Wiiuse is a library written in C that connects with several Nintendo Wii remotes. Supports motion sensing, IR tracking, nunchuk, classic controller, and the Guitar Hero 3 controller. Single threaded and nonblocking makes a light weight and clean API.

Licensed under GNU GPLv3 and GNU LGPLv3 (non-commercial).

Current Version: v0.12

Random Projects Using wiiuse

- WiiPuck Café
- Wing Commander ...
- AGR (Accelerome...
- libMT
- SDL-Ball

Get Listed Here See More

Wiiuse API Overview

Written by: Michael Laforest (para)
wiiuse version: v0.12

Index

About This Guide

This guide does not cover all the functionality of the wiiuse library. For more information visit the source documenation.

Notes are in blue boxes.

```
Code is in green boxes.
```

Warnings are in red boxes.

Before you Begin

Before you begin your project will need a few things:

1. A compiled version of the wiiuse library (Linux is *wiiuse.so* and Winodws is *wiiuse.dll* and *wiiuse.lib*).
2. The file include/wiiuse.h

To compile projects that use wiiuse for...

Windows:

- Include *include/wiiuse.h* in all files that use wiiuse.
- Link *wiiuse.lib* to your project.
- When you run your program it will try to load *wiiuse.dll* automatically.

Linux:

- Include *include/wiiuse.h* in all files that use wiiuse.
- Link *wiiuse.so* to your project.
- When you run your program it will try to load *wiiuse.so* automatically (use 'ld' to confirm it can find the path).

API Functions

The following functions are available through the API:

- wiiuse_init()
- wiiuse_cleanup()
- wiiuse_version()

- wiiuse_find()
- wiiuse_connect()
- wiiuse_disconnect()

- wiiuse_poll()

- wiiuse_rumble()
- wiiuse_toggle_rumble()
- wiiuse_set_leds()
- wiiuse_motion_sensing()
- wiiuse_read_data()
- wiiuse_write_data()
- wiiuse_status()
- wiiuse_get_by_id()
- wiiuse_set_flags()
- wiiuse_set_smooth_alpha()
- wiiuse_set_bluetooth_stack()
- wiiuse_set_orient_threshold()
- wiiuse_set_accel_threshold()
- wiiuse_set_nunchuk_orient_threshold()
- wiiuse_set_nunchuk_accel_threshold()
- wiiuse_resync()
- wiiuse_set_timeout()

- wiiuse_set_ir()
- wiiuse_set_ir_vres()
- wiiuse_set_ir_position()
- wiiuse_set_ir_sensitivity()
- wiiuse_set_aspect_ratio()

Not all of these functions are described here.
For more information about each function check the source documentation here.

There are also several macro functions available that are discussed throughout this document.

Finding and Connecting to Wiimotes

Wiiuse can connect up to as many wiimotes as your system will allow.

First you need to initialize the maximum number of wiimotes you want your program to use. It is okay if you actually use less than you initialize though. You can initialize them using the wiiuse_init() function.

In this example we tell wiiuse to create enough wiimote objects to connect up to two wiimotes. Each wiimote created will have an assoicated unique identifier so that they can be easily distinguished later, these are automatically generated by wiiuse_init(). The function returns an array of pointers to wiimote objects that have been initialized but not connected.

```
wiimote** wiimotes =  wiiuse_init(2);
```

Now we need to find some wiimotes that are in discovery mode. We can do this with the wiiuse_find() function.

Here we scan for a maximum of two wiimotes for a maximum duration of 5 seconds, and we want the information for each found wiimote to be stored in the *wiimotes* array returned by wiiuse_init(). The function will return the number of wiimotes found.

```
int found = wiiuse_find(wiimotes, 2, 5);
```

On Windows wiiuse_find() will find and connect to the available wiimote devices.
To keep your code consistent throughout all supported platforms you should still use wiiuse_connect() even though it will not do anything.

On Windows, wiiuse_find() will try to auto-detect the bluetooth stack the system is using. For more information on this and how to control it, see the Windows Bluetooth Auto-Detection section.

If we found some wiimotes we want to connect to them using the function wiiuse_connect(). Again you tell it the maxmimum number of wiimotes in the array, not how many were found. The function will return the total number of wiimotes that were successfully connected.

```
int connected = wiiuse_connect(wiimotes, 2);
if (connected)
	printf("Connected to %i wiimotes (of %i found).\n", connected, found);
else {
	printf("Failed to connect to any wiimote.\n");
	return 0;
}
```

Now that we are connected we can communicate fully with each wiimote. Later if we want to disconnect we can call the wiiuse_disconnect() function with the particular wiimote object that should be disconnected:

```
void wiiuse_disconnect(struct wiimote_t* wm);
```

Or you can call wiiuse_shutdown() to disconnect and clean up all wiimote connections:

```
void wiiuse_cleanup(struct wiimote_t** wm, int wiimotes);
```

Here *wiimotes* is the size of the wm array that was passed to wiimote_init().

Windows Bluetooth Stack Auto-Detection

This section only applies to Windows.

wiiuse_find() will try to auto-detect the bluetooth stack running on the system.

If wiiuse is unable to find the correct stack to use, or you already know the stack you would like to use, you can manually set it before calling wiiuse_find() with the following function:

```
void wiiuse_set_bluetooth_stack(struct wiimote_t** wm, int wiimotes, enum win_bt_stack_t type);
```

Here *wiimotes* is the size of the *wm* array that was passed to wiimote_init().

type can be either of the following:

- WIIUSE_STACK_BLUESOLEIL

  This will tell wiiuse to use the BlueSoleil stack.

- WIIUSE_STACK_MS

  This will tell wiiuse to use other stacks.
  Tested and work: Windows XP SP2 stack and Widcomm. Other stacks may also work.

The Polling System

Wiiuse works as a **nonblocking polling system**. This means that you need to constantly tell wiiuse to check for events.

This polling system is modeled after the SDL graphic rendering library and allows for both wiiuse to be single threaded and maximum compatibility with other languages.

To check for events simply call the wiiuse_poll() function with the array returned from wiiuse_init() and the maximum number of wiimotes that you passed to that function:

```
int wiiuse_poll(struct wiimote_t** wm, int wiimotes);
```

Here *wiimotes* is the size of the wm array that was passed to wiimote_init().

Since you must do this constantly you should put this in your main loop. You can of course fork off your own thread and place it in there if your main loop is very processor intensive.

wiiuse_poll() will return the number of wiimotes that had an event occur. If the number is 0 you do not need to do anything, otherwise you should loop through each wiimote and check what event was triggered. The following example code will illustrate a typical main loop that checks for wiimote events:

```
while (1) {
	if (wiiuse_poll(wiimotes, 2)) {
		int i = 0;
		for (; i < 2; ++i) {
			switch (wiimotes[i]->event) {
				/* check the events here */
			}
		}
	}
}
```

The *wiimote_t::event* variable is set by wiiuse_poll() to indicate if an event had occured on that wiimote for that poll. *event* may be set to any of the following:

- WIIUSE_NONE

  No event occured on the wiimote.

- WIIUSE_EVENT

  A generic event occured on the wiimote.

- WIIUSE_STATUS

  A status report was obtained from the wiimote.

- WIIUSE_DISCONNECT

  The wiimote disconnected.

- WIIUSE_READ_DATA

  Data was returned that was previously requested from the wiimote ROM/registers.

- WIIUSE_NUNCHUK_INSERTED

  A nunchuk has been inserted.
  This is a special case of the WIIUSE_STATUS event.

- WIIUSE_NUNCHUK_REMOVED

  A nunchuk has been removed.
  This is a special case of the WIIUSE_STATUS event.

- WIIUSE_CLASSIC_CTRL_INSERTED

  A classic controller has been inserted.
  This is a special case of the WIIUSE_STATUS event.

- WIIUSE_CLASSIC_CTRL_REMOVED

  A classic controller has been removed.
  This is a special case of the WIIUSE_STATUS event.

- WIIUSE_GUITAR_HERO_3_CTRL_INSERTED

  A Guitar Hero 3 controller has been inserted.
  This is a special case of the WIIUSE_STATUS event.

- WIIUSE_GUITAR_HERO_3_CTRL_REMOVED

  A Guitar Hero 3 controller has been removed.
  This is a special case of the WIIUSE_STATUS event.

The rest of this section only applies to Windows.
However, the function does exist on Linux, although it does nothing.

On Windows a timeout is used when polling the wiimotes. If you find the wiimote is responding too slowly you may try to lower the timeout values, however lowering them too much may cause problems. The timeouts are measured in milliseconds.

There are two timeout values:

- The normal timeout. This is used for normal polling.
- The expansion timeout. This is used when an expansion is detected until the expansion successfully handshakes.

The normal timeout is always used, except when an expansion is first plugged in. When an expansion is detected wiiuse will begin using the expansion timeout for that wiimote until the

expansion finishes its handshake.

If you find expansions are not being detected properly you might try increasing the expansion timeout. This will cause wiiuse to pause for a longer amount of time to wait for the handshake to finish before reverting back to the normal timeout value.

The function is:

```
void wiiuse_set_timeout(struct wiimote_t** wm, int wiimotes, byte normal_timeout, byte exp_timeout);
```

The Generic Event

The *event* event is set by wiiuse when a generic event occurs on a wiimote.

An event is generated when a *significant state change* has occured.
An significant state change is:

1. A button press
2. A button release
3. Joystick movement
4. The tilt (or orientation) of the device (if motion sensing is enabled) has changed by a significant amount
5. The position the IR camera is pointing at has changed

Orientation Threshold

The accelerometer is very sensitive and produces a lot of noise. Because of this the angle is almost always changing on every call to wiiuse_poll(), meaning that each call will result in raising a generic event if motion sensing is enabled. To fix this issue wiiuse will only generate an event for motion sensing if a significant orientation change has occured, or if any angle has changed by a particular degree.

By default this threshold is half a degree (0.5 degrees). This means if the angle changes by less than this wiiuse will not generate an event.

You can change the orientation theshold by calling the following function:

```
void wiiuse_set_orient_threshold(struct wiimote_t* wm, float threshold);
```

The *threshold* parameter is how many degrees any angle (roll, pitch, or yaw) must change to generate an event.
Note that this function only takes one wiimote structure so that you may dynamically make one wiimote more sensitive than another. A good time to set a different threshold if you want it to be applied to all of your wiimotes would be after calling wiimote_init().

There is also a function that applies to the nunchuk:

```
void wiiuse_set_nunchuk_orient_threshold(struct wiimote_t* wm, float threshold);
```

By default whenever a nunchuk is plugged in the orientation threshold is set to be the same as the wiimote. A good time to change this is when a WIIUSE_NUNCHUK_INSERTED event is generated.

Acceleration Threshold

The function

```
void wiiuse_set_accel_threshold(struct wiimote_t* wm, int threshold);
```

works the same as *wiiuse_set_orient_threshold()* but applies to the acceleration values rather than the orientation.

There is also a function that applies to the nunchuk:

```
void wiiuse_set_nunchuk_accel_threshold(struct wiimote_t* wm, float threshold);
```

By default whenever a nunchuk is plugged in the acceleration threshold is set to be the same as the wiimote. A good time to change this is when a WIIUSE_NUNCHUK_INSERTED event is generated.

The Status Event

The *status* event is set by wiiuse when a status change has occured on a wiimote.

A status change occurs when one of the following conditions are met:

1. An extension has been plugged into the wiimote extension port
2. An extension has been unplugged from the wiimote extension port
3. wiiuse_status() was called and the wiimote has responded

Important information that can be obtained from a status event include:

- If there is an attachment connected
- If the speaker is enabled
- What LEDs are set
- What the remaining battery life is (float between 0 and 1)

See the section on the wiimote structure for information on how to obtain these values.

The Disconnect Event

The *disconnect* event is set by wiiuse when a wiimote has disconnected.
A disconnect occurs when one of the following conditions are met:

1. The connection is dropped
2. The POWER button on the wiimote is held for a couple seconds
3. The battery is depleted and the wiimote turns off

The Data Read Event

The *read* event is set by wiiuse when the wiimote returns data that was previously requested to be read from either ROM or its registers.

Data can be requested to be read using the following function:

```
int wiiuse_read_data(struct wiimote_t* wm, byte* buffer, unsigned int offset, unsigned short len);
```

Where *buffer* is an allocated buffer big enough to hold the data to be read, *offset* is the wiimote address to read from, and *len* is the length of the block to read.

When the read event is returned you can obtain the data from wiimote_t::read_req. For example, to make sure the event corresponds to the request you asked for, check that wiimote_t.read_req.addr is the same as the *offset* you passed to wiiuse_read_data(). The actual data returned is in wiimote_t.read_req.buf.

The Wiimote Structure

The wiimote structure is passed to each callback and has all the information related to the devices current state and configuration. This structure is *read only* and should be treated as such.

Only key members are listed here, if you'd like to see the full structure it is defined in *include/wiiuse.h*.

int unid

The unique identifier assigned to the wiimote during the wiiuse_init() stage.

struct expansion_t exp

The expansion device plugged into the wiimote.
More information about this in the expansion structure section.

struct orient_t orient

The orientation of the accelerometer on each axis. This structure has *roll* and *pitch* floats ranging from -180 to 180 degrees.

The *yaw* float ranges from around -26 to 26 degrees and can only be calculated if IR tracking is enabled.
If IR tracking is disabled *yaw* will always be 0.

This structure also has floats *a_roll* and *a_pitch* that are the current absolute roll and pitch. These values are not influenced by any smoothing algorithms and represent the exact roll and pitch the wiimote reported for that event.

Accelerometers produce a lot of noise, so to reduce this wiiuse has implemented an exponential moving average for each angle.
You can use the function wiiuse_set_smooth_alpha() to change the alpha value of the equation.
You can also disable the averaging feature by disabling the corresponding flag with the wiiuse_set_flags() function.

struct gforce_t gforce

The gravity forces on each axis as reported by the accelerometer.
The accelerometer is sensitive to within +/- 3 gravity units.
This structure has a *x*, *y*, and *z* floats.

For example, if *y* is 2.3 then there are 2.3 gravity units applied on the positive direction of the y axis.

struct ir_t ir

This structure has all the information related to the IR pointing device.
See more information about it in the IR structure section.

unsigned short btns

The buttons that were just pressed this event.
Typically you do not need to use this directly, see the button section for how to use this.

unsigned short btns_held

The buttons that are being held.
Typically you do not need to use this directly, see the button section for how to use this.

unsigned short btns_released

The buttons that have just been released this event.
Typically you do not need to use this directly, see the button section for how to use this.

The Expansion Structure

The expansion structure keeps track of what type of expansion is connected to the expansion port and also all assoicated data.

type

The type parameter holds what type of expansion is attached, it can be one of the following:

- EXP_NONE
- EXP_NUNCHUK
- EXP_CLASSIC
- EXP_GUITAR_HERO_3

Based on what the type is, you can access the assoicated extension data by one of the following members:

- struct nunchuk_t nunchuk
- struct classic_ctrl_t classic
- struct guitar_hero_3_t gh3

The Nunchuk Structure

The nunchuk structure is accessible through the wiimote structure and has all the information related to the devices current state and configuration. This structure is *read only* and should be treated as such.

Only key members are listed here, if you'd like to see the full structure it is defined in *include/wiiuse.h*.

The joystick will only generate an event if the position has changed.
Holding the joystick in position will not cause continuous events unless the corresponding flag has been set for wiiuse.
For more information on setting flags see the Setting wiiuse Flags section.

struct orient_t orient

The orientation of the accelerometer on each axis.

This is the same as the wiimote orient member.

struct gforce_t gforce

The gravity forces on each axis as reported by the accelerometer.

This is the same as the wiimote gforce member.

struct joystick_t js

The joystick has *min*, *max*,and *center* members, each of which have *x* and *y* byte (unsigned char) members.
These members are probably not very interesting though.

The more imporant members are the *ang* and *mag* floats.

The *ang* is the angle at which the joystick is being held.
Straight up is 0 degrees, to the right is 90 degrees, down is 180 degrees, and to the left is 270 degrees.

The angle can often be 'not a number' (nan).
This may occur if the joystick is in the central position.

The *mag* is the magnitude at which the joystick is being held.
In the center is 0, and at the far edges is 1. So if the magnitude is 0.5 then the joystick is half way between the middle and outter edge.

byte btns

The buttons that were just pressed this event.
Typically you do not need to use this directly, see the button section for how to use this.

byte btns_held

The buttons that are being held.
Typically you do not need to use this directly, see the button section for how to use this.

byte btns_released

The buttons that have just been released this event.
Typically you do not need to use this directly, see the button section for how to use this.

The Classic Controller Structure

The classic controller structure is accessible through the wiimote structure and has all the information related to the devices current state and configuration. This structure is *read only* and should be treated as such.

Only key members are listed here, if you'd like to see the full structure it is defined in *include/wiiuse.h*.

> The joystick will only generate an event if the position has changed.
> Holding the joystick in position will not cause continuous events unless the corresponding flag has been set for wiiuse.
> For more information on setting flags see the Setting wiiuse Flags section.

struct joystick_t ljs

The left joystick. This is the same as the nunchuk joystick.

struct joystick_t rjs

The right joystick. This is the same as the nunchuk joystick.

short btns

The buttons that were just pressed this event.
Typically you do not need to use this directly, see the button section for how to use this.

short btns_held

The buttons that are being held.
Typically you do not need to use this directly, see the button section for how to use this.

short btns_released

The buttons that have just been released this event.
Typically you do not need to use this directly, see the button section for how to use this.

float r_shoulder

The right shoulder button is analog rather than digital like the rest of the buttons.
This ranges from 0 (not pressed) to 1 (fully pressed). So 0.5 is half pressed.

float l_shoulder

The left shoulder button is analog rather than digital like the rest of the buttons.
This ranges from 0 (not pressed) to 1 (fully pressed). So 0.5 is half pressed.

The Guitar Hero 3 Structure

The Guitar Hero 3 structure is accessible through the wiimote structure and has all the information related to the devices current state and configuration. This structure is *read only* and should be treated as such.

Only key members are listed here, if you'd like to see the full structure it is defined in *include/wiiuse.h*.

> The joystick will only generate an event if the position has changed.
> Holding the joystick in position will not cause continuous events unless the corresponding flag has been set for wiiuse.
> For more information on setting flags see the Setting wiiuse Flags section.

struct joystick_t js

The joystick. This is the same as the nunchuk joystick.

short btns

The buttons that were just pressed this event.
Typically you do not need to use this directly, see the button section for how to use this.

short btns_held

The buttons that are being held.
Typically you do not need to use this directly, see the button section for how to use

this.

short btns_released

The buttons that have just been released this event.
Typically you do not need to use this directly, see the button section for how to use this.

float whammy_bar

The whammy bar is an analog "button".
This ranges from 0 (not pressed) to 1 (fully pressed). So 0.5 is half pressed.

The IR Structure

The IR structure is accessible through the wiimote structure and has all the information related to the devices current state and configuration. This structure is *read only* and should be treated as such.

Only key members are listed here, if you'd like to see the full structure it is defined in *include/wiiuse.h*.

byte num_dots

This is how many IR sources the wiimote currently sees.
With the sensor bar that shipped with the Wii there are a maximum of 2 sources visible.

int x, y

The calculated X and Y coordinates of the cursor.
Remember, motion sensing should be enabled for this to be accurate.

These coordinates are in the range specified by the virtual screen resolution.

int z

An arbitrary number that represents how far away from the sensor bar the wiimote is.
This number increases as the distance increases.

This can only be calculated if there are at least 2 IR sources.

enum aspect_t aspect

The aspect ratio of the screen.
For more information see the aspect ratio section.

struct ir_dot_t dot[4]

A wiimote can see up to 4 IR sources, each sources data is stored in one of these objects.
The ir_dot_t structure has the following important data:

byte visible

This is set to 1 if the IR source is visible, 0 if it is not.
If the source is not visible then the rest of the data in this object is garbage from an older event and does no represent the current state.

unsigned int x, y

Corrected XY coordinates of the IR source.
These values are converted from *rx* and *ry* and are used directly in the calculation of the wiimotes cursor position.

The range of these values is determined by the *virtual screen resolution*. More information about this is discussed in the IR Tracking secion.

short rx, ry

Raw XY coordinates of the IR source as reported by the wiimote.

Checking Button States

Checking the state of buttons can be accomplished with a few built in macros.

Macros:

The *dev* parameter in the macros can be either a wiimote, nunchuk, classic controller, or Guitar Hero 3 object.

- IS_PRESSED()

  Will return 1 if the specified button *is currently pressed*.

  Unlike the other macros, this is the definitive button state macro.
  It does not matter if a button was just pressed this event or has been held,

it will return 1 if the button is pressed *at all*.

```
if (IS_PRESSED(dev, button)) {
	/* button is pressed */
} else {
	/* button is not pressed */
}
```

- IS_JUST_PRESSED()

  Will return 1 if the specified button *has just been pressed this event*.

```
if (IS_JUST_PRESSED(dev, button)) {
	/* button has just been pressed */
} else {
	/* button has not just been pressed */
}
```

- IS_RELEASED()

  Will return 1 if the specified button *has just been released this event*.

```
if (IS_RELEASED(dev, button)) {
	/* button has just been released */
} else {
	/* button has not just been released */
}
```

- IS_HELD()

  Will return 1 if the specified button *is being held* (that is it was previously pressed but not yet released).

```
if (IS_HELD(dev, button)) {
	/* button is being held */
} else {
	/* button is not being held */
}
```

Wiimote Button Codes:

- WIIMOTE_BUTTON_ONE
- WIIMOTE_BUTTON_TWO
- WIIMOTE_BUTTON_B
- WIIMOTE_BUTTON_A
- WIIMOTE_BUTTON_MINUS
- WIIMOTE_BUTTON_HOME
- WIIMOTE_BUTTON_LEFT
- WIIMOTE_BUTTON_RIGHT
- WIIMOTE_BUTTON_DOWN
- WIIMOTE_BUTTON_UP
- WIIMOTE_BUTTON_PLUS

Nunchuk Button Codes:

- NUNCHUK_BUTTON_C
- NUNCHUK_BUTTON_Z

Classic Controller Button Codes:

- CLASSIC_CTRL_BUTTON_UP
- CLASSIC_CTRL_BUTTON_LEFT
- CLASSIC_CTRL_BUTTON_DOWN
- CLASSIC_CTRL_BUTTON_RIGHT
- CLASSIC_CTRL_BUTTON_X
- CLASSIC_CTRL_BUTTON_A
- CLASSIC_CTRL_BUTTON_Y
- CLASSIC_CTRL_BUTTON_B
- CLASSIC_CTRL_BUTTON_PLUS
- CLASSIC_CTRL_BUTTON_HOME
- CLASSIC_CTRL_BUTTON_MINUS
- CLASSIC_CTRL_BUTTON_ZR
- CLASSIC_CTRL_BUTTON_ZL
- CLASSIC_CTRL_BUTTON_FULL_R *(R fully pressed)*
- CLASSIC_CTRL_BUTTON_FULL_L *(L fully pressed)*

Guitar Hero 3 Button Codes:

- GUITAR_HERO_3_BUTTON_YELLOW
- GUITAR_HERO_3_BUTTON_GREEN
- GUITAR_HERO_3_BUTTON_BLUE
- GUITAR_HERO_3_BUTTON_RED
- GUITAR_HERO_3_BUTTON_ORANGE
- GUITAR_HERO_3_BUTTON_PLUS
- GUITAR_HERO_3_BUTTON_MINUS
- GUITAR_HERO_3_BUTTON_STRUM_UP
- GUITAR_HERO_3_BUTTON_STRUM_DOWN

Motion Sensing

Motion sensing is not enabled by default.

To enable motion sensing for a wiimote device you must enable it by calling the wiiuse_motion_sensing() function.
This function can also be used to disable motion sensing.

Since the accelerometer produces a lot of noise wiiuse will only generate an event if any angle has changed by a significant degree. See the section on the orientation threshold for more information on how this works and how to control it.

The function is in the form:

```
void wiiuse_motion_sensing(struct wiimote_t* wm, int status);
```

Where *status* is 1 to enable the accelerometer and 0 to disable it.

Motion sensing is always enabled for the nunchuk and can not be disabled.

IR Tracking

IR tracking is not enabled by default.

To enable IR tracking for a wiimote device you must enable it by calling the wiiuse_set_ir() function.
This function can also be used to disable IR tracking.

The function is in the form:

```
void wiiuse_set_ir(struct wiimote_t* wm, int status);
```

Where *status* is 1 to enable IR tracking and 0 to disable it.

Wiiuse tries to approximate where the cursor should be by using the official Wii sensor bar.
As long as the sensor bar has two IR nodes spaced apart, and is level, wiiuse can use both sources to calculate where the cursor should be.

To properly calculate where the cursor is on the screen the accelerometer should be turned on.
If you enable IR sensing you should always enable motion sensing as well.

Virtual Screen Resolution

IR tracking reports an XY position on a virtual screen whose resolution is defined by the user. By default this resolution is dependent on the set aspect ratio and is:

- For 16:9, 660x370
- For 4:3, 560x420

This resolution can be changed by calling the function:

```
void wiiuse_set_ir_vres(struct wiimote_t* wm, unsigned int x, unsigned int y);
```

The virtual screen resolution only applies to the x and y members of the IR structure and does not apply to the individual IR source positions defined in ir_t::ir_dot_t. The individual IR source coordinates are on a fixed virtual screen resolution of 1024x768 and can not be changed.

The coordinate (0,0) is at the top left hand corner of the virtual screen.

TV/Monitor Screen Ratio

The screen ratio is important because it ensures that the vertical and horizontal sensitivity are equal.
By default the TV/monitor's aspect ratio is 4:3. To change this use the function:

```
void wiiuse_set_aspect_ratio(struct wiimote_t* wm, enum aspect_t aspect);
```

The *aspect* parameter can be either of the following:

- WIIUSE_ASPECT_4_3
- WIIUSE_ASPECT_16_9

The aspect ratio setting is stored in the IR structure.

Whenever this function is called the virtual screen resolution is changed to the default values listed in the screen ratio section.

Sensor Bar Position

By default the IR sensor bar is considered to be above the TV/monitor. To change this use the function:

```
void wiiuse_set_ir_position(struct wiimote_t* wm, enum ir_position_t pos);
```

The *pos* parameter can be either of the following:

- WIIUSE_IR_ABOVE

  This means that the IR sensor bar is **centered above** the TV/monitor.

- WIIUSE_IR_BELOW

  This means that the IR sensor bar is **centered below** the TV/monitor.

Whenever this function is called the virtual screen resolution is changed to the default values listed in the screen ratio section.

The following example demonstrates how to setup a 16:9 TV/monitor with the IR sensor bar above the screen and the virtual screen resolution 1066x600 is:

```
wiiuse_set_aspect_ratio(wm, WIIUSE_ASPECT_16_9);
wiiuse_set_ir_position(wm, WIIUSE_IR_ABOVE);
wiiuse_set_ir_vres(wm, 1066, 600);
```

IR Sensitivity

The sensitivity of the IR camera can be turned up or down depending on your needs.
Like the Wii, wiiuse can set the camera sensitivity to a degree between 1 (lowest) and 5 (highest).
The default is 3.

Use the following function to set the sensitivity:

```
void wiiuse_set_ir_sensitivity(struct wiimote_t* wm, int level);
```

If the level < 1 then it will be set to 1, and if level > 5 then it will be set to 5.
The current sensitivity setting can be obtained by using the following macro function:

```
WIIUSE_GET_IR_SENSITIVITY(wm)
```

Checking wiimote States

A few macros are provided to easily check the state of a wiimote.
Each macro takes one parameter, a pointer to a wiimote structure.

- WIIUSE_USING_ACC(wm)

  Returns 1 if the wiimote's accelerometer is currently in use (motion sensing is enabled).

- WIIUSE_USING_EXP(wm)

  Returns 1 if an expansion is plugged into the wiimote.

- WIIUSE_USING_IR(wm)

  Returns 1 if the wiimote's IR camera is currently in use.

- WIIUSE_USING_SPEAKER(wm)

  Returns 1 if the wiimote's speaker is enabled.

- WIIUSE_IS_LED_SET(wm, number)

  Returns 1 if the wiimote's *number* LED (between 1 and 4) is set.

Setting wiiuse Flags

Flags can be set to change the behavior of wiiuse.

Flags are set using the wiiuse_set_flags() function, in the form:

```
int wiiuse_set_flags(struct wiimote_t* wm, int enable, int disable);
```

The *enable* and *disable* parameters may be any of the following, and may be OR'ed together:

- WIIUSE_SMOOTHING

  This flag is set by default.

  Accelerometers generate a lot of noise and may not give relatively consistent angle calculations.
  If this flag is set then the angles reported by wiiuse will be smoothed.
  For more information see details for the orientation structure.

- WIIUSE_CONTINUOUS

  By default wiiuse will only generate an event if the status of the wiimote has changed (ex: button press, movement of a joystick, etc).

  By enabling this flag then wiiuse will constantly generate events even if the status of the device has not been altered since the last event.

Questions and More Information

If you have a question about something here or you need more information about a particular aspect of the library, please drop by the forums.

If you found a bug or have a feature request, the forums have a link to where you should post them (sticky topic at the top of the assoicated forum).

Wiiuse is packaged with a full example in *api/example.c* and is also available on the website.
The full source documentation (generated by doxygen) for the latest release is available on the website as well.
The website is http://wiiuse.net/.