DualSense Windows API
Version 0.1

Ludwig Füchsl

November 25, 2020

Contents

1 Important information

1.1 Trademarks and affiliation

1.2 Sources

This work is derivative from others work. Special thanks goes to:

- GitHub user dogtopus: `https://gist.github.com/dogtopus/894da226d73afb3bdd195df41b3a26aa`.
- Reddit user ginkgobitter: `https://www.reddit.com/r/gamedev/comments/jumvi5/dualsense_haptics_leds_and_more_hid_output_report/`
- GitHub user Ryochan7: `https://github.com/Ryochan7/DS4Windows/tree/dualsense-integration`
  Copyright (c) 2019 Travis Nickles - MIT License: `https://github.com/Ryochan7/DS4Windows/blob/jay/LICENSE.txt`
- And the amazing community at DS4Windows `https://github.com/Ryochan7/DS4Windows/issues/1545`

1.3 License

MIT License

2 Introduction

Welcome to the DualSense on Windows API documentation. This API will help you integrating the Sony PS5 DualSense controller into your application or game for Windows. This document will guide you through the complete flow of integration. Starting with learning the DualSense Features up to deeply understanding every feature of this api.
The documentation is structured as follows:

- Descriptions of the DualSense's features
- installation and self compiling
- getting started guide
- api references

We recommend starting with reading through all features as far as you are not familiar with the DualSense controllers feature. Continuing with the installation and getting started guide to get your own demo application up and running. Then you can use the API references to integrate the api into your application.

3 Features

In the following section the features of the DualSense controller will be explained.

3.1 Overview

- **Connectivity** The DualSense controller can be used via Bluetooth or USB (USB C).
- **Integrated battery** Featuring an integrated battery the DualSense controller is best used via Bluetooth. The controller can be charged via USB type-C.

(a) Front View (b) Rear View

Figure 1: The DualSense controller

The DualSense controller features the following peripherals:

- Two XY-Axis analog sticks with integrated push button.
- Two adaptive triggers (are able to provide feedback).
- Two shoulder buttons.
- DPad with the ability to press two neighbor buttons simultaneously.
- The default Square, Cross, Circle and Triangle PlayStation buttons.
- Dual-touch touchpad with integrated push button, surrounded with five player indication LEDs on the bottom and RGB-LED lightbar on the sides.
- Menu, share, microphone mute and PlayStation button.
- 3-Axis Accelerometer and Gyroscope.

- Two rumble motors (Hard and Soft one). Can alternatively used as haptic feedback (not supported yet).
- Integrated speaker and microphone.
- Stereo audio jack.

3.2 Feature List

Analog sticks Each analog stick has two axis with 8-Bit precision each. The analog sticks will automatically return to their center position if released. They are mapped to the range -128 to 127 where 0 means center, -128 means left/bottom on X/Y-Axis and 127 means right/top on X/Y-Axis. Using the analog values requires the correction of the dead zones, because a released stick will most likely not have the value $R_{xy}(0;0)$ it will be a bit off. Same goes for the extreme values witch will also be off and not be exactly $T_{xy}(0;127)$, $L_{xy}(-128;0)$, etc..

Figure 2: Analog sticks

Adaptive trigger The DualSense controller feature two 8-Bit analog triggers. It is possible to read the trigger values as 8-Bit continuous values or alternatively as binary button input. Aside of the normal trigger operation the adaptive triggers can be configured to simulate various force feedback effects. It is possible for example to simulate a gun trigger.

Figure 3: Adaptive triggers

Bumpers The two L/R Bumpers located over the adaptive triggers can be read as normal button inputs.

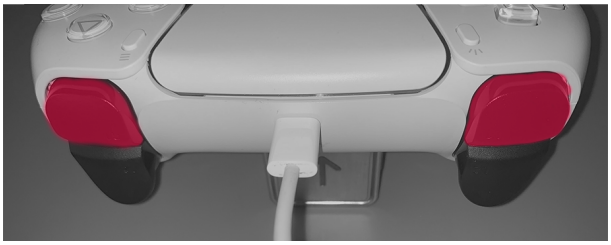

Figure 4: L/R Bumpers

DPAD and PS Buttons The DualSense controller feature a DPAD and the default well know PlayStation Square, Cross, Circle and Triangle buttons. The DPAD is capable of registering two simultaneously pressed buttons, however the two buttons must be neighbors. The PS-Buttons are being registered as four individual binary values.

Figure 5: DPAD and PS-Buttons

Other Buttons The DualSense controller feature several more buttons. Thees are:

- **Menu button** Should be used to open the in-game menu.
- **Share button** Should be used to open the in-game photo mode.
- **PlayStation button** Can be used to open a in-game overlay (Look at the know issues to get an additional use case of this button).
- **Mic button** Should be used to mute the microphone.

All the listed buttons are readable through individual binary values.

Figure 6: Left to right, top to bottom: Share, Menu, PlayStation and Mic Button

Touch Pad The touch-pad and the surrounding lightbar are featuring several functions:

- **Dual finger touch** The touch-pad itself is able to track two fingers simultaneously.
- **Integrated push button** The touch-pad integrates as momentary push button operated by pushing the pad down.
- **Lightbar** The left and right surrounding is able to light up in full 8-Bit RGB colors. The lib is providing several helpers to convert other color formats to the 8-Bit RGB UCHAR formate.
- **Player indication LEDs** On the bottom of the touch-pad are five player indication LEDs located. Thees LEDs are group in one left, three middle and one right led. The brightness of the LEDs is controllable and the LEDs are able to fade in.

When using the touch-pad make sure to implement hysteresis, dead zones and tolerances. It may also helpful to accumulate the values over multiple frame to get a more stable result but this will also increase latency.

Figure 7: Touch Pad

Accelerometer and Gyroscope The DualSense controller is able to measure its acceleration (By moving the controller around) and to track its rotation. Measurements are done with 16-Bit precision in all three XYZ-Axis. Make sure to implement hysteresis, dead zones and tolerances. It may also helpful to accumulate the values over multiple frame to get a more stable result but this will also increase latency.

- **Accelerometer** Measures the acceleration.
- **Gyroscope** Measures the controllers rotation. Currently you have to implement calibration in your own codebase.

Rumble motors / Haptic feedback The DualSense feature two Haptic feedback devices. Thees device work similar like normal speaker, but they are not good in producing tones, they are good in producing vibration. It is possible to send an audio signal directly to those haptic speakers (However currently not supported by this API).
The controller supports simulating the normal soft and hard rumble motors using the haptic speakers. When using this mode both motors can be controlled with the usual 8-Bit values. The left rumble feels hard, the right one soft.

Integrated speaker and microphone Featuring two microphones and one mono speaker, the DualSense is able to produce and pickup audio. This

features will be supported in the future. However it is possible to address thees devices with the default WASAPI independently.

Stereo audio jack Directly under the little microphone icon is a stereo headphone audio jack. It is possible to retrieve the connection status of this jack. However just like the speaker it is currently not supported to produce audio through the API.

4 Installation

We recommend using the prebuild dynamically linked release. But it is also possible to build the API from source. This might be the only way if you want to...

- statically link the library.
- get the latest (maybe unstable and undocumented) development release.
- sign and supply your company info inside the DLL info.
- make changes to the source code to better fit your application.

4.1 Prebuild installation

Before stating the integration process please make sure you have the latest release. You can find all releases here: `https://github.com/Ohjurot/DualSense-Windows/releases`

Inside the downloaded zip file you will find the following files:

```
DualSenseWindows.zip
├── ds5w.h
├── ds5w_x64.dll
├── ds5w_x64.lib
├── ds5w_x86.dll
├── ds5w_x86.lib
├── DualSenseWindows.pdf
└── LICENSE
```

The PDF file is the latest version of the document your are currently reading and LICENSE is a copy of the repository license file. All other files are required for integration into your project, thees steps will be explained next.

Including the header file When using the releases from the github page the single header `ds5w.h` is used. You can copy paste this file into the vendor directory of your project (The exact directory doesn't matter as far as it is accessible as a include directory).

Copying the binary DLL The API is coming with two different DLLs:

- **ds5w_x64.dll** Used when building x64 / 64-Bit projects.
- **ds5w_x86.dll** Used when building x86 / Win32 / 32-Bit projects.

Copy the DLL matching your target build to the output directory of your project.

Referencing the LIB file In order to compile your project successfully it is required to add the included lib file to the linker input. First you need to copy the lib file to your projects lib directory. Chose the lib file according to the name of the DLL you picked. After copying the file you have to add the files name to the linker input.

4.2 Self build installation

First you need to download or clone the project to your computer (We recommend to use the github tags to get the code that matches a specific release). Please check if you have the following prerequisites:

- VisualStudio `16.8.0` or later
- Windows SDK `10.0.19041.0` (Other version will probably work... Try to match the SDK version with the SDK version of your project!)
- Platform toolset `Visual Studio 2019 (v142)` Other version will probably work... Try to match with your project!)
- Windows Driver Kit (WDK) Version `10.0.19041.1` or later minor versions. (Should match your major SDK version. This part matters: 10.0.19041)

After downloading the project you can open the project by double-clicking the file `VS19_Solution\VS19_Solution.sln`
Building the project is very easy! Just execute a batch-built by selecting `Build → Batch Build → Select All → Build`. After building the library you can find several configurations under the following path `VS19_Solution\bin\DualSenseWindows\...`

- `DebugDll-x64 / DebugDll-x86` Debug DLL version for 64/32-Bit
- `ReleaseDll-x64 / ReleaseDll-x86` Release DLL version for 64/32-Bit

- `Debug-x64 / Debug-x64` Debug static link version for 64/32-Bit
- `Release-x64 / Release-x86` Release static link version for 64/32-Bit

Every self build version will be based on several header filed located in `VS19_Solution\DualSenseWindows\include\DualSenseWindows`.
Attention: When using the static link version your application needs additional linking to `hid.lib` and `setupapi.lib` from the Windows Driver Kit. It is also required to define `DS5W_USE_LIB` to prevent the DLL import attempt.

5 Quick start guide

In this section we will introduce you to the API step by step. You will need a DualSense controller connected via USB or Bluetooth.

5.1 Starting point

First start by creating the `main.cpp` file (assuming you already finished the installation steps from the former section). Include `ds5w.h` and create your main with an infinity loop inside.

Listing 1: DS5W Main

```
1 #include <Windows.h>
2 #include <ds5w.h>
3 #include <iostream>
4
5 int main(int argc, char** argv){
6
7     while(true){
8
9     }
10
11    return 0;
12 }
```

The library doesn't feature any global state or memory allocation, so it is not required to initialize the library. You can directly continue with the enumeration of all connected DualSense controllers.

5.2 Enumerate controllers

To enumerate DualSense controllers you need to supply an array of `DS5W::DeviceEnumInfo` or an array of pointers to `DS5W::DeviceEnumInfo` objects. The `DS5W::DeviceEnumInfo` object doesn't need any initializations, it will be initialized by the function you will call next. The function `DS5W::enumDevices(...)` will fill the supplied array with as many controllers as are available or the array can hold. Please consider looking at the API documentation to find the best way to robustly integrate that function call into your project.

Listing 2: Enumerate controllers

```
1 // ...
2 int main(int argc, char** argv){
3     // Array of controller infos
4     DS5W::DeviceEnumInfo infos[16];
5
6     // Number of controllers found
7     unsigned int controllersCount = 0;
8
9     // Call enumerate function and switch on return value
10    switch(DS5W::enumDevices(infos, 16, &controllersCount)){
11        // The buffer was not big enough. Ignore for now
12        case DS5W_E_INSUFFICIENT_BUFFER:
13            break;
14
15        // Any other error will terminate the application
16        default:
17            // Insert your error handling
18            return -1;
19    }
20 // ...
```

5.3 Selecting a controller

In this example we will select the first controller found. To enable a controller it is required to create a `DS5W::DeviceContext` context for it. The context will be initialized by the function call `DS5W::initDeviceContext(...)`. It is required to shutdown the controller after usage by calling `DS5W::freeDeviceContext(...)`.

Listing 3: Controller init / shutdown

```
1       // ...
2       // Check number of controllers
3       if(!controllersCount){
4           return -1;
5       }
6
7       // Context for controller
8       DS5W::DeviceContext con;
9
10      // Init controller and close application is failed
11      if(DS5W_FAILED(DS5W::initDeviceContext(&infos[0], &con))){
12          return -1;
13      }
14
15      // Main loop
16      while(true){
17          // ...
18      }
19
20      // Shutdown context
21      DS5W::freeDeviceContext(&con);
22  }
```

5.4 Reading controller input

The next step is read the input from the controller. We will check here if the PlayStation logo buttons is presses. When this is the case the application will exit the infinity loop and thus will shutdown. For reading the input data the `DS5W::DS5InputState` struct is required, it will be filled by the `DS5W::getDeviceInputState(...)` method call.

Listing 4: Reading the input

```
while(true){
    // Input state
    DS5W::DS5InputState inState;

    // Retrieve data
    if (DS5W_SUCCESS(DS5W::getDeviceInputState(&con, &inState))){
        // Check for the Logo button
        if(inState.buttonsB & DS5W_ISTATE_BTN_B_PLAYSTATION_LOGO){
            // Break from while loop
            break;
        }

        // ...
    }
}
```

5.5 Writing controller output

Writing to the controller is as simple as reading from it. It requires the struct `DS5W::DS5OutputState`, to prevent random data being sent please make sure to ZeroMemory the struct or setting every value. After setting all values you want to update, the data is written by calling the `DS5W::setDeviceOutputState(...)` function. In this example we will directly map the triggers input to the rumble motors output.

Listing 5: Writing the output

```
// ...

// Create struct and zero it
DS5W::DS5OutputState outState;
ZeroMemory(&outState, sizeof(DS5W::DS5OutputState));

// Set output data
outState.leftRumble = inState.leftTrigger;
outState.rightRumble = inState.rightTrigger;

// Send output to the controller
DS5W::setDeviceOutputState(&con, &outState);

// ...
```

Putting all the above code snippets together will give you a working example application.

6 API Reference

6.1 Preprocessor constants

Error codes

DS5W_OK
The operation completed without an error

DS5W_E_UNKNOWN
An unknown error occurred

DS5W_E_INSUFFICIENT_BUFFER
The user supplied buffer is to small

DS5W_E_EXTERNAL_WINAPI
An unsuspected Windows sided error occurred

DS5W_E_INVALID_ARGS
The user supplied arguments are invalid

DS5W_E_CURRENTLY_NOT_SUPPORTED
This feature is currently not supported

DS5W_E_DEVICE_REMOVED
The device was removed unexpectedly

DS5W_E_BT_COM
Bluetooth communication error

Error helpers

DS5W_SUCCESS(expr)
Check if the user supplied expression is an error success code

DS5W_FAILED(expr)
Check if the user supplied expression is an error code

I/O State helpers

DS5W_ISTATE_BTX_SQUARE
PlayStation Square button

DS5W_ISTATE_BTX_CROSS
PlayStation Cross button

DS5W_ISTATE_BTX_CIRCLE
PlayStation Circle button

DS5W_ISTATE_BTX_TRIANGLE
PlayStation Triangle button

DS5W_ISTATE_DPAD_LEFT
D-Pad left

DS5W_ISTATE_DPAD_DOWN
D-Pad down

DS5W_ISTATE_DPAD_RIGHT
D-Pad right

DS5W_ISTATE_DPAD_UP
D-Pad up

DS5W_ISTATE_BTN_A_LEFT_BUMPER
Left bumper button

DS5W_ISTATE_BTN_A_RIGHT_BUMPER
Right bumper button

DS5W_ISTATE_BTN_A_LEFT_TRIGGER
Left trigger binary input

DS5W_ISTATE_BTN_A_RIGHT_TRIGGER
Right trigger binary input

DS5W_ISTATE_BTN_A_SELECT
Select / Share button

DS5W_ISTATE_BTN_A_MENU
Menu Button

DS5W_ISTATE_BTN_A_LEFT_STICK
Left stick push button

DS5W_ISTATE_BTN_A_RIGHT_STICK
Right stick push button

DS5W_ISTATE_BTN_B_PLAYSTATION_LOGO
PlayStation logo button

DS5W_ISTATE_BTN_B_PAD_BUTTON
The touch-pads integrated button

DS5W_ISTATE_BTN_B_MIC_BUTTON
Microphone mute button

DS5W_OSTATE_PLAYER_LED_LEFT
Left player indicator LED bit-mask

DS5W_OSTATE_PLAYER_LED_MIDDLE_LEFT
Left middle player indicator LED bit-mask

DS5W_OSTATE_PLAYER_LED_MIDDLE
Middle player indicator LED bit-mask

DS5W_OSTATE_PLAYER_LED_MIDDLE_RIGHT
Right middle player indicator LED bit-mask

DS5W_OSTATE_PLAYER_LED_RIGHT
Right player indicator LED bit-mask

6.2 Types

DeviceEnumInfo This struct contains all internal data required for the controller enumeration. You should not read or write any of the internal data directly. The struct can be freely user allocated with random data. It will be initialized by the corresponding function call, don't use it before it got initialized by the corresponding function.

DeviceContext This struct contains all internal data for reading and writing to the controller. You should not read or write any of the internal data directly. The struct can be freely user allocated with random data. It will be initialized by the corresponding function call, don't use it before it got initialized by the corresponding function. It is very important to free this data with the corresponding function before the application exits or memory is reused.

AnalogStick This struct represent the XY position of one analog stick. Make sure to implement dead zones by yourself!

int8_t	x
X Position (left to right) of the analog stick.	

int8_t	y
Y Position (top to bottom) of the analog stick.	

Vector3, Vec3 Represents a three component 16-Bit vector

int16_t	x
X Component	

int16_t	y
Y Component	

int16_t	z
Z Component	

Color RGB 8-Bit color components. The library also provides several conversion functions to turn several color formats into 8-Bit RGB values.

uint8_t	r
R - Red color channel	

uint8_t	g
G - Green color channel	

uint8_t	b
B - Blue color channel	

Touch This struct contains information about a single fingers touch position.

int8_t	x
X Position of the finger (left to right).	

int8_t	y
Y Position of the finger (top to bottom).	

MicLed Enum class representation the state of the microphone LED.

`OFF`
Microphone LED is completely off

`ON`
Microphone LED is on

`PULSE`
Microphone LED is pulsing

TriggerEffectType Enum class: feedback / effect type of the adaptive trigger.

`NoResitance`
Adaptive trigger is disabled. Will provide no resistance.

`ContinuousResitance`
Adaptive trigger will provide a continuous resistance from a specific starting point.

`SectionResitance`
Adaptive trigger will provide a force fixed resistance on a defined section.

`EffectEx`
Adaptive trigger will execute an extended effect.

`Calibrate`
Adaptive trigger will enter an fixed function calibration program. Still experimental use only!

TriggerEffect This struct represents an adaptive trigger effect. The first param is the type. The other is a union over structs for each type.

TriggerEffectType	`effectType`
Type of the effect. Chose next data according to this parameter.	

When `effectType` == `NoResitance` no parameter needs to be set!
When `effectType` == `Calibrate` no parameter needs to be set!
When `effectType` == `ContinuousResitance`:

uint8_t	`Continuous.startPosition`
Start position of the continuous force.	

uint8_t	`Continuous.endPosition`
Force applied.	

When `effectType` == `SectionResitance`:

uint8_t	`Section.startPosition`
Start of force increased area.	

uint8_t	`Section.force`
End of force increased area.	

When `effectType` == `EffectEx`:

uint8_t	`EffectEx.startPosition`
Start positions of the effect.	

bool	`EffectEx.keepEffect`
Indicates weather the effect should keep playing (vibration) when the trigger is fully pressed.	

uint8_t	EffectEx.beginForce
Force for the section with trigger value ¿= 128.	

uint8_t	EffectEx.middleForce
Force for the section with trigger value ¡= 128.	

uint8_t	EffectEx.endForce
Force applied when the trigger is fully pressed / would go beyond 255.	

uint8_t	EffectEx.frequency
Frequency with witch the effect is executed. More a scalar value to scale between two fixed frequency than an read frequency parameter,	

LedBrightness Enum class representation the brightness of the player indication LEDs.

LOW
Low brightness player indication LEDs

MEDIUM
Medium brightness player indication LEDs

HIGH
High brightness player indication LEDs

PlayerLeds Struct defining the player LDEs state.

Bitmask / uint8_t	bitmask
Bitmask of the enabled player indication LEDs. Or together all enabled LEDs by using the DS5W_OSTATE_PLAYER_LED_XXXXX macros.	

bool	playerLedFade
Indicates weather the player LEDs should fade in when enabled.	

LedBrightness	brightness
Brightness of the player LEDs	

DS5InputState This struct represents the input state of a DualSense controller. It is used to read all input data.

AnalogStick	leftStick
Represents the position of the left analog stick.	

AnalogStick	rightStick
Represents the position of the right analog stick.	

uint8_t	leftTrigger
8-Bit position of the left trigger. No dead zones required!	

uint8_t	rightTrigger
8-Bit position of the right trigger. No dead zones required!	

Bitmask / uint8_t	buttonsAndDpad
Bitmask of the PlayStation button and the DPAD. Check the active state with the DS5W_ISTATE_BTX_XXXXX and DS5W_ISTATE_DPAD_XXXXX macros.	

Bitmask / uint8_t	buttonsA
Bitmask of the controllers buttons (Set A). Check the active state with the DS5W_ISTATE_BTN_A_XXXXX macros.	

Bitmask / uint8_t	buttonsB
Bitmask of the controllers buttons (Set B). Check the active state with the DS5W_ISTATE_BTN_B_XXXXX macros.	

Vector3	accelerometer
Acceleration vector.	

Vector3	gyroscope
Orientation vector.	

Touch	touchPoint1
First touch point.	

Touch	touchPoint2
Second touch point.	

bool	headPhoneConnected
Indicates weather a plug is present in the headphone jack. Will also trigger on an extension cord with no headphone connected!	

uint8_t	leftTriggerFeedback
Indicates the pressing force when the left adaptive trigger is active.	

uint8_t	rightTriggerFeedback
Indicates the pressing force when the right adaptive trigger is active.	

DS5OutputState This struct represents the output state of a DualSense controller. It is used to set all output data.

uint8_t	`leftRumble`
Force of the left (hard) rumble motor.	

uint8_t	`rightRumble`
Force of the right (soft) rumble motor.	

MicLed	`microphoneLed2`
State of the microphone LED.	

bool	`disableLeds`
When active the lightbar will be set to the default PS5 blue.	

PlayerLeds	`playerLeds`
State of the player LEDs.	

Color	`lightbar`
RGB Color of the lightbar. No affect when `disableLeds` is `true`	

TriggerEffect	`leftTriggerEffect`
Effect of the left adaptive trigger.	

TriggerEffect	`rightTriggerEffect`
Effect of the right adaptive trigger.	

6.3 Functions

DS5W::enumDevices(...) TODO

DS5W::initDeviceContext(...) TODO

DS5W::freeDeviceContext(...) TODO

DS5W::reconnectDevice(...) TODO

DS5W::getDeviceInputState(...) TODO

DS5W::setDeviceOutputState(...) TODO

DS5W::color_R32G32B32_FLOAT(...) TODO

DS5W::color_R32G32B32A32_FLOAT(...) TODO

DS5W::color_R8G8B8A8_UCHAR(...) TODO

DS5W::color_R8G8B8_UCHAR_A32_FLOAT(...) TODO