Introduction

Thank you for purchasing our plugin.

. SDL2InputDevice (SDL2ID) provides a new UnrealEngine 4 (UE4) input device that utilizes the Simple DirectMedia Layer (SDL2) library. This device can then be used to access or provide more advanced functionality than what is available in the engine by default. It was designed from the start to be as transparent to UE4 as possible.

By default, UE4 on windows utilizes Microsoft's XInput interface to provide controller input. This has several downsides, the biggest of which is the limit on the number of controllers that can activated at one time. This plugin was original designed to bypass that limitation and has grown to be a starting point to provide an additional functionality that the SDL provides.

SDL2InputDevice is as its namesake suggests a new input device. This solution means that input is passed into the various game systems of UE4 through the existing input pipeline. With SDL2InputDevice, there is no need for hacky blueprint glue code or special processing

SDL2 vs DirectInput

So why is it preferred to use SDL2 vs DirectInput for better control. The biggest reason is the SDL2 library is available on multiple platforms where DirectInput is available only for Windows. Additionally, SDL2 already contains a good database of game controllers and their mapping.

Usage

Since SDL2InputDevice provides a new input device to UE4, there is little that you need to do to utilize it. To enable the plugin, either select it via the plugins menu in the editor or add it manually to your project's **.uproject** file. Once the plug-in is enabled, SDL2 originated input events will occur via the input subsystem as usual. From a developer standpoint you do not need to do anything out of the ordinary.

However, you must keep in mind that SDL2ID exists concurrently with the default XInput system. This means by default for the first four game controllers, you will receive duplicate input events from the same physical controller via both systems. One event coming from XInput and one from SDL2InputDevice!

NOTE: To solve this, input coming from the SDL2ID plugin will have a ControllerId starting at 10.

You can explore the input system using a custom GameViewportClient and override the InputKey() and InputAxis() functions. These functions provide a convenient global access point to peek in on input.

C++ Classes and Interfaces

To work with the C++ classes and interface, you will want to add the SDL2InputDevice project to your PublicDependencyModuleNames in your project's **build.cs**. For example:

```
PublicDependencyModuleNames.AddRange(new string[]{
                    "Core",
                    "CoreUObject",
                    "Engine",
                    "InputCore",
                    "SDL2InputDevice"
            });
```

You can then access the SDL2InputDevice directly using the following code:

```
TSharedPtr<FSDL2InputDevice> SDLInputDevice = FSDL2InputDevicePlugin::Get().GetInputDevice();
if (SDLInputDevice.IsValid())
{
        // Do Something
}
```

Since input is provided via UE4's main subsystem, there really isn't anything you need to do in C++. Let us look at the public interface for the SDL2InputDevice:

bool IsGamepadAttached()

This function returns true if there is a Gamepad attached to the system.

void SetHapticFeedbackValues(int32 ControllerId, int32 Hand, const FHapticFeedbackValues& Values)

This function can be used to adjust the haptic feedback values for a given controller id. While you can call this function directly, it's better to use the APlayerController::SetHapticsByValue().

int32 GetNumberOfJoysticks()

Returns the total number of joysticks/gamepads being managed by the SDL2InputDevice

USDL2InputDeviceInfo* GetDeviceInfo(int32 ControllerId)

Returns the device info for a given controller id. See the section SDL2InputDeviceInfo below for more information.

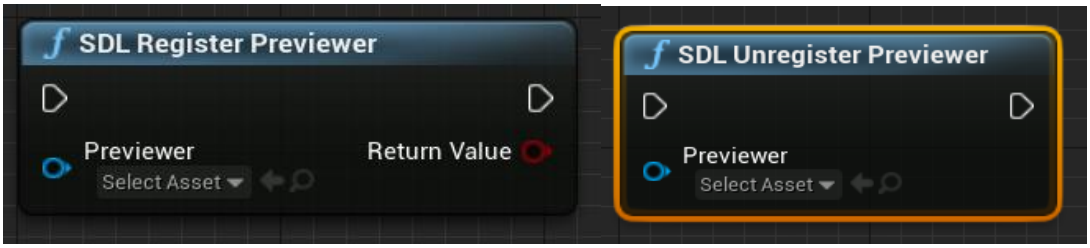
bool RegisterPreviewer(UObject* Previewer)
void UnregisterPreviewer(UObject* Previewer)

These two functions are used to register and unregister input preview objects. See the Key Notification Interface section below for more info on Input Previewers

Blueprint Library

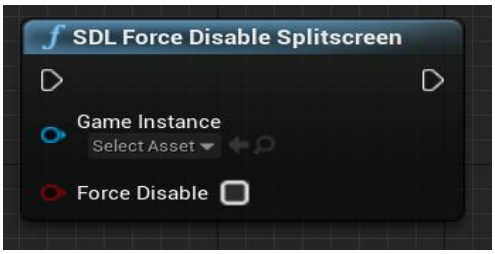
Blueprint access to the SDL2InputDevice plugin is provided by a Blueprint Library. While SDL2InputDevice provides input via the normal UE4 input subsystem, some of the critical infrastructure for effectively managing input devices is not accessible to Blueprints. Our Blueprint Library provides that functionality.

SDL Register Previewer and SDL Unregister Previewer



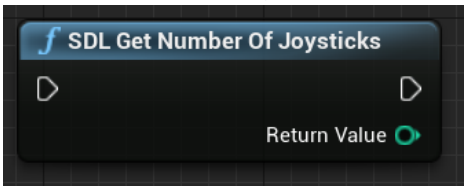
These two nodes are the two most used Blueprint nodes for tracking input. When you register an object that implements the Key Notification Interface (see below) you will have input events directed to the object for you to preview.

SDL Force Disable Splitscreen



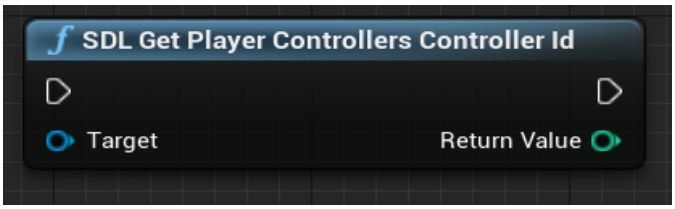
This node provides Blueprint access to the UGameViewportClient::SetForceDisableSplitscreen. This allows you to stop UE4 from automatically creating a splitscreen session when you add a player and is very useful if you are building a couch co-op style game.

SDL Get Number of Joysticks



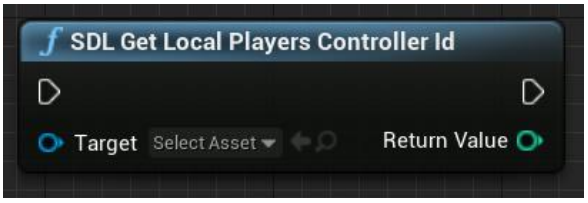
This is a simple utility function that queries the SDL2InputDevice and returns the # of joysticks/game controllers that are currently being tracked via SDL.

SDL Get Player Controllers Controller Id



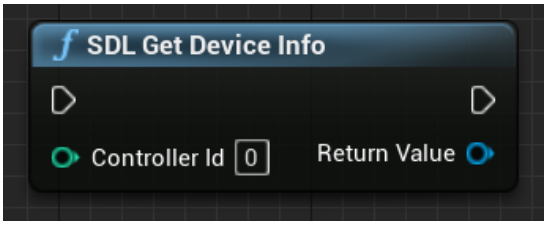
By default, UE4 does not expose a player controller's controller id to Blueprint. This node allows you to quickly get access to it.

SDL Get Local Player Controller Id



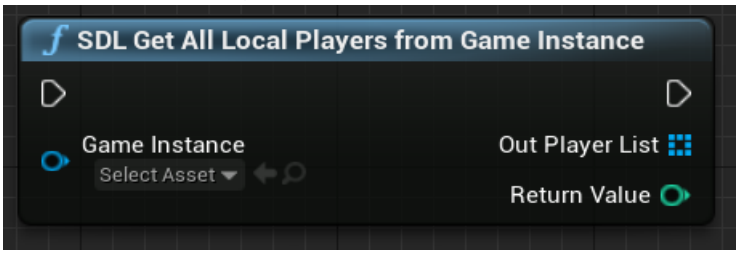
By default, UE4 does not expose a Local Player's controller id to Blueprint. This node allows you to quickly get access to it.

SDL Get Device Info



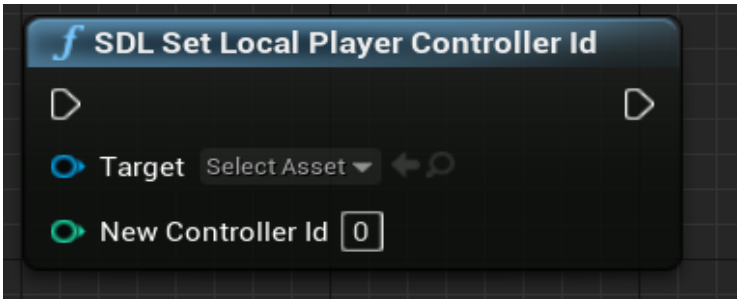
Given a controller id, this node will return a SDL Device Info for that controller id. See the section SDL2InputDeviceInfo below for more information on what it contains.

SDL Get All Local Players From Game Instance



This node provides a quick way to iterate over all the LocalPlayers present in the game.

SDL Set Local Player Controller Id



This is probably the most critical node that the Blueprint Library provides. Since SDL2InputDevice coexists with XInput but is loaded after the initial local player will never be associated with an SDL2 controller.

Key Notification Interface

SDL2InputDevice provides a new interface that makes previewing keys and getting notifications as simple as implementing a few events. Any UE4 Object that implements the event can register itself with the device to receive them. In C++ you call RegisterPreviewer() and UnregisterPreviewer() directly on the FSDL2InputDevice object. In Blueprints you utilize the SDL Register Previewer and SDL Unregister Previewer nodes.

The Key Notification Interface provides 4 functions that need to be implemented. These functions are defined as BlueprintNativeEvents so you can manage them at either native C++ or in Blueprints. They are:

Preview Input Key Events

```
UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = SDL2)
void SDL_PreviewInputKey(int32 ControllerId, FKey Key, EInputEvent Event,
       float AmountDepressed, bool bGamepad);
```

This event is triggered when a key event is sent via the UE4 Input Subsystem. You can use this preview to quickly manage the press without having to setup the various input binds. Typically, this is used to detect that first key when you do not already have a player controller / local player in place to manage it. It provides the **ControllerId** (starting at 10) of the input device, the **Key** value, the **Event** type (Pressed, Released, etc.) as well as **Amount Depressed** for analog keys and whether it is from a **GamePad.**

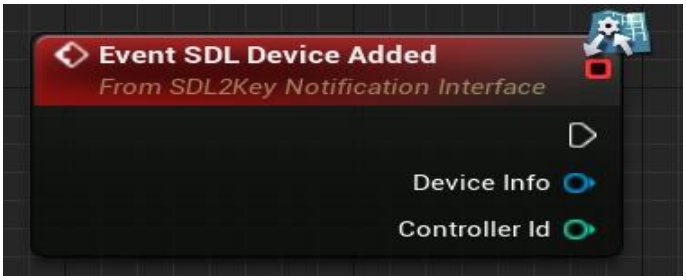
Preview Input Axis Event

```
UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = SDL2)
void SDL_PreviewInputAxis(int32 ControllerId, FKey Key, float Delta, float DeltaTime,
       int32 NumSamples, bool bGamepad);
```

This event is triggered when an analog axis is changed. You can use this event to get a preview of how a joystick is moving. It provides the **ControllerId** (starting at 10) of the input device, the **Key** value, the **Delta** which defines how much it has moved and the **DeltaTime** which is how long since the last update. **NumSamples** is currently unused.

SDL Device Added

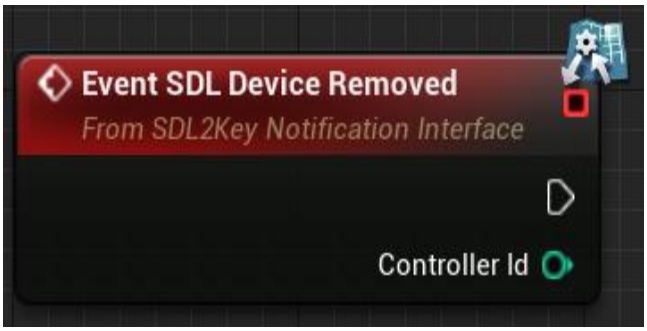
```
UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = SDL2)
void SDL_DeviceAdded(USDL2InputDeviceInfo* DeviceInfo, int32 ControllerId);
```



This event will be triggered every time a device is plugged in to the computer. It is typically used to reassign a controller when it is inadvertently unplugged or runs out of battery. It provides access to the **DeviceInfo** for the new device (See the section SDL2InputDeviceInfo below for more information on what it contains) and the **ControllerId** (starting at 10) of the input device.

SDL Device Removed

```
UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = SDL2)
void SDL_DeviceRemoved(int32 ControllerId);
```



The final event is the Device Removed event. It will be triggered whenever SDL2 detects that a managed device is removed from the computer. This typically happens when the batteries run out or the cable is unplugged. It provides the **ControllerId** (starting at 10) of the input device removed.

SDL2InputDeviceInfo

The USDL2InputDeviceInfo is meat of the controller support. Everyone joystick / game controller that is detected by SDL2 will have a DeviceInfo created for it. Let us look at its structure:

int32 InstanceID – This is the instance id that SDL2 uses to identify this device. If you are looking to modify the base code, you need to understand that the InstanceID can change under some circumstances and is not the same as the original DeviceIndex that is passed when a device is added.

int32 ControllerID – this is the UE4 ControllerId that is used to interface with the UE4 Input subsystem.

bool bSupportsHaptic – this value will be set to true if the joystick / game controller supports haptics.

int32 AxisCount – holds the # of axis values that this device supports.

int32 ButtonCount – holds the # of buttons this device supports.

int32 HatCount – holds the # of hats on this device (often it is the # of D-Pads).

int32 BallCount - holds the # of trackballs found on this device.

FString SDLDeviceName – holds the name of this device as provided by the SDL2 layer.

FString ControllerName – if the controller has an entry in the known controller table, this will hold its name.

FString CombiedID – is a string that holds a combined copy of the controller's Vendor Id and Product Id. It's used to look up the controller in the known controller table.

FString ControllerGUID - holds the GUID for this device as provided by the SDL2 layer.

Known Controllers

One other C++ structure that is important is the FKnownController struct. This struct stores human readable names for all the known joysticks / game controllers supported by the SDL layer. This list is maintained at the following github depot:

https://gist.github.com/nondebug/aec93dff7f0f1969f4cc2291b24a3171#file-known_gamepads-txt

Game Controller Mappings

SDL2 has a built-in ability to map devices as game controllers so that the devices return consistent key information. The layer comes preset with a list of controllers on each platform. SDL2InputDevice has a secondary list that is stored in the GameControllerDatabase directory of the plugin.

This datafile is loaded when the plugin is loaded so you can make changes to it to reflect devices that are missing. Refer to the SDL2 documentation for more information.

FAQ

Q. What platforms are supported?
A. Currently, only 32 and 64bit Windows are supported. However, this is more due to not having the required platforms to test. It should be trivial to get the plugin working on different platforms. If you would like to help ensure a port to alternate platforms works, please contact us.

Q. Is there example code available?
A. Yes, you can find an both a blueprint and C++ examples based on Epic's top-down sample on our github page located here:

> C++ Example based on the Top-Down Template
> https://github.com/JoeWilcox-WisEngineering/SDL2IDExample
>
> Blueprint only example based on the Top-Down Template
> https://github.com/JoeWilcox-WisEngineering/SDL2IDExampleBP

Q. How do I change the starting Controller Id for SDL2Input?
A. Change the define **STARTING_SDL_CONTROLLER_ID** in SDL2InputDEvice.h

Legal and Disclaimers