3dRudder SDK

02/16/2017

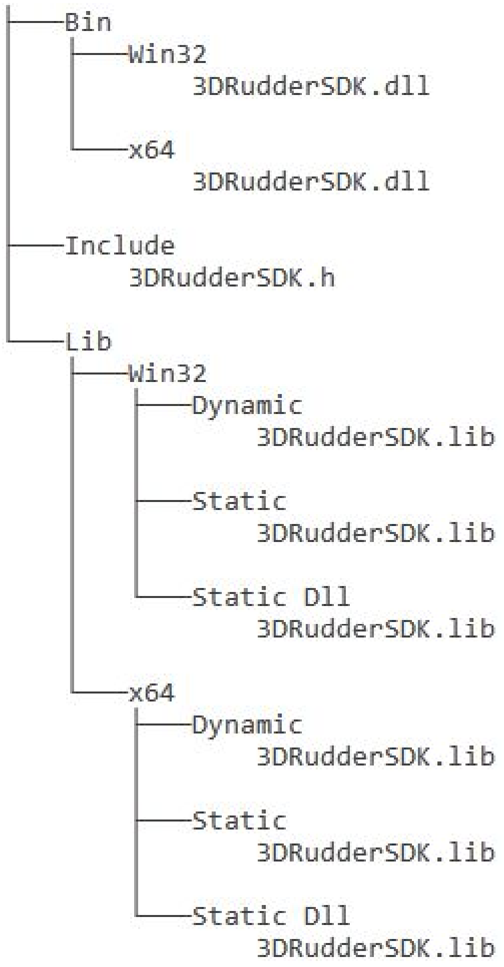
Version 0.7 for Windows

Warning: this version of the SDK works with firmware version 1.3.x.x and higher !

If you have 3dRudder version with the firmware 1.2.x.x or older, please contact us to get the updating software: support-dev@3drudder.com

3dRudder SDK

SDK Organization



```
├───Bin
│   ├───Win32
│   │       3DRudderSDK.dll
│   │
│   └───x64
│           3DRudderSDK.dll
│
├───Include
│       3DRudderSDK.h
│
└───Lib
    ├───Win32
    │   ├───Dynamic
    │   │       3DRudderSDK.lib
    │   │
    │   ├───Static
    │   │       3DRudderSDK.lib
    │   │
    │   └───Static Dll
    │           3DRudderSDK.lib
    │
    └───x64
        ├───Dynamic
        │       3DRudderSDK.lib
        │
        ├───Static
        │       3DRudderSDK.lib
        │
        └───Static Dll
                3DRudderSDK.lib
```

Type of library available

- With this release of the SDK, we provide the static and dynamic libraries in 32 and 64 bits.
- We only provide the multi-threaded version.
- The “Static Dll” is a static library with the DLL CRT compilation (/MD).
- Visual Studio 2015 has been used to compile these libraries.

Static library usage

To use the static libraries you need to define `_3DRUDDER_SDK_STATIC` to avoid `dllimport`

SDK Usage

Include the SDK definition

```
#include "3DRudderSDK.h"
```

Standard lib usage

The SDK uses stdint.h for the types definition.

3dRudder NameSpace

The SDK uses the namespace `ns3DRudder`

Get the SDK Class pointer

```
ns3DRudder::CSdk* pSdk=ns3DRudder::GetSDK();
```

Free the SDK

```
void ns3DRudder::EndSDK();
```

will free the sdk from memory.

SDK Reference

All the SDK is defined in the class `ns3DRudder::CSdk`. With this SDK it's possible to manage up to four 3DRudder `_3DRUDDER_SDK_MAX_DEVICE` defines the max ports number (from 0 to 3).

BASIC FUNCTIONS DESCRIPTION

Initialize the SDK

```
void Init()  const
```

Initializes the SDK.

Get the sdk version

```
uint16_t GetSDKVersion() const
```

Return the SDK version of the library, it's possible to compare this version with the `_3DRUDDER_SDK_VERSION` define included in the 3DRudderSDK.h to compare if the library and the .h match. The version is a fixed point unsigned short in hexadecimal: 0x0070 means version 0.7.

Get the number of connected 3DRudder devices

```
int32_t GetNumberOfConnectedDevice() const
```

Return the number of 3DRudder currently connected to the computer.

Check if a 3DRudder is connected to the port

```
bool IsDeviceConnected(uint32_t nPortNumber) const
```

Return true if a 3DRudder is connected to the `nPortNumber` port.

Get the Firmware version of a 3DRudder

```
uint16_t GetVersion(uint32_t nPortNumber) const
```

Return version number of the firmware of the 3DRudder connected to the `nPortNumber` port. The version is a fixed point unsigned short in hexadecimal: 0x1318 means version 1.3.1.8

Return 0xFFFF in case of error.

Play a sound on a 3DRudder

```
ErrorCode PlaySnd(uint32_t nPortNumber,uint16_t nFrequency,uint16_t nDuration) const
```

It's possible to play a sound on a 3DRudder connected to the `nPortNumber` port.

`nFrequency` defines the frequency of the sound in Hz (440 is a A).

`nDuration` defines the duration of the sound in ms.

`ErrorCode` is the possible error code returned by this method.

Hide the device

```
ErrorCode HideSystemDevice(uint32_t nPortNumber,bool bHide) const
```

By default the 3dRudder is seen by the system as a Directinput device, a mouse or a keyboard (this can be changed thanks to the dashboard).
The function HideSystemDevice allows to hide the 3dRudder from the system, so your game will not see it as a DirectInput device. Please think to put it back in standard mode when you exit your game !

`ErrorCode` is the possible error code returned by this method.

Test if the device is hidden

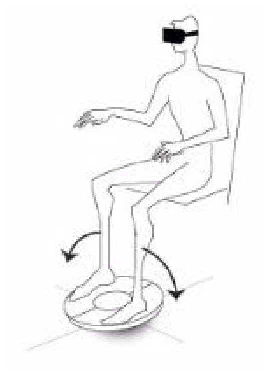
```
bool IsSystemDeviceHidden(uint32_t nPortNumber) const
```

Check if the device connected to `nPortNumber` is hidden

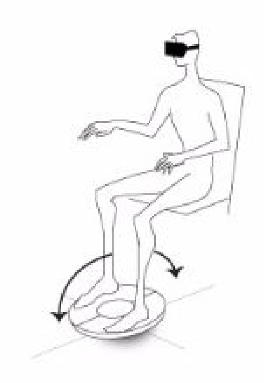
3D Axis definitions

The physical actions on the 3dRudder are converted from angle to move or rotation on 3D environment.

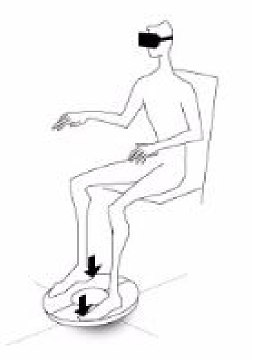
The physical actions are :



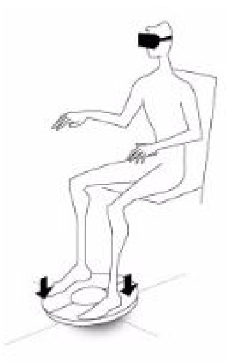
Roll



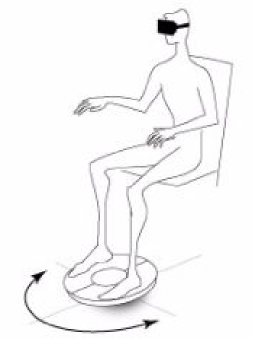
Pitch



Up

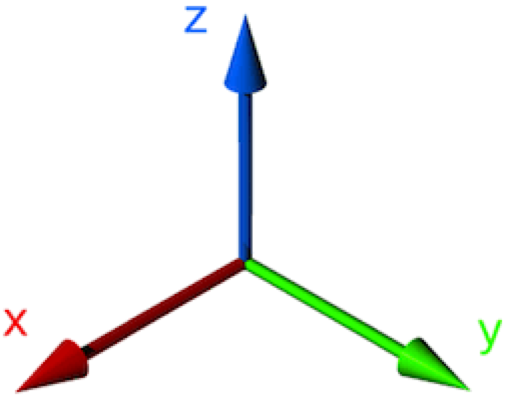


Down

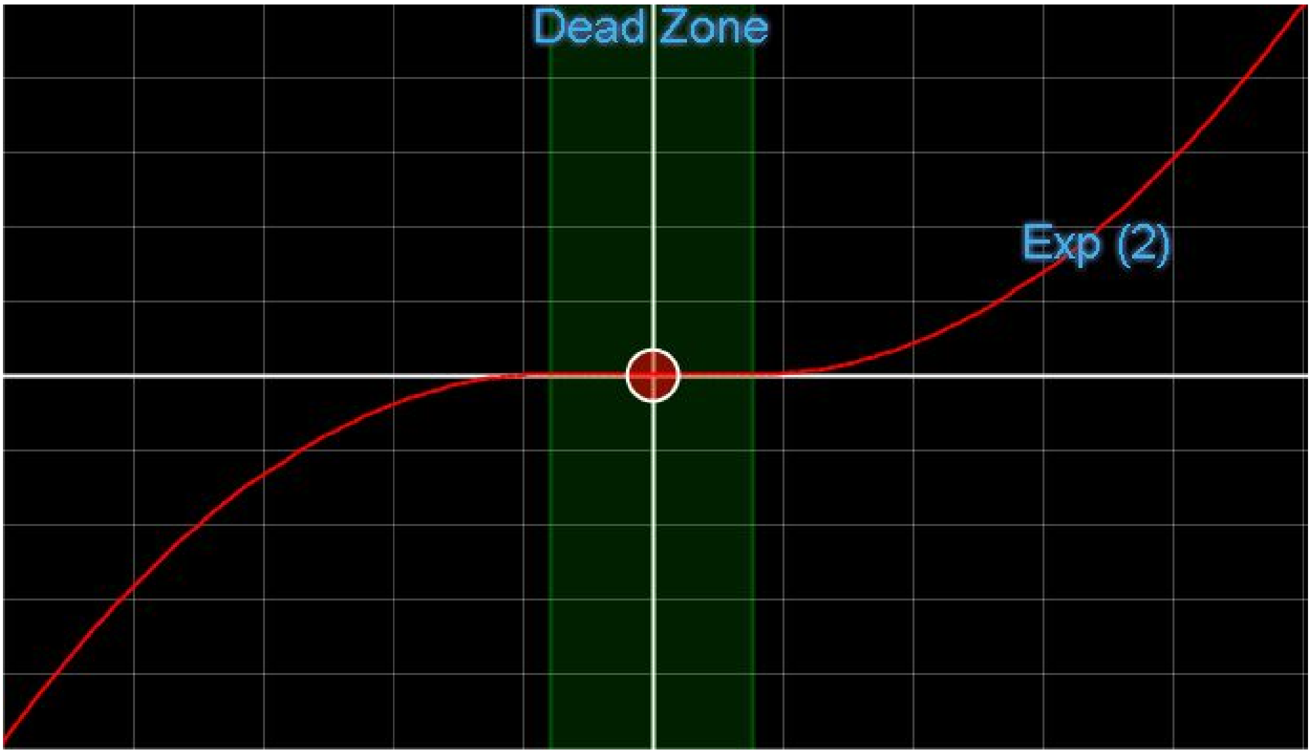


Yaw

This is the 3D axis définition used by the 3dRudder to move or rotate on the 3D world :



Curves



Note : before release 0.6 of the SDK, this functionality wasn't enabled.

The SDK manages the moving curves to help you to do the gameplay of your game. You have 4 parameters by curve but you will generally use mainly 2 as the others are the limits in and out.

The parameters are :

- **Dead Zone:** zone where the input has no impact on the output.
- **Exp:** exponent of the curve, Exponent 1 is linear, Exponent 2 is a square curve, etc.
- **xSat:** input limit, it's generally linked to the physical limit of the 3dRudder (for the Roll/X Axis and the Pitch/Y Axis) or of the human body (for the Yaw/Z Rotation).
- **yMax:** output limit, as we work in normalized values, this value is generally fixed to 1.0

There is one curve for each axis, defined in `CurveType` :

`CurveXAxis or CurveRoll` X Axis curve
`CurveYAxis or CurvePitch` YAxis curve
`CurveZAxis or CurveUpDown` ZAxis Curve
`CurveZRotation or CurveYaw` Z Rotation Curve

Custom Curve

It's possible to define your own custom curve by doing a derivation of the method :

```
virtual float CalcCurveValue(float fValue) const
```

of the class `Curve.`

Read DirectInput Mode Curve saved in the 3dRudder

```
ErrorCode ReadCurveFromDevice(uint32_t nPortNumber,CurveType nType,Curve *pCurve) const
```

This function reads the curve `CurveType` saved in the 3dRudder connected to the `nPortNumber` and saves it to the class `Curve`.

Write Curve in the DirectInput Mode Curve of the 3dRudder

```
ErrorCode WriteCurveToDevice(uint32_t nPortNumber,CurveType nType,Curve *pCurve) const
```

This function writes the curve `CurveType` in the 3dRudder connected to the `nPortNumber` from the class `Curve`.

This function doesn't save the value in the flash memory of the 3dRudder so it will be reset if you unplug the device.

Read the current status of the 3dRudder

```
Status GetStatus(uint32_t nPortNumber) const
```

This function read the current status of the 3dRudder connected to `nPortNumber`, the possible values of the `Status` are :

`NoFootStayStill`:

Puts the 3DRudder on the floor, curved side below, without putting your feet on the device. The user waits for approx. 5 seconds for the 3DRudder to boot up until 3 short beeps are heard.

`Initialisation`:

The 3DRudder initialize for about 2 seconds. Once done a long beep will be heard from the device. The 3DRudder is then operational.

`PutYourFeet`:

Put your first feet on the 3DRudder.

`PutSecondFoot`:

Put your second Foot on the 3DRudder.

`StayStill`:

The user must wait still for half a second for user calibration until a last short beep is heard from the device. The 3DRudder is then ready to be used.

`InUse`:

The 3DRudder is in use.

`ExtendedMode:`

The 3DRudder is in use and is fully operational with all the features enabled.

Read the User Offset Value

```
ErrorCode GetUserOffset(uint32_t nPortNumber,Axis *pAxis) const
```

This function reads the User Offset Value, i.e. the value (saved in `Axis`) of the yaw, pitch, roll and updown when the user is in its neutral position : those values could be used to calculate the Non Symmetrical Pitch value for instance.

Read Force Sensor Value

```
uint16_t GetSensor(uint32_t nPortNumber,uint32_t nIndex) const
```

This function reads the values of the 6 force sensors indexed by `nIndex` of the 3dRudder connected on `nPortNumber`. The unit of 16 bits returned value is given in grams.

Get Axis Value

```
ErrorCode GetAxis(uint32_t nPortNumber,ModeAxis nMode,Axis* pAxis,const CurveArray* pCurve) const
```

This function reads the values of the `Axis,` with the current `ModeAxis` with the optional usage of the curves defined by `CurveArray` for the 3dRudder Connected to `nPortNumber.` The values are only valid if the status is `InUse` or `ExtendedMode`.

`ErrorCode` is the possible error code returned by this method.

The class `Axis` contains the value of the Axis :

```
class Axis
{
public:
        float m_aX;
        float m_aY;
        float m_aZ;
        float m_rZ;
};
```

m_aX is the X Axis (you can use `GetXAxis()` or `GetPhysicalRoll()` to read it)
m_aY is the Y Axis (you can use `GetYAxis()` or `GetPhysicalPitch()` to read it)
m_aZ is the Z Axis (you can use `GetZAxis()` or `GetUpDown()` to read it)
m_rZ is the Z Rotation (you can use `GetZRotation()` or `GetPhysicalYaw()` read it)

The class `CurveArray` defines the setting of curve of each axis.

WARNING : since SDK version 0.7, the input values of the curves are normalized
This means that the x values should be in the range -1/1, corresponding to the following full scales of angles :

- yaw full scale : -25/+25 degrees, which is the maximum acceptable yaw angle for long-lasting play
- pitch full scale : -18/+18 degrees, which is the maximum reachable pitch angle, due to 3dRudder shape
- roll full scale : -18/+18 degrees, which is the maximum reachable pitch angle, due to 3dRudder shape
- updown full scale : -1/1 (unchanged, no unit)

`ModeAxis` defines the current mode to get the value :

In standard use, we strongly recommend to use on of the 2 ModeAxis :

- `NormalizedValueNonSymmetricalPitch`
- `ValueWithCurveNonSymmetricalPitch`

Which allows the user to reach the maximum value for backward movement whatever it's initial position.

`EarthRefAngle: (work in progress)`

returns 4 values :

- yaw : heading angle in degrees related to magnetic North
- pitch : angle in degrees related to vertical (Earth gravity)
- roll : angle in degrees related to vertical (Earth gravity)
- updown : raw up/down value between -1 and 1

These values are available at any time, as soon as the status is neither `NoFootStayStill` nor `Initialisation`

This mode doesn't use the curves

`UserRefAngle:`

Returns 4 values, depending on the status of the 3dRudder:

`If status is InUse or ExtendedMode :`

- yaw : angle in degrees related to neutral user position (i.e. feet position at init)
- pitch : angle in degrees related to neutral user position (i.e. feet position at init)
- roll : angle in degrees related to neutral user position (i.e. feet position at init)
- updown : raw up/down value between -1 and 1

`In all other status:`

- yaw : heading angle in degrees related to magnetic North
- pitch : value in degrees related to vertical (Earth gravity)
- roll : value in degrees related to vertical (Earth gravity)
- updown : raw up/down value between -1 and 1

This mode doesn't use the curves

`NormalizedValue:`

This function returns normalized values of each 4 axis between -1 and 1

It returns 4 values, depending on the status of the 3dRudder:

`If status is InUse or ExtendedMode :`

- yaw : heading value between -1 and 1, related to neutral user position (i.e. feet position at init).
  As physical Full Scale is 25 degrees, the returned value is yaw UserRefAngle/25

- pitch : pitch value between -1 and 1, related to neutral user position (i.e. feet position at init)
  As physical Full Scale is 18 degrees, the returned value is yaw UserRefAngle/18

- roll : roll value between -1 and 1, related to neutral user position (i.e. feet position at init)
  As physical Full Scale is 18 degrees, the returned value is yaw UserRefAngle/18

- updown : raw up/down value between -1 and 1

`In all other status:`

- Returns 0

This mode doesn't use the curves

`NormalizedValueNonSymmetricalPitch:`

With this ModeAxis value, the returned values are the same as in `NormalizedValue`, except for the pitch, for which the value is magnified, depending on the initial user position, to ensure to be able to reach the full scale. This is especially usefull when the user has a significant initial pitch to the rear, which is a standard position.

In this case, as the 18° angle (in reference to the user initial position) that leads to full scale in `NormalizedValue` mode cannot be reached, the value is magnified so that the full scale is reached when the pitch reaches 18° in reference to the vertical, which is the maximum value that the shape of the 3dRudder allows), without changing the neutral position.
In other words, the pitch value is magnified in the direction where the available angle is the smaller.

The calculation of the NonSymetricalPitch uses the unsymmetrical offset of the user calculated in `InUse` and `ExtendedMode`.

This mode doesn't use the curves.

With this ModeAxis value, the returned values are the same as in `NormalizedValue`, except for the pitch, for which the value is magnified, depending on the initial user position, to ensure to be able to reach the full scale. This is especially usefull when the user has a significant initial pitch to the rear, which is a standard position.

In this case, as the 18° angle (in reference to the user initial position) that leads to full scale in `NormalizedValue` mode cannot be reached, the value is magnified so that the full scale is reached when the pitch reaches 18° in reference to the vertical, which is the maximum value that the shape of the 3dRudder allows), without changing the neutral position.
In other words, the pitch value is magnified in the direction where the available angle is the smaller.

The calculation of the NonSymetricalPitch uses the unsymmetrical offset of the user calculated in `InUse` and `ExtendedMode`.

This mode doesn't use the curves.

`ValueWithCurve:`

returns the value of the axis, using the curves.
The input value of the curve is the `NormalizedValue`, the output of the function is the corresponding output of the curve, for each axis.
The curve should have an input range of -1/+1, and can include deadzone and progressivity.

`ValueWithCurveNonSymmetricalPitch:`

returns the value of the axis, using the curves.
The input value of the curve is the `NormalizedValueNonSymmetricalPitch`, the output of the function is the corresponding output of the curve, for each axis.
The curve should have an input range of -1/+1, and can include deadzone and progressivity.

Events

```
void SetEvent(IEvent *pEvent) const
```

For version 0.6 and further of the SDK, it's possible to get events. Currently the SDK manages two events, one for the connection and the other one for the disconnection.

to use it, you should create a class derived from `IEvent` and define two method from the virtual one :

```
class CEvent  : public IEvent
{
public:
        void OnConnect(uint32_t nDeviceNumber);
        void OnDisconnect(uint32_t nDeviceNumber);
};
```

`Warning: Those events are called from another thread !`

Error Code

`ns3DRudder::CSdk::ErrorCode` define the error code used by the SDK:

`Success`: No error
`NotConnected:` The 3DRudder is not connected.
`Fail:` Fail to execute the method.
`IncorrectCommand:` Incorrect command.
`Timeout:` Communication with the 3DRudder timeout.
`WrongSignature:` Wrong signature of the version of the Firmware.
`NotReady:` The data you try to read is not ready.

Get the text of the error

```
const char *GetErrorText(ErrorCode nError) const
```

Translates the error code to human-readable value.