Phi_interfaces

Version 20111230
1/22/2012 1:22:00 AM

Table of Contents

Bug List

File phi_interfaces.h

Not tested on, Arduino IDE 0023 or arduino MEGA hardware!

Class Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Class List

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

File Index

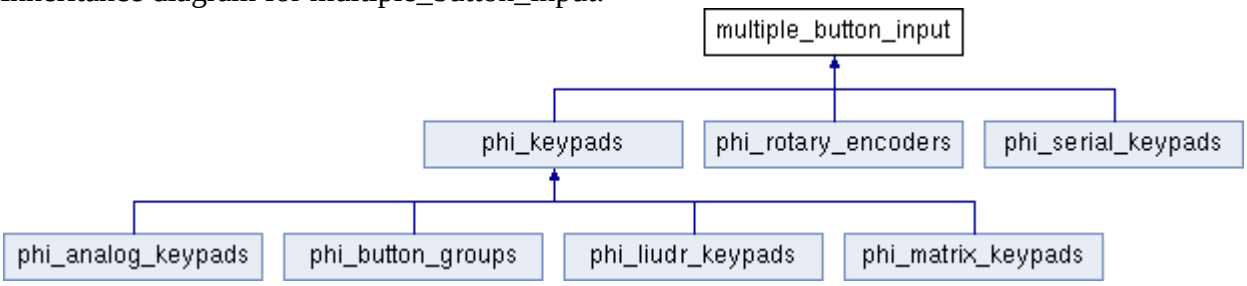
File List

Here is a list of all documented files with brief descriptions:

Class Documentation

multiple_button_input Class Reference

Virtual base class for inputs that contain multiple keys.
`#include <phi_interfaces.h>`
Inheritance diagram for multiple_button_input:



Public Member Functions

- virtual byte getKey ()=0
  This function is responsible for sensing the input for key press and update status.
- virtual byte get_device_type ()
  This returns device type. See device type defs.
- virtual byte get_status ()=0
  This should be run after getKey to get the up-to-date result.
- virtual byte get_sensed ()=0
  This should be run after getKey to get the up-to-date result.
- virtual void set_hold (unsigned int ht)
  This sets how long the button needs to be held before it repeats.
- virtual void set_debounce (unsigned int dt)
  This sets how long the button needs to be held before it is considered pressed.
- virtual void set_dash_threshold (unsigned int dt)
  This sets how long the button needs to be held before it repeats rapidly.
- virtual void set_repeat (unsigned int rt)
  This sets how often the button press repeats after being held.
- virtual void set_dash (unsigned int dt)
  This sets how often the button press rapidly repeats after being held.

Public Attributes

- byte device_type
  This stores the type of the device such as rotary encoder or keypad etc.

Static Protected Attributes

- static unsigned long t_last_action = 0
  This stores the last time any real keypad was active. You may use this to implement sleeping mode.
- static unsigned int buttons_hold_time = buttons_hold_time_def
  Key down time needed to be considered the key is held down.

- static unsigned int buttons_debounce_time = buttons_debounce_time_def
  Key down time needed to be considered the key is not bouncing anymore.
- static unsigned int buttons_dash_threshold = buttons_dash_threshold_def
  Key down time needed to be considered the key is held down long enough to repeat in a dash speed.
- static unsigned int buttons_repeat_time = buttons_repeat_time_def
  Delay between repeating of a held key.
- static unsigned int buttons_dash_time = buttons_dash_time_def
  Delay between dash repeating of a held key.

Detailed Description

Virtual base class for inputs that contain multiple keys.

This is a virtual base class meant to be inherited by child classes. You cannot instantiate any objects from this class. This class is inherited by classes to handle matrix keypads, PS/2 keyboards (planned), rotary encoders, analog buttons, button arrays, or anything that requires the program to not only sense the status of a digital or analog pin status, but also interpret the input and output keys pressed.

Member Function Documentation

virtual byte multiple_button_input::getKey () `[pure virtual]`

This function is responsible for sensing the input for key press and update status.

This function is responsible for sensing the input for key press and update status. Each child class implements this method to translate physical status changes into named buttons.

Implemented in phi_keypads, phi_serial_keypads, and phi_rotary_encoders.

The documentation for this class was generated from the following files:

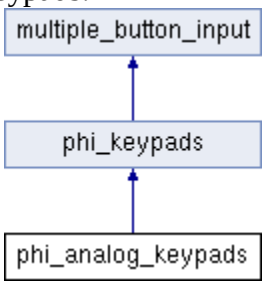
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

phi_analog_keypads Class Reference

a class for buttons connected to analog pin with resistors

`#include <phi_interfaces.h>`

Inheritance diagram for phi_analog_keypads:



Public Member Functions

- phi_analog_keypads (char *na, byte *sp, int *dp, byte r, byte c)
  Constructor for analog keypad.

Protected Member Functions

- byte sense_all ()
  This senses all analog input pins for change of key status.

Protected Attributes

- int * values
  This pointer points to an integer array with values of analog inputs. The number of dividers is equal to the number of buttons on each row. The values should increase monotonically, such as 0,146,342,513,744. A range of 10 between the stored and read values is taken as match to guarantee the match is good. These values apply to all columns so if you want to make a keypad with say three analog pins and 5 buttons on each pin, use the same button/resistor setup on all three pins.

Detailed Description

a class for buttons connected to analog pin with resistors

This class turns analogButton into a keypad. You may connect several buttons to one analog pin with resistors. You may also use multiple analog pins, with each pin connected to several buttons and resistors, to form a keypad. You have to use the same resistor values for all analog pins and thus the same amount of buttons per analog pin. If you need less buttons, just don't connect that many and leave the rest of the circuit with all resistors untouched. Only one function needs to be implemented, the sense_all(). Everything higher level is the same across all keypad subclasses, defined in phi_keypads. Find the sample circuit on my blog under http://liudr.wordpress.com/phi_interfaces/

Constructor & Destructor Documentation

phi_analog_keypads::phi_analog_keypads (char * *na*, byte * *sp*, int * *dp*, byte *r*, byte *c*)

Constructor for analog keypad.

Analog keypads are made up of several analog pins. Each pin is connected to several buttons and resistors. Find diagram in my blog. All analog pins need to be connected to the same resistor network. If you don't need as many buttons, you can omit some buttons but never omit any resistors.

Parameters:

na	This is the name of (or pointer to) a char array that stores the names corresponding to each key press.
sp	This is the name of (or pointer to) a byte array that stores all analog pins used by the keypad. Unlike the original analogbutton, you can use multiple pins, with each pin connected to a number of buttons to form a keypad.
dp	This is the name of (or pointer to) an integer array that stores the analog values of each button press. The array must be sorted from small to big. If you have 5 buttons, this array should have 5 elements.
r	This is the number of analog pins or "rows" of the analog keypad.
c	This is the number of buttons attached to each analog pin or "columns" of the analog keypad. All analog pins should connect to identical button/resistor configurations. If you don't need that many buttons for one particular pin, don't forget to connect all the resistors so that the analog values will be the same.

Member Function Documentation

byte phi_analog_keypads::sense_all () `[protected, virtual]`

This senses all analog input pins for change of key status.

This is the most physical layer of the phi_keypads. Senses all input pins for a valid status. This function is not intended to be call by arduino code but called within the library instead. If all you want is a key press, call getKey.

Returns:

It returns the button scan code (0-max_button-1) that is pressed down or NO_KEYs if no button is pressed down. The return is 0-based so the value is 0-15 if the array has 16 buttons.

Implements phi_keypads.

The documentation for this class was generated from the following files:

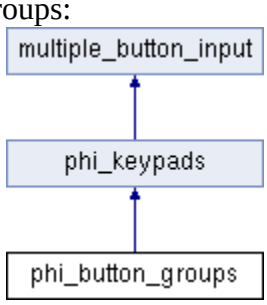
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

phi_button_groups Class Reference

a class for a group of buttons

`#include <phi_interfaces.h>`

Inheritance diagram for phi_button_groups:



Public Member Functions

- phi_button_groups (char *na, byte *sp, byte r)
  Constructor for phi_button_groups.

Protected Member Functions

- byte sense_all ()
  This senses all input pins.

Detailed Description

a class for a group of buttons

Collection of single buttons into a group and handled as a keypad so each button push is translated into a named key value such as '1'. The pointer to pins has no column or row lines. Each pin is connected to one button. This is the way to go if you want to start small with few buttons and intend to expand your interface into more buttons and add rotary encoders and keypads. Using this class instead of phi_buttons class also gives you a virtual layer, where you can develop your project without any buttons or keypads and simulate such input with serial You should only use the phi_buttons class if you are happy with just a few buttons and don't intend to expand your interface into mixtures of keypads, rotary encoders etc.

Constructor & Destructor Documentation

phi_button_groups::phi_button_groups (char * *na*, byte * *sp*, byte *r*)

Constructor for phi_button_groups.

Button group is a class that senses a group of push buttons. This class is preferred compared with phi_buttons class. You can assign names to each button so when the button is pressed, the name is returned such as '1', or 'A'. All buttons should connect arduino pins to GND. Internal pull-up resistors are automatically enabled.

Parameters:

na	This is the name of (or pointer to) a char array that stores the names

	corresponding to each key press.
sp	This is the name of (or pointer to) a byte array that stores all arduino pins used by each button. If you have 4 buttons, the array has 4 elements.
r	This is the number of buttons in this group.

Member Function Documentation

byte phi_button_groups::sense_all () [protected, virtual]

This senses all input pins.

This is the most physical layer of the phi_keypads. Senses all input pins for a valid status. The scanKeypad calls this function and interprets the return into status of the key. This function is not intended to be call by arduino code but called within the library instead. If all you want is a key press, call getKey.

Returns:

It returns the button scan code (0-max_button-1) that is pressed down or NO_KEYs if no button is pressed down. The return is 0-based so the value is 0-15 if the array has 16 buttons.

Implements phi_keypads.

The documentation for this class was generated from the following files:

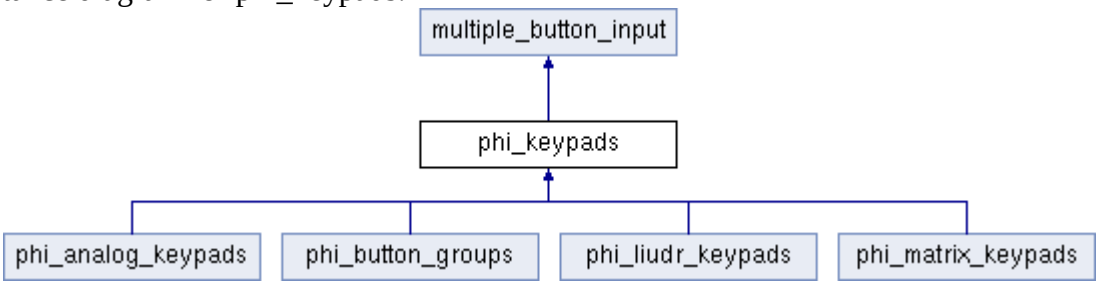
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

phi_keypads Class Reference

virtual class for all keypad subclasses

`#include <phi_interfaces.h>`

Inheritance diagram for phi_keypads:



Public Member Functions

- byte getKey ()
  Returns the key corresponding to the pressed button or NO_KEY.
- virtual byte get_sensed ()
  Get sensed button name. Replace this in children class if needed.
- virtual byte get_status ()
  Get status of the button being sensed. Replace this in children class if needed.

Public Attributes

- byte keyboard_type
  This stores the type of the keypad so a caller can use special functions for specific keypads.
- unsigned long button_status_t
  This is the time stamp of the sensed button first in the status stored in button_status.

Protected Member Functions

- byte scanKeypad ()
- virtual byte sense_all ()=0
  This senses all input pins.

Protected Attributes

- byte rows
  Number of rows on a keypad. Rows are input pins. In analog keypads, each row pin is an analog pin.
- byte columns
  Number of columns on a keypad. Columns are output pins when the column is addressed and tri-stated when the column is not addressed. In analog keypads, column represents number of buttons connected to each analog pin.
- byte buttonBits
  This is the button bits. It's a temporary variable.
- byte button_sensed
  This indicates which button is sensed or 255 if no button is sensed.
- byte button_status
  This indicates the status of the button if button_sensed is not 255.

- byte * mySensorPins
  Pointer to array of pins. Each subclass has a different convention of what pins are used, usually rows are followed by columns.
- char * key_names
  Pointer to array of characters. Each key press is translated into a name from this array such as '0'.

Detailed Description

virtual class for all keypad subclasses

This class provides the hierarchy for actual keypad classes to inherit from. It provides common high-level function codes. These function codes, coupled with the lower level function code of each inheriting child class, completes the translation from sensing physical pins to outputting named buttons with mapping array. The function hierarchy is getKey()<---scanKeypad()<---sense_all(). The sense_all reads digital pins for input. The scanKeypad turns these inputs into status changes for keys and provide scan code of the pressed key. It handles status change including debouncing and repeat. The getKey translates the key press from scan code (0 to max_key-1) into named keys with the mapping array.

Member Function Documentation

byte phi_keypads::getKey () `[virtual]`

Returns the key corresponding to the pressed button or NO_KEY.

This is the public method to get a key press from the keypad. The key press is translated into key_names or NO_KEY. This function is inherited from multiple_button_inputs. All Keypad subclasses such as phi_matrix_keypads and phi_analog_keypads share this code. Since all multiple_button_inputs devices have this method, you can treat all of them as multiple_button_inputs and call this method to get a key press. This is the function you should call to sense key presses. It is only few lines of code and is generic enough for all phi_keypads.

Returns:

It returns the name of the key that is pressed down.

Implements multiple_button_input.

byte phi_keypads::get_sensed () `[virtual]`

Get sensed button name. Replace this in children class if needed.

Outputs the name of the last sensed key or NO_KEY. If all you want is to sense a key press, use getKey instead. You can use this in conjunction with get_status to sense if a key is held.

Returns:

it returns the name of the last sensed key or NO_KEY. This key may not be currently pressed.

Implements multiple_button_input.

byte phi_keypads::get_status () `[virtual]`

Get status of the button being sensed. Replace this in children class if needed.

Return status of the sensed key. If there is no sensed key, this return should not be used.

Returns:

It returns status of the sensed key.

Implements multiple_button_input.

byte phi_keypads::scanKeypad () [protected]

Updates status of the keypad with button_sensed and button_status to provide information to getKey

This routine uses senseAll to scan the keypad, use debouncing to update button_sensed and button_status. This function is not intended to be call by arduino code but called within the library instead. If all you want is a key press, call getKey. The getKey calls this function and translates keypress from scan code (0 to max_key-1) into characters or key names.

Returns:

This function only returns scan code (0 to max_key-1).

The documentation for this class was generated from the following files:

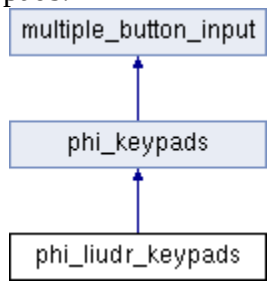
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

phi_liudr_keypads Class Reference

a class for Liudr's shift register LED keypad

`#include <phi_interfaces.h>`

Inheritance diagram for phi_liudr_keypads:



Public Member Functions

- phi_liudr_keypads (char *na, byte *sp, byte cp, byte dp, byte lp, byte r, byte c)
  Constructor for liudr keypad led panel.
- void setLed (byte led, byte on_off)
  Updates LED status using shift registers. Two bytes are shifted out.
- void setLedByte (byte led)
  Updates LED status using shift registers. Two bytes are shifted out.

Protected Member Functions

- byte sense_all ()
  This senses all input pins.
- void updateShiftRegister (byte first8, byte next8)
  This updates shift register with 2 bytes.

Protected Attributes

- byte clockPin
  Clock pin for liudr shift register pad.
- byte dataPin
  Data pin for liudr shift register pad.
- byte latchPin
  Latch or storage pin for liudr shift register pad.
- byte ledStatusBits
  Contains the LED status bits of liudr shift register pad.

Detailed Description

a class for Liudr's shift register LED keypad

This keypad class uses an undisclosed hardware design that incorporates a keypad and LED indicators. The details may be published in a future date and is not the focus of this library.

Constructor & Destructor Documentation

phi_liudr_keypads::phi_liudr_keypads (char * *na*, byte * *sp*, byte *cp*, byte *dp*, byte *lp*, byte *r*, byte *c*)

Constructor for liudr keypad led panel.

Liudr keypad has column pins and row pins. Column pins are on shift register 0 with 8 total column pins. They are pulled LOW when that column is addressed and left in HIGH when those columns are not addressed so you should not press two buttons together. Row pins are always inputs with pull-up resistors and will be read when that row is addressed. By default there are 2 row pins.

Parameters:

na	This is the name of (or pointer to) a char array that stores the names corresponding to each key press.
sp	This is the name of (or pointer to) a byte array that stores all row pins used by the keypad.
cp	This is the arduino pin for shift register clock.
dp	This is the arduino pin for shift register data.
lp	This is the arduino pin for shift register latch.
r	This is the number of rows of the liudr keypad.
c	This is the number of columns of the liudr keypad. By default it is 8.

Member Function Documentation

void phi_liudr_keypads::setLed (byte *led*, byte *on_off*)

Updates LED status using shift registers. Two bytes are shifted out.

You may connect a second shift register and connect up to 8 LEDs to this register. This function can set the status of each of these 8 LEDs.

Parameters:

led	This is the LED number to be set. 0-7.
on_off	This is the status you want to set the LED to, either LOW or HIGH.

void phi_liudr_keypads::setLedByte (byte *led*)

Updates LED status using shift registers. Two bytes are shifted out.

You may connect a second shift register and connect up to 8 LEDs to this register. This function can set the status of each of these 8 LEDs.

Parameters:

led	This is the binary status of all 8 LEDs. If you decide to turn on all LEDs, use 255.

byte phi_liudr_keypads::sense_all () `[protected, virtual]`

This senses all input pins.

This is the most physical layer of the phi_keypads. Senses all input pins for a valid status. The scanKeypad calls this function and interprets the return into status of the key. This function is not

intended to be call by arduino code but called within the library instead. If all you want is a key press, call getKey.

Returns:

It returns the button scan code (0-max_button-1) that is pressed down or NO_KEYs if no button is pressed down. The return is 0-based so the value is 0-15 if the array has 16 buttons.

Implements phi_keypads.

The documentation for this class was generated from the following files:

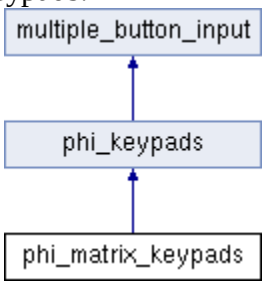
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

phi_matrix_keypads Class Reference

a class for matrix keypads of any size.

`#include <phi_interfaces.h>`

Inheritance diagram for phi_matrix_keypads:



Public Member Functions

- phi_matrix_keypads (char *na, byte *sp, byte r, byte c)
  Constructor for matrix keypad.

Protected Member Functions

- byte sense_all ()
  This senses all input pins.

Detailed Description

a class for matrix keypads of any size.

This is the actual class for matrix keypads, not the phi_keypads, which is a virtual class to support all keypad type of inputs. Only one function needs to be implemented, the sense_all(). Everything higher level is the same across all keypad subclasses, defined in phi_keypads.

Constructor & Destructor Documentation

phi_matrix_keypads::phi_matrix_keypads (char * *na*, byte * *sp*, byte *r*, byte *c*)

Constructor for matrix keypad.

Matrix keypad has column pins and row pins. Column pins are pulled LOW when that column is addressed and left in tristate with pull-up resistor when those columns are not addressed. Row pins are always inputs with pull-up resistors and will be read when that row is addressed.

Parameters:

na	This is the name of (or pointer to) a char array that stores the names corresponding to each key press.
sp	This is the name of (or pointer to) a byte array that stores all pins used by the keypad, with row pins and then column pins.
r	This is the number of rows of the matrix keypad.
c	This is the number of columns of the matrix keypad.

Member Function Documentation

byte phi_matrix_keypads::sense_all () [protected, virtual]

This senses all input pins.

This is the most physical layer of the phi_keypads. Senses all input pins for a valid status. The scanKeypad calls this function and interprets the return into status of the key. This function is not intended to be call by arduino code but called within the library instead. If all you want is a key press, call getKey.

Returns:

It returns the button scan code (0-max_button-1) that is pressed down or NO_KEYs if no button is pressed down. The return is 0-based so the value is 0-15 if the array has 16 buttons.

Implements phi_keypads.

The documentation for this class was generated from the following files:

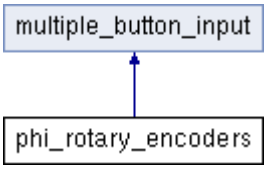
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

phi_rotary_encoders Class Reference

a class for rotary encoders

`#include <phi_interfaces.h>`

Inheritance diagram for phi_rotary_encoders:



Public Member Functions

- phi_rotary_encoders (char *na, byte ChnA, byte ChnB, byte det)
  Constructor for rotary encoder.
- byte getKey ()
  Returns the key corresponding to dial up or down or NO_KEY.
- byte get_status ()
  Always returns buttons_up since the encoder works differently than other keypads.
- byte get_sensed ()
  Always returns NO_KEY since the encoder works differently than other keypads.
- byte get_angle ()
  Get the angle or orientation of the rotary encoder between 0 and detent-1.

Protected Attributes

- byte EncoderChnA
  Arduino pin connected to channel A of the encoder.
- byte EncoderChnB
  Arduino pin connected to channel B of the encoder.
- byte detent
  Number of detents per rotation of the encoder.
- byte stat_seq_ptr
  Current status of the encoder in gray code.
- byte counter
  Counts for get_angle() to calculate knob orientation.
- char * key_names
  Pointer to array of characters two elements long. Each click up or down is translated into a name from this array such as 'U'.

Detailed Description

a class for rotary encoders

This class senses a rotary encoder and reports when the rotary knob is turned one detent up or down. You may use this similarly to a keypad. A call to getKey will yield say 'U' or 'D' for dial up or down. You can also call get_angle to get the orientation of the dial. To use a rotary encoder with a clickable shaft, define a button with phi_buttons class or phi_button_arrays class, with the latter as preferred method. By default

both channels are off when the knob is in a groove. I strongly suggest you purchase an encoder that does that instead of both channels on when the knob is in a groove. This class supports important functions such as getKey(), which you need to call periodically inside a loop to update the status of the encoder and sense a dial up or down when they happen. Then if the return is up or down, you can trigger actions. This library is not interrupt driven and thus has no call-back functions.

Constructor & Destructor Documentation

phi_rotary_encoders::phi_rotary_encoders (char * *na*, byte *ChnA*, byte *ChnB*, byte *det*)

Constructor for rotary encoder.

Constructor for rotary encoder. Provide the names of up and down actions such as 1, and 2, or 'U' and 'D', arduino pins for channels A and B, and number of detent per rotation. Please define the shaft click as a regular phi_buttons or phi_button_arrays object.

Parameters:

na	This is the name of (or pointer to) a char array that stores the names corresponding to the rotary encoder dial up and down.
ChnA	This is the arduino pin connected to the encoder channel A.
ChnB	This is the arduino pin connected to the encoder channel B.
det	This is the number of detent per rotation.

Example:

char mapping[]={'U','D'}; // This is a rotary encoder that returns U for up and D for down rotation on the dial.

phi_rotary_encoders my_encoder(mapping, Encoder1ChnA, Encoder1ChnB, EncoderDetent); // Replace Encoder1ChnA, Encoder1ChnB, EncoderDetent with actual numbers.

Member Function Documentation

byte phi_rotary_encoders::getKey () [virtual]

Returns the key corresponding to dial up or down or NO_KEY.

This actually performs the encoder read and returns up or down dials with the translation done by key_names. If you are not very interested in the inner working of this library, this is the only function you need to call to get a response on the rotary encoder. It assumes the channels are off when the knob is in a groove. To assume the channels are on when the knob is in a groove, read the code on stat_deq. To properly sense the encoder, call this function inside of a loop.

Returns:

It returns the named keys defined by the constructor such as 'U' and 'D' for up and down dial rotations.

Implements multiple_button_input.

byte phi_rotary_encoders::get_status () [virtual]

Always returns buttons_up since the encoder works differently than other keypads.

This always returns buttons_up due to the fact that rotary encoders can't assume other status.

Returns:

This function is defined only to be compatible with the parent class and always returns buttons_up.

Implements multiple_button_input.

byte phi_rotary_encoders::get_sensed () [virtual]

Always returns NO_KEY since the encoder works differently than other keypads.

This always returns NO_KEY due to the nature of rotary encoders.

Returns:

This function is defined only to be compatible with the parent class and always returns NO_KEY.

Implements multiple_button_input.

byte phi_rotary_encoders::get_angle ()

Get the angle or orientation of the rotary encoder between 0 and detent-1.

Get the angle or orientation of the rotary encoder between 0 and detent-1. This function calls getKey to update the angle. If you call getKey BEFORE get_angle, you get the dial up/down from getKey and the correct angle from get_angle. If you call get_angle BEFORM getKey, the dial up/down is read and lost but you get the correct angle. So make your decision. Do you want just dial up/down actions? Then only call getKey. Do you want just angle? Then only call get_angle. Chances of you need them both is very slim but as mentioned you should call getKey first. To properly update the angle, you need to call this function inside of a loop.

Returns:

It returns a value between 0 and detent-1. You can calculate angle with it return.

The documentation for this class was generated from the following files:

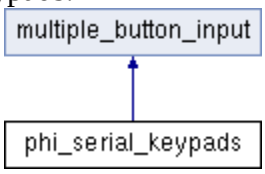
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

phi_serial_keypads Class Reference

a class for phi-panel serial LCD keypads and for serial port to simulate key presses

`#include <phi_interfaces.h>`

Inheritance diagram for phi_serial_keypads:



Public Member Functions

- phi_serial_keypads (Stream *ser, unsigned long bau)
  Constructor for phi-panel serial LCD keypads or serial port input.
- byte getKey ()
- virtual byte get_sensed ()
  Get sensed button name. No serial port read will be done and it always return NO_KEY.
- virtual byte get_status ()
  Get status of the button being sensed. No serial port read will be done and it always return buttons_up.

Protected Attributes

- Stream * ser_port
  Pointer to a Stream object such as hardware serial port.
- unsigned long ser_baud
  Baud rate of the Stream object.

Detailed Description

a class for phi-panel serial LCD keypads and for serial port to simulate key presses

This class provides a way to seamlessly integrate a phi-panel serial LCD keypad or to simulate key presses with serial port under the multiple_button_input virtual base class. The purpose of this class is to integrate a phi-panel serial LCD keypad or simulate key presses with serial port so a project can carry on without considering how to lay out user interfaces such as do you want single buttons or a keypad or with some rotary encoders? Any actual multiple_button_input devices can do can be done over serial. With this class, you can start working on your project's function either with a phi-panel serial LCD keypad or serial port instead of worrying about its interface with a user. Later you can decide what type of user interface and layout you want once you have developed all the project functions, an appropriate time to discuss user interface layout after all. In Arduino IDE 1.0, both software and hardware serials are supported. In Arduino IDE 0022, only hardware serial is supported since the software serial library in this and previous versions don't inherit from Stream. The getKey simply reads from serial port and returns either a character or NO_KEY. The serial port has to be initialized with begin method before it can be passed to this object.

Constructor & Destructor Documentation

phi_serial_keypads::phi_serial_keypads (Stream * *ser*, unsigned long *bau*)

Constructor for phi-panel serial LCD keypads or serial port input.

This is the constructor of phi_serial_keypads class. This class is intended for phi-panel serial LCD keypad or debugging and prototyping. You can create a keypad with this class and use serial keypads or type in the serial monitor as key presses. Later once you decide what actual keypad to use, you can just replace this class with the proper class.

Parameters:

ser	This is the address of your serial object, which you already used begin() on, such as &SERIAL or &NSS.
bau	This is the baud rate in unsigned long integer. At the moment it is not utilized but just reserved for future code.

Member Function Documentation

byte phi_serial_keypads::getKey () `[virtual]`

Returns the key coming from serial port or NO_KEY.

This acquires one character from a serial port as a key press. If the port is empty then it returns NO_KEY. If you are not very interested in the inner working of this library, this is the only function you need to call to get a response on the rotary encoder.

Returns:

It returns the serial port content in byte data type or NO_KEY.

Implements multiple_button_input.

The documentation for this class was generated from the following files:

- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h
- C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.cpp

File Documentation

C:/Users/Liu/Documents/arduino sketchbooks/libraries/phi_interfaces/phi_interfaces.h File Reference

This is the first official release of the phi_interfaces library.

```
#include <WProgram.h>
```

Classes

- class multiple_button_input *Virtual base class for inputs that contain multiple keys.*
- class phi_rotary_encoders *a class for rotary encoders*
- class phi_serial_keypads *a class for phi-panel serial LCD keypads and for serial port to simulate key presses*
- class phi_keypads *virtual class for all keypad subclasses*
- class phi_analog_keypads *a class for buttons connected to analog pin with resistors*
- class phi_matrix_keypads *a class for matrix keypads of any size.*
- class phi_button_groups *a class for a group of buttons*
- class phi_liudr_keypads *a class for Liudr's shift register LED keypad*

Defines

- #define Liudr_shift_register_pad 0
  Liudr shift register pad used on phi-panels.
- #define Single_button 1
  Single buttons need to connect an arduino pin to GND.
- #define Keypad 2
  Generic keypad.
- #define Matrix3X4 3
  3X4 matrix keypad
- #define Matrix4X4 4
  4X4 matrix keypad
- #define Button_group 5
  A group of arduino pins, each connected to a single button. Single buttons need to connect an arduino pin to GND.
- #define Rotary_encoder 6
  Digital rotary encoder with two channels and a common connected to GND.
- #define PS2_keyboard 7
  A PS/2 keyboard. This is not yet supported.
- #define Analog_keypad 8
  A number of buttons connected together with some resistors and one analog input.
- #define Serial_keypad 9
  Phi-panel serial LCD keypad as input.
- #define buttons_up 0
  Non-transitional button status.
- #define buttons_pressed 1
  Transitional button status.
- #define buttons_down 2

Non-transitional button status.

- #define buttons_held 3
  Non-transitional button status.
- #define buttons_released 4
  Transitional button status.
- #define buttons_debounce 5
  One needs to wait till debounce status is over to become pressed status to confirm a press.
- #define buttons_hold_time_def 1000
  Default key down time needed to be considered the key is held down.
- #define buttons_debounce_time_def 50
  Default key down time needed to be considered the key is not bouncing anymore.
- #define buttons_dash_threshold_def 10
  Default key down time needed to be considered the key is held down long enough to repeat in a dash speed.
- #define buttons_repeat_time_def 200
  Default delay between repeating of a held key.
- #define buttons_dash_time_def 50
  Default delay between dash repeating of a held key.
- #define NO_KEYs 255
  This is no key in scan code, internal to the library.
- #define NO_KEY 0
  This is no key that the library outputs to a caller, to be compatible with keypad.h.

Detailed Description

This is the first official release of the phi_interfaces library.

This library unites various button, rotary encoder and keypad input libraries under one library, the phi_interfaces library, for easy of use. This is the first official release. All currently supported input devices are single buttons, array of single buttons, matrix keypads, rotary encoders, analog buttons, and liudr pads. User is encouraged to obtain compatible hardware from liudr or is solely responsible for converting it to work on other shields or configurations.

Author:

Dr. John Liu

Version:

1.0

Date:

01/16/2012

Precondition:

Compatible with Arduino IDE 1.0 and 0022. Please remove your previous phi_buttons library before installing this library.

Bug:

Not tested on, Arduino IDE 0023 or arduino MEGA hardware!

Warning:

PLEASE DO NOT REMOVE THIS COMMENT WHEN REDISTRIBUTING! No warranty!

Contact

Obtain the documentation or find details of the phi_interfaces, phi_prompt TUI library, Phi-2 shield, and Phi-panel hardware or contact Dr. Liu at:

http://liudr.wordpress.com/phi_interfaces/

http://liudr.wordpress.com/phi-panel/

http://liudr.wordpress.com/phi_prompt/

http://liudr.wordpress.com/phi-2-shield/

Index