DualSense Unity Documentation

Contents

Introduction

This document is intended to serve as a quick guide for getting started using this Unity plugin and its code to interact with your DualSense controllers as well as a reference to look back at when you need help doing a specific thing with your DualSense controller. The plugin's code should allow you to detect when a DuaSense controller is connected through USB or through Bluetooth. You will be able to detect when things happen to your controller like a button being pressed and you will be able to configure your controller in various ways like setting the haptic trigger effects.

Getting Started

Setup

Installing the plugin should incorporate all necessary files to be ready for use, but in case that does not happen, you will need to bring the following files over to your Assets folder.

- ds5w_x64.dll
- DualSenseWindowsNative.dll

In any script where you want to communicate with a DualSense controller, a single import of 'DualSenseUnity' is all that is needed to get started.

```
using DualSenseUnity;
```

Controller Retrieval

Before you are able to do the fun stuff with your DualSense controller like set the trigger effects or detect which buttons are pressed, you need to detect when controllers are connected and know how to interact with them. The following code shows you how to:

- Get controller counter
- Get list of controls to interact with
- Detect when the number of controllers changes

```
// Private variables to keep track of controllers
private uint _controllerCount;
private List<DualSenseController> _dualSenseControllers;

// Initializing these variables somewhere
_controllerCount = DualSense.GetControllerCount();
_dualSenseControllers = DualSense.GetControllers();

// Listen for the event when the controller count changes
DualSense.ControllerCountChanged += RefreshControllers;

// Calling code to initialize controller information
// This is often the same code used when the controller
// count changes
RefreshControllers();

private void RefreshControllers()
{
   _controllerCount = DualSense.GetControllerCount();
   _dualSenseControllers = DualSense.GetControllers();

   // You can now use the controller count and the list of controllers in your
   // further scripts to interact with the DualSense controllers
}
```

Reference

static class DualSense

static uint GetControllerCount()

Returns the number of DualSense controllers that are currently connected.

static List<DualSenseController> GetControllers()

Returns a list of all currently connected DualSense controller. You can then use these DualSenseController objects to perform other actions like configuring the haptic trigger effects or detecting when a button is pressed.

static event Action ControllerCountChanged

An event you can attach to detect when DualSense controllers are disconnected or new DualSense controllers are connected.

Controller Input

Controller Input refers to detecting when various things happen to your DualSense controller like a button being pressed or the controller being tilted and turned. The following code shows how to determine if:

- The D-pad up button is pressed
- The Circle button is released
- The left stick is pushed in as a button
- The Right Trigger is pulled at least halfway
- The left stick is pushed to the left

```
// Get the input state from the first controller
var controllerToUse = _dualSenseControllers[0];
var inputState = controllerToUse.GetInputState();

// The normal states to check for are Down and Up, but NewDown
// and NewUp are also available to easily determine when the state
// has changed
var dPadUpIsPressed = inputState.DPadUpButton == ButtonState.Down ||
   inputState.DPadUpButton == ButtonState.NewDown;
var circleIsPressed = inputState.CircleButton == ButtonState.Up ||
   inputState.CircleButton == ButtonState.NewUp;
var leftStickIsPressed = inputState.LeftStick.PushButton == ButtonState.Down ||
   inputState.LeftStick.PushButton == ButtonState.NewDown;

// Numerical values that have a range like how far a trigger is pulled
// go from 0.0-1.0
var rTriggerPulledHalf = inputState.RightTrigger.TriggerValue >= .5;

// The stick values for either axis go from -1.0-1.0
var leftStickPointingLeft = inputState.LeftStick.XAxis < 0;
```

Reference

class DualSenseController

ControllerInputState GetInputState()

Returns the information about the input state of the controller including if buttons are pressed, triggers are pulled, or sticks are rotated.

<u>class ControllerInputState</u>

bool IsValid

This is true is retrieving the controller’s input state succeeded

TriggerState LeftTrigger

Tells you how much the left trigger is pulled as well as interpreting the trigger as a button where pulling at all is counted as a button press.

TriggerState RightTrigger

Tells you how much the right trigger is pulled as well as interpreting the trigger as a button where pulling at all is counted as a button press.

ButtonState LeftBumper

Tells you if the left bumper is pressed or not.

ButtonState RightBumper

Tells you if the right bumper is pressed or not.

StickState LeftStick

Tells you how far the left stick is pushed on either the X or Y axis as well as whether the left stick is pushed in as a button or not.

StickState RightStick

Tells you how far the right stick is pushed on either the X or Y axis as well as whether the right stick is pushed in as a button or not.

ButtonState DPadUpButton

Tells you if the D-pad up button is pressed or not.

ButtonState DPadDownButton

Tells you if the D-pad down button is pressed or not.

ButtonState DPadLeftButton

Tells you if the D-pad left button is pressed or not.

ButtonState DPadRightButton

Tells you if the D-pad right button is pressed or not.

ButtonState TriangleButton

Tells you if the Triangle button is pressed or not.

ButtonState CircleButton

Tells you if the Circle button is pressed or not.

ButtonState CrossButton

Tells you if the Cross / X button is pressed or not.

ButtonState SquareButton

Tells you if the Square button is pressed or not.

ButtonState PSButton

Tells you if the PlayStation button is pressed or not.

ButtonState CreateButton

Tells you if the Create button is pressed or not.

ButtonState OptionsButton

Tells you if the Options button is pressed or not.

ButtonState MicrophoneButton

Tells you if the Microphone button is pressed or not.

TouchPadState TouchPad

Tells you where one or two fingers are touching the touch pad.

VectorState Accelerometer

Tells you the three-dimensional sensor readings for the controller's accelerometer. This can tell you how fast a controller is being moved in any direction.

VectorState Gyroscope

Tells you the three-dimensional sensor readings for the controller's gyroscope. This can tell you how a controller is rotated and tilted in any direction.

bool HeadPhonesConnected

Tells you if something is plugged in to the headphone jack of the controller. This library does not currently let you output audio to the headphone jack, but you can determine if something is plugged in there in case that is useful.

enum ButtonState

Up

The button is currently released.

Down

The button is currently pushed.

NewUp

The button was just released. This will only be reported for one frame / update cycle.

NewDown

The button was just pushed. This will only be reported for one frame / update cycle.

struct TriggerState

double TriggerValue

Tells you how far the trigger is pulled. The value goes from 0.0 (not pulled at all) to 1.0 (pulled all the way).

ButtonState ActiveState

Tells you if the trigger is pressed like a button. If the Trigger value is anything above 0, that counts as a button press.

struct StickState

double XAxis

Tells you how far the stick is pushed left or right. The value goes from -1 (all the way left) to 1.0 (all the way right).

double YAxis

Tells you how far the stick is pushed up or down. The value goes from -1 (all the way down) to 1.0 (all the way up).

ButtonState PushButton

Tells you if the stick is pressed like a button.

struct VectorState

double XAxis

A value for the X axis of a sensor reading like the accelerometer of the gyroscope. This is usually a raw value and not converted to a range of 0.0-1.0 like other values.

double YAxis

A value for the Y axis of a sensor reading like the accelerometer of the gyroscope. This is usually a raw value and not converted to a range of 0.0-1.0 like other values.

double ZAxis

A value for the Z axis of a sensor reading like the accelerometer of the gyroscope. This is usually a raw value and not converted to a range of 0.0-1.0 like other values.

struct TouchPadState

TouchPointState TouchPoint1

Tells you the X and Y position of the first finger touching the pad. If you let go, the last position is continuously reported. You would need to detect if the finger is no longer touching by seeing if the position changes.

TouchPointState TouchPoint2

Tells you the X and Y position of the second finger touching the pad. If you let go, the last position is continuously reported. You would need to detect if the finger is no longer touching by seeing if the position changes.

struct TouchPointState

double X

Tells you where your finger is on the left to right axis. The goes from 0.0 (all the way left) to 1.0 (all the way right).

double Y

Tells you where your finger is on the up to down axis. The goes from 0.0 (all the way up) to 1.0 (all the way down).

Controller Output

Controller output refers to setting the output state of the DualSense controller, which includes things like the trigger effects, the lights on the controller, and the rumble motors. The following code:

- Make the player lights fade in and out when their state changes
- Turns the first and last of 5 player lights on
- Turns on the lights bar and makes it purple at 80% brightness
- Turns on the right rumble motor at 20% strength
- Sets the left trigger to use continuous resistance at half strength for the entire trigger pull
- Sets the right trigger to have the strongest possible resistance between the 20% and 60% positions of the trigger sort of like pulling the trigger of a gun

```
var output = new ControllerOutputState();

// Make the player lights animate with a fade and
// turn on the first and last player light
output.FadePlayerLight = true;
output.LeftPlayerLightEnabled = true;
output.RightPlayerLightEnabled = true;

// Turn on the light bar and make it purple
// with 80% brightness
output.LightBarEnabled = true;
output.LightBarIntensity = .8;
output.LightBarR = 1.0;
output.LightBarG = 0;
output.LightBarB = 1.0;

// Make the right side rumble at 20% strength
output.RightRumbleIntensity = .2;

// Set the left trigger to use half force
// for the entire trigger pull
var startPosition = 0.0;
var force = .5;
output.LeftTriggerEffect.InitializeContinuousResistanceEffect(
   startPosition, force );

// Set the right trigger to feel pulling a gun's trigger
// with resistance between 20% and 80% of the pull
// with a strong resistance
startPosition = .2;
var endPosition = .6;
force = 1.0;
output.RightTriggerEffect.InitializeSectionResistanceEffect(
   startPosition, endPosition, force );


// Set the output state of the first controller
var controllerToUse = _dualSenseControllers[0];
controllerToUse.SetOutputState( output );
```

Reference

class DualSenseController

bool SetOutputState(ControllerOutputState state)

Sets the output state of the controller. This lets you set things like the haptic trigger effects, turn rumble on, and turn lights on. This returns true if setting the output state succeeds.

struct ControllerOutputState

TriggerEffect LeftTriggerEffect

The effect applied to the left trigger. This lets you change the resistance settings for the haptic trigger.

TriggerEffect RightTriggerEffect

The effect applied to the right trigger. This lets you change the resistance settings for the haptic trigger.

double LeftRumbleIntensity

The strength of the left rumble motor. This value goes from 0.0 (no rumble) to 1.0 (max rumble). The left rumble motor is stronger than the right rumble motor.

double RightRumbleIntensity

The strength of the right rumble motor. This value goes from 0.0 (no rumble) to 1.0 (max rumble). The right rumble motor is weaker than the left rumble motor.

bool LightBarEnabled

True turns the light bar on and false turns it off. You need to also set the LightBarIntensity to a value greater than 0 to turn the light on.

double LightBarIntensity

The brightness of the light bar. This value goes from 0.0 (off) to 1.0 (brightest). You need to also set LightBarEnabled to true to turn the light on.

double LightBarR

The red value of the light bar. The value goes from 0.0 (no red) to 1.0 (highest red). That translates to the values of 0-255 if you are used to dealing with color values that way. You need to also set LightBarEnabled to true and LightBarIntensity to a value greater than 0 to turn the light on.

double LightBarG

The green value of the light bar. The value goes from 0.0 (no green) to 1.0 (highest green). That translates to the values of 0-255 if you are used to dealing with color values that way. You need to also set LightBarEnabled to true and LightBarIntensity to a value greater than 0 to turn the light on.

double LightBarB

The blue value of the light bar. The value goes from 0.0 (no blue) to 1.0 (highest blue). That translates to the values of 0-255 if you are used to dealing with color values that way. You need to also set LightBarEnabled to true and LightBarIntensity to a value greater than 0 to turn the light on.

bool LeftPlayerLightEnabled

True turns the leftmost player light on.

bool MiddleLeftPlayerLightEnabled

True turns the second from the left player light on.

bool MiddlePlayerLightEnabled

True turns the middle player light on.

bool MiddleRightPlayerLightEnabled

True turns the second from the right player light on.

bool RightPlayerLightEnabled

True turns the rightmost light on.

bool FadePlayerLight

True makes the player lights animate with a fade when the player lights turn on or off.

struct TriggerEffect

There are public member variables that you can set manually if you would like, but it is recommended that you use the following helper methods to define the trigger effects that you want to use.

void InitializeNoResistanceEffect

Makes the trigger have no resistance at all and work like a normal controller trigger.

void InitializeContinuousResistanceEffect(double startPos, double force)

Makes the trigger have a specified resistance starting at a trigger position.

- double startPos – The position to start the resistance at. The value goes from 0.0 (trigger resting position) to 1.0 (full trigger pull).
- double force – The resistance strength. The value goes from 0.0 (weakest) to 1.0 (strongest).

void InitializeSectionResistanceEffect(double startPos, double endPos, double force)

Makes the trigger have a specified resistance between two positions in a trigger pull. This is good for mimicking pulling a gun's trigger.

- double startPos – The position to start the resistance at. The value goes from 0.0 (trigger resting position) to 1.0 (full trigger pull).
- double endPos – The position to end the resistance at. The value goes from 0.0 (trigger resting position) to 1.0 (full trigger pull).
- double force – The resistance strength. The value goes from 0.0 (weakest) to 1.0 (strongest).

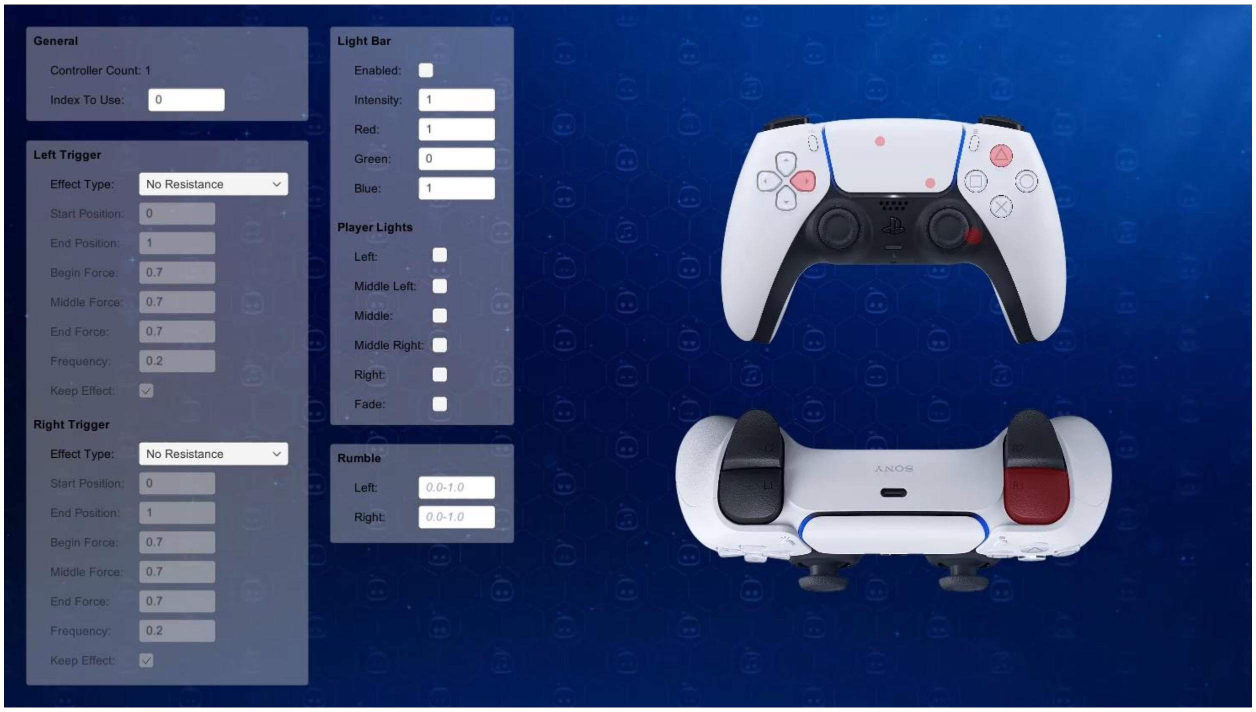
void InitializeExtendedEffect(double startPos, double beginForce, double middleForce, double endForce, double freq, bool keepEffect)

Makes the trigger have a varied starting at a trigger position with some vibration. This is good for mimicking and automatic machine gun firing fast.

- double startPos – The position to start the resistance at. The value goes from 0.0 (trigger resting position) to 1.0 (full trigger pull).
- double beginForce – The resistance strength at the start of the effect. The value goes from 0.0 (weakest) to 1.0 (strongest).
- double middleForce – The resistance strength at the middle of the effect. The value goes from 0.0 (weakest) to 1.0 (strongest).
- double endForce – The resistance strength at the end of the effect. The value goes from 0.0 (weakest) to 1.0 (strongest).
- double freq – How fast the vibration occurs. The value goes from 0.0 (slowest) to 1.0 (fastest).
- bool keepEffect – True makes the effect keep going continuously during the trigger pull.

Using The Sample Scene

A Unity scene named 'DualSenseExample' is included in this plugin's assets. When running the scene, you can set the output state of the controller for things like haptic triggers and lights as well as get feedback on things like which buttons are pressed. The scene currently does not show information for the controller's accelerometer and gyroscope sensors, but that functionality is available for you to use.



General

Controller Count: 1
Index To Use: 0

Left Trigger

Effect Type: No Resistance
Start Position: 0
End Position: 1
Begin Force: 0.7
Middle Force: 0.7
End Force: 0.7
Frequency: 0.2
Keep Effect: ✓

Right Trigger

Effect Type: No Resistance
Start Position: 0
End Position: 1
Begin Force: 0.7
Middle Force: 0.7
End Force: 0.7
Frequency: 0.2
Keep Effect: ✓

Light Bar

Enabled: ☐
Intensity: 1
Red: 1
Green: 0
Blue: 1

Player Lights

Left: ☐
Middle Left: ☐
Middle: ☐
Middle Right: ☐
Right: ☐
Fade: ☐

Rumble

Left: 0.0-1.0
Right: 0.0-1.0

The scene contains GameObject named 'Controller', which includes separate GameObjects to show the four different sample scripts included.

Controller Provider - DualSenseControllerProvider.cs

This script allows you to retrieve the number of DualSense controllers currently connected and the light of controllers to communicate with. It also allow you attach to an event that lets you know when controllers are disconnected or new controller are connected.

Controller Provider - SampleDualSenseInputUIController.cs

This script shows you how to detect when certain actions are taken on a DualSense controller. For example, you can see which buttons are pressed, how far a trigger is pulled, how much a stick is moved, and where fingers are touching the touch pad.

Controller Output - SampleDualSenseOutputUIController.cs

This script shows you how to change the configuration settings for various features of the DualSense controller. For example, you can change the resistance settings of the haptic triggers, the strength of the two rumble motors, and you can change how the controller lights up.

Controller Debugger - DualSenseDebugger.cs

This script is turned off by default. This script does a lot of similar things that the scripts above do but allows you to more easily see exactly what is going on with your DualSense controller. If you enable this script, you can print information to the screen and you use the inspector to see which controller inputs are being used as well as set the output state for things like changing the resistance of the haptic triggers and the strength of the rumble.

License

Portions of this software make use of a library named 'DualSense-Windows' to communicate with a DualSense controller. That software attribution is included below.

The MIT license below does not translate to the Unity plugin code that you are using right now. That code can only be distributed / sold as part of your software if you acquired it appropriately through the Unity Asset Store. It does not require further attribution, but that is always appreciated.