

Copyright (C) BenaTong 1991 All rights reserved

## **Serial**

INHERITS FROM

Object

DECLARED IN

Serial.h

WRITTEN BY

Charles G. Bennett

Version 2.0, Aug 4, 1992

### **CLASS DESCRIPTION**

The Serial class is a subclass of Object and is used to implement serial communication in an object oriented manner.

## FEATURES

- Easy to use, object oriented interface to serial communications .

## INSTANCE VARIABLES

*Inherited from Object*

Class      isa;

*Declared in Serial*

int ttyptr;

```
int baudrate;  
int par;  
id serialTarget;  
SEL theSelector;  
BOOL notifyStatus;  
char deviceName[20];  
unsigned timeout;  
unsigned int notifyThreshold;
```

ttyptr  
baudrate  
par  
targetobject ID  
theSelector

The serial port file handle.  
Serial port baudrate  
Serial port parity  
Used by SetNotify  
method sent to serialTarget

BOOL notifyStatus	Enables or Disables SetNotify
deviceName	The string used to open the serial port.
timeout	Sets the timeout for incoming
characters.	
notifyThreshold	Sets the threshold to trigger setNotify

## METHOD TYPES

### Opening and initializing a serial port

- init;
- (int) openCBREAK: (char \*) portStr Baudrate:(int) baud  
Parity : (int) parity;
- (int) openRAW: (char \*) portStr Baudrate:(int) baud;
- (BOOL) portIsOpen

- (unsigned) setCharacterTimeout:(unsigned) time;
- (unsigned) timeout;

#### Testing for input

- (int) waitForInput;
- (int) setNotify:(SEL)aSelector with:anObject;
- setNotifyStatus:(BOOL)status;
- setNotifyThreshold:(unsigned int) threshold;
- (int) getNotifyThreshold;
- (BOOL) getNotifyStatus;
- (int) inputQueueSize;

#### Checking the output

- (int) outputQueueSize;

#### Reading the serial port

- (int) expectPattern:(int ) seconds: (char\*) pattern;
- (int) expectStr:(int ) seconds , ... ;
- (int) readString: (char \*) str forMaxOf:(int) max;
- (int) readString: (char \*) str forMaxOf:(int) max term:(char) termChar;
- (int) readBlock: (char\*) buf for:(int) num;
- (int) readBlock: (char\*) buf for:(int) num term:(char) termChar;

#### Writing to the serial port

- (int) serialPrintf: (const char\* ) format, ...;
- (int) writeString:(char\*)str;
- (int) writeBlock:(char\*) buf for:(int) num;

#### Controlling the serial port

- (int) baudrate;
- (int) flushOutput;
- (int) flushInput;

- (int) flushAll;
- (int) setBaudRate:(int) baud;
- (int) setParity:(int) parity;
- (int) parity;
- (int) ttyptr;
- (int) dropDTR;
- (int) raiseDTR;
- (int) break;
- (int) dropBREAK;
- (int) raiseBREAK;
- (int) dcd;
- (int) cts;
- (int) dtr;
- (int) useXONXOFF:(BOOL) status;
- (int) modifyCRLF:(BOOL) status;
- serialTarget;

- (SEL) selector;

Closing the serial port

- (int) closePort;

## INTERNAL METHODS

```
void myTtyHandler(int fd, void *userData);  
unsigned char addParity(register unsigned char ch,int parity);
```

## CLASS METHODS



None

## INSTANCE METHODS

### **baudrate**

- (int) **baudrate**;

Returns the baudrate setting of the serial port..

Returns `PORTNOTOPEN` if there was no port open.

### **break**

- (int) **break;**

This is a convenience method. It invokes raiseBreak sleeps 1 second and calls dropBreak;

Returns PORTNOTOPEN if there was no port open.

## **closePort**

- (int) **closePort**

Closes the Serial Port.

Returns PORTNOTOPEN if there was no port open  
SERIALOK otherwise.

## **cts**

- (int) **cts;**

Returns ON if the Clear To Send line is active (ok to send) , OFF otherwise.  
The line is controlled by the device connected to the serial port. Flow control is managed by the hardware when you open a port with a "f" in it's name IE /dev/cu~~fa~~.

Returns PORTNOTOPEN if there was no port open.

### **dcd**

- (int) **dcd**;

Returns ON if the Data Carrier Detect line is active, OFF otherwise.  
The line goes active when the modem detects carrier OR the line is floating.  
Returns PORTNOTOPEN if there was no port open.

### **deviceName**

- (char \*) **deviceName**;

Returns the string used to open this port.

### **dropBREAK**

- (int) **dropBREAK;**

Clears a BREAK signal.

Returns `PORTNOTOPEN` if there was no port open `SERIALOK` otherwise.

### **dropDTR**

- (int) **dropDTR;**

Sets the DTR line on the serial port to `FALSE`. This will cause most

modems to disconnect or reset.

Returns `PORTNOTOPEN` if there was no port open `SERIALOK` otherwise.

### **dtr**

- (int) **dtr**;

Returns ON if theData Terminal Ready line is active, OFF otherwise.

You control DTR by using the `dropDTR` or `raiseDTR` methods. The method simply lets you check the status.

Returns `PORTNOTOPEN` if there was no port open.

### **expectPattern::**

- (int) **expectPattern:(int) seconds : (char\*) pattern**;

This method looks for a pattern in the input stream. If **seconds** elapse without a match then the method returns `TIMEOUT`.

Returns `PORTNOTOPEN` if there was no port open , `SERIALOK` if a match was found or `TIMEOUT` if **seconds** have elapsed without a match. This method "slides" a **pattern** size window along the input stream looking for the match, the input stream is consumed so all characters up to and including the **pattern** string are lost.

#### **expectStr:**

- (int) **expectStr:(int) seconds, ...;**

This method accepts a variable number of strings and reads one LINE of input at a time and then matches the read line against the pattern string(s). It expects the line to be an exact match. If **seconds** elapse without a match then the method returns `TIMEOUT`. If a match is found the methods returns the index of the matched string. This methods consumes the input stream, so all lines up to and including the matched line are lost.

Returns `PORTNOTOPEN` if there was no port open index of the matched

string otherwise.

### **flushAll**

- (int) **flushAll**;

Empties both the input queue and the output queue.  
Returns PORTNOTOPEN if there was no port open  
SERIALOK otherwise.

### **flushInput**

- (int) **flushInput**;

Empties the input queue.  
Returns PORTNOTOPEN if there was no port open  
SERIALOK otherwise.

## **flushOutput**

- (int) **flushOutput**;

Empties the output queue.

Returns PORTNOTOPEN if there was no port open

SERIALOK otherwise.

## **getNotifyStatus**

- (BOOL) **getNotifyStatus**;

Returns whether or not the setNotify: with: is enabled.



## **getNotifyThreshold**

- (int) **getNotifyThreshold**;

Returns whether the number of characters that must be in the input queue before the setNotify method is invoked. The default value is 0, so even one character will trigger it.

## **init**

- **init**;

Set up for internal variables and signal handlers.

## **inputQueueSize**

- (int) **inputQueueSize**;

Returns the number of characters available in the input Queue.  
Returns PORTNOTOPEN if there was no port open.

### **modifyCRLF**

- (int) **modifyCRLF**:(BOOL) status;

If the port was opened in CBREAK mode you can cause incoming carriage returns to be converted to New Line Characters. Outgoing new line characters will be converted to CR LF pairs.

Returns PORTNOTOPEN if there was no port open SERIALOK otherwise.

### **openCBREAK: Baudrate: Parity:**

- (int) **openCBREAK**: (char\*) portSt r **Baudrate**:(int) baud **Parity**:(int)

parity;

Opens the serial port named by portStr IE. "/dev/ttydfa"  
Returns PORTNOTOPEN if the port could not be opened.  
BADPARITY if an illegal parity was requested SERIALOK otherwise.

### **openRAW: Baudrate:**

- (int) **openRAW:** (char\*) portStr **Baudrate:**(int) baud ;

Opens the serial port named by portStr IE. "/dev/ttydfa". The port is  
ALWAYS opened N,8,1. Returns PORTNOTOPEN if the port could not  
be opened.  
SERIALOK otherwise.

### **outputQueueSize**

- (int) **outputQueueSize**;

Returns the number of characters waiting to be sent.  
Returns PORTNOTOPEN if there was no port open.

### **parity**

- (int) **parity**;

Returns the parity setting of the serial port.  
Returns PORTNOTOPEN if there was no port open.

### **portIsOpen**

- (BOOL) **portIsOpen**;

Returns YES if the port is open NO if not.

### **raiseBREAK**

- (int) **raiseBREAK**;

Sends a BREAK signal. Usually used to get another computer systems

Returns PORTNOTOPEN if there was no port open SERIALOK otherwise.

### **raiseDTR**

- (int) **raiseDTR**;

Sets the DTR line on the serial port to TRUE.

Returns PORTNOTOPEN if there was no port open SERIALOK otherwise.

**readBlock: for:**

- (int) **readBlock:** (char\*) buf **for:** (int) num;

Reads a block of characters from the serial Port. num characters are received. Except for parity, there is no special treatment of any characters. If the input queue has less than num characters then the number available is read and the number of characters actually read is returned. Parity is stripped unless NONE was specified.

Returns PORTNOTOPEN if there was no port open.

Returns TIMEOUT if no characters are available.

Returns num otherwise.

### **readBlock: for: term:**

- (int) **readBlock:** (char\*) buf **for:** (int) num **term:** (char) termChar;

Reads a block of characters from the serial Port. num characters are received or termChar. Except for parity, is no special treatment of any characters. If the input queue has less than num characters then the number available is read (or until termChar is received) and the number of characters actually read is returned. Parity is stripped unless NONE was specified.

Returns PORTNOTOPEN if there was no port open  
Returns TIMEOUT if no characters are available.  
num otherwise.

### **readString: forMaxOf:**

- (int) **readStr:** (char\*) str **forMaxOf:**(int) max ;

Reads a string from the serial port.

Returns PORTNOTOPEN if there was no port open.

Returns TIMEOUT if none arrive.

Returns Number of Characters Received otherwise.

Note: strips \r (carriage returns) terminates on \n (new lines) or on max characters.

Will NULL terminate string in case of max characters. Parity is stripped unless NONE was specified.

Returns PORTNOTOPEN if there was no port open.

Returns TIMEOUT if none arrive.



Returns Number of Characters Received otherwise.

**readString: forMaxOf: term:**

- (int) **readStr:** (char\*) str **forMaxOf:** (int) max **term:** (char) termChar;

Reads a string from the serial port.

Note: strips \r (carriage returns) terminates on \n (new lines), on max characters or when termChar is received. Parity is stripped unless NONE was specified.

Will NULL terminate string in case of max characters.

Returns PORTNOTOPEN if there was no port open.

Returns TIMEOUT if none arrive.

Returns Number of Characters Received otherwise

**selector**

- (SEL) **selector**;

Returns the method selector of the Object that will receive SetNotify messages.

**serialPrintf:**

- (int) **serialPrintf:** (const char\* ) format, ...;

This is a convinence method to make it easy to "print" to the serial port. You use it exactaly the same way as a normal printf.

Returns PORTNOTOPEN if there was no port open.

SERIALOK otherwise.

**setBaudRate:**

- (int) **setBaudRate:** (int) baud;

Set the serial port baudrate. Serial.h includes sgTTY.h so standard constants I.E. B9600 are also available. Look in sgTTY.h for a complete list or just use the number you want.

Returns PORTNOTOPEN if there was no port open.

BADBAUD if an illegal baudrate is selected.

SERIALOK otherwise.

**setNotifyStatus:**

- **setNotifyStatus:** (BOOL) status;

Enables or Disables the method SetNotify: with:. You would use this routine to keep from being swamped with input notifications.

**setNotifyThreshold:**

- **setNotifyThreshold:** (unsigned int) threshold;

Controls the number of characters that must be in the input queue BEFORE the object passed to the setNotify:with: method is invoked. You would use this routine to keep from being swamped with input notifications.

**setNotify: with:**

- (int) **setNotify:** (SEL) aSelector **with:** anObject;

Messages anObject with the message aSelector when the serial port has characters available for reading.

**setParity:**

- (int) **setParity:** (int) parity;

Set the serial port parity. Constants for EVEN, ODD, MARK, SPACE and NONE are provided.

Returns PORTNOTOPEN if there was no port open  
SERIALOK otherwise.

**setCharacterTimeout:**

- (unsigned) **setCharacterTimeout:** (unsigned) time;

Set the serial port timeout variable ( In microseconds). The default is 1 Second (timeout value = 1,000,000 ). This timeout is used to specify how long the serial object will wait for input. On a block read this value is used

to wait for the required number of characters to be in the input queue before reading them.

Returns the old timeout value.

### **serialTarget**

- **serialTarget;**

Returns the ID of the Object that will receive SetNotify messages.

### **timeout**

- (unsigned) **timeout;**

Returns the current timeout setting for the serial object. The returned value is in microseconds. ( 1,000,000 = one second).

## **ttyptr**

- (int) **ttyptr**;

Returns the port handle for the serial object. The returned value is -1 if there is no port.

## **useXONXOFF**

- (int) **useXONXOFF**:(BOOL) status;

If the port was opened in CBREAK mode you can enable or disable the processing on XON and XOFF characters (ctrl-S and ctrl-Q). The default is to NOT enable this feature.

Returns PORTNOTOPEN if there was no port open SERIALOK otherwise.

### **waitForInput**

- (int) **WaitForInput**;

Waits for up to timeout microseconds for the input buffer to have characters available for reading. Returns SERIALOK if there are characters available or TIMEOUT.

Returns PORTNOTOPEN if there was no port open.

### **writeBlock:**

- (int) **writeBlock:** (char\*) buf **for:** (int) num;



Sends a block of characters to the serial Port. num characters are sent and there is no special treatment of any characters.

Returns PORTNOTOPEN if there was no port open  
SERIALOK otherwise.

### **writeString:**

- (int) **writeStr:** (char\*) str;

Sends a NULL terminated string to the serial Port.  
Returns PORTNOTOPEN if there was no port open  
SERIALOK otherwise.

## CONSTANTS AND DEFINED TYPES

OFF	0
ON	1
EVEN	1
ODD	2
NONE	3
SPACE	4
MARK	5
SEC	1000000
SERIALOK	0
PORTNOTOPEN	-1

TIMEOUT	-2
BADPARITY	-3
BADBAUD	-4