Input Device Manager Manual

Pixel Crushers Common Library

Contents

Chapter 1: Input Device Manager Overview

The Input Device Manager is an optional component that handles input for Pixel Crushers assets. It provides these features:

- Automatically detect when mouse, joystick, or keyboard is being used.
- When joystick or keyboard are used, it can be configured to hide the mouse cursor, and Pixel Crushers UIs ensure that a button is focused (selected) so the user can navigate buttons.
- Ability to use traditional Unity Input manager, new Input System, or other input systems.

The Input Device Manager detects changes by monitoring mouse movement and joystick & keyboard presses. You can configure which inputs to check in the Input Device Manager's Inspector.

The Input Device Manager may be found on its own GameObject. If using the Dialogue System for Unity, it is usually placed on the Dialogue Manager GameObject. By default, the Input Device Manager makes its GameObject into a singleton that survives scene changes.

How to Get Help

We're here to help! If you get stuck or have any questions, please contact us any time at support@pixelcrushers.com or visit http://pixelcrushers.com.

We do our very best to reply to all emails within 24 hours. If you haven't received a reply within 24 hours, please check your spam folder.

Chapter 2: Reading Input

The Input Device Manager provides an abstract wrapper that can read input from traditional Unity Input, Unity's new Input System, or other input systems such as Rewired.

Use these methods to read input:

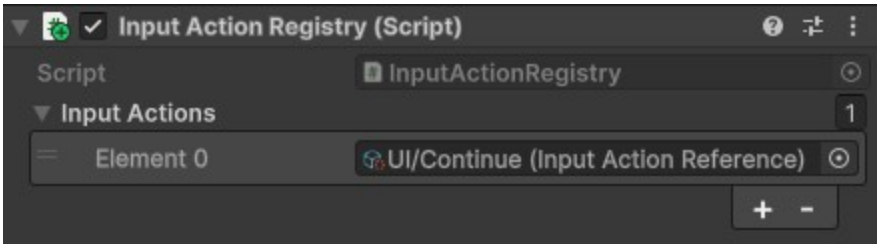
```
bool InputDeviceManager.IsButtonDown("button name")
bool InputDeviceManager.IsButtonUp("button name")
bool InputDeviceManager.IsKeyDown(KeyCode)
float InputDeviceManager.GetAxis("axis name")
Vector3 InputDeviceManager.GetMousePosition()
```

See also: Input Device Manager API Reference

Chapter 2: New Input System

To use Unity's new Input System package, define the scripting symbol `USE_NEW_INPUT`. If you're using the Dialogue System, the Welcome Window provides a checkbox to automatically define this symbol for you.

After you define your inputs, typically in an Input Actions asset, you must register them with the Input Device Manager before calling any of the InputDeviceManager input query functions such as `InputDeviceManager.IsButtonDown` and `InputDeviceManager.GetAxis`. To register the inputs, add an Input Actions Registry component to your Input Device Manager GameObject and assign the action inputs to it:



Or, if you want to use your own C# script, use `InputDeviceManager.RegisterInputAction`. Use `InputDeviceManager.UnregisterInputAction` to unregister them. Example:

```
using UnityEngine;
using UnityEngine.InputSystem;
using PixelCrushers;

public class RegisterMyControls : MonoBehaviour
{
    protected static bool isRegistered = false;
    private bool didIRegister = false;
    private Controls controls;

    void Awake()
    {
        controls = new MyControls();
    }

    void OnEnable()
    {
        if (!isRegistered)
        {
            isRegistered = true;
            didIRegister = true;
            controls.Enable();
            InputDeviceManager.RegisterInputAction("Back", controls.Gameplay.Back);
            InputDeviceManager.RegisterInputAction("Interact", controls.Gameplay.Interact);
        }
    }

    void OnDisable()
    {
        if (didIRegister)
```

```
        {
            isRegistered = false;
            didIRegister = false;
            controls.Disable();
            InputDeviceManager.UnregisterInputAction("Back");
            InputDeviceManager.UnregisterInputAction("Interact");
        }
    }
}
```

The code above registers two inputs: “Back” and “Interact” so you can use them in InputDeviceManager queries. Add it to your scene (e.g., to the Dialogue Manager GameObject if using the Dialogue System.)

Review the input values already assigned by default to the Input Device Manager, such as “Back”, and also including any key code dropdowns, which the Input Device Manager will attempt to interpret as Input System inputs. For example, the key code “Joystick Button 0” will make the Input System look for a registered input named “joystickbutton0”. Unless you register equivalent inputs as described above, the Input System will report errors.

UI Button Hotkeys

To map a UI button to a new Input System input, register the input as described above. Then add a UI Button Key Trigger component to the UI button. Set the Key dropdown to None, or to a key code if you’d like to map a key to the button. Set the Button field to the name of the registered input.

Unity UI Navigation

Check your scenes’ EventSystem GameObjects. They should have an InputSystemUIInputModule component, not a StandaloneInputModule.

Chapter 3: Rewired

To use Rewired, import this package:

Plugins > Pixel Crushers > Common > Third Party Support > Rewired Support

Then add an Input Device Manager Rewired component to the same GameObject as the Input Device Manager. Both components are required to use Rewired with the Input Device Manager. Queries such as `InputDeviceManager.IsButtonDown` will use Rewired instead of Unity Input.

Chapter 4: Other Input Systems

To use other input systems, you can assign your own methods to the Input Device Manager component’s `GetButtonDown`, `GetButtonUp`, and `GetAxis` delegates.