Wii Library

0.1.0

Project Overview

Contents

1 Changelog for the Wii Library

Release v0.1.0 - [2018-11-07]

1. Initial release of library. Validated to work on:
    - PIC32MX370F512L
    - PIC32MX795F512H
2. Developed using Microchip's legacy library on MPLAB 8 and then ported over to MPLAB X.
3. Created and validated for Hackaday Supercon 2018.

2 Background

The 'Wii Library' provides support for communicating with external Wii peripherals on the following platform(s):

- PIC32MX

Dependencies

This project is dependent upon the following projects:

1. doxygen
    - Used to generate <docs> folder output.
    - Cloned instance should be named "doxygen" and live as a sibling to this repository.
2. lib-timing
    - Directory cloned into must be <lib-timing>.
    - Directory must be a sibling to the clone of this repository.

1. lib-i2c
    - Directory cloned into must be <lib-i2c>.
    - Directory must be a sibling to the clone of this repository.

Detailed Overview

For complete details on how to use, modify, and expand this utility, please see the provided Doxygen Summary

Development History

For complete details on the what was changed for the latest release, please see the Changelog.md

Licensing

All code is provided 'as is'. You are free to modify, distribute, etc. the code within the bounds of the Mozilla Public License (v2.0).

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Class Documentation

5.1 _WiiClassic_StatusNormal Struct Reference

Defines bitfield interaction for status field queries from classic controllers.

`#include <wii_classic_controller.h>`

Collaboration diagram for _WiiClassic_StatusNormal:

Public Attributes

- uint8_t analogLeftX: 6
  Bits <5:0> for the left analog joystic along the x-axis.
- uint8_t analogRightXHigh: 2
  Bits <4:3> for the right analog joystic along the x-axis.
- uint8_t analogLeftY: 6
  Bits <5:0> for the left analog joystic along the y-axis.
- uint8_t analogRightXMid: 2
  Bits <2:1> for the right analog joystic along the x-axis.
- uint8_t analogRightY: 5
  Bits <4:0> for the right analog joystic along the y-axis.
- uint8_t leftTriggerHigh: 2
  Bits <4:3> for the left trigger.
- uint8_t analogRightXLow: 1
  Bit <0> for the right analog joystick along the x-axis.
- uint8_t rightTrigger: 5
  Bits <4:0> for the right trigger.
- uint8_t leftTriggerLow: 3
  Bits <2:0> for the left trigger.
- uint8_t RESERVED: 1
  Reserved bit used when passing through the Wii Motion Plus (should be high == 1).
- uint8_t buttonTriggerRight: 1
  Bit indicating status of button->-trigger-right.
- uint8_t buttonPlus: 1
  Bit indicating status of button->+.
- uint8_t buttonHome: 1
  Bit indicating status of button->home.
- uint8_t buttonMinus: 1
  Bit indicating status of button->-.
- uint8_t buttonTriggerLeft: 1
  Bit indicating status of button->-trigger-left.
- uint8_t dpadDown: 1
  Bit indicating status of d-pad->down.
- uint8_t dpadRight: 1
  Bit indicating status of d-pad->right.
- uint8_t dpadUp: 1
  Bit indicating status of d-pad->up.
- uint8_t dpadLeft: 1
  Bit indicating status of d-pad->left.

- uint8_t buttonZRight: 1

  Bit indicating status of button->z-right.
- uint8_t buttonX: 1

  Bit indicating status of button->x.
- uint8_t buttonA: 1

  Bit indicating status of button->a.
- uint8_t buttonY: 1

  Bit indicating status of button->y.
- uint8_t buttonB: 1

  Bit indicating status of button->b.
- uint8_t buttonZLeft: 1

  Bit indicating status of button->z-left.

5.1.1 Detailed Description

Defines bitfield interaction for status field queries from classic controllers.

Note

The definitions below are for use when directly connected to a classic controller.

Warning

The following is in little-endian format.

5.1.2 Member Data Documentation

5.1.2.1 analogLeftX

```
uint8_t _WiiClassic_StatusNormal::analogLeftX
```

Bits <5:0> for the left analog joystic along the x-axis.

5.1.2.2 analogLeftY

```
uint8_t _WiiClassic_StatusNormal::analogLeftY
```

Bits <5:0> for the left analog joystic along the y-axis.

5.1.2.3 analogRightXHigh

```
uint8_t _WiiClassic_StatusNormal::analogRightXHigh
```

Bits <4:3> for the right analog joystic along the x-axis.

5.1.2.4 analogRightXLow

`uint8_t _WiiClassic_StatusNormal::analogRightXLow`

Bit <0> for the right analog joystick along the x-axis.

5.1.2.5 analogRightXMid

`uint8_t _WiiClassic_StatusNormal::analogRightXMid`

Bits <2:1> for the right analog joystic along the x-axis.

5.1.2.6 analogRightY

`uint8_t _WiiClassic_StatusNormal::analogRightY`

Bits <4:0> for the right analog joystic along the y-axis.

5.1.2.7 buttonA

`uint8_t _WiiClassic_StatusNormal::buttonA`

Bit indicating status of button->a.

5.1.2.8 buttonB

`uint8_t _WiiClassic_StatusNormal::buttonB`

Bit indicating status of button->b.

5.1.2.9 buttonHome

`uint8_t _WiiClassic_StatusNormal::buttonHome`

Bit indicating status of button->home.

5.1.2.10 buttonMinus

`uint8_t _WiiClassic_StatusNormal::buttonMinus`

Bit indicating status of button->-.

5.1.2.11 buttonPlus

```
uint8_t _WiiClassic_StatusNormal::buttonPlus
```

Bit indicating status of button->+.

5.1.2.12 buttonTriggerLeft

```
uint8_t _WiiClassic_StatusNormal::buttonTriggerLeft
```

Bit indicating status of button->-trigger-left.

5.1.2.13 buttonTriggerRight

```
uint8_t _WiiClassic_StatusNormal::buttonTriggerRight
```

Bit indicating status of button->-trigger-right.

5.1.2.14 buttonX

```
uint8_t _WiiClassic_StatusNormal::buttonX
```

Bit indicating status of button->x.

5.1.2.15 buttonY

```
uint8_t _WiiClassic_StatusNormal::buttonY
```

Bit indicating status of button->y.

5.1.2.16 buttonZLeft

```
uint8_t _WiiClassic_StatusNormal::buttonZLeft
```

Bit indicating status of button->z-left.

5.1.2.17 buttonZRight

```
uint8_t _WiiClassic_StatusNormal::buttonZRight
```

Bit indicating status of button->z-right.

5.1.2.18 dpadDown

`uint8_t _WiiClassic_StatusNormal::dpadDown`

Bit indicating status of d-pad->down.

5.1.2.19 dpadLeft

`uint8_t _WiiClassic_StatusNormal::dpadLeft`

Bit indicating status of d-pad->left.

5.1.2.20 dpadRight

`uint8_t _WiiClassic_StatusNormal::dpadRight`

Bit indicating status of d-pad->right.

5.1.2.21 dpadUp

`uint8_t _WiiClassic_StatusNormal::dpadUp`

Bit indicating status of d-pad->up.

5.1.2.22 leftTriggerHigh

`uint8_t _WiiClassic_StatusNormal::leftTriggerHigh`

Bits <4:3> for the left trigger.

5.1.2.23 leftTriggerLow

`uint8_t _WiiClassic_StatusNormal::leftTriggerLow`

Bits <2:0> for the left trigger.

5.1.2.24 RESERVED

`uint8_t _WiiClassic_StatusNormal::RESERVED`

Reserved bit used when passing through the Wii Motion Plus (should be high == 1).

5.1.2.25 rightTrigger

```
uint8_t _WiiClassic_StatusNormal::rightTrigger
```

Bits <4:0> for the right trigger.

The documentation for this struct was generated from the following file:

- wii_classic_controller.h

5.2 _WiiClassic_StatusPassThrough Struct Reference

Defines bitfield interaction for status field queries from classic controllers.

```
#include <wii_classic_controller.h>
```

Collaboration diagram for _WiiClassic_StatusPassThrough:

Public Attributes

- uint8_t dpadUp: 1
  Bit indicating status of d-pad->up.
- uint8_t analogLeftX: 5
  Bits <5:1> for the left analog joystic along the x-axis.
- uint8_t analogRightXHigh: 2
  Bits <4:3> for the right analog joystic along the x-axis.
- uint8_t dpadLeft: 1
  Bit indicating status of d-pad->left.
- uint8_t analogLeftY: 5
  Bits <5:1> for the left analog joystic along the y-axis.
- uint8_t analogRightXMid: 2
  Bits <2:1> for the right analog joystic along the x-axis.
- uint8_t analogRightY: 5
  Bits <4:0> for the right analog joystic along the y-axis.
- uint8_t leftTriggerHigh: 2
  Bits <4:3> for the left trigger.
- uint8_t analogRightXLow: 1
  Bit <0> for the right analog joystick along the x-axis.
- uint8_t rightTrigger: 5
  Bits <4:0> for the right trigger.
- uint8_t leftTriggerLow: 3
  Bits <2:0> for the left trigger.
- uint8_t extensionConnected: 1
  Pass through bit used when passing through the Wii Motion Plus (should be high == 1).
- uint8_t buttonTriggerRight: 1
  Bit indicating status of button->-trigger-right.
- uint8_t buttonPlus: 1
  Bit indicating status of button->+.
- uint8_t buttonHome: 1
  Bit indicating status of button->home.

- uint8_t buttonMinus: 1

  Bit indicating status of button->-.
- uint8_t buttonTriggerLeft: 1

  Bit indicating status of button->-trigger-left.
- uint8_t dpadDown: 1

  Bit indicating status of d-pad->down.
- uint8_t dpadRight: 1

  Bit indicating status of d-pad->right.
- uint8_t RESERVED: 2

  Rserved bits (should be == 0b00).
- uint8_t buttonZRight: 1

  Bit indicating status of button->z-right.
- uint8_t buttonX: 1

  Bit indicating status of button->x.
- uint8_t buttonA: 1

  Bit indicating status of button->a.
- uint8_t buttonY: 1

  Bit indicating status of button->y.
- uint8_t buttonB: 1

  Bit indicating status of button->b.
- uint8_t buttonZLeft: 1

  Bit indicating status of button->z-left.

5.2.1 Detailed Description

Defines bitfield interaction for status field queries from classic controllers.

Note

The definitions below are for use when connected to a classic controller in pass-through mode (e.g. when using Wii Motion Plus + classic controller).

Warning

The following is in little-endian format.

5.2.2 Member Data Documentation

5.2.2.1 analogLeftX

`uint8_t _WiiClassic_StatusPassThrough::analogLeftX`

Bits <5:1> for the left analog joystic along the x-axis.

5.2.2.2 analogLeftY

```
uint8_t _WiiClassic_StatusPassThrough::analogLeftY
```

Bits <5:1> for the left analog joystic along the y-axis.

5.2.2.3 analogRightXHigh

```
uint8_t _WiiClassic_StatusPassThrough::analogRightXHigh
```

Bits <4:3> for the right analog joystic along the x-axis.

5.2.2.4 analogRightXLow

```
uint8_t _WiiClassic_StatusPassThrough::analogRightXLow
```

Bit <0> for the right analog joystick along the x-axis.

5.2.2.5 analogRightXMid

```
uint8_t _WiiClassic_StatusPassThrough::analogRightXMid
```

Bits <2:1> for the right analog joystic along the x-axis.

5.2.2.6 analogRightY

```
uint8_t _WiiClassic_StatusPassThrough::analogRightY
```

Bits <4:0> for the right analog joystic along the y-axis.

5.2.2.7 buttonA

```
uint8_t _WiiClassic_StatusPassThrough::buttonA
```

Bit indicating status of button->a.

5.2.2.8 buttonB

```
uint8_t _WiiClassic_StatusPassThrough::buttonB
```

Bit indicating status of button->b.

5.2.2.9 buttonHome

```
uint8_t _WiiClassic_StatusPassThrough::buttonHome
```

Bit indicating status of button->home.

5.2.2.10 buttonMinus

```
uint8_t _WiiClassic_StatusPassThrough::buttonMinus
```

Bit indicating status of button->-.

5.2.2.11 buttonPlus

```
uint8_t _WiiClassic_StatusPassThrough::buttonPlus
```

Bit indicating status of button->+.

5.2.2.12 buttonTriggerLeft

```
uint8_t _WiiClassic_StatusPassThrough::buttonTriggerLeft
```

Bit indicating status of button->-trigger-left.

5.2.2.13 buttonTriggerRight

```
uint8_t _WiiClassic_StatusPassThrough::buttonTriggerRight
```

Bit indicating status of button->-trigger-right.

5.2.2.14 buttonX

```
uint8_t _WiiClassic_StatusPassThrough::buttonX
```

Bit indicating status of button->x.

5.2.2.15 buttonY

```
uint8_t _WiiClassic_StatusPassThrough::buttonY
```

Bit indicating status of button->y.

5.2.2.16 buttonZLeft

```
uint8_t _WiiClassic_StatusPassThrough::buttonZLeft
```

Bit indicating status of button->z-left.

5.2.2.17 buttonZRight

```
uint8_t _WiiClassic_StatusPassThrough::buttonZRight
```

Bit indicating status of button->z-right.

5.2.2.18 dpadDown

```
uint8_t _WiiClassic_StatusPassThrough::dpadDown
```

Bit indicating status of d-pad->down.

5.2.2.19 dpadLeft

```
uint8_t _WiiClassic_StatusPassThrough::dpadLeft
```

Bit indicating status of d-pad->left.

5.2.2.20 dpadRight

```
uint8_t _WiiClassic_StatusPassThrough::dpadRight
```

Bit indicating status of d-pad->right.

5.2.2.21 dpadUp

```
uint8_t _WiiClassic_StatusPassThrough::dpadUp
```

Bit indicating status of d-pad->up.

5.2.2.22 extensionConnected

```
uint8_t _WiiClassic_StatusPassThrough::extensionConnected
```

Pass through bit used when passing through the Wii Motion Plus (should be high == 1).

5.2.2.23 leftTriggerHigh

```
uint8_t _WiiClassic_StatusPassThrough::leftTriggerHigh
```

Bits <4:3> for the left trigger.

5.2.2.24 leftTriggerLow

```
uint8_t _WiiClassic_StatusPassThrough::leftTriggerLow
```

Bits <2:0> for the left trigger.

5.2.2.25 RESERVED

```
uint8_t _WiiClassic_StatusPassThrough::RESERVED
```

Rserved bits (should be == 0b00).

5.2.2.26 rightTrigger

```
uint8_t _WiiClassic_StatusPassThrough::rightTrigger
```

Bits <4:0> for the right trigger.

The documentation for this struct was generated from the following file:

- wii_classic_controller.h

5.3 _WiiLib_Device Struct Reference

Defines the tracking information used when communicating with Wii targets.

```
#include <wii_lib.h>
```

Collaboration diagram for _WiiLib_Device:

Public Attributes

- I2C_Device i2c

  I2C device information. Used when communicating with Wii device over I2C.
- WII_LIB_TARGET_DEVICE target

  Target device type intended for communication.
- uint8_t dataEncrypted

  Flag indicating if data read is encrypted.
- uint8_t calculateRelativePosition

  Flag inidicating if the relative position values should be calculated (defaults to 'WII_LIB_DEFAULT_CALCULATE_↩ RELATIVE_POSITION').
- uint8_t dataCurrent [WII_LIB_MAX_PAYLOAD_SIZE]

  Payload used when storing the most recently read data in from the target device.
- WiiLib_Interface interfaceCurrent

  Instance of most recently read-in status values for interface (buttons, accelerometers, etc.) on the target device.
- WiiLib_Interface interfaceHome

  Instance of status values associated with the home position for the interface (buttons, accelerometers, etc.) on the target device.
- WiiLib_Interface interfaceRelative

  Relative interface values obtained by taking 'interfaceCurrent' and subtracting 'interfaceHome' for all interface values.

5.3.1 Detailed Description

Defines the tracking information used when communicating with Wii targets.

Note

All data presented has been processed and can be easily indexed by treating it as a structure of bitfields where the corresponding bitfield mapping is defined in the target-specific header file.

5.3.2 Member Data Documentation

5.3.2.1 calculateRelativePosition

```
uint8_t _WiiLib_Device::calculateRelativePosition
```

Flag inidicating if the relative position values should be calculated (defaults to 'WII_LIB_DEFAULT_CALCULATE↩ _RELATIVE_POSITION').

5.3.2.2 dataCurrent

```
uint8_t _WiiLib_Device::dataCurrent[WII_LIB_MAX_PAYLOAD_SIZE]
```

Payload used when storing the most recently read data in from the target device.

5.3.2.3 dataEncrypted

```
uint8_t _WiiLib_Device::dataEncrypted
```

Flag indicating if data read is encrypted.

5.3.2.4 i2c

```
I2C_Device _WiiLib_Device::i2c
```

I2C device information. Used when communicating with Wii device over I2C.

5.3.2.5 interfaceCurrent

```
WiiLib_Interface _WiiLib_Device::interfaceCurrent
```

Instance of most recently read-in status values for interface (buttons, accelerometers, etc.) on the target device.

5.3.2.6 interfaceHome

```
WiiLib_Interface _WiiLib_Device::interfaceHome
```

Instance of status values associated with the home position for the interface (buttons, accelerometers, etc.) on the target device.

5.3.2.7 interfaceRelative

```
WiiLib_Interface _WiiLib_Device::interfaceRelative
```

Relative interface values obtained by taking 'interfaceCurrent' and subtracting 'interfaceHome' for all interface values.

5.3.2.8 target

```
WII_LIB_TARGET_DEVICE _WiiLib_Device::target
```

Target device type intended for communication.

The documentation for this struct was generated from the following file:

- wii_lib.h

5.4 _WiiLib_Interface Struct Reference

Used to track the state of a Wii controller's buttons, accel, etc.

`#include <wii_lib.h>`

Collaboration diagram for _WiiLib_Interface:

Public Attributes

- uint8_t buttonA
  Flag indicating status of A button (pressed == high).
- uint8_t buttonB
  Flag indicating status of B button (pressed == high).
- uint8_t buttonC
  Flag indicating status of C button (pressed == high).
- uint8_t buttonX
  Flag indicating status of X button (pressed == high).
- uint8_t buttonY
  Flag indicating status of Y button (pressed == high).
- uint8_t buttonZL
  Flag indicating status of the left z button (pressed == high).
- uint8_t buttonZR
  Flag indicating status of the right z button (pressed == high).
- uint8_t buttonMinus
  Flag indicating status of minus [-] button.
- uint8_t buttonHome
  Flag indicating status of home button.
- uint8_t buttonPlus
  Flag indicating status of plus [+] button.
- uint8_t dpadLeft
  Flag indicating status of the left d-pad button (pressed == high).
- uint8_t dpadUp
  Flag indicating status of the top d-pad button (pressed == high).
- uint8_t dpadRight
  Flag indicating status of the right d-pad button (pressed == high).
- uint8_t dpadDown
  Flag indicating status of the bottom d-pad button (pressed == high).
- uint8_t buttonLeftTrigger
  Flag indicating status of left trigger button.
- uint8_t buttonRightTrigger
  Flag indicating status of right trigger button.
- int8_t triggerLeft
  Value of the left [analog] trigger.
- int8_t triggerRight
  Value of the right [analog] trigger.
- int16_t analogLeftX
  Value of the left analog joystick along the x-axis.
- int16_t analogLeftY
  Value of the left analog joystick along the y-axis.
- int16_t analogRightX

Value of the right analog joystick along the x-axis.

- int16_t analogRightY

  Value of the right analog joystick along the y-axis.

- int16_t accelX

  Value of the [10-bit] accelerometer along the x-axis.

- int16_t accelY

  Value of the [10-bit] accelerometer along the y-axis.

- int16_t accelZ

  Value of the [10-bit] accelerometer along the z-axis.

- int16_t gyroX

  Value of the gyroscope along the x-axis.

- int16_t gyroY

  Value of the gyroscope along the y-axis.

- int16_t gyroZ

  Value of the gyroscope along the z-axis.

5.4.1 Detailed Description

Used to track the state of a Wii controller's buttons, accel, etc.

Defines every known type of feature across Wii controllers.

Note

Wii nunchuck's use a single Z button and have one joystick, however the classic controller has a left and right version of both. For the purposes of tracking, a non-sided / generic joystick and z button options are not provided.
Using signed integers to make it easier to do a reltive position tracking array.

5.4.2 Member Data Documentation

5.4.2.1 accelX

```
int16_t _WiiLib_Interface::accelX
```

Value of the [10-bit] accelerometer along the x-axis.

5.4.2.2 accelY

```
int16_t _WiiLib_Interface::accelY
```

Value of the [10-bit] accelerometer along the y-axis.

5.4.2.3 accelZ

```
int16_t _WiiLib_Interface::accelZ
```

Value of the [10-bit] accelerometer along the z-axis.

5.4.2.4 analogLeftX

```
int16_t _WiiLib_Interface::analogLeftX
```

Value of the left analog joystick along the x-axis.

5.4.2.5 analogLeftY

```
int16_t _WiiLib_Interface::analogLeftY
```

Value of the left analog joystick along the y-axis.

5.4.2.6 analogRightX

```
int16_t _WiiLib_Interface::analogRightX
```

Value of the right analog joystick along the x-axis.

5.4.2.7 analogRightY

```
int16_t _WiiLib_Interface::analogRightY
```

Value of the right analog joystick along the y-axis.

5.4.2.8 buttonA

```
uint8_t _WiiLib_Interface::buttonA
```

Flag indicating status of A button (pressed == high).

5.4.2.9 buttonB

```
uint8_t _WiiLib_Interface::buttonB
```

Flag indicating status of B button (pressed == high).

5.4.2.10 buttonC

```
uint8_t _WiiLib_Interface::buttonC
```

Flag indicating status of C button (pressed == high).

5.4.2.11 buttonHome

```
uint8_t _WiiLib_Interface::buttonHome
```

Flag indicating status of home button.

5.4.2.12 buttonLeftTrigger

```
uint8_t _WiiLib_Interface::buttonLeftTrigger
```

Flag indicating status of left trigger button.

5.4.2.13 buttonMinus

```
uint8_t _WiiLib_Interface::buttonMinus
```

Flag indicating status of minus [-] button.

5.4.2.14 buttonPlus

```
uint8_t _WiiLib_Interface::buttonPlus
```

Flag indicating status of plus [+] button.

5.4.2.15 buttonRightTrigger

```
uint8_t _WiiLib_Interface::buttonRightTrigger
```

Flag indicating status of right trigger button.

5.4.2.16 buttonX

```
uint8_t _WiiLib_Interface::buttonX
```

Flag indicating status of X button (pressed == high).

5.4.2.17 buttonY

```
uint8_t _WiiLib_Interface::buttonY
```

Flag indicating status of Y button (pressed == high).

5.4.2.18 buttonZL

```
uint8_t _WiiLib_Interface::buttonZL
```

Flag indicating status of the left z button (pressed == high).

5.4.2.19 buttonZR

```
uint8_t _WiiLib_Interface::buttonZR
```

Flag indicating status of the right z button (pressed == high).

5.4.2.20 dpadDown

```
uint8_t _WiiLib_Interface::dpadDown
```

Flag indicating status of the bottom d-pad button (pressed == high).

5.4.2.21 dpadLeft

```
uint8_t _WiiLib_Interface::dpadLeft
```

Flag indicating status of the left d-pad button (pressed == high).

5.4.2.22 dpadRight

```
uint8_t _WiiLib_Interface::dpadRight
```

Flag indicating status of the right d-pad button (pressed == high).

5.4.2.23 dpadUp

```
uint8_t _WiiLib_Interface::dpadUp
```

Flag indicating status of the top d-pad button (pressed == high).

5.4.2.24 gyroX

```
int16_t _WiiLib_Interface::gyroX
```

Value of the gyroscope along the x-axis.

5.4.2.25 gyroY

```
int16_t _WiiLib_Interface::gyroY
```

Value of the gyroscope along the y-axis.

5.4.2.26 gyroZ

```
int16_t _WiiLib_Interface::gyroZ
```

Value of the gyroscope along the z-axis.

5.4.2.27 triggerLeft

```
int8_t _WiiLib_Interface::triggerLeft
```

Value of the left [analog] trigger.

5.4.2.28 triggerRight

```
int8_t _WiiLib_Interface::triggerRight
```

Value of the right [analog] trigger.

The documentation for this struct was generated from the following file:

- wii_lib.h

5.5 _WiiNunchuck_StatusNormal Struct Reference

Defines bitfield interaction for status field queries from nunchuck controllers.

```
#include <wii_nunchuck.h>
```

Collaboration diagram for _WiiNunchuck_StatusNormal:

Public Attributes

- uint8_t analogX: 8

  Bits <7:0> for the analog joystic along the x-axis.
- uint8_t analogY: 8

  Bits <7:0> for the analog joystic along the y-axis.
- uint8_t accelXHigh: 8

  Bits <9:2> for the accelerometer along the x-axis.
- uint8_t accelYHigh: 8

  Bits <9:2> for the accelerometer along the y-axis.
- uint8_t accelZHigh: 8

  Bits <9:2> for the accelerometer along the z-axis.
- uint8_t buttonZ: 1

  Bit indicating status of z button (pressed == low).
- uint8_t buttonC: 1

  Bit indicating status of c button (pressed == low).
- uint8_t accelXLow: 2

  Bits <1:0> for the accelerometer along the z-axis.
- uint8_t accelYLow: 2

  Bits <1:0> for the accelerometer along the y-axis.
- uint8_t accelZLow: 2

  Bits <1:0> for the accelerometer along the x-axis.

5.5.1 Detailed Description

Defines bitfield interaction for status field queries from nunchuck controllers.

Note

The definitions below are for use when directly connected to a nunchuck.

Warning

The following is in little-endian format.

5.5.2 Member Data Documentation

5.5.2.1 accelXHigh

```
uint8_t _WiiNunchuck_StatusNormal::accelXHigh
```

Bits <9:2> for the accelerometer along the x-axis.

5.5.2.2 accelXLow

`uint8_t _WiiNunchuck_StatusNormal::accelXLow`

Bits <1:0> for the accelerometer along the z-axis.

5.5.2.3 accelYHigh

`uint8_t _WiiNunchuck_StatusNormal::accelYHigh`

Bits <9:2> for the accelerometer along the y-axis.

5.5.2.4 accelYLow

`uint8_t _WiiNunchuck_StatusNormal::accelYLow`

Bits <1:0> for the accelerometer along the y-axis.

5.5.2.5 accelZHigh

`uint8_t _WiiNunchuck_StatusNormal::accelZHigh`

Bits <9:2> for the accelerometer along the z-axis.

5.5.2.6 accelZLow

`uint8_t _WiiNunchuck_StatusNormal::accelZLow`

Bits <1:0> for the accelerometer along the x-axis.

5.5.2.7 analogX

`uint8_t _WiiNunchuck_StatusNormal::analogX`

Bits <7:0> for the analog joystic along the x-axis.

5.5.2.8 analogY

`uint8_t _WiiNunchuck_StatusNormal::analogY`

Bits <7:0> for the analog joystic along the y-axis.

5.5.2.9 buttonC

`uint8_t _WiiNunchuck_StatusNormal::buttonC`

Bit indicating status of c button (pressed == low).

5.5.2.10 buttonZ

`uint8_t _WiiNunchuck_StatusNormal::buttonZ`

Bit indicating status of z button (pressed == low).

The documentation for this struct was generated from the following file:

- wii_nunchuck.h

5.6 _WiiNunchuck_StatusPassThrough Struct Reference

Defines bitfield interaction for status field queries from nunchuck controllers.

`#include <wii_nunchuck.h>`

Collaboration diagram for _WiiNunchuck_StatusPassThrough:

Public Attributes

- uint8_t analogX: 8
  Bits <7:0> for the analog joystic along the x-axis.
- uint8_t analogY: 8
  Bits <7:0> for the analog joystic along the y-axis.
- uint8_t accelXHigh: 8
  Bits <9:2> for the accelerometer along the x-axis.
- uint8_t accelYHigh: 8
  Bits <9:2> for the accelerometer along the y-axis.
- uint8_t extensionConnected: 1
  Bit indicating if extension is connected (1 == active).
- uint8_t accelZHigh: 7
  Bits <9:3> for the accelerometer along the z-axis.
- uint8_t RESERVED: 2
  Rserved bits (should be == 0b00).
- uint8_t buttonZ: 1
  Bit indicating status of z button (pressed == low).
- uint8_t buttonC: 1
  Bit indicating status of c button (pressed == low).
- uint8_t accelXLow: 1
  Bit <1> for the accelerometer along the z-axis.
- uint8_t accelYLow: 1
  Bit <1> for the accelerometer along the y-axis.
- uint8_t accelZLow: 2
  Bits <2:1> for the accelerometer along the x-axis.

5.6.1 Detailed Description

Defines bitfield interaction for status field queries from nunchuck controllers.

Note

The definitions below are for use when connected to a nunchuck in pass-through mode (e.g. when using Wii Motion Plus + nunchuck).
To accomidate the pass-through flag, the least significant bit of all accelerometer values is dropped.

Warning

The following is in little-endian format.

5.6.2 Member Data Documentation

5.6.2.1 accelXHigh

```
uint8_t _WiiNunchuck_StatusPassThrough::accelXHigh
```

Bits <9:2> for the accelerometer along the x-axis.

5.6.2.2 accelXLow

```
uint8_t _WiiNunchuck_StatusPassThrough::accelXLow
```

Bit <1> for the accelerometer along the z-axis.

5.6.2.3 accelYHigh

```
uint8_t _WiiNunchuck_StatusPassThrough::accelYHigh
```

Bits <9:2> for the accelerometer along the y-axis.

5.6.2.4 accelYLow

```
uint8_t _WiiNunchuck_StatusPassThrough::accelYLow
```

Bit <1> for the accelerometer along the y-axis.

5.6.2.5 accelZHigh

```
uint8_t _WiiNunchuck_StatusPassThrough::accelZHigh
```

Bits <9:3> for the accelerometer along the z-axis.

5.6.2.6 accelZLow

```
uint8_t _WiiNunchuck_StatusPassThrough::accelZLow
```

Bits <2:1> for the accelerometer along the x-axis.

5.6.2.7 analogX

```
uint8_t _WiiNunchuck_StatusPassThrough::analogX
```

Bits <7:0> for the analog joystic along the x-axis.

5.6.2.8 analogY

```
uint8_t _WiiNunchuck_StatusPassThrough::analogY
```

Bits <7:0> for the analog joystic along the y-axis.

5.6.2.9 buttonC

```
uint8_t _WiiNunchuck_StatusPassThrough::buttonC
```

Bit indicating status of c button (pressed == low).

5.6.2.10 buttonZ

```
uint8_t _WiiNunchuck_StatusPassThrough::buttonZ
```

Bit indicating status of z button (pressed == low).

5.6.2.11 extensionConnected

```
uint8_t _WiiNunchuck_StatusPassThrough::extensionConnected
```

Bit indicating if extension is connected (1 == active).

5.6.2.12 RESERVED

```
uint8_t _WiiNunchuck_StatusPassThrough::RESERVED
```

Rserved bits (should be == 0b00).

The documentation for this struct was generated from the following file:

- wii_nunchuck.h

6 File Documentation

6.1 Changelog.md File Reference

6.2 README.md File Reference

6.3 wii_classic_controller.c File Reference

Implements functions used to interpret data read from a Wii classic controller device.

```
#include "i2c.h"
#include "wii_classic_controller.h"
```

Include dependency graph for wii_classic_controller.c:

6.4 wii_classic_controller.h File Reference

Defines public constants, macros, and constant functions available for the "wii classic controller" support.

```
#include <stdint.h>
#include "wii_lib.h"
```

Include dependency graph for wii_classic_controller.h: This graph shows which files directly or indirectly include this file:

Classes

- struct _WiiClassic_StatusNormal

  Defines bitfield interaction for status field queries from classic controllers.
- struct _WiiClassic_StatusPassThrough

  Defines bitfield interaction for status field queries from classic controllers.

Typedefs

- typedef struct _WiiClassic_StatusNormal WiiClassic_StatusNormal

  Defines bitfield interaction for status field queries from classic controllers.
- typedef struct _WiiClassic_StatusPassThrough WiiClassic_StatusPassThrough

  Defines bitfield interaction for status field queries from classic controllers.

Functions

- WII_LIB_RC WiiClassic_ProcessStatusParam (WiiLib_Device *device)

  Process current data for device as the response field from querying the device status register.

6.4.1 Detailed Description

Defines public constants, macros, and constant functions available for the "wii classic controller" support.

Note

This file is pulled into "wii_lib.h" automatically.

6.4.2 Typedef Documentation

6.4.2.1 WiiClassic_StatusNormal

```
typedef struct _WiiClassic_StatusNormal WiiClassic_StatusNormal
```

Defines bitfield interaction for status field queries from classic controllers.

Note

The definitions below are for use when directly connected to a classic controller.

Warning

The following is in little-endian format.

6.4.2.2 WiiClassic_StatusPassThrough

```
typedef struct _WiiClassic_StatusPassThrough WiiClassic_StatusPassThrough
```

Defines bitfield interaction for status field queries from classic controllers.

Note

The definitions below are for use when connected to a classic controller in pass-through mode (e.g. when using Wii Motion Plus + classic controller).

Warning

The following is in little-endian format.

6.4.3 Function Documentation

6.4.3.1 WiiClassic_ProcessStatusParam()

```
WII_LIB_RC WiiClassic_ProcessStatusParam (
            WiiLib_Device * device )
```

Process current data for device as the response field from querying the device status register.

Populates the relevant 'device->interfaceCurrent' values by applying the appropriate bitfield mapping and merging values.

Parameters

in	*∗device*	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.5 wii_lib.c File Reference

Implements functions used to abstract away interacting with Wii devices over I2C.

```
#include "i2c.h"
#include "wii_lib.h"
```

Include dependency graph for wii_lib.c:

Functions

- static WII_LIB_TARGET_DEVICE WiiLib_DetermineDeviceType (WiiLib_Device ∗device)

  Handles the process of decrypting data received from a target device.
- static BOOL WiiLib_ValidateDataReceived (uint8_t ∗data, uint32_t len)

  Verifies the data provided is not a known set of invalid byte(s).
- static WII_LIB_RC WiiLib_Decrypt (uint8_t ∗data, int8_t len)

  Handles the process of decrypting data received from a target device.
- static WII_LIB_RC WiiLib_UpdateInterfaceTracking (WiiLib_Device ∗device)

  Wrapper to invoke the appropriate target-specific processing function to interpret the current status data.
- WII_LIB_RC WiiLib_Init (I2C_MODULE module, uint32_t pbClk, WII_LIB_TARGET_DEVICE target, BOOL decryptData, WiiLib_Device ∗device)

  Initializes the Wii target device (e.g. nunchuck).
- WII_LIB_RC WiiLib_ConnectToTarget (WiiLib_Device ∗device)

  Attempts to connect to target device.
- WII_LIB_RC WiiLib_ConfigureDevice (WiiLib_Device ∗device)

  Pushes out configuration to target device.
- WII_LIB_RC WiiLib_QueryParameter (WiiLib_Device ∗device, WII_LIB_PARAM param)

  Hanldes process of initiating and reading the response for querying a parameter value from the target device.
- WII_LIB_RC WiiLib_PollStatus (WiiLib_Device ∗device)

  Refreshes tracking values for the target device's status bits.
- WII_LIB_RC WiiLib_SetNewHomePosition (WiiLib_Device ∗device)

  Refreshes tracking values for the target device's status bits.
- WII_LIB_RC WiiLib_EnableRelativePosition (WiiLib_Device ∗device)

  Simple wrapper to handle enabling of relative positioning.
- WII_LIB_RC WiiLib_DisableRelativePosition (WiiLib_Device ∗device)

  Simple wrapper to handle enabling of relative positioning.

6.5.1 Detailed Description

Implements functions used to abstract away interacting with Wii devices over I2C.

6.5.2 Function Documentation

6.5.2.1 WiiLib_ConfigureDevice()

```
WII_LIB_RC WiiLib_ConfigureDevice (
            WiiLib_Device * device )
```

Pushes out configuration to target device.

Initializes target device in an encrypted or decrypted state based on the configuration flags in the provided device.

Parameters

in	*device	Instance of 'WiiLib_Device{}' defining target device interaction.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.5.2.2 WiiLib_ConnectToTarget()

```
WII_LIB_RC WiiLib_ConnectToTarget (
            WiiLib_Device * device )
```

Attempts to connect to target device.

Pushes out initialization messages to target device. If device ack's messages, attempts to validate the target ID. If successful, device is up and running, but before exiting the function, grabs the initial device status (queries WII_↩ LIB_PARAM_STATUS).

Note

Only attempts to connect once. Repeated connectoin attempts (and any desired delays) should be handled by caller.

Parameters

in	*device	Instance of 'WiiLib_Device{}' to utilize.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.2.3 WiiLib_Decrypt()

```
static WII_LIB_RC WiiLib_Decrypt (
            uint8_t * data,
            int8_t len )  [static]
```

Handles the process of decrypting data received from a target device.

Executes the following to decrypt:

- x = (x [xor] 0x17) + 0x17

Parameters

in	*data	Pointer to data to decrypt.
in	*len*	Number of bytes of data to decrypt.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.5.2.4 WiiLib_DetermineDeviceType()

```
static WII_LIB_TARGET_DEVICE WiiLib_DetermineDeviceType (
            WiiLib_Device * device )  [static]
```

Handles the process of decrypting data received from a target device.

Queries the device for it's identifier by writing 'WII_LIB_PARAM_DEVICE_TYPE' to the target and reading back the 6-byte value. The value is decrypted if necessary before then comparing it against the expected ID values.

Parameters

in	*device	Instance of 'WiiLib_Device{}'.

Returns

Entry from 'WII_LIB_TARGET_DEVICE{}' that represents the target device determined.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.2.5 WiiLib_DisableRelativePosition()

```
WII_LIB_RC WiiLib_DisableRelativePosition (
            WiiLib_Device * device )
```

Simple wrapper to handle enabling of relative positioning.

Note

No values interface tracking values are modified by this function. This function sole aim is to wrap the enable/disable flag for if relative position information is tracked and calculated.

Parameters

in	**device*	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.5.2.6 WiiLib_EnableRelativePosition()

```
WII_LIB_RC WiiLib_EnableRelativePosition (
            WiiLib_Device * device )
```

Simple wrapper to handle enabling of relative positioning.

Note

No values interface tracking values are modified by this function. This function sole aim is to wrap the enable/disable flag for if relative position information is tracked and calculated.

Parameters

in	**device*	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.5.2.7 WiiLib_Init()

```
WII_LIB_RC WiiLib_Init (
            I2C_MODULE module,
```

```
            uint32_t pbClk,
            WII_LIB_TARGET_DEVICE target,
            BOOL decryptData,
            WiiLib_Device * device )
```

Initializes the Wii target device (e.g. nunchuck).

Initializes the I2C bus and pushes initialization messages to target device.

Parameters

in	*module*	Which I2C module (port) to use(e.g. I2C1) when communicating to target device.
in	*pbClk*	Current peripheral bus clock for device (referenced during I2C initialization).
in	*target*	Target type. Should be of type 'WII_LIB_TARGET_DEVICE'.
in	*decryptData*	Boolean flag indicating if data should be initialized as deecrypted.
in	*∗device*	Instance of 'WiiLib_Device{}' to populate/utilize.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function:

6.5.2.8 WiiLib_PollStatus()

```
WII_LIB_RC WiiLib_PollStatus (
            WiiLib_Device * device )
```

Refreshes tracking values for the target device's status bits.

Uses the 'WiiLib_QueryParameter()' to execute a query for 'WII_LIB_PARAM_STATUS' and store the result within the buffer in the 'device->dataCurrent[]' buffer.

Note

This is mainly meant to serve as a simple wrapper to make it easier for app development to not need to know much about the internals of the I2C query process.

Parameters

in	*∗device*	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.2.9 WiiLib_QueryParameter()

```
WII_LIB_RC WiiLib_QueryParameter (
            WiiLib_Device * device,
            WII_LIB_PARAM param )
```

Hanldes process of initiating and reading the response for querying a parameter value from the target device.

Verifies the parameter requested is supported [a known parameter] before utilizing temporary buffers to request and read data over the I2C bus. If the data is read correctly (valid reply, decrypted appropriately, etc.), the results are copied into the 'device->dataCurrent[]' before returning success.

Parameters

in	**device*	Instance of 'WiiLib_Device{}' defining target device interaction.
in	*param*	Parameter value to query. Must match one of the supported values defined in the 'WII_LIB_PARAM{}' enum.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.2.10 WiiLib_SetNewHomePosition()

```
WII_LIB_RC WiiLib_SetNewHomePosition (
            WiiLib_Device * device )
```

Refreshes tracking values for the target device's status bits.

Uses the 'WiiLib_QueryParameter()' to execute a query for 'WII_LIB_PARAM_STATUS' and store the result within the buffer in the 'device->dataBaseline[]' buffer.

Note

This is mainly meant to serve as a simple wrapper to make it easier for app development to not need to know much about the internals of the I2C query process.
This could be handled more efficiently, but presently focused on encapsulation and not too worried about the secondary memcpy() event.

Parameters

in	**device*	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.2.11 WiiLib_UpdateInterfaceTracking()

```
static WII_LIB_RC WiiLib_UpdateInterfaceTracking (
            WiiLib_Device * device )  [static]
```

Wrapper to invoke the appropriate target-specific processing function to interpret the current status data.

Note

Presumes data available in 'device->dataCurrent[]' is a valid payload from querying status data.

Parameters

in	*device	Instance of 'WiiLib_Device{}'.

Returns

Entry from 'WII_LIB_TARGET_DEVICE{}' that represents the target device determined.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.2.12 WiiLib_ValidateDataReceived()

```
static BOOL WiiLib_ValidateDataReceived (
            uint8_t * data,
            uint32_t len )  [static]
```

Verifies the data provided is not a known set of invalid byte(s).

Confirms data is ready was ready to be read from the target device (did not receive all 0xFF bytes) and returns the result. Long term, any error codes can and should be checked by this function.

Parameters

in	*data	Pointer to data to validate.
in	*len*	Number of bytes of data to validate.

Return values

TRUE	Data is valid.
TRUE	Data is not valid.

Here is the caller graph for this function:

6.6 wii_lib.h File Reference

Defines public constants, macros, and constant functions available for the "wii" library module.

```
#include <stdint.h>
#include "i2c.h"
```

```
#include "wii_nunchuck.h"
#include "wii_classic_controller.h"
```

Include dependency graph for wii_lib.h: This graph shows which files directly or indirectly include this file:

Classes

- struct _WiiLib_Interface

  Used to track the state of a Wii controller's buttons, accel, etc.
- struct _WiiLib_Device

  Defines the tracking information used when communicating with Wii targets.

Macros

- #define WII_LIB_MAX_CONNECTION_ATTEMPTS 5

  Maximum number of connectoin attempts to try before presuming device not available. May not exceed 255.
- #define WII_LIB_DEFAULT_CALCULATE_RELATIVE_POSITION TRUE

  Default value for flag controlling whether or not relative position is automatically calculated.
- #define WII_LIB_ID_LENGTH 6

  Length (in bytes) for a ID read from a target device.
- #define WII_LIB_ID_NUNCHUCK { 0x00, 0x00, 0xA4, 0x20, 0x00, 0x00 }

  Identifier read when device is Wii Nunchuck.
- #define WII_LIB_ID_CLASSIC_CONTROLLER { 0x00, 0x00, 0xA4, 0x20, 0x01, 0x01 }

  Identifier read when device is Wii Classic Controller.
- #define WII_LIB_ID_WII_MOTION_PLUS { 0x00, 0x00, 0xA4, 0x20, 0x04, 0x05 }

  Identifier read when device is Wii Motion Plus.
- #define WII_LIB_ID_WII_MOTION_PLUS_PASS_NUNCHUCK { 0x00, 0x00, 0xA4, 0x20, 0x05, 0x05 }

  Identifier read when device is Wii Motion Plus passing through the Wii Nunchuck.
- #define WII_LIB_ID_WII_MOTION_PLUS_PASS_CLASSIC { 0x00, 0x00, 0xA4, 0x20, 0x07, 0x05 }

  Identifier read when device is Wii Motion Plus passing through the Wii Classic Controller.
- #define WII_LIB_MAX_PAYLOAD_SIZE 20

  Largest size (in bytes) of a I2C payload supported by Wii targets.
- #define WII_LIB_PARAM_REQUEST_LEN 1

  Number of bytes to push when starting parameter query.
- #define WII_LIB_PARAM_RESPONSE_LEN_DEFAULT 6

  Number of bytes to read for standard [most] parameter queries.
- #define WII_LIB_PARAM_RESPONSE_LEN_EXTENDED 20

  Number of bytes to read for long parameter queries.
- #define WII_LIB_DELAY_I2C_SETTLE_TIME_MS 10

  Time to delay in milliseconds after initializing the I2C bus before sending any traffic.
- #define WII_LIB_DELAY_AFTER_CONFIRM_ID_MS 10

  Time to delay in milliseconds after confirming the target device ID and before determining the home position.
- #define WII_LIB_DELAY_AFTER_CONNECTION_ATTEMPT_MS 500

  Time to delay in milliseconds after a failed connection attempt (before next attempt in the initialization function).
- #define WII_LIB_DELAY_AFTER_CONFIG_MESSAGE_MS 20

  Time to delay in milliseconds after after sending a configuration message to the target.
- #define WII_LIB_I2C_DELAY_POST_SEND_MS 0

  Delay in milliseconds after trasnmitting a payload across the I2C bus.
- #define WII_LIB_I2C_DELAY_POST_READ_MS 10

  Delay in milliseconds after reading a payload from the I2C bus.
- #define WII_LIB_I2C_DELAY_BETWEEN_TX_RX_MS 1

  Delay in milliseconds between sending a TX request and starting the following RX reqeust to read the reply.

Typedefs

- typedef enum _WII_LIB_RC WII_LIB_RC

  Enum of return code values.
- typedef enum _WII_LIB_TARGET_DEVICE WII_LIB_TARGET_DEVICE

  Defines constants used as abstractions to indicate target device type. Referenced to determine initialization process, register settings, and how to interpret received data.
- typedef enum _WII_LIB_I2C_ADDR WII_LIB_I2C_ADDR

  Defines all known I2C address values for communicating with Wii targets.
- typedef enum _WII_LIB_PARAM WII_LIB_PARAM

  Defines all known paramters (registers) available for library to read and/or write.
- typedef struct _WiiLib_Interface WiiLib_Interface

  Used to track the state of a Wii controller's buttons, accel, etc.
- typedef struct _WiiLib_Device WiiLib_Device

  Defines the tracking information used when communicating with Wii targets.

Enumerations

- enum _WII_LIB_RC {
  WII_LIB_RC_SUCCESS = 0, WII_LIB_RC_UNSUPPORTED_DEVICE = 1, WII_LIB_RC_TARGET_NOT_↩
  INITIALIZED = 2, WII_LIB_RC_I2C_ERROR = 3,
  WII_LIB_RC_TARGET_ID_MISMATCH = 4, WII_LIB_RC_UNKOWN_PARAMETER = 5, WII_LIB_RC_D↩
  ATA_RECEIVED_IS_INVALID = 6, WII_LIB_RC_UNABLE_TO_DECRYPT_DATA_RECEIVED = 7,
  WII_LIB_RC_RELATIVE_POSITION_FEATURE_DISABLED = 8 }

  Enum of return code values.
- enum _WII_LIB_TARGET_DEVICE {
  WII_LIB_TARGET_DEVICE_UNKNOWN = -1, WII_LIB_TARGET_DEVICE_UNSUPPORTED = 0, WII_LI↩
  B_TARGET_DEVICE_NUNCHUCK = 1, WII_LIB_TARGET_DEVICE_CLASSIC_CONTROLLER = 2,
  WII_LIB_TARGET_DEVICE_MOTION_PLUS = 3, WII_LIB_TARGET_DEVICE_MOTION_PLUS_PASS_↩
  NUNCHUCK = 4, WII_LIB_TARGET_DEVICE_MOTION_PLUS_PASS_CLASSIC = 5 }

  Defines constants used as abstractions to indicate target device type. Referenced to determine initialization process, register settings, and how to interpret received data.
- enum _WII_LIB_I2C_ADDR { WII_LIB_I2C_ADDR_STANDARD = 0x52, WII_LIB_I2C_ADDR_WII_MOTI↩
  ON_PLUS = 0x53 }

  Defines all known I2C address values for communicating with Wii targets.
- enum _WII_LIB_PARAM { WII_LIB_PARAM_STATUS = 0x00, WII_LIB_PARAM_RAW_DATA = 0x20, WI↩
  I_LIB_PARAM_DEVICE_TYPE = 0xFA }

  Defines all known paramters (registers) available for library to read and/or write.

Functions

- WII_LIB_RC WiiLib_Init (I2C_MODULE module, uint32_t pbClk, WII_LIB_TARGET_DEVICE target, BOOL decryptData, WiiLib_Device ∗device)

  Initializes the Wii target device (e.g. nunchuck).
- WII_LIB_RC WiiLib_ConnectToTarget (WiiLib_Device ∗device)

  Attempts to connect to target device.
- WII_LIB_RC WiiLib_ConfigureDevice (WiiLib_Device ∗device)

  Pushes out configuration to target device.
- WII_LIB_RC WiiLib_QueryParameter (WiiLib_Device ∗device, WII_LIB_PARAM param)

  Hanldes process of initiating and reading the response for querying a parameter value from the target device.
- WII_LIB_RC WiiLib_SetNewHomePosition (WiiLib_Device ∗device)

  Refreshes tracking values for the target device's status bits.

- WII_LIB_RC WiiLib_PollStatus (WiiLib_Device *device)
  Refreshes tracking values for the target device's status bits.
- WII_LIB_RC WiiLib_EnableRelativePosition (WiiLib_Device *device)
  Simple wrapper to handle enabling of relative positioning.
- WII_LIB_RC WiiLib_DisableRelativePosition (WiiLib_Device *device)
  Simple wrapper to handle enabling of relative positioning.

6.6.1 Detailed Description

Defines public constants, macros, and constant functions available for the "wii" library module.

Note

This is the core header file for the Wii library support (includes all other "wii_...h" header files).

6.6.2 Macro Definition Documentation

6.6.2.1 WII_LIB_DEFAULT_CALCULATE_RELATIVE_POSITION

```
#define WII_LIB_DEFAULT_CALCULATE_RELATIVE_POSITION TRUE
```

Default value for flag controlling whether or not relative position is automatically calculated.

6.6.2.2 WII_LIB_DELAY_AFTER_CONFIG_MESSAGE_MS

```
#define WII_LIB_DELAY_AFTER_CONFIG_MESSAGE_MS 20
```

Time to delay in milliseconds after after sending a configuration message to the target.

6.6.2.3 WII_LIB_DELAY_AFTER_CONFIRM_ID_MS

```
#define WII_LIB_DELAY_AFTER_CONFIRM_ID_MS 10
```

Time to delay in milliseconds after confirming the target device ID and before determining the home position.

6.6.2.4 WII_LIB_DELAY_AFTER_CONNECTION_ATTEMPT_MS

```
#define WII_LIB_DELAY_AFTER_CONNECTION_ATTEMPT_MS 500
```

Time to delay in milliseconds after a failed connection attempt (before next attempt in the initialization function).

6.6.2.5 WII_LIB_DELAY_I2C_SETTLE_TIME_MS

```
#define WII_LIB_DELAY_I2C_SETTLE_TIME_MS 10
```

Time to delay in milliseconds after initializing the I2C bus before sending any traffic.

6.6.2.6 WII_LIB_I2C_DELAY_BETWEEN_TX_RX_MS

```
#define WII_LIB_I2C_DELAY_BETWEEN_TX_RX_MS 1
```

Delay in milliseconds between sending a TX request and starting the following RX reqeust to read the reply.

6.6.2.7 WII_LIB_I2C_DELAY_POST_READ_MS

```
#define WII_LIB_I2C_DELAY_POST_READ_MS 10
```

Delay in milliseconds after reading a payload from the I2C bus.

6.6.2.8 WII_LIB_I2C_DELAY_POST_SEND_MS

```
#define WII_LIB_I2C_DELAY_POST_SEND_MS 0
```

Delay in milliseconds after trasnmitting a payload across the I2C bus.

6.6.2.9 WII_LIB_ID_CLASSIC_CONTROLLER

```
#define WII_LIB_ID_CLASSIC_CONTROLLER { 0x00, 0x00, 0xA4, 0x20, 0x01, 0x01 }
```

Identifier read when device is Wii Classic Controller.

6.6.2.10 WII_LIB_ID_LENGTH

```
#define WII_LIB_ID_LENGTH 6
```

Length (in bytes) for a ID read from a target device.

6.6.2.11 WII_LIB_ID_NUNCHUCK

```
#define WII_LIB_ID_NUNCHUCK { 0x00, 0x00, 0xA4, 0x20, 0x00, 0x00 }
```

Identifier read when device is Wii Nunchuck.

6.6.2.12 WII_LIB_ID_WII_MOTION_PLUS

```
#define WII_LIB_ID_WII_MOTION_PLUS { 0x00, 0x00, 0xA4, 0x20, 0x04, 0x05 }
```

Identifier read when device is Wii Motion Plus.

6.6.2.13 WII_LIB_ID_WII_MOTION_PLUS_PASS_CLASSIC

```
#define WII_LIB_ID_WII_MOTION_PLUS_PASS_CLASSIC { 0x00, 0x00, 0xA4, 0x20, 0x07, 0x05 }
```

Identifier read when device is Wii Motion Plus passing through the Wii Classic Controller.

6.6.2.14 WII_LIB_ID_WII_MOTION_PLUS_PASS_NUNCHUCK

```
#define WII_LIB_ID_WII_MOTION_PLUS_PASS_NUNCHUCK { 0x00, 0x00, 0xA4, 0x20, 0x05, 0x05 }
```

Identifier read when device is Wii Motion Plus passing through the Wii Nunchuck.

6.6.2.15 WII_LIB_MAX_CONNECTION_ATTEMPTS

```
#define WII_LIB_MAX_CONNECTION_ATTEMPTS 5
```

Maximum number of connectoin attempts to try before presuming device not available. May not exceed 255.

6.6.2.16 WII_LIB_MAX_PAYLOAD_SIZE

```
#define WII_LIB_MAX_PAYLOAD_SIZE 20
```

Largest size (in bytes) of a I2C payload supported by Wii targets.

6.6.2.17 WII_LIB_PARAM_REQUEST_LEN

```
#define WII_LIB_PARAM_REQUEST_LEN 1
```

Number of bytes to push when starting parameter query.

6.6.2.18 WII_LIB_PARAM_RESPONSE_LEN_DEFAULT

```
#define WII_LIB_PARAM_RESPONSE_LEN_DEFAULT 6
```

Number of bytes to read for standard [most] parameter queries.

6.6.2.19 WII_LIB_PARAM_RESPONSE_LEN_EXTENDED

```
#define WII_LIB_PARAM_RESPONSE_LEN_EXTENDED 20
```

Number of bytes to read for long parameter queries.

6.6.3 Typedef Documentation

6.6.3.1 WII_LIB_I2C_ADDR

```
typedef enum _WII_LIB_I2C_ADDR WII_LIB_I2C_ADDR
```

Defines all known I2C address values for communicating with Wii targets.

6.6.3.2 WII_LIB_PARAM

```
typedef enum _WII_LIB_PARAM WII_LIB_PARAM
```

Defines all known paramters (registers) available for library to read and/or write.

6.6.3.3 WII_LIB_RC

```
typedef enum _WII_LIB_RC WII_LIB_RC
```

Enum of return code values.

6.6.3.4 WII_LIB_TARGET_DEVICE

```
typedef enum _WII_LIB_TARGET_DEVICE WII_LIB_TARGET_DEVICE
```

Defines constants used as abstractions to indicate target device type. Referenced to determine initialization process, register settings, and how to interpret received data.

6.6.3.5 WiiLib_Device

```
typedef struct _WiiLib_Device WiiLib_Device
```

Defines the tracking information used when communicating with Wii targets.

Note

All data presented has been processed and can be easily indexed by treating it as a structure of bitfields where the corresponding bitfield mapping is defined in the target-specific header file.

6.6.3.6 WiiLib_Interface

```
typedef struct _WiiLib_Interface WiiLib_Interface
```

Used to track the state of a Wii controller's buttons, accel, etc.

Defines every known type of feature across Wii controllers.

Note

Wii nunchuck's use a single Z button and have one joystick, however the classic controller has a left and right version of both. For the purposes of tracking, a non-sided / generic joystick and z button options are not provided.
Using signed integers to make it easier to do a reltive position tracking array.

6.6.4 Enumeration Type Documentation

6.6.4.1 _WII_LIB_I2C_ADDR

```
enum _WII_LIB_I2C_ADDR
```

Defines all known I2C address values for communicating with Wii targets.

Enumerator

WII_LIB_I2C_ADDR_STANDARD	Standard I2C address for Wii extension controllers. The same address is used across most devices.
WII_LIB_I2C_ADDR_WII_MOTION_PLUS	I2C address for Wii Motion Plus.

6.6.4.2 _WII_LIB_PARAM

```
enum _WII_LIB_PARAM
```

Defines all known paramters (registers) available for library to read and/or write.

Enumerator

WII_LIB_PARAM_STATUS	Parameter ID (register address) for querying the status flags from a target device.
WII_LIB_PARAM_RAW_DATA	Parameter ID (register address) for querying raw data from a target dvice.
WII_LIB_PARAM_DEVICE_TYPE	Parameter ID (register address) for querying the device identifier from a target device.

6.6.4.3 _WII_LIB_RC

```
enum _WII_LIB_RC
```

Enum of return code values.

Enumerator

WII_LIB_RC_SUCCESS	Successfully completed task(s).
WII_LIB_RC_UNSUPPORTED_DEVICE	Wii target type presently unsupported.
WII_LIB_RC_TARGET_NOT_INITIALIZED	Target not initialized.
WII_LIB_RC_I2C_ERROR	Failed to communicate with device over I2C.
WII_LIB_RC_TARGET_ID_MISMATCH	Value read from target does not match expected value.
WII_LIB_RC_UNKOWN_PARAMETER	Parameter requested is unknown to this library.
WII_LIB_RC_DATA_RECEIVED_IS_INVALID	Data received from target device but value(s) is(are) invalid.
WII_LIB_RC_UNABLE_TO_DECRYPT_DATA_RE↩CEIVED	Unable to decrypt data received over I2C.
WII_LIB_RC_RELATIVE_POSITION_FEATURE_↩DISABLED	Relative positoin feature disabled presently.

6.6.4.4 _WII_LIB_TARGET_DEVICE

```
enum _WII_LIB_TARGET_DEVICE
```

Defines constants used as abstractions to indicate target device type. Referenced to determine initialization process, register settings, and how to interpret received data.

Enumerator

WII_LIB_TARGET_DEVICE_UNKNOWN	Placeholder for when a target device ID cannot be read.
WII_LIB_TARGET_DEVICE_UNSUPPORTED	Placeholder for unsupported target device type.
WII_LIB_TARGET_DEVICE_NUNCHUCK	Wii target type == Nunchuk.
WII_LIB_TARGET_DEVICE_CLASSIC_CONTRO↩LLER	Wii target type == Classic Controller.
WII_LIB_TARGET_DEVICE_MOTION_PLUS	Wii target type == Wii Motion Plus.
WII_LIB_TARGET_DEVICE_MOTION_PLUS_PA↩SS_NUNCHUCK	Wii target type == Wii Motion Plus that is passing through the Wii Nunchuck.
WII_LIB_TARGET_DEVICE_MOTION_PLUS_PA↩SS_CLASSIC	Wii target type == Wii Motion Plus that is passing through the Wii Classic Controller.

6.6.5 Function Documentation

6.6.5.1 WiiLib_ConfigureDevice()

```
WII_LIB_RC WiiLib_ConfigureDevice (
            WiiLib_Device * device )
```

Pushes out configuration to target device.

Initializes target device in an encrypted or decrypted state based on the configuration flags in the provided device.

Parameters

in	*device	Instance of 'WiiLib_Device{}' defining target device interaction.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.6.5.2 WiiLib_ConnectToTarget()

```
WII_LIB_RC WiiLib_ConnectToTarget (
            WiiLib_Device * device )
```

Attempts to connect to target device.

Pushes out initialization messages to target device. If device ack's messages, attempts to validate the target ID. If successful, device is up and running, but before exiting the function, grabs the initial device status (queries WII_↩ LIB_PARAM_STATUS).

Note

Only attempts to connect once. Repeated connectoin attempts (and any desired delays) should be handled by caller.

Parameters

in	*device	Instance of 'WiiLib_Device{}' to utilize.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.6.5.3 WiiLib_DisableRelativePosition()

```
WII_LIB_RC WiiLib_DisableRelativePosition (
            WiiLib_Device * device )
```

Simple wrapper to handle enabling of relative positioning.

Note

No values interface tracking values are modified by this function. This function sole aim is to wrap the enable/disable flag for if relative position information is tracked and calculated.

Parameters

in	*device	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.6.5.4 WiiLib_EnableRelativePosition()

```
WII_LIB_RC WiiLib_EnableRelativePosition (
            WiiLib_Device * device )
```

Simple wrapper to handle enabling of relative positioning.

Note

No values interface tracking values are modified by this function. This function sole aim is to wrap the enable/disable flag for if relative position information is tracked and calculated.

Parameters

in	*device	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.6.5.5 WiiLib_Init()

```
WII_LIB_RC WiiLib_Init (
            I2C_MODULE module,
            uint32_t pbClk,
            WII_LIB_TARGET_DEVICE target,
            BOOL decryptData,
            WiiLib_Device * device )
```

Initializes the Wii target device (e.g. nunchuck).

Initializes the I2C bus and pushes initialization messages to target device.

Parameters

in	*module*	Which I2C module (port) to use(e.g. I2C1) when communicating to target device.
in	*pbClk*	Current peripheral bus clock for device (referenced during I2C initialization).
in	*target*	Target type. Should be of type 'WII_LIB_TARGET_DEVICE'.
in	*decryptData*	Boolean flag indicating if data should be initialized as deecrypted.
in	*device	Instance of 'WiiLib_Device{}' to populate/utilize.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function:

6.6.5.6 WiiLib_PollStatus()

```
WII_LIB_RC WiiLib_PollStatus (
            WiiLib_Device * device )
```

Refreshes tracking values for the target device's status bits.

Uses the 'WiiLib_QueryParameter()' to execute a query for 'WII_LIB_PARAM_STATUS' and store the result within the buffer in the 'device->dataCurrent[]' buffer.

Note

This is mainly meant to serve as a simple wrapper to make it easier for app development to not need to know much about the internals of the I2C query process.

Parameters

in	**device*	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.6.5.7 WiiLib_QueryParameter()

```
WII_LIB_RC WiiLib_QueryParameter (
            WiiLib_Device * device,
            WII_LIB_PARAM param )
```

Hanldes process of initiating and reading the response for querying a parameter value from the target device.

Verifies the parameter requested is supported [a known parameter] before utilizing temporary buffers to request and read data over the I2C bus. If the data is read correctly (valid reply, decrypted appropriately, etc.), the results are copied into the 'device->dataCurrent[]' before returning success.

Parameters

in	**device*	Instance of 'WiiLib_Device{}' defining target device interaction.
in	*param*	Parameter value to query. Must match one of the supported values defined in the 'WII_LIB_PARAM{}' enum.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.6.5.8 WiiLib_SetNewHomePosition()

```
WII_LIB_RC WiiLib_SetNewHomePosition (
            WiiLib_Device * device )
```

Refreshes tracking values for the target device's status bits.

Uses the 'WiiLib_QueryParameter()' to execute a query for 'WII_LIB_PARAM_STATUS' and store the result within the buffer in the 'device->dataBaseline[]' buffer.

Note

This is mainly meant to serve as a simple wrapper to make it easier for app development to not need to know much about the internals of the I2C query process.
This could be handled more efficiently, but presently focused on encapsulation and not too worried about the secondary memcpy() event.

Parameters

in	*∗device*	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the call graph for this function: Here is the caller graph for this function:

6.7 wii_nunchuck.c File Reference

Implements functions used to interpret data read from a Wii nunchuck device.

```
#include "i2c.h"
#include "wii_nunchuck.h"
```

Include dependency graph for wii_nunchuck.c:

Functions

- WII_LIB_RC WiiNunchuck_ProcessStatusParam (WiiLib_Device ∗device)

  Process current data for device as the response field from querying the device status register.

6.7.1 Detailed Description

Implements functions used to interpret data read from a Wii nunchuck device.

6.7.2 Function Documentation

6.7.2.1 WiiNunchuck_ProcessStatusParam()

```
WII_LIB_RC WiiNunchuck_ProcessStatusParam (
            WiiLib_Device * device )
```

Process current data for device as the response field from querying the device status register.

Populates the relevant 'device->interfaceCurrent' values by applying the appropriate bitfield mapping and merging values.

Note

The nunchuck does not have multiple (left and right) fields. For situations where the nunchuck uses a 'instance' member that is tracked across multipe fields, all values receive the current status values (e.g. ZL and ZR both populated with the current status value).

Parameters

in	*device	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function:

6.8 wii_nunchuck.h File Reference

Defines public constants, macros, and constant functions available for the "wii nunchuck" support.

```
#include <stdint.h>
#include "wii_lib.h"
```

Include dependency graph for wii_nunchuck.h: This graph shows which files directly or indirectly include this file:

Classes

- struct _WiiNunchuck_StatusNormal
  Defines bitfield interaction for status field queries from nunchuck controllers.
- struct _WiiNunchuck_StatusPassThrough
  Defines bitfield interaction for status field queries from nunchuck controllers.

Typedefs

- typedef struct _WiiNunchuck_StatusNormal WiiNunchuck_StatusNormal
  Defines bitfield interaction for status field queries from nunchuck controllers.
- typedef struct _WiiNunchuck_StatusPassThrough WiiNunchuck_StatusPassThrough
  Defines bitfield interaction for status field queries from nunchuck controllers.

Functions

- WII_LIB_RC WiiNunchuck_ProcessStatusParam (WiiLib_Device *device)

  Process current data for device as the response field from querying the device status register.

6.8.1 Detailed Description

Defines public constants, macros, and constant functions available for the "wii nunchuck" support.

Note

This file is pulled into "wii_lib.h" automatically.

6.8.2 Typedef Documentation

6.8.2.1 WiiNunchuck_StatusNormal

```
typedef struct _WiiNunchuck_StatusNormal WiiNunchuck_StatusNormal
```

Defines bitfield interaction for status field queries from nunchuck controllers.

Note

The definitions below are for use when directly connected to a nunchuck.

Warning

The following is in little-endian format.

6.8.2.2 WiiNunchuck_StatusPassThrough

```
typedef struct _WiiNunchuck_StatusPassThrough WiiNunchuck_StatusPassThrough
```

Defines bitfield interaction for status field queries from nunchuck controllers.

Note

The definitions below are for use when connected to a nunchuck in pass-through mode (e.g. when using Wii Motion Plus + nunchuck).
To accomidate the pass-through flag, the least significant bit of all accelerometer values is dropped.

Warning

The following is in little-endian format.

6.8.3 Function Documentation

6.8.3.1 WiiNunchuck_ProcessStatusParam()

```
WII_LIB_RC WiiNunchuck_ProcessStatusParam (
            WiiLib_Device * device )
```

Process current data for device as the response field from querying the device status register.

Populates the relevant 'device->interfaceCurrent' values by applying the appropriate bitfield mapping and merging values.

Note

The nunchuck does not have multiple (left and right) fields. For situations where the nunchuck uses a 'instance' member that is tracked across multipe fields, all values receive the current status values (e.g. ZL and ZR both populated with the current status value).

Parameters

in	*device	Instance of 'WiiLib_Device{}'.

Returns

Return code corresponding to an entry in the 'WII_LIB_RC' enum (zero == success; non-zero == error code). Please see enum definition for details.

Here is the caller graph for this function: