

# **CyWee Z Motion SDK**

## **Manual**

**For Windows**

# Table of Contents

Table of Contents	1
Introduction	2
Requirements	2
CyWee motion controller Features	2
Overview of the CyWee Z API Process	3
CyWee Z Motion SDK File Notes	4
A Simple Example	5
Sample program usage	5
APIs usage in Sample code	7
Transmitter X-Y-Z definition in 3D space	10
Transmitter buttons definition	10
Function List	11
CWZ_init	11
CWZ_term	11
CWZ_open_Rx	11
CWZ_close_Rx	12
CWZ_check_Rx	12
CWZ_check_Tx	12
CWZ_get_data	13
CWZ_set_browser_mode	14
CWZ_set_Tx_indicator	15
CWZ_detect_motion	16
CWZ_send_key	19
CWZ_send_mouse	20
CWZ_send_vibration	21

## Introduction

Welcome the CyWee Z Motion SDK!.

CyWee Z Motion SDK is a C language function library that makes up CyWee's innovative motion controller technology. This modular library is designed to be integrated into a broad range of applications. These include games, leisure software, and other applications suitable for motion control interactions.

## Requirements

- 8 MB RAM or more recommended.
- 2 MB free hard disk space.
- Microsoft Windows XP or above.
- Microsoft Visual C++ 6.0 or later.
- CyWee controller set, including a USB dongle (Receiver, "Rx") and a motion-detect controller (Transmitter, "Tx").

## CyWee motion controller Features

- With X-Y-Z accelerometer, and Wy-Wz gyroscope for motion detection.
- 9 buttons plus a switch mode button available on the Transmitter.
- Standard HID compliant device (Rx) for Keyboard and Mouse.
- Uses 2.4GHz wireless transmission protocol between Transmitter and Receiver.
- Supports up to 4 players (4 controller sets) at the same time.
- Output sensor data ranges from 0 to 4095 for Ax, Ay, Az, Wy, and Wz.
- Supports hot plug and detection of Receivers.
- No need for installation of driver.
- Supports 4 paly modes for the Transmitter: Stick mode, Gun mode, Racing mode, and Flying mode. These 4 modes are suitable for playing all types of sports games, shooting games, racing games, and flying games.

## Overview of the CyWee Z API Process

This section provides an overview of the CyWee Z API process, which can be divided into the following steps:

1. Hardware setup.
2. Initialization.
3. Open connection.
4. Polling status.
5. Get data.
6. Termination.

### Hardware setup

First, make sure that you have inserted the Receiver into the USB port of your PC, turned on the Transmitter, and that both of them are communicating correctly. If you have problems with this step, please refer to the user's manual.

### Initialization

To start a CyWee Z API session, you must first call **CWZ\_init**.

**CWZ\_init** needs to be called only once per CWZ API session. Only one session at a time can be established.

### Open connection

Call **CWZ\_open\_Rx** to establish the connection between the controller and the PC.

**CWZ\_open\_Rx** will search for a Receiver, and start to receive data.

### Polling status

CyWee recommends that you call **CWZ\_check\_Rx** and **CWZ\_check\_Tx** every 200ms (for example, in a timer loop), to check the status of the Receiver and Transmitter to ensure the devices are still working correctly (not unplugged, not turned off, etc.).

### Get data

Once the Receiver and Transmitter are working, you can use **CWZ\_get\_data** to get motion and button data from the controller. The data will be stored in a **CWZ\_DATA** structure, which is passed as the parameter of the **CWZ\_get\_data** function.

You may use **CWZ\_detect\_motion** to detect controller motion and get related information. The data will be returned by a callback function.

### Termination

You should call **CWZ\_term** once the CyWee Z API session is no longer needed.

### CyWee Z Motion SDK File Notes

Doc\CyWeeZ\_SDK.pdf -> This document file  
C\_Sample\  
include\CWZ\_sdk.h -> include file of CyWee Z Motion SDK  
lib\IIMR\_Main.lib -> library file of CyWee Z Motion SDK  
SDK-distribution\IIMR\_Main.dll  
-> Redistribution file (put it in the same folder  
with your application execution file)

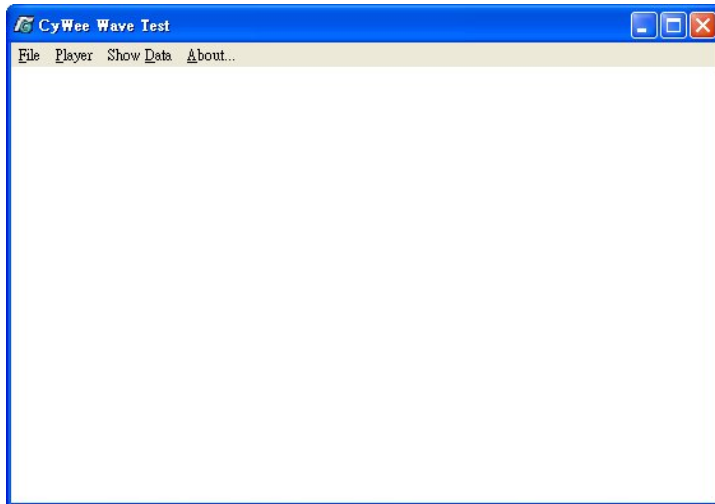
## A Simple Example

### Sample program usage

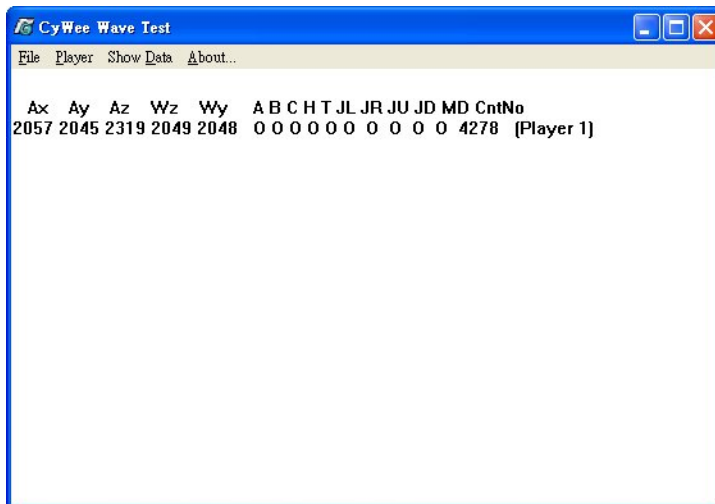
First let's run the sample program and see how it works:

C\_Sample\workdir\WaveTest.exe

1. Make sure you have inserted the Receiver, and turned on the Transmitter, and that both of them are communicating correctly.
2. Run the program WaveTest.exe. It looks like this:



3. Click on "Show Data->Get Receiver Data START", and you will see a data display like this:



Ax, Ay, Az correspond to the X-Y-Z accelerometer values. (For more information, refer to the section on Transmitter X-Y-Z definition in 3D space)

Wz, Wy correspond to the gyro value of axis z and axis y.

---

## CyWee Z SDK

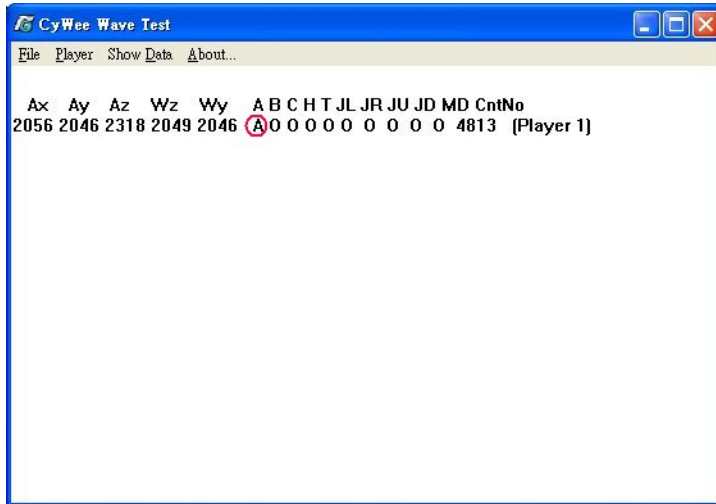
---

A, B, C, H, T, JL, JR, JU, JD, and MD correspond to the 9 buttons and the mode switch button. (For more information, refer to the section on Transmitter button definitions.)

CntNo is the count number of the current data sent by the Transmitter.

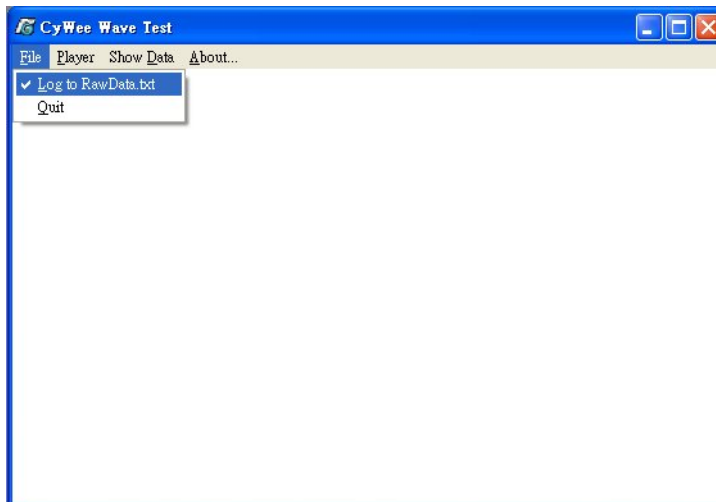
(Player 1) refers to the data of the first CyWee Z controller.

4. If you press a button on the Transmitter (for example, button A), you will see your action reflected on the screen:

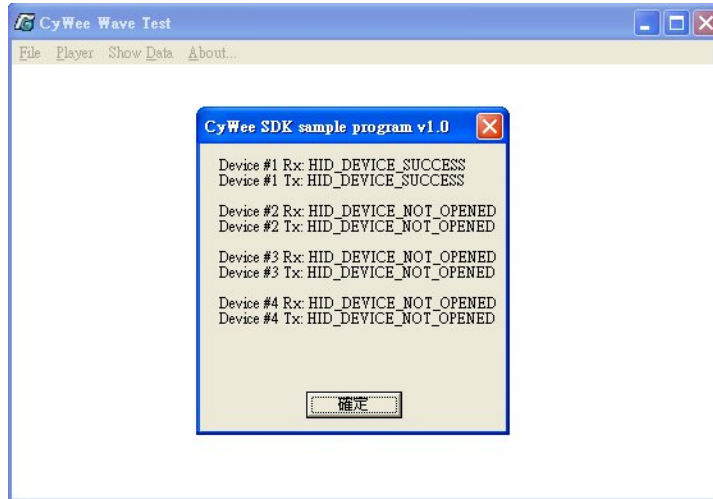


Wave your Transmitter and you can see the values of Ax, Ay, Az, Wz, Wy change according to your movements.

5. The values of the motion parameters range from 0 to 4095.
6. You may stop the data by clicking on "Show Data->Get Receiver Data STOP"
7. There will also be a log file called "RawData.txt", if this option is selected in the menu "File->Log to RawData.txt"



8. Click on "About", and you can see the status of the Receiver and Transmitter:



## API usage in sample code

Now let's look at a sample source code segment:

### 1. Initialization

[winmain.cpp]

```
#include "CWZ_sdk.h"
int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst, LPSTR
    lpCmdLine, int nCmdShow)
{
    .
    .
    .
    SetTimer(hWnd, ID_TIMER_01, 10, NULL);
    SetTimer(hWnd, ID_TIMER_02, 200, NULL);
    Init_IIMR();
    .
    .
    .
}
```

[WaveTest.cpp.cpp]

```
void Init_IIMR()
{
    if (init_flag==0) {
        CWZ_init();
        CWZ_open_Rx();
        init_flag = 1;
    }
}
```

When using the CyWee Z Motion SDK APIs, include the header file CWZ\_sdk.h.

To initialize the session and open the Receiver connection, use the sample calls CWZ\_init() and CWZ\_open\_Rx() at the beginning of the program.

This initialization also sets up two timers: Timer1 (10 ms) is for polling to get the controller data, and Timer2 (200 ms) is for polling to check the controller status.

### 2. Polling:

```
[winmain.cpp]
LONG FAR PASCAL MainWndProc(HWND hWnd, unsigned message, WPARAM wParam,
LPARAM lParam)
{
    switch (message) {
        case WM_COMMAND:
            switch (wParam) {
                .
                case WM_TIMER:
                    if (wParam == ID_TIMER_01)
                        Show_data();
                        KB_polling();
                        Mouse_polling();
                    if (wParam == ID_TIMER_02)
                        PollingAllStatus();
                    return (DefWindowProc(hWnd, message, wParam, lParam));
                    break;
                .
            }
    }
}
```

Timer1 will call show\_data() every 10 ms, and Timer2 will call PollingAllStatus() every 200 ms.

### 3. Check status:

```
[WaveTest.cpp.cpp]
void PollingAllStatus()
{
    int i;

    for (i=1; i<=4; i++) {
        RxStatus[i] = CWZ_check_Rx(i);
        TxStatus[i] = CWZ_check_Tx(i);
        CWZ_set_browser_mode(i, browser_mode_flag);
        CWZ_set_Tx_indicator(i, i);
    }
}
```

PollingAllStatus() checks Receiver and Transmitter by calling CWZ\_check\_Rx(i) and CWZ\_check\_Tx(i), then stores their status in RxStatus[] and TxStatus[].

### 4. Get data:

```
[WaveTest.cpp.cpp]
void Show_data()
{
    .
    .
    CWZ_get_data(p_no, &g_cwz_data);

    if (g_cwz_data.btn_A) strcpy(LineText0, "A ");
    else                    strcpy(LineText0, "O ");
    .
}
}
```

Show\_data() gets data by calling CWZ\_get\_data(), the results will be stored in the g\_cwz\_data:

```
static CWZ_DATA g_cwz_data;
```

You may refer to the CWZ\_sdk.h for the definition of the structure CWZ\_DATA.

### 5. Hot plug treatment:

```
[winmain.cpp]
LONG FAR PASCAL MainWndProc(HWND hWnd, unsigned message, WPARAM wParam,
LPARAM lParam)
{
    switch (message) {
        case WM_COMMAND:
            switch (wParam) {
                .
                .
            case WM_DEVICECHANGE:
                switch(wParam) {
                    case DBT_DEVICEARRIVAL:
                        CWZ_open_Rx();
                        break;
                    case DBT_DEVICEREMOVECOMPLETE:
                        int i;
                        for (i=1; i<=4; i++) {
                            if (CWZ_check_Rx(i)==HID_DEVICE_NOT_OPENED)
                                CWZ_close_Rx(i);
                        }
                        break;
                }
            }
        break
        .
        .
    }
}
```

It is recommended that you treat events of Receiver hot plugging by using the WM\_DEVICECHANGE windows message case statement, as shown in the code segment above.

### 6. Termination

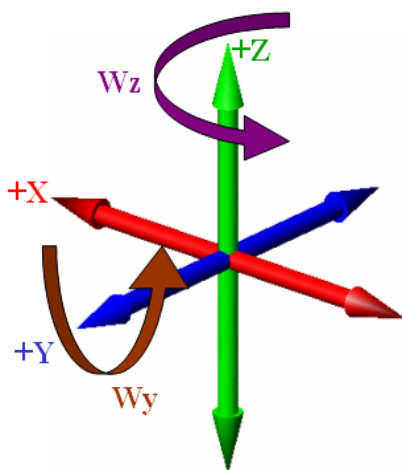
```
[winmain.cpp]
LONG FAR PASCAL MainWndProc(HWND hWnd, unsigned message, WPARAM wParam,
LPARAM lParam)
{
    switch (message) {
        case WM_COMMAND:
            switch (wParam) {
                .
                .
            case IDC_QUIT:
                Term_IIMR();
                .
                .
            }
        break;
    }
}

[WaveTest.cpp.cpp]
void Term_IIMR()
{
    if (init_flag) {
        CWZ_term();
        init_flag = 0;
    }
}
```

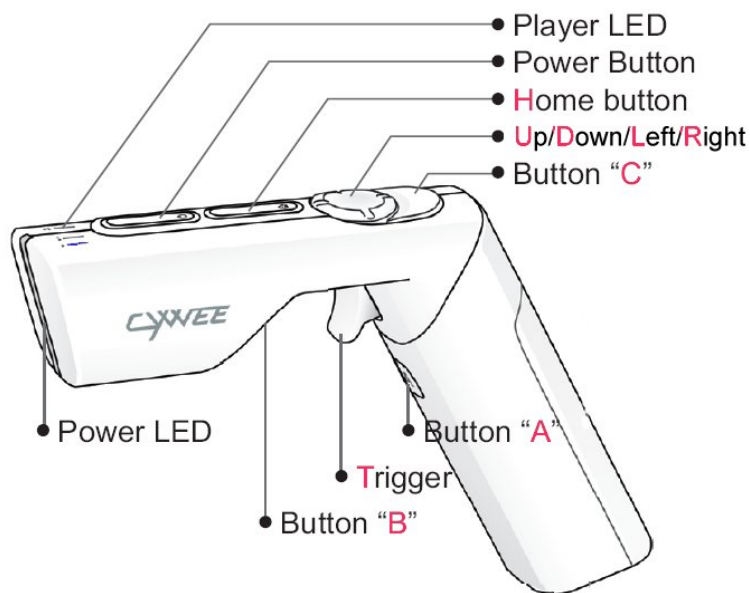
In this sample, CWZ\_term() is called when the program quits.

7. Summation of the program flow in the sample project:
  - a. Initialization: initialize the SDK and set the two timers.
  - b. Poll to check and store the status of Receiver and Transmitter.
  - c. Poll to check for motion and button data, and display them (or you may save the results, or analyze and recognize them).
  - d. Perform error handling of the Windows USB hot plug message.
  - e. Termination.

### Transmitter X-Y-Z definition in 3D space:



### Transmitter buttons definition:



## Function List

In the following function list, we will use “Receiver” to indicate the USB receiver, and “Transmitter” to indicate the CyWee Z motion controller.

### **CWZ\_init()**

#### **Description**

This function is required for starting the CyWee Z API session. It does the necessary initialization for the session. You should call it before all the other CWZ\_ functions.

#### **Parameters**

None

#### **Usage**

```
CWZ_init();
```

### **CWZ\_term()**

#### **Description**

This function is required to terminate the CyWee Z API session. It does the necessary cleanup for the session. You should call it when all the CWZ\_ functions are no longer needed.

#### **Parameters**

None

#### **Refer to**

CWZ\_Init

### **CWZ\_open\_Rx()**

#### **Description**

This function establishes the connection between the Receiver, the Transmitter, and the PC. It will search for all available Receivers (up to a maximum of four), and start to get data from the Transmitter if any is available. You should also call this function if a new Receiver is plugged in and detected, to establish its connection with the PC.

#### **Parameters**

None

#### **Usage**

```
CWZ_init();  
CWZ_open_Rx();
```

## CWZ\_close\_Rx(int DeviceNum)

### Description

Close the connection between the controller and the PC. You should call this function if the connection is no longer needed, or the Receiver is unplugged.

### Parameters

DeviceNum

The device number of the Receiver to close, which should be either 1, 2, 3, or 4.

### Refer to

CWZ\_open\_Rx

## CWZ\_check\_Rx(int DeviceNum)

### Description

Check the status of the Receiver and report the status via the return value. It is recommended that you periodically poll the status of the Receiver by using this function. The time period recommended is 200 ms or longer.

### Parameters

DeviceNum

The device number of the Receiver to check, which should be either 1, 2, 3, or 4.

### Refer to

CWZ\_check\_Tx

### Return value

Defined in CWZ\_sdk.h:

HID_DEVICE_SUCCESS:	successful
HID_DEVICE_RESET_DONE:	reset OK
HID_DEVICE_RESET_NOT_DONE:	reset failed
HID_DEVICE_NOT_OPENED:	not opened (not connected)
HID_DEVICE_TRANSFER_FAILED:	transfer failed
HID_DEVICE_TRANSFER_TIMEOUT:	transfer time out (no data)
HID_DEVICE_HANDLE_ERROR:	internal handle error

## CWZ\_check\_Tx(int DeviceNum)

### Description

This function enables you to check the status of the Transmitter and report that via the return value. It is recommended that you periodically poll the Transmitter using this function. The time period recommended is 200 ms or longer.

### Parameters

DeviceNum

The device number of the Transmitter to check, either 1, 2, 3, or 4.

## Refer to

CWZ\_check\_Rx

## Return value

Defined in CWZ\_sdk.h:

HID_DEVICE_SUCCESS:	successful
HID_DEVICE_RESET_DONE:	reset OK
HID_DEVICE_RESET_NOT_DONE:	reset failed
HID_DEVICE_NOT_OPENED:	not opened (not connected)
HID_DEVICE_TRANSFER_FAILED:	transfer failed
HID_DEVICE_TRANSFER_TIMEOUT:	transfer time out (no data)
HID_DEVICE_HANDLE_ERROR:	internal handle error

## CWZ\_get\_data(int DeviceNum, LP\_CWZ\_DATA lpCWZ\_data)

### Description

This function gets motion and button data from the controller. The data transfer sampling rate between Transmitter and Receiver is 10 ms. It is recommended that you poll the controller data at a comparable frequency. The data will be stored in the structure given by the lpCWZ\_data parameter.

### Parameters

DeviceNum

The device number to get the data from, either 1, 2, 3, or 4.

lpCWZ\_data

This is a pointer to the CWZ\_DATA structure, to store the controller data defined in CWZ\_sdk.h:

```
typedef struct CWZ_DATA
{
    int ax[MAX_CWZ_SENSOR_DATA_LEN];
    int ay[MAX_CWZ_SENSOR_DATA_LEN];
    int az[MAX_CWZ_SENSOR_DATA_LEN];
    int wy[MAX_CWZ_SENSOR_DATA_LEN];
    int wz[MAX_CWZ_SENSOR_DATA_LEN];
    int sensor_amount;
    int btn_A;
    int btn_B;
    int btn_C;
    int btn_H;
    int btn_T;
    int btn_L;
    int btn_R;
    int btn_U;
    int btn_D;
    int mode_switch;
    long int count_num;
} CWZ_DATA, *LP_CWZ_DATA;
```

## Description of the structure items in CWZ\_DATA:

**count\_num:** count number of the current data, from 1 to 10000 (then resets to 1 and cycles again); used as an id for the data  
**sensor\_amount:** counts of the motion data for the current count\_num, presents the total number to get in ax[], ay[], az[], wy[], wz[] for the current count\_number  
**ax, ay, az:** accelerometer data array of Ax, Ay, Az. For more information, refer to the section "Transmitter X-Y-Z definition in 3D space".  
**wy, wz:** gyro data array of Wy, and Wz.  
**btn\_A, btn\_B, btn\_C, btn\_H, btn\_T, btn\_L, btn\_R, btn\_U, btn\_D,** stands for the buttons on Tx:  
A, B, C, Home, Trigger, Left, Right, Up, and Down, respectively. Please refer to the section of "Transmitter buttons definition" for the location of the buttons on the Transmitter.  
**mode\_switch:** the mode switch detection button on Tx.

## Usage

```
static CWZ_DATA g_cwz_data;
memset(&g_cwz_data, 0, sizeof(CWZ_DATA));

CWZ_init();

CWZ_get_data(1, &g_cwz_data); // get data of device #1
if (g_cwz_data.btn_A) // check button A status
    strcpy(LineText, "Button A is pressed.");
for (i=0; i <g_cwz_data.sensor_amount; i++) { // loop through ax[]
    printf("%4d\n", g_cwz_data.ax[i]); // get ax[] data
}
```

## CWZ\_set\_browser\_mode(int DeviceNum, int on\_off)

### Description

This function sets the browser mode to either on or off. The default mode is on. When the browser mode is on, the cursor is on, and some key settings of the buttons on the Transmitter are mapped:

Trigger = mouse left button

Button C = mouse right button

Up/Down/Left/Right buttons = keyboard Up/Down/Left/Right keys

Button Home = Backspace key

If you don't need the controller cursor, you can turn off the browser mode.

### Parameters

DeviceNum

The device number to set the browser mode, either 1, 2, 3, or 4.

on\_off

Could be one of the following values:

BROWSER\_MODE\_ON = on,

BROWSER\_MODE\_OFF = off,

BROWSER\_MODE\_FOR\_SHOOTING = special mode for shooting games.

### Usage

```
CWZ_set_browser_mode(1,0); // turn off browser mode of device #1
```

### Notice:

This function needs to be called periodically about once per second (1000 ms cycle). Otherwise, the setting will be reset to its default value.

## CWZ\_set\_Tx\_indicator(int DeviceNum, int PlayerNum)

### Description

This function sets the LED indicator on the Transmitter of DeviceNum according to the PlayerNum specified.

### Parameters

DeviceNum

The device number of the Transmitter whose LED is to be set, either 1, 2, 3, or 4.

PlayerNum

The player# of LED indicator to set, either 1, 2, 3, or 4.

### Usage

```
// set the device#1 of controller to be player#2  
// i.e. turn on the 2nd LED on Tx#1  
CWZ_set_Tx_indicator(1, 2);
```

### Notice:

This function needs to be called periodically about once per second (1000 ms cycle). Otherwise, the setting will be reset to its default value.

## **CWZ\_detect\_motion(int mode, CWZ\_MotionCallback fn\_MotionCallback)**

### **Description**

This function initiates motion detection and assigns a callback function for receiving the results.

### **Parameters**

mode

The Transmitter mode, should be one of the following: [MODE\\_STICK](#), [MODE\\_GUN](#), [MODE\\_RACING](#), [MODE\\_FLYING](#), or [MODE\\_STICK\\_MORE](#).

fn\_MotionCallback

Pointer to a callback function prepared by the user for receiving the results. The prototype of the callback function is defined as:

```
void (*CWZ_MotionCallback)(LP_CWZ_MOTION pCWZ_motion);
```

where the pCWZ\_motion is a pointer to a structure CWZ\_MOTION defined as:

```
typedef struct CWZ_MOTION
{
    int DeviceNum;
    int mode;
    int MotionType;
    int strength;
    int angle_x;
    int angle_y;
    int cursor_x;
    int cursor_y;
} CWZ_MOTION, *LP_CWZ_MOTION;
```

### **Usage**

```
static CWZ_MotionCallback fn_MotionCallback = 0;

void fn_Motion_detect_call_back(LP_CWZ_MOTION pCWZ_motion)
{
    .
    // You may get the motion detection result,
    // and do the actions you want here.
    .
}

fn_MotionCallback = &(fn_Motion_detect_call_back);
CWZ_detect_motion(g_motion_detect_mode, fn_MotionCallback);
```

### **Notice:**

To stop the motion detection function, set fn\_MotionCallback to 0, and no more results will be passed to the callback function.

```
fn_MotionCallback = 0;
CWZ_detect_motion(g_motion_detect_mode, fn_MotionCallback);
```

### Description of the structure items in CWZ\_MOTION:

DeviceNum: Device number of the detected motion, should be 1, 2, 3, or 4.

mode: Transmitter playing mode for motion detection, should be one of the followings:

- MODE\_STICK
- MODE\_GUN
- MODE\_RACING
- MODE\_FLYING
- MODE\_STICK\_MORE

MotionType: The detected motion type. For MODE\_STICK (Transmitter in Stick or Gun mode), should be one of the followings:  
`MOTION_NONE`, `MOTION_UP`, `MOTION_DOWN`, `MOTION_LEFT`, `MOTION_RIGHT`  
And for MODE\_STICK\_MORE (Transmitter in stick mode), 6 more additional types:  
`MOTION_FORWARD`, `MOTION_BACKWARD`, `MOTION_UPPERLEFT`,  
`MOTION_LOWERLEFT`, `MOTION_UPPERRIGHT`, `MOTION_LOWERRIGHT`

strength: The detected strength value for Stick or Gun mode, should be ranged from 0 to 9. 0: no motion, 1: min. strength, 9: max. strength

angle\_x: The detected angle X value for Racing or Flying mode, should be ranged from -90 to 90. Positive value: turning right, Negative value: turning left.

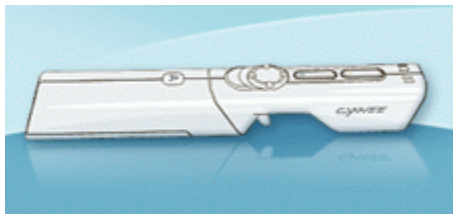
angle\_y: The detected angle Y value for Racing or Flying mode, should be ranged from -90 to 90.  
For Racing mode: Positive value: turning forward, Negative value: turning backward.  
For Flying mode: Positive value: turning up, Negative value: turning down.

cursor\_x: The detected cursor X movement value for Gun mode. Positive value: moving right, Negative value: moving left.

cursor\_y: The detected cursor Y movement value for Gun mode. Positive value: moving down, Negative value: moving up.

### Transmitter mode figures:

Stick mode:



Gun mode:

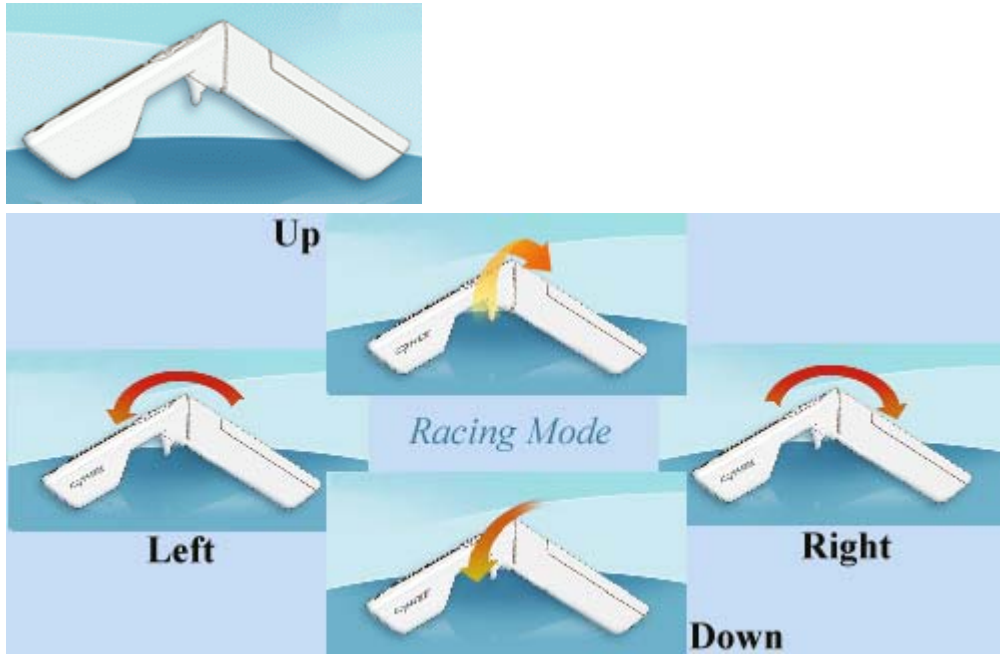
---

## CyWee Z SDK

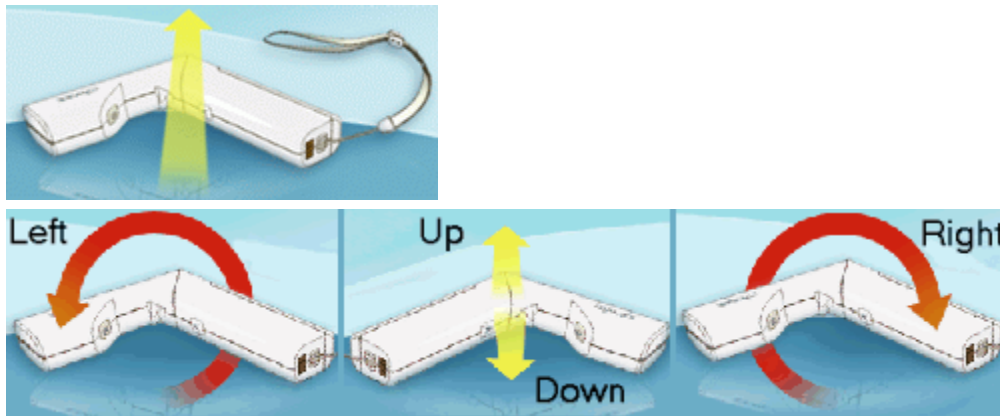
---



Racing mode:



Flying mode:



## **CWZ\_send\_key(int DeviceNum, int SysKey, int key1, int key2, int key3, int key4, int key5, int key6)**

### **Description**

This function passes motion information to the Receiver in the form of keyboard strokes.

### **Parameters**

DeviceNum

The device number of the Receiver to send the keyboard stroke, either 1, 2, 3, or 4.

SysKey

Key combinations of Ctrl, Alt, Shift, or Win. For more information, refer to CWZ\_sdk.h. You can use (L\_Ctrl | L\_Alt) for combinations Left-Ctrl and Left-Alt.

Key1, Key2, Key3, Key4, Key5, Key6

Keys to send to the Receiver. For example, use Keyboard\_A for 'A'. Refer to CWZ\_sdk.h for more key definitions.

### **Usage**

```
// The following statement will send out "GREAT!" via Receiver of player #1
CWZ_send_key(1, L_Sht, Keyboard_G, Keyboard_R, Keyboard_E,
             Keyboard_A, Keyboard_T, Keyboard_1);
// Use the following statement to clear the key setting:
CWZ_send_key(1, 0, 0, 0, 0, 0, 0, 0);
```

### **Example:**

Please refer to the sample code in [WaveTest.cpp], the function KB\_polling (). First, run the sample program WaveTest.exe. Click on Player->Set Button A = "GREAT!" to turn this option on. Then, open a text editor such as Notepad.exe, and click the button A on the Transmitter. You should see the output result in Notepad showing the word "GREAT!"



**Notice:**

This function is an auxiliary function. It is intended to enhance the user's convenience only. In general, it is not necessary to send the keys via the Receiver.

### **CWZ\_send\_mouse(int DeviceNum, int l\_btn, int r\_btn, int m\_btn, int x\_move, int y\_move, int wheel\_move)**

**Description**

This function emulates mouse clicks and mouse cursor movements.

**Parameters**

DeviceNum

The device number to send the mouse event, either 1, 2, 3, or 4.

l\_btn

Mouse left button, the value should be 0=don't click, or 1=click.

r\_btn

Mouse right button, the value should be 0=don't click, or 1=click.

m\_btn

Mouse middle button, the value should be 0=don't click, or 1=click.

x\_move

Mouse cursor movement value, positive=move right, negative=move left.

y\_move

Mouse cursor movement value, positive=move down, negative=move up.

wheel\_move

Mouse wheel movement value, positive=move forward, negative=move backward.

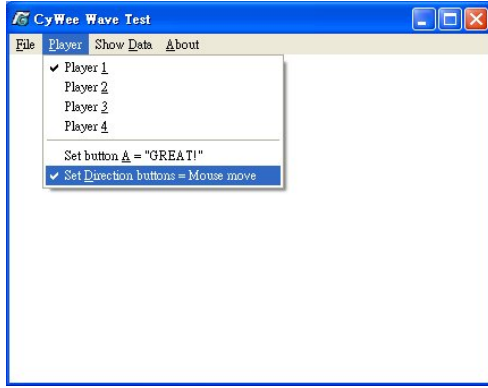
**Usage**

Please refer to the sample code in [WaveTest.cpp], the function Mouse\_polling().

```
// The following statement will move the mouse cursor of player #1
// upward 20 pixels.
CWZ_send_mouse(1, 0, 0, 0, 0, -20, 0);
```

**Example:**

First, run the sample program WaveTest.exe. Click on "Player->Set Direction buttons = Mouse move" to turn this option on. Then, press and hold the "Down" button on the Transmitter; you should see the mouse cursor is moving down slowly. If you press the "Trigger" on the Transmitter while keeping the "Down" button pressed, you can see the cursor moves down much faster.



## CWZ\_send\_vibration(int DeviceNum, int VibrationMode)

### Description

Send a vibration command to the Transmitter to make it vibrate. The Transmitter should have the built-in vibrator, otherwise this function won't work.

### Parameters

DeviceNum

The device number to vibrate, either 1, 2, 3, or 4.

VibrationMode

The vibration mode, it should be one of the followings:

```
VIBRATION_MODE_NORMAL: vibrate in normal strength for 2sec.  
VIBRATION_MODE_SHORT:  vibrate in normal strength for 1sec.  
VIBRATION_MODE_LONG:   vibrate in normal strength for 4sec.  
VIBRATION_MODE_WEAK:   vibrate in weak strength for 1sec.
```

### Usage

```
// The following statement will vibrate the Transmitter of player #1  
// in normal strength for 2 sec.  
CWZ_send_vibration(1, VIBRATION_MODE_NORMAL);
```

### Example:

First, run the sample program WaveTest.exe. Click on “Vibration->Normal” to make the Transmitter to vibrate in normal mode. You should feel the vibration of the controller for about 2 seconds. Try other modes to feel the different effects. Please be sure that your Transmitter has the built-in vibrator.

