModule.ConvertKeyCode() and attempt to find if any key was pressed. This is not particularly efficient, though the impact this will have on performance is not known as this uses a method bound to the event InputSystem.onAfterUpdate in order to run.

However, by disabling this setting, buttons pressed on Gamepads will also count towards whether any key is pressed or held down.

Default: Enabled

EIM_RESTRICTCFT_JoystickNameGamepadNamesOnly

Restricts EmbeddedInputModule.GetJoystickNames() from returning the name of every connected device or peripheral. This method will then instead return the names of all connected Gamepads stored within the module's internal array.

Default: Disabled

EIM_DISABLEOPT_SingleGamepadKeycodeCheck

Changes how the EmbeddedInputModule internally identifies a keycode. This is provided in the event that Unity Technologies adds more KeyCodes which are not Gamepad KeyCodes to the list. As of writing all Gamepad KeyCodes exist at value 330 and onwards, however, future versions of Unity could add new KeyCodes beyond value 509, which is the last currently defined KeyCode.

Currently not particularly useful outside of an optimization case for versions of Unity Engine with 509 KeyCode values defined. Included as a precaution.

Default: Disabled

EIM_ENABLEOPT_DisableDeviceChangedExtraCases

EmbeddedInputModule listens to the event InputSystem.onDeviceChange to determine when a device has been added, removed or otherwise modified. Currently, only the following cases are listened to, with the other cases Included as a preparation for any potential use cases.

- InputDeviceChange.Enabled
- InputDeviceChange.Disabled
- InputDeviceChange.Reconnected
- InputDeviceChange.Disconnected
- InputDeviceChange.Added
- InputDeviceChange.Removed

This feature hides all other cases within the switch statement, in case there is any potential micro-optimization to be gained from doing so. This is currently being investigated.

Default: Enabled