libctb Reference Manual

0.15

Generated by Doxygen 1.5.1

Mon Jan 4 14:49:10 2010

Contents

1 ctb overview **1**

2 libctb Class Documentation **1**

3 libctb File Documentation **33**

1 ctb overview

2 libctb Class Documentation

2.1 ctb::Fifo Class Reference

```
#include <fifo.h>
```

Public Member Functions

- Fifo (size_t size)

  the constructor initialize a fifo with the given size.

- virtual ~Fifo ()

  the destructor destroys all internal memory.

- virtual void clear ()

  clear all internal memory and set the read and write pointers to the start of the internal memory.

  Note:

  > *This function is not thread safe! Don't use it, if another thread takes access to the fifo instance. Use a looping get() or read() call instead of this.*

- virtual int get (char ∗ch)

  fetch the next available byte from the fifo.

- size_t items ()

  query the fifo for it's available bytes.

- virtual int put (char ch)

  put a character into the fifo.

- virtual int read (char ∗data, int count)

  read a given count of bytes out of the fifo.

- virtual int write (char ∗data, int count)

  write a given count of bytes into the fifo.

Protected Attributes

- size_t m_size
- char ∗ m_begin
- char ∗ m_end
- char ∗ m_rdptr
- char ∗ m_wrptr

2.1.1 Detailed Description

A simple thread safe fifo to realize a put back mechanism for the wxIOBase and it's derivated classes.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 ctb::Fifo::Fifo (size_t *size*)

the constructor initialize a fifo with the given size.

Parameters:

size size of the fifo

2.1.2.2 ctb::Fifo::∼Fifo () `[virtual]`

the destructor destroys all internal memory.

2.1.3 Member Function Documentation

2.1.3.1 void ctb::Fifo::clear () `[virtual]`

clear all internal memory and set the read and write pointers to the start of the internal memory.

Note:

This function is not thread safe! Don't use it, if another thread takes access to the fifo instance. Use a looping get() or read() call instead of this.

2.1.3.2 int ctb::Fifo::get (char ∗ *ch*) `[virtual]`

fetch the next available byte from the fifo.

Parameters:

ch points to a charater to store the result

Returns:

1 if successful, 0 otherwise

2.1.3.3 size_t ctb::Fifo::items ()

query the fifo for it's available bytes.

Returns:

count of readable bytes, storing in the fifo

2.1.3.4 int ctb::Fifo::put (char *ch*) `[virtual]`

put a character into the fifo.

Parameters:

ch the character to put in

Returns:

1 if successful, 0 otherwise

2.1.3.5 int ctb::Fifo::read (char ∗ *data*, int *count*) `[virtual]`

read a given count of bytes out of the fifo.

Parameters:

data memory to store the readed data

count number of bytes to read

Returns:

On success, the number of bytes read are returned, 0 otherwise

2.1.3.6 int ctb::Fifo::write (char ∗ *data*, int *count*) `[virtual]`

write a given count of bytes into the fifo.

Parameters:

data start of the data to write

count number of bytes to write

Returns:

On success, the number of bytes written are returned, 0 otherwise

2.1.4 Member Data Documentation

2.1.4.1 size_t ctb::Fifo::m_size `[protected]`

the size of the fifo

2.1.4.2 char∗ ctb::Fifo::m_begin `[protected]`

the start of the internal fifo buffer

2.1.4.3 char∗ ctb::Fifo::m_end `[protected]`

the end of the internal fifo buffer (m_end marks the first invalid byte AFTER the internal buffer)

2.1.4.4 char∗ ctb::Fifo::m_rdptr `[protected]`

the current read position

2.1.4.5 char∗ ctb::Fifo::m_wrptr `[protected]`

the current write position

2.2 ctb::Gpib_DCS Struct Reference

```
#include <gpib.h>
```

Public Member Functions

- ∼Gpib_DCS ()
- Gpib_DCS ()

  the constructor initiate the device control struct with the common useful values and set the internal timeout for the GPIB controller to 1ms to avoid (or better reduce) blocking

- char ∗ GetSettings ()

  returns the internal parameters in a more human readable string format like 'Adr: (1,0) to:1ms'.

Public Attributes

- int m_address1
- int m_address2
- GpibTimeout m_timeout
- bool m_eot
- unsigned char m_eosChar
- unsigned char m_eosMode
- char m_buf [32]

2.2.1 Detailed Description

The device control struct for the gpib communication class. This struct should be used, if you refer advanced parameter.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 ctb::Gpib_DCS::∼Gpib_DCS () `[inline]`

to avoid memory leak warnings generated by swig

2.2.2.2 ctb::Gpib_DCS::Gpib_DCS () `[inline]`

the constructor initiate the device control struct with the common useful values and set the internal timeout for the GPIB controller to 1ms to avoid (or better reduce) blocking

set default device address to 1

set the timeout to a short value to avoid blocking (default are 1msec)

EOS character, see above!

EOS mode, see above!

2.2.3 Member Function Documentation

2.2.3.1 char ∗ ctb::Gpib_DCS::GetSettings ()

returns the internal parameters in a more human readable string format like ’Adr: (1,0) to:1ms’.

Returns:

the settings as a null terminated string

2.2.4 Member Data Documentation

2.2.4.1 int ctb::Gpib_DCS::m_address1

primary address of GPIB device

2.2.4.2 int ctb::Gpib_DCS::m_address2

secondary address of GPIB device

2.2.4.3 GpibTimeout ctb::Gpib_DCS::m_timeout

I/O timeout

2.2.4.4 bool ctb::Gpib_DCS::m_eot

EOT enable

2.2.4.5 unsigned char ctb::Gpib_DCS::m_eosChar

Defines the EOS character. Note! Defining an EOS byte does not cause the driver to automatically send that byte at the end of write I/O operations. The application is responsible for placing the EOS byte at the end of the data strings that it defines. (National Instruments NI-488.2M Function Reference Manual)

2.2.4.6 unsigned char ctb::Gpib_DCS::m_eosMode

Set the EOS mode (handling).m_eosMode may be a combination of bits ORed together. The following bits can be used: 0x04: Terminate read when EOS is detected. 0x08: Set EOI (End or identify line) with EOS on write function 0x10: Compare all 8 bits of EOS byte rather than low 7 bits (all read and write functions).

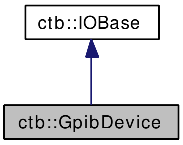
2.2.4.7 char ctb::Gpib_DCS::m_buf[32]

buffer for internal use

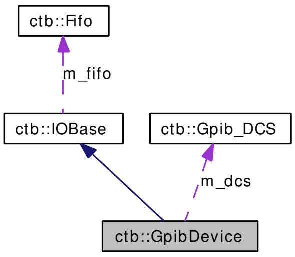
2.3 ctb::GpibDevice Class Reference

```
#include <gpib.h>
```

Inheritance diagram for ctb::GpibDevice:



Collaboration diagram for ctb::GpibDevice:



Public Member Functions

- const char ∗ ClassName ()

  returns the name of the class instance. You find this useful, if you handle different devices like a serial port or a gpib device via a IOBase pointer.

- virtual const char ∗ GetErrorDescription (int error)

  returns a more detail description of the given error number.

- virtual const char ∗ GetErrorNotation (int error)

  returns a short notation like 'EABO' of the given error number.

- virtual char ∗ GetSettingsAsString ()

  request the current settings of the connected gpib device as a null terminated string.

- int Ibrd (char ∗buf, size_t len)

  This is only for internal usage.

- int Ibwrt (char ∗buf, size_t len)

  This is only for internal usage.

- virtual int Ioctl (int cmd, void ∗args)

Many operating characteristics are only possible for special devices. To avoid the need of a lot of different functions and to give the user a uniform interface, all this special operating instructions will covered by one Ioctl methode (like the linux ioctl call). The Ioctl command (cmd) has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument args in bytes. Macros and defines used in specifying an ioctl request are located in iobase.h and the header file for the derivated device (for example in gpib.h).

- int IsOpen ()
- int Read (char ∗buf, size_t len)
- int Write (char ∗buf, size_t len)

Static Public Member Functions

- static int FindListeners (int board=0)

  FindListener returns all listening devices connected to the GPIB bus of the given board. This function is not member of the GPIB class, becauce it should do it's job before you open any GPIB connection.

Protected Member Functions

- int CloseDevice ()
- virtual const char ∗ GetErrorString (int error, bool detailed)

  returns a short notation or more detail description of the given GPIB error number.

- int OpenDevice (const char ∗devname, void ∗dcs)

Protected Attributes

- int m_board

  the internal board identifier, 0 for the first gpib controller, 1 for the second one

- int m_hd

  the file descriptor of the connected gpib device

- int m_state

  contains the internal conditions of the GPIB communication like GPIB error, timeout and so on...

- int m_error
- int m_count
- Gpib_DCS m_dcs

  contains the internal settings of the GPIB connection like address, timeout, end of string character and so one...

2.3.1 Detailed Description

GpibDevice is the basic class for communication via the GPIB bus.

2.3.2 Member Function Documentation

2.3.2.1 int ctb::GpibDevice::CloseDevice () `[protected, virtual]`

Close the interface (internally the file descriptor, which was connected with the interface).

Returns:

zero on success, otherwise -1.

Implements ctb::IOBase.

2.3.2.2 const char ∗ ctb::GpibDevice::GetErrorString (int *error*, bool *detailed*) `[protected, virtual]`

returns a short notation or more detail description of the given GPIB error number.

Parameters:

error the occured GPIB error

detailed true for a more detailed description, false otherwise

Returns:

a null terminated string with the short or detailed error message.

2.3.2.3 int ctb::GpibDevice::OpenDevice (const char ∗ *devname*, void ∗ *dcs*) `[protected, virtual]`

Open the interface (internally to request a file descriptor for the given interface). The second parameter is a undefined pointer of a Gpib_DCS data struct.

Parameters:

devname the name of the GPIB device, GPIB1 means the first GPIB controller, GPIB2 the second (if available).

dcs untyped pointer of advanced device parameters,

See also:

struct Gpib_DCS (data struct for the gpib device)

Returns:

zero on success, otherwise -1

Implements ctb::IOBase.

2.3.2.4 const char∗ ctb::GpibDevice::ClassName () `[inline, virtual]`

returns the name of the class instance. You find this useful, if you handle different devices like a serial port or a gpib device via a IOBase pointer.

Returns:

name of the class.

Reimplemented from ctb::IOBase.

2.3.2.5 virtual const char∗ ctb::GpibDevice::GetErrorDescription (int *error*) `[inline, virtual]`

returns a more detail description of the given error number.

Parameters:

error the occured error number

Returns:

null terminated string with the error description

2.3.2.6 virtual const char∗ ctb::GpibDevice::GetErrorNotation (int *error*) `[inline, virtual]`

returns a short notation like 'EABO' of the given error number.

Parameters:

error the occured error number

Returns:

null terminated string with the short error notation

2.3.2.7 virtual char∗ ctb::GpibDevice::GetSettingsAsString () `[inline, virtual]`

request the current settings of the connected gpib device as a null terminated string.

Returns:

the settings as a string like 'Adr: (1,0) to:1ms'

2.3.2.8 int ctb::GpibDevice::Ibrd (char ∗ *buf*, size_t *len*)

This is only for internal usage.

2.3.2.9 int ctb::GpibDevice::Ibwrt (char ∗ *buf*, size_t *len*)

This is only for internal usage.

2.3.2.10 int ctb::GpibDevice::Ioctl (int *cmd*, void ∗ *args*) `[virtual]`

Many operating characteristics are only possible for special devices. To avoid the need of a lot of different functions and to give the user a uniform interface, all this special operating instructions will covered by one Ioctl methode (like the linux ioctl call). The Ioctl command (cmd) has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument args in bytes. Macros and defines used in specifying an ioctl request are located in iobase.h and the header file for the derivated device (for example in gpib.h).

Parameters:

cmd one of GpibIoctls specify the ioctl request.

args is a typeless pointer to a memory location, where Ioctl reads the request arguments or write the results. Please note, that an invalid memory location or size involving a buffer overflow or segmention fault!

Reimplemented from ctb::IOBase.

2.3.2.11 int ctb::GpibDevice::IsOpen () `[inline, virtual]`

Returns the current state of the device.

Returns:

1 if device is valid and open, otherwise 0

Implements ctb::IOBase.

2.3.2.12 int ctb::GpibDevice::Read (char ∗ *buf*, size_t *len*) `[virtual]`

Read attempt to read len bytes from the interface into the buffer starting with buf. Read never blocks. If there are no bytes for reading, Read returns zero otherwise the count of bytes been readed.

Parameters:

buf starting adress of the buffer

len count of bytes, we want to read

Returns:

-1 on fails, otherwise the count of readed bytes

Implements ctb::IOBase.

2.3.2.13 int ctb::GpibDevice::Write (char ∗ *buf*, size_t *len*) `[virtual]`

Write writes up to len bytes from the buffer starting with buf into the interface.

Parameters:

buf start adress of the buffer

len count of bytes, we want to write

Returns:

on success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned.

Implements ctb::IOBase.

2.3.2.14 int ctb::GpibDevice::FindListeners (int *board* = `0`) `[static]`

FindListener returns all listening devices connected to the GPIB bus of the given board. This function is not member of the GPIB class, becauce it should do it's job before you open any GPIB connection.

Parameters:

board the board nummber. Default is the first board (=0). Valid board numbers are 0 and 1.

Returns:

-1 if an error occurred, otherwise a setting bit for each listener address. Bit0 is always 0 (address 0 isn't valid, Bit1 means address 1, Bit2 address 2 and so on...

2.3.3 Member Data Documentation

2.3.3.1 int ctb::GpibDevice::m_board `[protected]`

the internal board identifier, 0 for the first gpib controller, 1 for the second one

2.3.3.2 int ctb::GpibDevice::m_hd `[protected]`

the file descriptor of the connected gpib device

2.3.3.3 int ctb::GpibDevice::m_state `[protected]`

contains the internal conditions of the GPIB communication like GPIB error, timeout and so on...

2.3.3.4 int ctb::GpibDevice::m_error `[protected]`

the internal GPIB error number

2.3.3.5 int ctb::GpibDevice::m_count `[protected]`

the count of data read or written

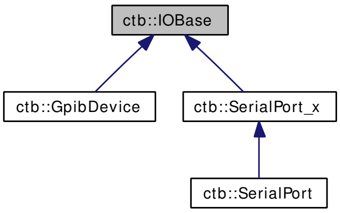
2.3.3.6 Gpib_DCS ctb::GpibDevice::m_dcs `[protected]`

contains the internal settings of the GPIB connection like address, timeout, end of string character and so one...

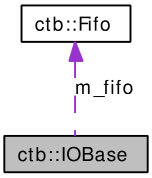
2.4 ctb::IOBase Class Reference

`#include <iobase.h>`

Inheritance diagram for ctb::IOBase:



Collaboration diagram for ctb::IOBase:



Public Member Functions

- IOBase ()
- virtual ∼IOBase ()
- virtual const char ∗ ClassName ()

  A little helper function to detect the class name.

- int Close ()
- virtual int Ioctl (int cmd, void ∗args)
- virtual int IsOpen ()=0
- int Open (const char ∗devname, void ∗dcs=0L)
- int PutBack (char ch)

  In some circumstances you want to put back a already readed byte (for instance, you have overreaded it and like to parse the recieving bytes again). The internal fifo stores fifoSize characters until you have to read again.

- virtual int Read (char ∗buf, size_t len)=0
- virtual int ReadUntilEOS (char ∗&readbuf, size_t ∗readedBytes, char ∗eosString="\n", long timeout_in_ms=1000L, char quota=0)

  ReadUntilEos read bytes from the interface until the EOS string was received or a timeout occurs. ReadUntilEos returns the count of bytes been readed. The received bytes are stored on the heap point by the readbuf pointer and must delete by the caller.

- int Readv (char ∗buf, size_t len, unsigned int timeout_in_ms)

  readv() attempts to read up to len bytes from the interface into the buffer starting at buf. readv() is blocked till len bytes are readed or the given timeout in milliseconds was reached.

- int Readv (char ∗buf, size_t len, int ∗timeout_flag, bool nice=false)

  readv() attempts to read up to len bytes from the interface into the buffer starting at buf. readv() is blocked till len bytes are readed or the timeout_flag points on a int greater then zero.

- virtual int Write (char ∗buf, size_t len)=0
- int Writev (char ∗buf, size_t len, unsigned int timeout_in_ms)
- int Writev (char ∗buf, size_t len, int ∗timeout_flag, bool nice=false)

Protected Types

- enum { fifoSize = 256 }

Protected Member Functions

- virtual int CloseDevice ()=0
- virtual int OpenDevice (const char ∗devname, void ∗dcs=0L)=0

Protected Attributes

- Fifo ∗ m_fifo

  internal fifo (first in, first out queue) to put back already readed bytes into the reading stream. After put back a single byte or sequence of characters, you can read them again with the next Read call.

2.4.1 Detailed Description

A abstract class for different interfaces. The idea behind this: Similar to the virtual file system this class defines a lot of preset member functions, which the derivate classes must be overload. In the main thing these are: open a interface (such as RS232), reading and writing non blocked through the interface and at last, close it. For special interface settings the method ioctl was defined. (control interface). ioctl covers some interface dependent settings like switch on/off the RS232 status lines and must also be defined from each derivated class.

2.4.2 Member Enumeration Documentation

2.4.2.1 anonymous enum `[protected]`

Enumerator:

fifoSize fifosize of the putback fifo

2.4.3 Constructor & Destructor Documentation

2.4.3.1 ctb::IOBase::IOBase () `[inline]`

Default constructor

2.4.3.2 virtual ctb::IOBase::~IOBase () `[inline, virtual]`

Default destructor

2.4.4 Member Function Documentation

2.4.4.1 virtual int ctb::IOBase::CloseDevice () `[protected, pure virtual]`

Close the interface (internally the file descriptor, which was connected with the interface).

Returns:

zero on success, otherwise -1.

Implemented in ctb::GpibDevice, and ctb::SerialPort.

2.4.4.2 virtual int ctb::IOBase::OpenDevice (const char ∗ *devname*, void ∗ *dcs* = `0L`) `[protected, pure virtual]`

Open the interface (internally to request a file descriptor for the given interface). The second parameter is a undefined pointer of a device dependent data struct. It must be undefined, because different devices have different settings. A serial device like the com ports points here to a data struct, includes information like baudrate, parity, count of stopbits and wordlen and so on. Another devices (for example a IEEE) needs a adress and EOS (end of string character) and don't use baudrate or parity.

Parameters:

devname the name of the device, presents the given interface. Under windows for example COM1, under Linux /dev/cua0. Use wxCOMn to avoid plattform depended code (n is the serial port number, beginning with 1).

dcs untyped pointer of advanced device parameters,

See also:

struct dcs_devCUA (data struct for the serail com ports)

Returns:

zero on success, otherwise -1

Implemented in ctb::GpibDevice, and ctb::SerialPort.

2.4.4.3 virtual const char∗ ctb::IOBase::ClassName () `[inline, virtual]`

A little helper function to detect the class name.

Returns:

the name of the class

Reimplemented in ctb::GpibDevice, and ctb::SerialPort_x.

2.4.4.4 int ctb::IOBase::Close () `[inline]`

Closed the interface. Internally it calls the CloseDevice() method, which must be defined in the derivated class.

Returns:

zero on success, or -1 if an error occurred.

2.4.4.5 virtual int ctb::IOBase::Ioctl (int *cmd*, void ∗ *args*) `[inline, virtual]`

In this method we can do all things, which are different between the discrete interfaces. The method is similar to the C ioctl function. We take a command number and a integer pointer as command parameter. An example for this is the reset of a connection between a PC and one ore more other instruments. On serial (RS232) connections mostly a break will be send, GPIB on the other hand defines a special line on the GPIB bus, to reset all connected devices. If you only want to reset your connection, you should use the Ioctl methode for doing this, independent of the real type of the connection.

Parameters:

cmd a command identifier, (under Posix such as TIOCMBIS for RS232 interfaces), IOBaseIoctls

args typeless parameter pointer for the command above.

Returns:

zero on success, or -1 if an error occurred.

Reimplemented in ctb::GpibDevice, ctb::SerialPort_x, and ctb::SerialPort.

2.4.4.6 virtual int ctb::IOBase::IsOpen () `[pure virtual]`

Returns the current state of the device.

Returns:

1 if device is valid and open, otherwise 0

Implemented in ctb::GpibDevice, and ctb::SerialPort.

2.4.4.7 int ctb::IOBase::Open (const char ∗ *devname*, void ∗ *dcs* = `0L`) `[inline]`

Parameters:

devname name of the interface, we want to open

dcs a untyped pointer to a device control struct. If he is NULL, the default device parameter will be used.

Returns:

the new file descriptor, or -1 if an error occurred

The pointer dcs will be used for special device dependent settings. Because this is very specific, the struct or destination of the pointer will be defined by every device itself. (For example: a serial device class should refer things like parity, word length and count of stop bits, a IEEE class adress and EOS character).

2.4.4.8 int ctb::IOBase::PutBack (char *ch*) `[inline]`

In some circumstances you want to put back a already readed byte (for instance, you have overreaded it and like to parse the recieving bytes again). The internal fifo stores fifoSize characters until you have to read again.

Parameters:

ch the character to put back in the input stream

Returns:

1, if successful, otherwise 0

2.4.4.9 virtual int ctb::IOBase::Read (char ∗ *buf*, size_t *len*) `[pure virtual]`

Read attempt to read len bytes from the interface into the buffer starting with buf. Read never blocks. If there are no bytes for reading, Read returns zero otherwise the count of bytes been readed.

Parameters:

buf starting adress of the buffer

len count of bytes, we want to read

Returns:

-1 on fails, otherwise the count of readed bytes

Implemented in ctb::GpibDevice, and ctb::SerialPort.

2.4.4.10 int ctb::IOBase::ReadUntilEOS (char ∗& *readbuf*, size_t ∗ *readedBytes*, char ∗ *eosString* = "\n", long *timeout_in_ms* = 1000L, char *quota* = 0) [virtual]

ReadUntilEos read bytes from the interface until the EOS string was received or a timeout occurs. ReadUntilEos returns the count of bytes been readed. The received bytes are stored on the heap point by the readbuf pointer and must delete by the caller.

Parameters:

readbuf points to the start of the readed bytes. You must delete them, also if you received no byte.

readedBytes A pointer to the variable that receives the number of bytes read.

eosString is the null terminated end of string sequence. Default is the linefeed character.

timeout_in_ms the function returns after this time, also if no eos occured (default is 1s).

quota defines a character between those an EOS doesn't terminate the string

Returns:

1 on sucess (the operation ends successfull without a timeout), 0 if a timeout occurred and -1 otherwise

2.4.4.11 int ctb::IOBase::Readv (char ∗ *buf*, size_t *len*, unsigned int *timeout_in_ms*)

readv() attempts to read up to len bytes from the interface into the buffer starting at buf. readv() is blocked till len bytes are readed or the given timeout in milliseconds was reached.

Parameters:

buf starting address of the buffer

len count bytes, we want to read

timeout_in_ms in milliseconds. If you don't want any timeout, you give the wxTIMEOUT_-INFINITY here. But think of it: In this case, this function comes never back, if there a not enough bytes to read.

Returns:

the number of data bytes successfully read

2.4.4.12 int ctb::IOBase::Readv (char ∗ *buf*, size_t *len*, int ∗ *timeout_flag*, bool *nice* = false)

readv() attempts to read up to len bytes from the interface into the buffer starting at buf. readv() is blocked till len bytes are readed or the timeout_flag points on a int greater then zero.

Parameters:

buf starting adress of the buffer

len count bytes, we want to read

timeout_flag a pointer to an integer. If you don't want any timeout, you given a null pointer here. But think of it: In this case, this function comes never back, if there a not enough bytes to read.

nice if true go to sleep for one ms (reduce CPU last), if there is no byte available (default is false)

2.4.4.13 virtual int ctb::IOBase::Write (char * *buf*, size_t *len*) [pure virtual]

Write writes up to len bytes from the buffer starting with buf into the interface.

Parameters:

buf start adress of the buffer

len count of bytes, we want to write

Returns:

on success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned.

Implemented in ctb::GpibDevice, and ctb::SerialPort.

2.4.4.14 int ctb::IOBase::Writev (char * *buf*, size_t *len*, unsigned int *timeout_in_ms*)

Writev() writes up to len bytes to the interface from the buffer, starting at buf. Also Writev() blocks till all bytes are written or the given timeout in milliseconds was reached.

Parameters:

buf starting address of the buffer

len count bytes, we want to write

timeout_in_ms timeout in milliseconds. If you give wxTIMEOUT_INFINITY here, the function blocks, till all data was written.

Returns:

the number of data bytes successfully written.

2.4.4.15 int ctb::IOBase::Writev (char * *buf*, size_t *len*, int * *timeout_flag*, bool *nice* = false)

Writev() writes up to len bytes to the interface from the buffer, starting at buf. Also Writev() blocks till all bytes are written or the timeout_flag points to an integer greater then zero.

Parameters:

buf starting adress of the buffer

len count bytes, we want to write

timeout_flag a pointer to an integer. You also can give a null pointer here. This blocks, til all data is writen.

nice if true go to sleep for one ms (reduce CPU last), if there is no byte available (default is false)

2.4.5 Member Data Documentation

2.4.5.1 Fifo* ctb::IOBase::m_fifo [protected]

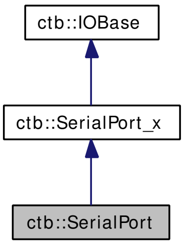
internal fifo (first in, first out queue) to put back already readed bytes into the reading stream. After put back a single byte or sequence of characters, you can read them again with the next Read call.

2.5 ctb::SerialPort Class Reference

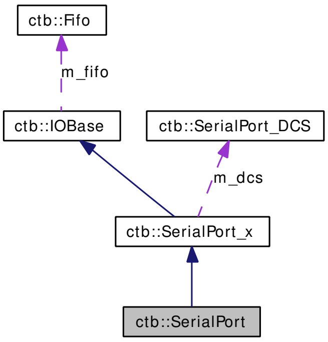
the linux version

`#include <serport.h>`

Inheritance diagram for ctb::SerialPort:



Collaboration diagram for ctb::SerialPort:



Public Member Functions

- int ChangeLineState (SerialLineState flags)

  change the linestates according to which bits are set/unset in flags.

- int ClrLineState (SerialLineState flags)

  turn off status lines depending upon which bits (DSR and/or RTS) are set in flags.

- int GetLineState ()

  Read the line states of DCD, CTS, DSR and RING.

- int Ioctl (int cmd, void ∗args)

  Many operating characteristics are only possible for special devices. To avoid the need of a lot of different functions and to give the user a uniform interface, all this special operating instructions will covered by one Ioctl methode (like the linux ioctl call). The Ioctl command (cmd) has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument args in bytes. Macros and defines used in specifying an ioctl request are located in iobase.h and the header file for the derivated device (for example in serportx.h).

- int IsOpen ()

- int Read (char ∗buf, size_t len)
- int SendBreak (int duration)

  Sendbreak transmits a continuous stream of zero-valued bits for a specific duration.

- int SetBaudrate (int baudrate)

  Set the baudrate (also non-standard) Please note: Non-standard baudrates like 70000 are not supported by each UART and depends on the RS232 chipset you apply.

- int SetParityBit (bool parity)

  Set the parity bit to a firm state, for instance to use the parity bit as the ninth bit in a 9 bit dataword communication.

- int SetLineState (SerialLineState flags)

  turn on status lines depending upon which bits (DSR and/or RTS) are set in flags.

- int Write (char ∗buf, size_t len)

Protected Member Functions

- speed_t AdaptBaudrate (int baud)

  adaptor member function, to convert the plattform independent type wxBaud into a linux conform value.

- int CloseDevice ()
- int OpenDevice (const char ∗devname, void ∗dcs)
- int SetBaudrateAny (int baudrate)

  internal member function to set an unusal (non-standard) baudrate. Called by SetBaudrate.

- int SetBaudrateStandard (int baudrate)

  internal member function to set a standard baudrate. Called by SetBaudrate.

Protected Attributes

- int fd

  under Linux, the serial ports are normal file descriptor

- termios t save_t

  Linux defines this struct termios for controling asynchronous communication. t covered the active settings, save_t the original settings.

- serial_icounter_struct save_info last_info

  The Linux serial driver summing all breaks, framings, overruns and parity errors for each port during system runtime. Because we only need the errors during a active connection, we must save the actual error numbers in this separate structurs.

2.5.1 Detailed Description

the linux version

2.5.2 Member Function Documentation

2.5.2.1 speed_t ctb::SerialPort::AdaptBaudrate (int *baud*) `[protected]`

adaptor member function, to convert the plattform independent type wxBaud into a linux conform value.

Parameters:

baud the baudrate as wxBaud type

Returns:

speed_t linux specific data type, defined in termios.h

2.5.2.2 int ctb::SerialPort::CloseDevice () `[protected, virtual]`

Close the interface (internally the file descriptor, which was connected with the interface).

Returns:

zero on success, otherwise -1.

Implements ctb::IOBase.

2.5.2.3 int ctb::SerialPort::OpenDevice (const char ∗ *devname*, void ∗ *dcs*) `[protected, virtual]`

Open the interface (internally to request a file descriptor for the given interface). The second parameter is a undefined pointer of a device dependent data struct. It must be undefined, because different devices have different settings. A serial device like the com ports points here to a data struct, includes information like baudrate, parity, count of stopbits and wordlen and so on. Another devices (for example a IEEE) needs a adress and EOS (end of string character) and don't use baudrate or parity.

Parameters:

devname the name of the device, presents the given interface. Under windows for example COM1, under Linux /dev/cua0. Use wxCOMn to avoid plattform depended code (n is the serial port number, beginning with 1).

dcs untyped pointer of advanced device parameters,

See also:

struct dcs_devCUA (data struct for the serail com ports)

Returns:

zero on success, otherwise -1

Implements ctb::IOBase.

2.5.2.4 int ctb::SerialPort::SetBaudrateAny (int *baudrate*) `[protected]`

internal member function to set an unusal (non-standard) baudrate. Called by SetBaudrate.

2.5.2.5 int ctb::SerialPort::SetBaudrateStandard (int *baudrate*) `[protected]`

internal member function to set a standard baudrate. Called by SetBaudrate.

2.5.2.6 int ctb::SerialPort::ChangeLineState (SerialLineState *flags*) `[virtual]`

change the linestates according to which bits are set/unset in flags.

Parameters:

flags valid line flags are SERIAL_LINESTATE_DSR and/or SERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implements ctb::SerialPort_x.

2.5.2.7 int ctb::SerialPort::ClrLineState (SerialLineState *flags*) `[virtual]`

turn off status lines depending upon which bits (DSR and/or RTS) are set in flags.

Parameters:

flags valid line flags are SERIAL_LINESTATE_DSR and/or SERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implements ctb::SerialPort_x.

2.5.2.8 int ctb::SerialPort::GetLineState () `[virtual]`

Read the line states of DCD, CTS, DSR and RING.

Returns:

returns the appropriate bits on sucess, otherwise -1

Implements ctb::SerialPort_x.

2.5.2.9 int ctb::SerialPort::Ioctl (int *cmd*, void $*$ *args*) `[virtual]`

Many operating characteristics are only possible for special devices. To avoid the need of a lot of different functions and to give the user a uniform interface, all this special operating instructions will covered by one Ioctl methode (like the linux ioctl call). The Ioctl command (cmd) has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument args in bytes. Macros and defines used in specifying an ioctl request are located in iobase.h and the header file for the derivated device (for example in serportx.h).

Parameters:

cmd one of SerialPortIoctls specify the ioctl request.

args is a typeless pointer to a memory location, where Ioctl reads the request arguments or write the results. Please note, that an invalid memory location or size involving a buffer overflow or segmention fault!

Reimplemented from ctb::SerialPort_x.

2.5.2.10 int ctb::SerialPort::IsOpen () `[virtual]`

Returns the current state of the device.

Returns:

1 if device is valid and open, otherwise 0

Implements ctb::IOBase.

2.5.2.11 int ctb::SerialPort::Read (char $*$ *buf*, size_t *len*) `[virtual]`

Read attempt to read len bytes from the interface into the buffer starting with buf. Read never blocks. If there are no bytes for reading, Read returns zero otherwise the count of bytes been readed.

Parameters:

buf starting adress of the buffer

len count of bytes, we want to read

Returns:

-1 on fails, otherwise the count of readed bytes

Implements ctb::IOBase.

2.5.2.12 int ctb::SerialPort::SendBreak (int *duration*) `[virtual]`

Sendbreak transmits a continuous stream of zero-valued bits for a specific duration.

Parameters:

duration If duration is zero, it transmits zero-valued bits for at least 0.25 seconds, and not more that 0.5 seconds. If duration is not zero, it sends zero-valued bits for duration$*$N seconds, where N is at least 0.25, and not more than 0.5.

Returns:

zero on success, -1 if an error occurs.

Implements ctb::SerialPort_x.

2.5.2.13 int ctb::SerialPort::SetBaudrate (int *baudrate*) `[virtual]`

Set the baudrate (also non-standard) Please note: Non-standard baudrates like 70000 are not supported by each UART and depends on the RS232 chipset you apply.

Parameters:

baudrate the new baudrate

Returns:

zero on success, -1 if an error occurs

Implements ctb::SerialPort_x.

2.5.2.14 int ctb::SerialPort::SetParityBit (bool *parity*) [virtual]

Set the parity bit to a firm state, for instance to use the parity bit as the ninth bit in a 9 bit dataword communication.

Returns:

zero on succes, a negative value if an error occurs

Implements ctb::SerialPort_x.

2.5.2.15 int ctb::SerialPort::SetLineState (SerialLineState *flags*) [virtual]

turn on status lines depending upon which bits (DSR and/or RTS) are set in flags.

Parameters:

flags valid line flags are SERIAL_LINESTATE_DSR and/or SERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implements ctb::SerialPort_x.

2.5.2.16 int ctb::SerialPort::Write (char ∗ *buf*, size_t *len*) [virtual]

Write writes up to len bytes from the buffer starting with buf into the interface.

Parameters:

buf start adress of the buffer

len count of bytes, we want to write

Returns:

on success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned.

Implements ctb::IOBase.

2.5.3 Member Data Documentation

2.5.3.1 int ctb::SerialPort::fd [protected]

under Linux, the serial ports are normal file descriptor

2.5.3.2 struct termios t ctb::SerialPort::save_t [protected]

Linux defines this struct termios for controling asynchronous communication. t covered the active settings, save_t the original settings.

2.5.3.3 struct serial_icounter_struct save_info ctb::SerialPort::last_info [protected]

The Linux serial driver summing all breaks, framings, overruns and parity errors for each port during system runtime. Because we only need the errors during a active connection, we must save the actual error numbers in this separate structurs.

2.6 ctb::SerialPort_DCS Struct Reference

```
#include <serportx.h>
```

Public Member Functions

- char ∗ GetSettings ()

  returns the internal settings of the DCS as a human readable string like '8N1 115200'.

Public Attributes

- int baud
- Parity parity
- unsigned char wordlen
- unsigned char stopbits
- bool rtscts
- bool xonxoff
- char buf [16]

2.6.1 Detailed Description

The device control struct for the serial communication class. This struct should be used, if you refer advanced parameter.

2.6.2 Member Function Documentation

2.6.2.1 char∗ ctb::SerialPort_DCS::GetSettings () `[inline]`

returns the internal settings of the DCS as a human readable string like '8N1 115200'.

Returns:

the internal settings as null terminated string

2.6.3 Member Data Documentation

2.6.3.1 int ctb::SerialPort_DCS::baud

the baudrate

2.6.3.2 Parity ctb::SerialPort_DCS::parity

the parity

2.6.3.3 unsigned char ctb::SerialPort_DCS::wordlen

the wordlen

2.6.3.4 unsigned char ctb::SerialPort_DCS::stopbits

count of stopbits

2.6.3.5 bool ctb::SerialPort_DCS::rtscts

rtscts flow control

2.6.3.6 bool ctb::SerialPort_DCS::xonxoff

XON/XOFF flow control

2.6.3.7 char ctb::SerialPort_DCS::buf[16]

buffer for internal use

2.7 ctb::SerialPort_EINFO Struct Reference

```
#include <serportx.h>
```

Public Attributes

- int brk
- int frame
- int overrun
- int parity

2.7.1 Detailed Description

The internal communication error struct. It contains the number of each error (break, framing, overrun and parity) since opening the serial port. Each error number will be cleared if the open methode was called.

2.7.2 Member Data Documentation

2.7.2.1 int ctb::SerialPort_EINFO::brk

number of breaks

2.7.2.2 int ctb::SerialPort_EINFO::frame

number of framing errors

2.7.2.3 int ctb::SerialPort_EINFO::overrun

number of overrun errors

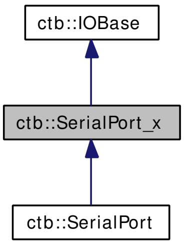
2.7.2.4 int ctb::SerialPort_EINFO::parity

number of parity errors

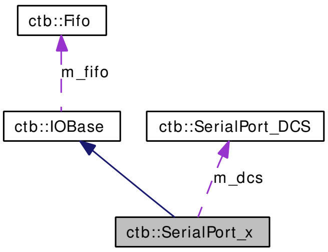
2.8 ctb::SerialPort_x Class Reference

```
#include <serportx.h>
```

Inheritance diagram for ctb::SerialPort_x:



Collaboration diagram for ctb::SerialPort_x:



Public Member Functions

- const char ∗ ClassName ()

  returns the name of the class instance. You find this useful, if you handle different devices like a serial port or a gpib device via a IOBase pointer.

- virtual int ChangeLineState (SerialLineState flags)=0

  change the linestates according to which bits are set/unset in flags.

- virtual int ClrLineState (SerialLineState flags)=0

  turn off status lines depending upon which bits (DSR and/or RTS) are set in flags.

- virtual int GetLineState ()=0

  Read the line states of DCD, CTS, DSR and RING.

- virtual char ∗ GetSettingsAsString ()

  request the current settings of the connected serial port as a null terminated string.

- virtual int Ioctl (int cmd, void ∗args)

  Many operating characteristics are only possible for special devices. To avoid the need of a lot of different functions and to give the user a uniform interface, all this special operating instructions will covered by one Ioctl methode (like the linux ioctl call). The Ioctl command (cmd) has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument args in bytes. Macros and defines used in specifying an ioctl request are located in iobase.h and the header file for the derivated device (for example in serportx.h).

- virtual int SendBreak (int duration)=0

  Sendbreak transmits a continuous stream of zero-valued bits for a specific duration.

- virtual int SetBaudrate (int baudrate)=0

  Set the baudrate (also non-standard) Please note: Non-standard baudrates like 70000 are not supported by each UART and depends on the RS232 chipset you apply.

- virtual int SetLineState (SerialLineState flags)=0

  turn on status lines depending upon which bits (DSR and/or RTS) are set in flags.

- virtual int SetParityBit (bool parity)=0

  Set the parity bit to a firm state, for instance to use the parity bit as the ninth bit in a 9 bit dataword communication.

Static Public Member Functions

- static bool IsStandardRate (int rate)

  check the given baudrate against a list of standard rates. \ return true, if the baudrate is a standard value, false otherwise

Protected Attributes

- SerialPort_DCS m_dcs

  contains the internal settings of the serial port like baudrate, protocol, wordlen and so on.

- char m_devname [SERIALPORT_NAME_LEN]

  contains the internal (os specific) name of the serial device.

2.8.1 Detailed Description

SerialPort_x is the basic class for serial communication via the serial comports. It is also an abstract class and defines all necessary methods, which the derivated plattform depended classes must be invoke.

2.8.2 Member Function Documentation

2.8.2.1 const char∗ ctb::SerialPort_x::ClassName () `[inline, virtual]`

returns the name of the class instance. You find this useful, if you handle different devices like a serial port or a gpib device via a IOBase pointer.

Returns:

name of the class.

Reimplemented from ctb::IOBase.

2.8.2.2 virtual int ctb::SerialPort_x::ChangeLineState (SerialLineState *flags*) `[pure virtual]`

change the linestates according to which bits are set/unset in flags.

Parameters:

flags valid line flags are SERIAL_LINESTATE_DSR and/or SERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implemented in ctb::SerialPort.

2.8.2.3 virtual int ctb::SerialPort_x::ClrLineState (SerialLineState *flags*) `[pure virtual]`

turn off status lines depending upon which bits (DSR and/or RTS) are set in flags.

Parameters:

flags valid line flags are SERIAL_LINESTATE_DSR and/or SERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implemented in ctb::SerialPort.

2.8.2.4 virtual int ctb::SerialPort_x::GetLineState () `[pure virtual]`

Read the line states of DCD, CTS, DSR and RING.

Returns:

returns the appropriate bits on sucess, otherwise -1

Implemented in ctb::SerialPort.

2.8.2.5 virtual char∗ ctb::SerialPort_x::GetSettingsAsString () `[inline, virtual]`

request the current settings of the connected serial port as a null terminated string.

Returns:

the settings as a string like ’8N1 115200’

2.8.2.6 virtual int ctb::SerialPort_x::Ioctl (int *cmd*, void ∗ *args*) `[inline, virtual]`

Many operating characteristics are only possible for special devices. To avoid the need of a lot of different functions and to give the user a uniform interface, all this special operating instructions will covered by one Ioctl methode (like the linux ioctl call). The Ioctl command (cmd) has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument args in bytes. Macros and defines used in specifying an ioctl request are located in iobase.h and the header file for the derivated device (for example in serportx.h).

Parameters:

cmd one of SerialPortIoctls specify the ioctl request.

args is a typeless pointer to a memory location, where Ioctl reads the request arguments or write the results. Please note, that an invalid memory location or size involving a buffer overflow or segmention fault!

Reimplemented from ctb::IOBase.

Reimplemented in ctb::SerialPort.

2.8.2.7 virtual int ctb::SerialPort_x::SendBreak (int *duration*) `[pure virtual]`

Sendbreak transmits a continuous stream of zero-valued bits for a specific duration.

Parameters:

duration If duration is zero, it transmits zero-valued bits for at least 0.25 seconds, and not more that 0.5 seconds. If duration is not zero, it sends zero-valued bits for duration∗N seconds, where N is at least 0.25, and not more than 0.5.

Returns:

zero on success, -1 if an error occurs.

Implemented in ctb::SerialPort.

2.8.2.8 virtual int ctb::SerialPort_x::SetBaudrate (int *baudrate*) `[pure virtual]`

Set the baudrate (also non-standard) Please note: Non-standard baudrates like 70000 are not supported by each UART and depends on the RS232 chipset you apply.

Parameters:

baudrate the new baudrate

Returns:

zero on success, -1 if an error occurs

Implemented in ctb::SerialPort.

2.8.2.9 virtual int ctb::SerialPort_x::SetLineState (SerialLineState *flags*) `[pure virtual]`

turn on status lines depending upon which bits (DSR and/or RTS) are set in flags.

Parameters:

flags valid line flags are SERIAL_LINESTATE_DSR and/or SERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implemented in ctb::SerialPort.

2.8.2.10 virtual int ctb::SerialPort_x::SetParityBit (bool *parity*) `[pure virtual]`

Set the parity bit to a firm state, for instance to use the parity bit as the ninth bit in a 9 bit dataword communication.

Returns:

zero on succes, a negative value if an error occurs

Implemented in ctb::SerialPort.

2.8.2.11 bool ctb::SerialPort_x::IsStandardRate (int *rate*) `[static]`

check the given baudrate against a list of standard rates. \ return true, if the baudrate is a standard value, false otherwise

2.8.3 Member Data Documentation

2.8.3.1 SerialPort_DCS ctb::SerialPort_x::m_dcs `[protected]`

contains the internal settings of the serial port like baudrate, protocol, wordlen and so on.

2.8.3.2 char ctb::SerialPort_x::m_devname[SERIALPORT_NAME_LEN] `[protected]`

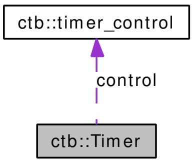
contains the internal (os specific) name of the serial device.

2.9 ctb::Timer Class Reference

A thread based timer class for handling timeouts in an easier way.

`#include <timer.h>`

Collaboration diagram for ctb::Timer:



Public Member Functions

- Timer (unsigned int msec, int ∗exitflag, void ∗(∗exitfnc)(void ∗))
- ∼Timer ()
- int start ()
- int stop ()

Protected Attributes

- timer_control control
- int stopped
- pthread_t tid
- unsigned int timer_secs

2.9.1 Detailed Description

A thread based timer class for handling timeouts in an easier way.

On starting every timer instance will create it's own thread. The thread makes simply nothing, until it's given time is over. After that, he set a variable, refer by it's adress to one and exit.

There are a lot of situations, which the timer class must handle. The timer instance leaves his valid range (for example, the timer instance is local inside a function, and the function fished) BEFORE the thread was ending. In this case, the destructor must terminate the thread in a correct way. (This is very different between the OS. threads are a system resource like file descriptors and must be deallocated after using it).

The thread should be asynchronously stopped. Means, under all circumstance, it must be possible, to finish the timer and start it again.

Several timer instance can be used simultanously.

2.9.2 Constructor & Destructor Documentation

2.9.2.1 ctb::Timer::Timer (unsigned int *msec*, int ∗ *exitflag*, void ∗(∗)(void ∗) *exitfnc*)

The constructor creates an timer object with the given properties. The timer at this moment is not started. This will be done with the start() member function.

Parameters:

msec time interval after that the the variable pointed by exitflag is setting to one.

exitflag the adress of an integer, which was set to one after the given time interval.

Warning:

The integer variable shouldn't leave it's valid range, before the timer was finished. So never take a local variable.

Parameters:

exitfnc A function, which was called after msec. If you don't want this, refer a NULL pointer.

2.9.2.2 ctb::Timer::∼Timer ()

the destructor. If his was called (for example by leaving the valid range of the timer object), the timer thread automaticaly will finished. The exitflag wouldn't be set, also the exitfnc wouldn't be called.

2.9.3 Member Function Documentation

2.9.3.1 int ctb::Timer::start ()

starts the timer. But now a thread will created and started. After this, the timer thread will be running until he was stopped by calling stop() or reached his given time interval.

2.9.3.2 int ctb::Timer::stop ()

stops the timer and canceled the timer thread. After timer::stop() a new start() will started the timer from beginning.

2.9.4 Member Data Documentation

2.9.4.1 timer_control ctb::Timer::control `[protected]`

control covers the time interval, the adress of the exitflag, and if not null, a function, which will be called on the end.

2.9.4.2 int ctb::Timer::stopped `[protected]`

stopped will be set by calling the stop() method. Internaly the timer thread steadily tests the state of this variable. If stopped not zero, the thread will be finished.

2.9.4.3 pthread_t ctb::Timer::tid `[protected]`

under linux we use the pthread library. tid covers the identifier for a separate threads.

2.9.4.4 unsigned int ctb::Timer::timer_secs `[protected]`

here we store the time interval, whilst the timer run. This is waste!!!

2.10 ctb::timer_control Struct Reference

A data struct, using from class timer.

```
#include <timer.h>
```

Public Attributes

- unsigned int usecs
- int $*$ exitflag
- void $*$($*$ exitfnc)(void $*$)

2.10.1 Detailed Description

A data struct, using from class timer.

2.10.2 Member Data Documentation

2.10.2.1 unsigned int ctb::timer_control::usecs

under linux, we used usec internally

2.10.2.2 int$*$ ctb::timer_control::exitflag

covers the adress of the exitflag

2.10.2.3 void$*$($*$ ctb::timer_control::exitfnc)(void $*$)

covers the adress of the exit function. NULL, if there was no exit function.

3 libctb File Documentation

3.1 gpib.h File Reference

Namespaces

- namespace **ctb**

Classes

- struct ctb::Gpib_DCS
- class ctb::GpibDevice

Enumerations

- enum **GpibTimeout** {
  ctb::GpibTimeoutNone = 0, ctb::GpibTimeout10us, ctb::GpibTimeout30us, ctb::GpibTimeout100us, ctb::GpibTimeout300us, ctb::GpibTimeout1ms, ctb::GpibTimeout3ms, ctb::GpibTimeout10ms, ctb::GpibTimeout30ms, ctb::GpibTimeout100ms, ctb::GpibTimeout300ms, ctb::GpibTimeout1s, ctb::GpibTimeout3s, ctb::GpibTimeout10s, ctb::GpibTimeout30s, ctb::GpibTimeout100s, ctb::GpibTimeout300s, ctb::GpibTimeout1000s }
- enum **GpibIoctls** {
  ctb::CTB_GPIB_SETADR = CTB_GPIB, ctb::CTB_GPIB_GETRSP, ctb::CTB_GPIB_GETSTA, ctb::CTB_GPIB_GETERR,

  ctb::CTB_GPIB_GETLINES, ctb::CTB_GPIB_SETTIMEOUT, ctb::CTB_GPIB_GTL, ctb::CTB_GPIB_REN,

  ctb::CTB_GPIB_RESET_BUS, ctb::CTB_GPIB_SET_EOS_CHAR, ctb::CTB_GPIB_GET_EOS_CHAR, ctb::CTB_GPIB_SET_EOS_MODE,

  ctb::CTB_GPIB_GET_EOS_MODE }

Variables

- const char ∗ ctb::GPIB1
- const char ∗ ctb::GPIB2

3.1.1 Detailed Description

3.2 serportx.h File Reference

Namespaces

- namespace **ctb**

Classes

- struct ctb::SerialPort_DCS
- struct ctb::SerialPort_EINFO
- class ctb::SerialPort_x

Defines

- #define SERIALPORT_NAME_LEN 32

Enumerations

- enum **Parity** {
  ctb::ParityNone, ctb::ParityOdd, ctb::ParityEven, ctb::ParityMark,
  ctb::ParitySpace }
- enum **SerialLineState** {
  ctb::LinestateDcd = 0x040, ctb::LinestateCts = 0x020, ctb::LinestateDsr = 0x100, ctb::LinestateDtr = 0x002,
  ctb::LinestateRing = 0x080, ctb::LinestateRts = 0x004, ctb::LinestateNull = 0x000 }
- enum **SerialPortIoctls** {
  ctb::CTB_SER_GETEINFO = CTB_SERIAL, ctb::CTB_SER_GETBRK, ctb::CTB_SER_GETFRM, ctb::CTB_SER_GETOVR,
  ctb::CTB_SER_GETPAR, ctb::CTB_SER_GETINQUE, ctb::CTB_SER_SETPAR }

Variables

- const char ∗ ctb::COM1
- const char ∗ ctb::COM2
- const char ∗ ctb::COM3
- const char ∗ ctb::COM4
- const char ∗ ctb::COM5
- const char ∗ ctb::COM6
- const char ∗ ctb::COM7
- const char ∗ ctb::COM8
- const char ∗ ctb::COM9
- const char ∗ ctb::COM10
- const char ∗ ctb::COM11
- const char ∗ ctb::COM12
- const char ∗ ctb::COM13
- const char ∗ ctb::COM14
- const char ∗ ctb::COM15
- const char ∗ ctb::COM16
- const char ∗ ctb::COM17
- const char ∗ ctb::COM18
- const char ∗ ctb::COM19
- const char ∗ ctb::COM20

3.2.1 Detailed Description

3.2.2 Define Documentation

3.2.2.1 #define SERIALPORT_NAME_LEN 32

defines the maximum length of the os depending serial port names

Index

~Fifo
    ctb::Fifo, 2
~Gpib_DCS
    ctb::Gpib_DCS, 4
~IOBase
    ctb::IOBase, 13
~Timer
    ctb::Timer, 31

AdaptBaudrate
    ctb::SerialPort, 19

baud
    ctb::SerialPort_DCS, 24
brk
    ctb::SerialPort_EINFO, 25
buf
    ctb::SerialPort_DCS, 25

ChangeLineState
    ctb::SerialPort, 20
    ctb::SerialPort_x, 27
ClassName
    ctb::GpibDevice, 8
    ctb::IOBase, 13
    ctb::SerialPort_x, 27
clear
    ctb::Fifo, 2
Close
    ctb::IOBase, 14
CloseDevice
    ctb::GpibDevice, 7
    ctb::IOBase, 13
    ctb::SerialPort, 19
ClrLineState
    ctb::SerialPort, 20
    ctb::SerialPort_x, 28
control
    ctb::Timer, 32
ctb::Fifo, 1
    ~Fifo, 2
    clear, 2
    Fifo, 2
    get, 2
    items, 2
    m_begin, 3
    m_end, 3
    m_rdptr, 3
    m_size, 3
    m_wrptr, 3
    put, 2
    read, 3
    write, 3
ctb::Gpib_DCS, 4
    ~Gpib_DCS, 4
    GetSettings, 5
    Gpib_DCS, 4
    m_address1, 5
    m_address2, 5
    m_buf, 5
    m_eosChar, 5
    m_eosMode, 5
    m_eot, 5
    m_timeout, 5
ctb::GpibDevice, 5
ctb::GpibDevice
    ClassName, 8
    CloseDevice, 7
    FindListeners, 10
    GetErrorDescription, 8
    GetErrorNotation, 8
    GetErrorString, 7
    GetSettingsAsString, 9
    Ibrd, 9
    Ibwrt, 9
    Ioctl, 9
    IsOpen, 9
    m_board, 10
    m_count, 11
    m_dcs, 11
    m_error, 11
    m_hd, 10
    m_state, 10
    OpenDevice, 8
    Read, 9
    Write, 10
ctb::IOBase, 11
    ~IOBase, 13
    ClassName, 13
    Close, 14
    CloseDevice, 13
    fifoSize, 13
    IOBase, 13
    Ioctl, 14
    IsOpen, 14
    m_fifo, 17
    Open, 14
    OpenDevice, 13
    PutBack, 15
    Read, 15
    ReadUntilEOS, 15

Readv, 16
Write, 16
Writev, 16, 17
ctb::SerialPort, 17
ctb::SerialPort
AdaptBaudrate, 19
ChangeLineState, 20
CloseDevice, 19
ClrLineState, 20
fd, 23
GetLineState, 21
Ioctl, 21
IsOpen, 21
last_info, 23
OpenDevice, 20
Read, 21
save_t, 23
SendBreak, 22
SetBaudrate, 22
SetBaudrateAny, 20
SetBaudrateStandard, 20
SetLineState, 22
SetParityBit, 22
Write, 23
ctb::SerialPort_DCS, 23
ctb::SerialPort_DCS
baud, 24
buf, 25
GetSettings, 24
parity, 24
rtscts, 24
stopbits, 24
wordlen, 24
xonxoff, 24
ctb::SerialPort_EINFO, 25
ctb::SerialPort_EINFO
brk, 25
frame, 25
overrun, 25
parity, 25
ctb::SerialPort_x, 25
ctb::SerialPort_x
ChangeLineState, 27
ClassName, 27
ClrLineState, 28
GetLineState, 28
GetSettingsAsString, 28
Ioctl, 28
IsStandardRate, 30
m_dcs, 30
m_devname, 30
SendBreak, 29
SetBaudrate, 29
SetLineState, 29
SetParityBit, 29
ctb::Timer, 30
~Timer, 31
control, 32
start, 31
stop, 31
stopped, 32
tid, 32
Timer, 31
timer_secs, 32
ctb::timer_control, 32
exitflag, 32
exitfnc, 32
usecs, 32

exitflag
ctb::timer_control, 32
exitfnc
ctb::timer_control, 32

fd
ctb::SerialPort, 23
Fifo
ctb::Fifo, 2
fifoSize
ctb::IOBase, 13
FindListeners
ctb::GpibDevice, 10
frame
ctb::SerialPort_EINFO, 25

get
ctb::Fifo, 2
GetErrorDescription
ctb::GpibDevice, 8
GetErrorNotation
ctb::GpibDevice, 8
GetErrorString
ctb::GpibDevice, 7
GetLineState
ctb::SerialPort, 21
ctb::SerialPort_x, 28
GetSettings
ctb::Gpib_DCS, 5
ctb::SerialPort_DCS, 24
GetSettingsAsString
ctb::GpibDevice, 9
ctb::SerialPort_x, 28
gpib.h, 33
Gpib_DCS
ctb::Gpib_DCS, 4

Ibrd
ctb::GpibDevice, 9

Ibwrt
ctb::GpibDevice, 9
IOBase
ctb::IOBase, 13
Ioctl
ctb::GpibDevice, 9
ctb::IOBase, 14
ctb::SerialPort, 21
ctb::SerialPort_x, 28
IsOpen
ctb::GpibDevice, 9
ctb::IOBase, 14
ctb::SerialPort, 21
IsStandardRate
ctb::SerialPort_x, 30
items
ctb::Fifo, 2

last_info
ctb::SerialPort, 23

m_address1
ctb::Gpib_DCS, 5
m_address2
ctb::Gpib_DCS, 5
m_begin
ctb::Fifo, 3
m_board
ctb::GpibDevice, 10
m_buf
ctb::Gpib_DCS, 5
m_count
ctb::GpibDevice, 11
m_dcs
ctb::GpibDevice, 11
ctb::SerialPort_x, 30
m_devname
ctb::SerialPort_x, 30
m_end
ctb::Fifo, 3
m_eosChar
ctb::Gpib_DCS, 5
m_eosMode
ctb::Gpib_DCS, 5
m_eot
ctb::Gpib_DCS, 5
m_error
ctb::GpibDevice, 11
m_fifo
ctb::IOBase, 17
m_hd
ctb::GpibDevice, 10
m_rdptr
ctb::Fifo, 3
m_size
ctb::Fifo, 3
m_state
ctb::GpibDevice, 10
m_timeout
ctb::Gpib_DCS, 5
m_wrptr
ctb::Fifo, 3

Open
ctb::IOBase, 14
OpenDevice
ctb::GpibDevice, 8
ctb::IOBase, 13
ctb::SerialPort, 20
overrun
ctb::SerialPort_EINFO, 25

parity
ctb::SerialPort_DCS, 24
ctb::SerialPort_EINFO, 25
put
ctb::Fifo, 2
PutBack
ctb::IOBase, 15

Read
ctb::GpibDevice, 9
ctb::IOBase, 15
ctb::SerialPort, 21
read
ctb::Fifo, 3
ReadUntilEOS
ctb::IOBase, 15
Readv
ctb::IOBase, 16
rtscts
ctb::SerialPort_DCS, 24

save_t
ctb::SerialPort, 23
SendBreak
ctb::SerialPort, 22
ctb::SerialPort_x, 29
SERIALPORT_NAME_LEN
serportx.h, 34
serportx.h, 33
SERIALPORT_NAME_LEN, 34
SetBaudrate
ctb::SerialPort, 22
ctb::SerialPort_x, 29
SetBaudrateAny
ctb::SerialPort, 20
SetBaudrateStandard

ctb::SerialPort, 20
SetLineState
ctb::SerialPort, 22
ctb::SerialPort_x, 29
SetParityBit
ctb::SerialPort, 22
ctb::SerialPort_x, 29
start
ctb::Timer, 31
stop
ctb::Timer, 31
stopbits
ctb::SerialPort_DCS, 24
stopped
ctb::Timer, 32

tid
ctb::Timer, 32
Timer
ctb::Timer, 31
timer_secs
ctb::Timer, 32

usecs
ctb::timer_control, 32

wordlen
ctb::SerialPort_DCS, 24
Write
ctb::GpibDevice, 10
ctb::IOBase, 16
ctb::SerialPort, 23
write
ctb::Fifo, 3
Writev
ctb::IOBase, 16, 17

xonxoff
ctb::SerialPort_DCS, 24