DualSense Windows API
Version 0.3

Ludwig Füchsl Matthew Hall

April 29, 2022

Contents

1 Important information

1.1 Trademarks and affiliation

"PlayStation", "PlayStation Family Mark", "PS5 logo", "PS5", "DualSense" and "DUALSHOCK" are registered trademarks or trademarks of Sony Interactive Entertainment Inc. "SONY" is a registered trademark of Sony Corporation.

"Windows" is a registered trademark of Microsoft Corporation.

The Author is not affiliated in any kind with Sony Interactive Entertainment Inc.!
The Author is not affiliated in any kind with Microsoft Corporation!
Using this library may void your / your clients / your users / your customers controllers warranty! You as the redistributor of the precompiled or self compiled library have to make sure the controller will not be damaged by the functionality you use or at least point out the possible risk to your users / clients / customer!

1.2 Sources

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

- The original developer Ohjurot: `https://github.com/Ohjurot/DualSense-Windows`.
- 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

Info

This library is a fork of, "DualSense Windows API" by Ludwig Füchsl. As such, portions of the project are held by the original copyright: Ludwig Füchsl (2020).

Only the implementation is licensed under the MIT license. The protocol of the DualSense 5 controller has been discovered independently and collectively by several people. You can find my sources in the GitHub repository.

2 Introduction

Welcome to the DualSense Windows API documentation. This API will help you with 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 from an overview of the DualSense controller, up to using the advanced features of this API.

The documentation is structured as follows:

- Device Features
- Installation
- Quick Start Guide
- API Reference

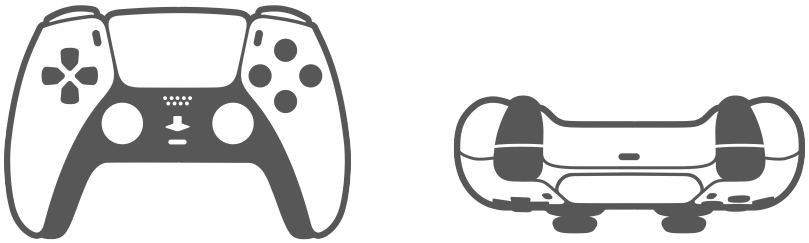
If you are not familiar with the DualSense controller, we recommend starting by reading through the list of controller features before continuing with the installation and getting started guide. Then you can use the API reference to integrate this library into your application.

3 Device 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 a USB C cable or Bluetooth.
- **Integrated battery** Featuring an integrated battery, the DualSense controller is best used via Bluetooth. The controller can be charged via USB C.



(a) Front View (b) Rear View

Figure 1: The DualSense controller

The DualSense controller conatins the following features:

- Two XY-Axis analog sticks with integrated push button.
- Two adaptive triggers (are able to provide resistance feedback).
- Two shoulder buttons.
- Directional-pad with the ability to press up to two neighboring buttons simultaneously.
- Four circular face buttons (Square, Cross, Circle and Triangle).
- Dual-touch touchpad with integrated push button, surrounded by an RGB-LED lightbar and a five LED player indicator on the bottom.
- Menu, share, PlayStation and microphone mute button.
- 6-Axis IMU (gyroscope and accelerometer).

- Left and right rumble motors (Hard and Soft respectively). Can alternatively used as haptic feedback (not supported).
- Integrated speaker and microphone (both not supported).
- Stereo audio jack (not supported).

3.2 Feature List

Analog sticks Each analog stick has two axes with 8-Bit precision. The analog sticks will return to their centered position if released. Output values are mapped to the range -128 to 127 where 0 indicates the stick is centered. The minimum values are located on the left/bottom of the respective X/Y-Axis and maximum values at the right/top. For best results the analog values should incorporate a dead zone as analog sticks are notorious for not resetting exactly to $R_{xy}(0;0)$. 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 features two 8-Bit analog triggers. It is possible to read each trigger's value as an 8-Bit continuous value (0-255) or alternatively as a binary button input. Aside of the normal trigger operation, the adaptive triggers can be configured to simulate various force feedback effects, e.g. to simulate a gun trigger.

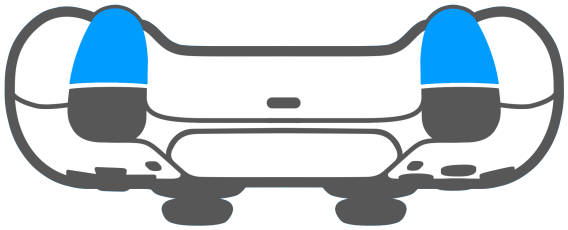


Figure 3: Adaptive triggers

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

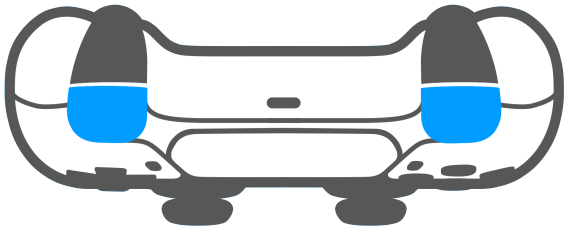


Figure 4: L/R Bumpers

DPAD and PS Buttons The DualSense controller face features a four-directional DPAD and the well known PlayStation buttons: Square, Cross, Circle and Triangle. The DPAD is capable of registering two simultaneously pressed buttons, however the two buttons must be neighbors. The PS-Buttons are registered as four individual binary values and can all be pressed simultaneously.

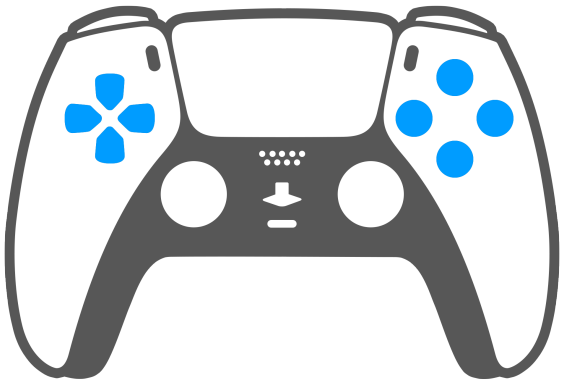


Figure 5: DPAD and PS-Buttons

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

- **Menu button** Often used to open the in-game menu.
- **Share button** Often used to open the in-game photo mode.
- **PlayStation button** Can be used to open a in-game overlay (also used to shutdown the device in Bluetooth mode if long-pressed).
- **Mic button** Should be used to mute the microphone.

All the listed buttons are readable as 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 have featuring several functions:

- **Dual finger touch** The touch-pad itself is able to track two fingers simultaneously as well as know how many times it has been touched (*loops after 127 touches).
- **Integrated push button** The touch-pad includes a binary button operated by pushing the pad down.
- **Lightbar** The left and right surrounding is able to light up in full 8-Bit RGB colors.
- **Player indication LEDs** On the bottom of the touch-pad are five player indication LEDs located. These 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 controller is able to measure its positional acceleration and angular velocity. 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 positional acceleration (meters/second).

- **Gyroscope** Measures the angular velocity (degrees/second).

These values should be calibrated in your own code base, particularly the gyroscope which usually has per axis biases.

Rumble motors / Haptic feedback The ccontroller feature two Haptic feedback devices. Thees device work similar like normal speakers, 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. These features are not currently supported by this API, however it is possible to address these devices through Windows independently.

Stereo audio jack Directly under the microphone is a stereo, 3.5mm audio jack. It is possible to retrieve the connection status of this jack. However, like the speaker it is currently not supported by the API.

Internal timer The DualSense controller has an inbuilt timer which can be used to more accurately integrate the data from the accelerometer and gyroscope. The timer starts when the device is connected and is measured in units of 0.33 microseconds. The value is stored as a 32-Bit unsigned integer which will overflow after roughly 72 minutes and restart from 0. This API accounts for the overflow and calculates the correct delta time when reading the input state.

4 Installation

We recommend using the prebuilt 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.
- make changes to the source code to better fit your application.

4.1 Prebuilt installation

Before starting the installation process please make sure you have the latest release. You can find all releases here: `https://github.com/mattdevv/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` should be 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 comes 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 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 file to project's 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 `DualSenseWindows\DualSenseWindows.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 `DualSenseWindows\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 `DualSenseWindows\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        case DS5W_OK:
12        // Can ignore, just there were more controllers
              found than slots in the buffer
13        case DS5W_E_INSUFFICIENT_BUFFER:
14            break;
15
16        // Any other error will terminate the
            application
17        default:
18            // Insert your error handling
19            return -1;
20    }
21 // ...
```

5.3 Selecting a controller

In this example we will select the first controller found to connect to. 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 if the PS button flag is set on
        if (inState.buttonMap & DS5W_ISTATE_BTN_SELECT)
            {
            // 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 names + codes

DS5W_OK 0
The operation completed without an error

DS5W_E_UNKNOWN 1
An unknown error occurred

DS5W_E_INSUFFICIENT_BUFFER 2
The user supplied buffer is to small

DS5W_E_EXTERNAL_WINAPI 3
An unsuspected Windows sided error occurred

DS5W_E_STACK_OVERFLOW 4
The API tried to allocate memory on the stack but failed

DS5W_E_INVALID_ARGS 5
The user supplied arguments are invalid

DS5W_E_CURRENTLY_NOT_SUPPORTED 6
This feature is currently not supported

DS5W_E_DEVICE_REMOVED 7
The device is not connected either physically or digitally

DS5W_E_BT_COM 8
Bluetooth communication error

DS5W_E_IO_TIMEOUT 9
Windows IO operation was cancelled due to taking too long

DS5W_E_IO_FAILED 10
Windows IO operation failed

DS5W_E_IO_NOT_FOUND 11
Windows IO operation was missing

DS5W_E_IO_PENDING 12
Windows IO operation did not complete but is running in background

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_BTN_SQUARE
PlayStation Square button

DS5W_ISTATE_BTN_CROSS
PlayStation Cross button

DS5W_ISTATE_BTN_CIRCLE
PlayStation Circle button

DS5W_ISTATE_BTN_TRIANGLE
PlayStation Triangle button

DS5W_ISTATE_BTN_DPAD_LEFT
D-Pad left

DS5W_ISTATE_BTN_DPAD_DOWN
D-Pad down

DS5W_ISTATE_BTN_DPAD_RIGHT
D-Pad right

DS5W_ISTATE_BTN_DPAD_UP
D-Pad up

DS5W_ISTATE_BTN_BUMPER_LEFT
Left bumper button

DS5W_ISTATE_BTN_BUMPER_RIGHT
Right bumper button

DS5W_ISTATE_BTN_TRIGGER_LEFT
Left trigger binary input

DS5W_ISTATE_BTN_TRIGGER_RIGHT
Right trigger binary input

DS5W_ISTATE_BTN_SELECT
Select / Share button

DS5W_ISTATE_BTN_MENU
Menu Button

DS5W_ISTATE_BTN_STICK_LEFT
Left stick push button

DS5W_ISTATE_BTN_STICK_RIGHT
Right stick push button

DS5W_ISTATE_BTN_PLAYSTATION_LOGO
PlayStation logo button

DS5W_ISTATE_BTN_PAD_BUTTON
The touch-pads integrated button

DS5W_ISTATE_BTN_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).	

bool	`down`
Whether a finger is touching the touchpad	

uint8_t	`id`
7-Bit id of last touch (resets after 127). Counter is shared by each touch point and is not unique.	

Battery This struct contains information about the battery level. (work in progress)

bool	`charging`
Whether the device is currently charging.	

boolt	`fullyCharged`
Whether the battery is full	

uint8_t	`level`
current battery level (0-100)	

MicLed Enum class representation the state of the orange microphone LED.

`OFF`
Microphone LED is 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.

`ReleaseAll`
Adaptive trigger will disable and release any active effects immediately.

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

TriggerEffect This struct represents an adaptive trigger effect. The struct is 11 bytes long with the first byte being the effect type and the remaining being parameters. Not all parameters are used in trigger effects.

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 real 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 parsed form of the raw input state from a DualSense controller. It is used to easily read input data from your own programs.

AnalogStick	`leftStick`
Represents the position of the left analog stick.	

AnalogStick	`rightStick`
Represents the position of the right analog stick.	

Bitmask / uint32_t	`buttonsMap`
Bitmask of all binary buttons. Check individual button states by doing bitwise & with button flag macros.	

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!	

Vector3	`accelerometer`
Positional acceleration vector (m/s).	

Vector3	`gyroscope`
Angular velocity vector (degrees/s).	

Touch	`touchPoint1`
First touch point.	

Touch	`touchPoint2`
Second touch point.	

uint32_t	`currentTime`
Time that the input state was read at (measured in 0.33 microseconds)	

uint32_t	`deltaTime`
Time since the previous input state was read (measured in 0.33 microseconds)	

Battery	`battery`
Battery info.	

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.	

uint8_t	`rumbleStrength`
Strength of the rumble/haptic (trigger) motors. First 4 bits (0-3) represent the rumble motors and last 4 (4-7) represent the trigger's haptics strength.	

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(...) Fill array with enumerable list of DualSense devices.

DS5W::enumUnknownDevices(...) Fill array with enumerable list of DualSense devices without doubles by passing in an array of known device IDs

DS5W::initDeviceContext(...) Initializes the context for a specific controller.

DS5W::freeDeviceContext(...) Frees a context from a controller witch is no longer required. Context cannot be reconnected and must be re-enumerated to be used again.

DS5W::shutDownDevice(...) Takes a working DualSense controller and releases it in Windows but does not forget it. Device can be reconnected again.

DS5W::reconnectDevice(...) Tries to reconnect a disconnected device. Device could have been lost due to a shutdown or been unplugged.

DS5W::getDeviceInputState(...) Retrieve the current input state of the device. Blocks thread for up to 100 milliseconds until input is received and parsed.

DS5W::setDeviceOutputState(...) Sets the desired output state of the device. Blocks thread for up to 100 milliseconds until output is parsed and sent.

DS5W::startInputRequest(...) Begin a non-blocking request for an input report. Returns whether report was read instantly, or Windows is waiting for next report.

DS5W::awaitInputRequest(...) Blocks thread until previous input request was fulfilled (or timeout).

DS5W::getHeldInputState(...) Parses internal buffer into an input state. Uses whatever content was last read. Intended for use after startInputRequest/getHeldInputState.

DS5W::color_R32G32B32_FLOAT(...) Converts a three component (RGB) normalized float color to the internal RGB 8-Bit formate.

DS5W::color_R32G32B32A32_FLOAT(...) Converts a four component (RGBA) normalized float color to the internal RGB 8-Bit formate.

DS5W::color_R8G8B8A8_UCHAR(...) Converts a four component (RGBA) unsigned char color (8-Bit) to the internal RGB 8-Bit formate.

DS5W::color_R8G8B8_UCHAR_A32_FLOAT(...) Scales a three component (RGB) unsigned char color (8-Bit) by a normalized float (A).

7 Advanced Examples

In this section some more advanced features will be explained with examples.

7.1 Enumerate Unknown Devices

During your program you may wish to check for new DualSense controllers which have been connected since the program began. The quick start guide shows how to use `DS5W::enumDevices(...)` to get a list of controllers, but this function lists all DualSense controllers not just the new ones. To solve this problem the API generates a unique ID for every controller and stores it in `DS5W::DeviceEnumInfo`. Once all the IDs of known controllers are placed in an array, a pointer to the array and the count of known controllers can be used to call `DS5W::enumKnownDevices(...)` which gets a list of devices excluding those with IDs found in the input array.

Listing 6: Enumerate Unknown Controllers

```
1     // ...
2     // Following on from original example of Enumerate
         Devices
3     // some time has passed and new devices might have
         been connected
4
5     unsigned int numKnownDevices = controllersCount;
6     unsigned int* knownIDs = new unsigned int[
         numKnownDevices];
7     for (int i = 0; i < controllersCount; i++) {
8         knownIDs[i] = infos[i]._internal.uniqueID;
9     }
10
11    // Call enumerate function again with extra
         parameters and switch on return value
12    // start of buffer is now after found devices
13    // remaining size has shrunk
14    switch (DS5W::enumUnknownDevices(&infos[
         numKnownDevices], 16 - numKnownDevices, knownIDs
         , numKnownDevices, &controllersCount)) {
15    case DS5W_OK:
16        // Can ignore, just there were more controllers
             found than slots in the buffer
17    case DS5W_E_INSUFFICIENT_BUFFER:
18        break;
19        // Any other error will terminate the
             application
20    default:
21        // Insert your error handling
22        return -1;
23    }
24
25    delete[] knownIDs;
26
27    // total num devices has increased
28    numKnownDevices += controllersCount;
29 // ...
```

7.2 Overlapped IO

Reading and writing to the device blocks the calling thread until the request is fulfilled. To try and reduce the amount of time waiting overlapped IO can be used. Overlapped functions allow running other code while the IO request is processed in the background by the OS. It has only been implemented here for input as output does not seem to cause bottlenecks.

As a note, the term "input report" means the raw input data straight from the controller. An "input state" is the processed version outputted by this API.

Listing 7: Overlapped IO

```
1     // Input state
2     DS5W::DS5InputState inState;
3
4     while (true) {
5         // start request for input report
6         DS5W_ReturnValue err = DS5W::startInputRequest
              (&con);
7
8         // request has started but could have failed
              already
9         // check if needed
10
11        // can run code while waiting
12        // don't run code longer than the device
              polling rate as that will add latency
13        DoExtraWork();
14
15        // The pending error says that the request
              started, but did not finish instantly
16        // wait for it to finish here
17        if (err == DS5W_E_IO_PENDING) {
18            // will get a new error code too
19            err = DS5W::awaitInputRequest(&con);
20        }
21
22        // Check that request completed correctly
23        if (DS5W_SUCCESS(err)) {
24            // get the newly recieved input state
25            DS5W::getHeldInputState(&con, &inState);
26
27            // can continue as normal now
28            // ...
29        }
30    }
31 // ...
```