Wiring Pi

GPIO Interface library for the Raspberry Pi

Serial Library

WiringPi includes a simplified serial port handling library. It can use the on-board serial port, or any USB serial device with no special distinctions between them. You just specify the device name in the initial open function.

To use, you need to make sure your program includes the following file:

```
#include <wiringSerial.h>
```

Then the following functions are available:

- **int serialOpen (char *device, int baud) ;**

This opens and initialises the serial device and sets the baud rate. It sets the port into “raw” mode (character at a time and no translations), and sets the read timeout to 10 seconds. The return value is the file descriptor or -1 for any error, in which case *errno* will be set as appropriate.

- **void serialClose (int fd) ;**

Closes the device identified by the file descriptor given.

- **void serialPutchar (int fd, unsigned char c) ;**

Sends the single byte to the serial device identified by the given file descriptor.

- **void serialPuts (int fd, char *s) ;**

Sends the nul-terminated string to the serial device identified by the given file descriptor.

- **void serialPrintf (int fd, char *message, …) ;**

Emulates the system printf function to the serial device.

- **int serialDataAvail (int fd) ;**

Returns the number of characters available for reading, or -1 for any error condition, in which case *errno* will be set appropriately.

- **int serialGetchar (int fd) ;**

Returns the next character available on the serial device. This call will block for up to 10 seconds if no data is available (when it will return -1)

- **void serialFlush (int fd) ;**

This discards all data received, or waiting to be send down the given device.

Note: The file descriptor (***fd***) returned is a standard Linux file descriptor. You can use the standard read(), write(), etc. system calls on this file descriptor as required. E.g. you may wish to write a larger block of binary data where the serialPutchar() or serialPuts() function may not be the most appropriate function to use, in which case, you can use write() to send the data.

Advanced Serial Port Control

The ***wiringSerial*** library is intended to provide simplified control – suitable for most applications, however if you need advanced control – e.g. parity control, modem control lines (via a USB adapter, there are none on the Pi's on-board UART!) and so on, then you need to do some of this the "old fashioned" way.

For example – To set the serial line into 7 bit mode plus even parity, you need to do this…

In your program:

```
#include <termios.h>
```

and in a function:

```
  struct termios options ;

  tcgetattr (fd, &options) ;   // Read current options
  options.c_cflag &= ~CSIZE ;  // Mask out size
  options.c_cflag |= CS7 ;     // Or in 7-bits
  options.c_cflag |= PARENB ;  // Enable Parity - even by default
  tcsetattr (fd, &options) ;   // Set new options
```

The 'fd' variable above is the file descriptor that serialOpen() returns.

Please see the man page for tcgetattr for all the options that you can set.

```
man tcgetattr
```