PS3_USB library

The PS3_USB library adds support for the Sony PS3 Game controller to the Arduino by USB cable on the USB Host Shield from Circuits@Home. It allows a sketch to gather information from the PS3 Game Controller including buttons, joysticks and accelerometers. It also allows output to the LED and rumble features of the Game controller. Additional function is provided to read and write Bluetooth addresses for pairing in Bluetooth mode.

The library requires the USB Host Shield to be installed and uses the libraries developed for the shield, it is not compatible with other USB ports at present. The shield and the libraries do consume memory considerable resources on the Arduino; though they can be supported on most Arduino variants even the ATMega168 based boards with 1K of RAM..

Connections:

The USB Host shield is assembled and links configures depending on the Arduino variant used. The shield supports 3.3V and 5V Arduino in different configurations. The GPIO connections are not required by this library, but another library MaxLCD exists to support an optional LCD display on this interface if required. The USB Host shield uses digital pins D7 through D13 and these pins are not available to the sketch for other purposes. Power for the USB device is derived from the Arduino, so sufficient power must be provide from the USB or external power supply.

The Sony PS3 Game controller is connected by USB cable to the USB A connector on the host shield. The library is tested with the Sony Controller PS3 Game controller and also the Madcatz wireless PS3 game controller which is wireless connected from the game controller to a Madcatz USB dongle. The Madcatz does not support the rumble feature.

Functional overview:

Methods:

void init(void);

a method to perform low level initialisation of the PS3 class, should be called in the setup() function to perform hardware initialisation of the USB Host Shield before calling task().

void task(void);

this method must be called regularly to manage the all events occurring on the USB port. Calls to this routine should be made in the range of once per millisecond to 20 times per second. The report data from the PS3 Game Controller will not be available if this routine is not called regularly.

unsigned char getStatus(void);

this method returns an unsigned character to show the status of the USB and PS3 connection. Bits are used as follows:
bit 0: statusDeviceConnected Set when a USB device is plugged into the USB port on the Host Shield

bit 1: statusUSBConfigured Set when a valid PS3 device is detected and configured
bit 2: statusPS3Connected Set when the PS3 device is connected and sending reports based on changes on the controller (buttons pressed, joystick moved,..)
bits 3: statusReportReceived Set when a report is received from the PS3 controller that a change has occurred on one of the parameters. This bit is cleared by the stat_report_received() method;

bool statConnected(void);

This method returns the state of the statusPS3Connected bit for the attached device.

bool statReportReceived(void);

This method returns the state of the statusReportReceived bit for the attached device. Calling this method also clears the bit if set

bool buttonChanged(void);

This method returns true if a Game Controller button has been pressed or released, since the last call to this method.

bool buttonPressed(unsigned char button);

This method returns the state of the Game Controller button specified in the call. True if button pressed. Buttons are:

```
#define buSelect    0
#define buLAnalog   1
#define buRAnalog   2
#define buStart     3
#define buUp        4
#define buRight     5
#define buDown      6
#define buLeft      7
#define buL2        8
#define buR2        9
#define buL1        10
#define buR1        11
#define buTriangle  12
#define buCircle    13
#define buCross     14
#define buSquare    15
#define buPS        16
```

void LEDRumble(unsigned char ledrum);

This method sets the state of the LEDs and rumble motors on the Game Controller. Ledrum values are:

```
#define psLED1 0x01
#define psLED2 0x02
#define psLED3 0x04
#define psLED4 0x08
```

```
#define psRumbleHigh 0x10
#define psRumbleLow 0x20
```

unsigned int getMotion(unsigned char accel);

This method returns the value read from the accelerometer or Gyro in the Game Controller. Values are in range 0 to 1023 with 512 as centre. Accel values are:

```
#define AccelerometerX 0
#define AccelerometerY 1
#define AccelerometerZ 2
#define GyrometerZ 3
```

unsigned char getJoystick(unsigned char joystick);

This method returns the value read from the joystick on the Game Controller. Values are in range 0 to 255. Joystick values are:

```
#define leftJoystickX 0
#define leftJoystickY 1
#define rightJoystickX 2
#define rightJoystickY 3
```

unsigned char getPressure(unsigned char button);

This method returns the value read for the button pressure on the Game Controller. Values are in range 0 to 255. Button values are:

```
#define buUp        4
#define buRight     5
#define buDown      6
#define buLeft      7
#define buL2        8
#define buR2        9
#define buL1        10
#define buR1        11
#define buTriangle  12
#define buCircle    13
#define buCross     14
#define buSquare    15
```

void setBDADDR(unsigned char * bdaddr);

This method sets the master Bluetooth address to the PS3 Game Controller. The master Bluetooth address is the address where the Game controller will attempt to make a Bluetooth connection when pairing. Bdaddr[6] is a six unsigned char array containing the Bluetooth address with least significant byte first.

void getBDADDR(unsigned char * bdaddr);

This method gets the master Bluetooth address in the PS3 Game Controller. The master Bluetooth address is the address where the Game controller will attempt to make a Bluetooth connection when pairing. Bdaddr[6] is a six unsigned char array containing the Bluetooth address with least significant byte first.

Memory usage:

The library is fairly complex and takes about 5000 bytes of program memory depending on the methods actually used.
Due to the complexity and data buffers the data memory usage is about 350 bytes. Since no warning is generated at compile time of data memory usage, care should be taken to monitor and manage consumption. MemoryFree() can be used to check data memory usage.
While this fits onto the AtMega168, it may be better to use a larger device if additional code/libraries are complex or if a large number of serial.print() functions are used.

Example sketch:

The test sketch included in the download demonstrates many of the capabilities of the library.

```
#include <Servo.h>
#include <Spi.h>
#include <ps3_usb.h>
#include <MemoryFree.h>
void setup(void);
void loop(void);

Servo servoOne;  // create servo object for first Servo
Servo servoTwo;  // create servo object for second Servo
PS3_USB  PS3Game;   // create an object for the PS3 Game Controller

int PositionOne, PositionTwo; //storage for servo positions
char servomode; // mode for servo control 0 = joystick 1 = accelerometer

void setup()
{
  servoOne.attach(2);  // attaches the first servo on pin 0 to the servo object
  servoTwo.attach(3);  // attaches the first servo on pin 1 to the servo object
  PS3Game.init();
  Serial.begin( 9600 );
  Serial.println("PS3 Controller");
  Serial.print("freeMemory() reports ");
  Serial.println( freeMemory() );
}

void loop()
{
  PS3Game.task();                  // perform the regular USB routines

  if ((PS3Game.statConnected()) && (PS3Game.statReportReceived())){ // report received ?

    if(PS3Game.buttonChanged()){   // right and left buttons change mode joystick/Accelerometer

      if(PS3Game.buttonPressed(buLeft)) {

        servomode = 0;
        PS3Game.LEDRumble(psLED1);
      }
      if(PS3Game.buttonPressed(buRight)) {

        servomode = 1;
        PS3Game.LEDRumble(psLED2);
      }
    }

    if(servomode){ // Accelerometer mode

      PositionOne = constrain(PS3Game.getMotion(AccelerometerX), 400, 600);    // constrain to +/- 1g
      PositionOne = map(PositionOne, 400, 600, 0, 179);    // scale it to use it with the servo
      PositionTwo = constrain(PS3Game.getMotion(AccelerometerY), 400, 600);    // constrain to +/- 1g
      PositionTwo = map(PositionTwo, 400, 600, 0, 179);    // scale it to use it with the servo
    }
    else{  // Joystick mode
      PositionOne = map(PS3Game.getJoystick(leftJoystickX), 0, 255, 0, 179);    // scale it to use it with the servo
      PositionTwo = map(PS3Game.getJoystick(leftJoystickY), 0, 255, 0, 179);    // scale it to use it with the servo
    }
  }



  servoOne.write(PositionOne);   //sets the first servo position according to the scaled value
  servoTwo.write(PositionTwo);   // sets the first servo position according to the scaled value
  delay(15);                     // waits for the servo low time


}
```

[Get Code] **Change URL!!**

On using and modifying libraries

- http://www.arduino.cc/en/Main/Libraries
- http://www.arduino.cc/en/Reference/Libraries

Coming soon:

This library is the first of a small collection for the support of PS3 and Wiimote Game controllers on the Arduino. Subsequent libraries support the PS3 controller and the Wiimote controller over Bluetooth wireless. The libraries may also be modified to support other Arduino hardware with USB Host capability as it becomes available.

The background and development of the libraries is described in a series of articles.

ADD URL

Questions, comments and suggestions on the library and documentation

Further documentation and background on this library and the support of PS3 and Wiimote game controllers on the Arduino is available here:

ADD URL

Comments and suggestions should be sent here:

ADD URL