0.1 ctb overview

The ctb (communication toolbox) library was realized, to simplify the communication with other instruments throughout the serial com ports (at first). To make my life easier, it should works with Linux and all win32 plattforms (excepted windows 3.1, which is a only 16bit OS) because I develope my applications for both plattforms).

Some times later GPIB support was added to make ctb an integrated part for the extensive test and calibration system of a company I worked these days.

The main goal of the library was a non-blocked communication to avoid frozen GUIs waiting for data which in some conditions never arrives.

On the base ctb defines an abstract class IOBase, which must be derivate for several interfaces (at now this was done for the RS232 comports and GPIB IEEE488 interface).

This leads to another feature: Because all classes depends on one super class, you have just open your wanted interface and don't worry about it's special typ later. This is like the 'Virtual Instrument' featured by Nation Instruments LabView.

Last not least: ctb provides one written code for Linux and Windows (compiles well with GNU G++ and VC++). Without any dependences (execept for a standard C++ compilier) ctb runs also in small enviroments like embedded systems and doesn't need any graphic stuff for use.

ctb is composed of five parts:

- ctb::IOBase class
- ctb::SerialPort class
- ctb::GpibDevice class
- ctb::Timer class
- ctb::Fifo class

0.1.1 IOBase class

An 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.

0.1.2 SerialPort class

The class for the serial ports is named as ctb::SerialPort. SerialPort is a wrapper for non blocked reading and writing. This is easy under linux, but with windows a lot more tricky. SerialPort is as simple as possible. It doesn't create any gui events or signals, so it works also standalone. It's also not a device driver, means, you must call the read method, if you look for receiving data.

You can write any desired data with any length (length type is size_t, I think, on win32 and linux this is a 32Bit integer) and SerialPort returns the really writen data length, also you can read a lot of data and SerialPort returns the really received data count.

Both, read and write returns immediatelly. Using these, the program never blocks. Also IOBase implements a blocked read and write. You can use these functions, if you want a definitiv count of data and never accept less than this. Because there is a difficulty, when the communication is interrupted or death, both blocked functions get a timeout flag to returns after a given time interval. The timeouts will be handled with the second timer class.

As an additional benefit ctb features also 9 Bit transmission (with take advantage of the parity bit), non-standard baudrates (depending on your hardware but not on ctb) and all parity eventualities including static parity settings like Mark and Space.

0.1.3 GpibDevice class

Named as ctb::GpibDevice. In the philosophy of the SerialPort class GpibDevice also supports non-blocking communication. You can instant as many GpibDevice objects as you need for instance to communicate with a lot of different bus participants in a typical GPIB enviroment. GPIB support was tested with PCI cards and USB adapter from Nation Instrument and Keithley.

0.1.4 Timer class

The idea of the ctb::Timer class is to base on the Unix C alarm function. You create a Timer with a given alarm time and a adress of flag, which the timer must set after the time is over.

Because the alarm function cannot used more than once in the same process (under windows I don't know a similar function), every timer instance will be a separate thread after starting it. So you can start a timer and continue in your program, make a lot of things and test the flag whenever you want this. (For example, you read/write a given count of data).

Note:

I think, it's a better style, to request a given count of data in 100ms (for example) and trap the situation, if there are not enough data after this time. And not do this for every byte!

0.1.5 Fifo cass

Provides a simple thread safe fifo to realize a fast and simple communication pipe between two threads (and was used also as a put back mechanism for the wxIOBase and it's derivated classes).

ctb::Fifo tackles the concurrently access from different threads with an internal temporary pointer asignment which was atomic. From there no mutex or semaphore is involved and lead to a fast access.

Please note:

The thread safeness is limited to the put/get write/read methods but which should be sufficent for a fifo.

0.2 libctb Namespace Documentation

0.2.1 ctb Namespace Reference

Classes

- class Fifo
- struct Gpib_DCS
- class GpibDevice
- class IOBase
- class SerialPort

  the linux version

- struct SerialPort_DCS
- struct SerialPort_EINFO
- class SerialPort_x
- class Timer

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

- struct timer_control

  A data struct, using from class timer.

Enumerations

- enum { **CTB_COMMON** = 0x0000, **CTB_SERIAL** = 0x0100, **CTB_GPIB** = 0x0200, **CTB_-TIMEOUT_INFINITY** = 0xFFFFFFFF }
- enum GpibIoctls {

  CTB_GPIB_SETADR = CTB_GPIB, CTB_GPIB_GETRSP, CTB_GPIB_GETSTA, CTB_-GPIB_GETERR,

  CTB_GPIB_GETLINES, CTB_GPIB_SETTIMEOUT, CTB_GPIB_GTL, CTB_GPIB_REN,

  CTB_GPIB_RESET_BUS, CTB_GPIB_SET_EOS_CHAR, CTB_GPIB_GET_EOS_CHAR, CTB_GPIB_SET_EOS_MODE,

  CTB_GPIB_GET_EOS_MODE }
- enum GpibTimeout {

  GpibTimeoutNone = 0, GpibTimeout10us, GpibTimeout30us, GpibTimeout100us,

  GpibTimeout300us, GpibTimeout1ms, GpibTimeout3ms, GpibTimeout10ms,

  GpibTimeout30ms, GpibTimeout100ms, GpibTimeout300ms, GpibTimeout1s,

  GpibTimeout3s, GpibTimeout10s, GpibTimeout30s, GpibTimeout100s,

  GpibTimeout300s, GpibTimeout1000s }
- enum IOBaseIoctls { CTB_RESET = CTB_COMMON }
- enum Parity {

  ParityNone, ParityOdd, ParityEven, ParityMark,

  ParitySpace }

  Defines the different modes of parity checking. Under Linux, the struct termios will be set to provide the wanted behaviour.

- enum SerialLineState {
  LinestateDcd = 0x040, LinestateCts = 0x020, LinestateDsr = 0x100, LinestateDtr = 0x002,
  LinestateRing = 0x080, LinestateRts = 0x004, LinestateNull = 0x000 }
- enum SerialPortIoctls {
  CTB_SER_GETEINFO = CTB_SERIAL, CTB_SER_GETBRK, CTB_SER_GETFRM, CTB_SER_GETOVR,
  CTB_SER_GETPAR, CTB_SER_GETINQUE, CTB_SER_SETPAR }

Functions

- bool GetAvailablePorts (std::vector$<$ std::string $>$ &result, bool checkInUse=true)

  returns all available COM ports as an array of strings.
- char GetKey ()
- void sleepms (unsigned int ms)

  sleepms A plattform independent function, to go to sleep for the given time interval.
- static void timer_exit (void ∗arg)
- static void ∗ timer_fnc (void ∗arg)

Variables

- const char ∗ COM1
- const char ∗ COM1 = "/dev/ttyS0"
- const char ∗ COM10
- const char ∗ COM10 = "/dev/ttyS9"
- const char ∗ COM11
- const char ∗ COM11 = "/dev/ttyS10"
- const char ∗ COM12
- const char ∗ COM12 = "/dev/ttyS11"
- const char ∗ COM13
- const char ∗ COM13 = "/dev/ttyS12"
- const char ∗ COM14
- const char ∗ COM14 = "/dev/ttyS13"
- const char ∗ COM15
- const char ∗ COM15 = "/dev/ttyS14"
- const char ∗ COM16
- const char ∗ COM16 = "/dev/ttyS15"
- const char ∗ COM17
- const char ∗ COM17 = "/dev/ttyS16"
- const char ∗ COM18
- const char ∗ COM18 = "/dev/ttyS17"
- const char ∗ COM19
- const char ∗ COM19 = "/dev/ttyS18"
- const char ∗ COM2
- const char ∗ COM2 = "/dev/ttyS1"
- const char ∗ COM20
- const char ∗ COM20 = "/dev/ttyS19"

- const char ∗ COM3
- const char ∗ COM3 = "/dev/ttyS2"
- const char ∗ COM4
- const char ∗ COM4 = "/dev/ttyS3"
- const char ∗ COM5
- const char ∗ COM5 = "/dev/ttyS4"
- const char ∗ COM6
- const char ∗ COM6 = "/dev/ttyS5"
- const char ∗ COM7
- const char ∗ COM7 = "/dev/ttyS6"
- const char ∗ COM8
- const char ∗ COM8 = "/dev/ttyS7"
- const char ∗ COM9
- const char ∗ COM9 = "/dev/ttyS8"
- const char ∗ GPIB1
- const char ∗ GPIB1 = "gpib1"
- const char ∗ GPIB2
- const char ∗ GPIB2 = "gpib2"
- static gpibErr_t gpibErrors [ ]

0.2.1.1 Enumeration Type Documentation

enum ctb::GpibIoctls

The following Ioctl calls are only valid for the GpibDevice class.

Enumerator:

CTB_GPIB_SETADR Set the adress of the via gpib connected device.

CTB_GPIB_GETRSP Get the serial poll byte

CTB_GPIB_GETSTA Get the GPIB status

CTB_GPIB_GETERR Get the last GPIB error number

CTB_GPIB_GETLINES Get the GPIB line status (hardware control lines) as an integer. The lowest 8 bits correspond to the current state of the lines.

CTB_GPIB_SETTIMEOUT Set the GPIB specific timeout

CTB_GPIB_GTL Forces the specified device to go to local program mode

CTB_GPIB_REN This routine can only be used if the specified GPIB Interface Board is the System Controller. Remember that even though the REN line is asserted, the device(s) will not be put into remote state until is addressed to listen by the Active Controller

CTB_GPIB_RESET_BUS The command asserts the GPIB interface clear (IFC) line for ast least 100us if the GPIB board is the system controller. This initializes the GPIB and makes the interface CIC and active controller with ATN asserted. Note! The IFC signal resets only the GPIB interface functions of the bus devices and not the internal device functions. For a device reset you should use the CTB_RESET command above.

CTB_GPIB_SET_EOS_CHAR Configure the end-of-string (EOS) termination 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)

CTB_GPIB_GET_EOS_CHAR Get the internal EOS termination character (see above).

CTB_GPIB_SET_EOS_MODE 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).

CTB_GPIB_GET_EOS_MODE Get the internal EOS mode (see above).

Definition at line 141 of file gpib.h.

enum ctb::GpibTimeout

NI488.2 API defines the following valid timeouts.

Enumerator:

GpibTimeoutNone no timeout (infinity)

GpibTimeout10us 10 micro seconds

GpibTimeout30us 30 micro seconds

GpibTimeout100us 100 micro seconds

GpibTimeout300us 300 micro seconds

GpibTimeout1ms 1 milli second

GpibTimeout3ms 3 milli seconds

GpibTimeout10ms 10 milli seconds

GpibTimeout30ms 30 milli seconds

GpibTimeout100ms 0.1 seconds

GpibTimeout300ms 0.3 seconds

GpibTimeout1s 1 second

GpibTimeout3s 3 seconds

GpibTimeout10s 10 seconds

GpibTimeout30s 30 seconds

GpibTimeout100s 100 seconds

GpibTimeout300s 300 seconds (5 minutes)

GpibTimeout1000s 1000 seconds

Definition at line 30 of file gpib.h.

enum ctb::IOBaseIoctls

Defines the ioctl calls for derivated classes. The following Ioctl calls are valid for all from wxIOBase derivated classes.

Enumerator:

CTB_RESET Reset the connected device. For a serial (RS232) connection, a break is send. For GPIB the IFC (Interface Clear) line is set.

Definition at line 37 of file iobase.h.

enum ctb::Parity

Defines the different modes of parity checking. Under Linux, the struct termios will be set to provide the wanted behaviour.

Enumerator:

ParityNone no parity check

ParityOdd odd parity check

ParityEven even parity check

ParityMark mark (not implemented yet)

ParitySpace space (not implemented yet)

Definition at line 80 of file serportx.h.

enum ctb::SerialLineState

Defines the different modem control lines. The value for each item are defined in /usr/include/bits/ioctl-types.h. This is the linux definition. The window version translate each item in it's own value. modem lines defined in ioctl-types.h

```
#define TIOCM_LE    0x001
#define TIOCM_DTR   0x002
#define TIOCM_RTS   0x004
#define TIOCM_ST    0x008
#define TIOCM_SR    0x010
#define TIOCM_CTS   0x020
#define TIOCM_CAR   0x040
#define TIOCM_RNG   0x080
#define TIOCM_DSR   0x100
#define TIOCM_CD    TIOCM_CAR
#define TIOCM_RI    TIOCM_RNG
```

Enumerator:

LinestateDcd Data Carrier Detect (read only)

LinestateCts Clear To Send (read only)

LinestateDsr Data Set Ready (read only)

LinestateDtr Data Terminal Ready (write only)

LinestateRing Ring Detect (read only)

LinestateRts Request To Send (write only)

LinestateNull no active line state, use this for clear

Definition at line 116 of file serportx.h.

enum ctb::SerialPortIoctls

The following Ioctl calls are only valid for the SerialPort class.

Enumerator:

CTB_SER_GETEINFO Get all numbers of occured communication errors (breaks framing, overrun and parity), so the args parameter of the Ioctl call must pointed to a SerialPort_-EINFO struct.

CTB_SER_GETBRK Get integer 1, if a break occured since the last call so the args parameter of the Ioctl methode must pointed to an integer value. If there was no break, the result is integer 0.

CTB_SER_GETFRM Get integer 1, if a framing occured since the last call so the args parameter of the Ioctl methode must pointed to an integer value. If there was no break, the result is integer 0.

CTB_SER_GETOVR Get integer 1, if a overrun occured since the last call so the args parameter of the Ioctl methode must pointed to an integer value. If there was no break, the result is integer 0.

CTB_SER_GETPAR Get integer 1, if a parity occured since the last call so the args parameter of the Ioctl methode must pointed to an integer value. If there was no break, the result is integer 0.

CTB_SER_GETINQUE Get the number of bytes received by the serial port driver but not yet read by a Read or Readv Operation.

CTB_SER_SETPAR Set the parity bit on or off to use it as a ninth bit.

Definition at line 212 of file serportx.h.

0.2.1.2 Function Documentation

bool ctb::GetAvailablePorts (std::vector< std::string > & *result*, bool *checkInUse* = `true`**)**

returns all available COM ports as an array of strings.

Parameters:

result stores the available COM ports
checkInUse return only ports which are available AND unused (default)

Returns:

true if successful, false otherwise

Definition at line 12 of file portscan.cpp.

References ctb::SerialPort_x::Open().

void ctb::sleepms (unsigned int *ms*)

sleepms A plattform independent function, to go to sleep for the given time interval.

Parameters:

ms time interval in milli seconds

Definition at line 92 of file timer.cpp.

Referenced by ctb::IOBase::ReadUntilEOS(), ctb::IOBase::Readv(), and ctb::IOBase::Writev().

0.2.1.3 Variable Documentation

const char* ctb::COM1

specifices the first serial port

Definition at line 24 of file serport.cpp.

const char∗ ctb::COM1 = "/dev/ttyS0"

specifices the first serial port

Definition at line 24 of file serport.cpp.

const char∗ ctb::COM10

specifies the tenth serial port

Definition at line 33 of file serport.cpp.

const char∗ ctb::COM10 = "/dev/ttyS9"

specifies the tenth serial port

Definition at line 33 of file serport.cpp.

const char∗ ctb::COM11

specifies the eleventh serial port

Definition at line 34 of file serport.cpp.

const char∗ ctb::COM11 = "/dev/ttyS10"

specifies the eleventh serial port

Definition at line 34 of file serport.cpp.

const char∗ ctb::COM12

specifies the twelfth serial port

Definition at line 35 of file serport.cpp.

const char∗ ctb::COM12 = "/dev/ttyS11"

specifies the twelfth serial port

Definition at line 35 of file serport.cpp.

const char∗ ctb::COM13

specifies the thriteenth serial port

Definition at line 36 of file serport.cpp.

const char∗ ctb::COM13 = "/dev/ttyS12"

specifies the thriteenth serial port

Definition at line 36 of file serport.cpp.

const char∗ ctb::COM14

specifies the fourteenth serial port

Definition at line 37 of file serport.cpp.

const char∗ ctb::COM14 = "/dev/ttyS13"

specifies the fourteenth serial port

Definition at line 37 of file serport.cpp.

const char∗ ctb::COM15

specifies the fiveteenth serial port

Definition at line 38 of file serport.cpp.

const char∗ ctb::COM15 = "/dev/ttyS14"

specifies the fiveteenth serial port

Definition at line 38 of file serport.cpp.

const char∗ ctb::COM16

specifies the sixteenth serial port

Definition at line 39 of file serport.cpp.

const char∗ ctb::COM16 = "/dev/ttyS15"

specifies the sixteenth serial port

Definition at line 39 of file serport.cpp.

const char∗ ctb::COM17

specifies the seventeenth serial port

Definition at line 40 of file serport.cpp.

const char∗ ctb::COM17 = "/dev/ttyS16"

specifies the seventeenth serial port

Definition at line 40 of file serport.cpp.

const char∗ ctb::COM18

specifies the eighteenth serial port

Definition at line 41 of file serport.cpp.

const char∗ ctb::COM18 = "/dev/ttyS17"

specifies the eighteenth serial port

Definition at line 41 of file serport.cpp.

const char∗ ctb::COM19

specifies the nineteenth serial port

Definition at line 42 of file serport.cpp.

const char∗ ctb::COM19 = "/dev/ttyS18"

specifies the nineteenth serial port

Definition at line 42 of file serport.cpp.

const char∗ ctb::COM2

specifies the second serial port

Definition at line 25 of file serport.cpp.

const char∗ ctb::COM2 = "/dev/ttyS1"

specifies the second serial port

Definition at line 25 of file serport.cpp.

const char∗ ctb::COM20

specifies the twentieth serial port

Definition at line 43 of file serport.cpp.

const char∗ ctb::COM20 = "/dev/ttyS19"

specifies the twentieth serial port

Definition at line 43 of file serport.cpp.

const char∗ ctb::COM3

specifies the third serial port

Definition at line 26 of file serport.cpp.

const char∗ ctb::COM3 = "/dev/ttyS2"

specifies the third serial port

Definition at line 26 of file serport.cpp.

const char∗ ctb::COM4

specifies the fourth serial port

Definition at line 27 of file serport.cpp.

const char∗ ctb::COM4 = "/dev/ttyS3"

specifies the fourth serial port

Definition at line 27 of file serport.cpp.

const char∗ ctb::COM5

specifies the fifth serial port

Definition at line 28 of file serport.cpp.

const char∗ ctb::COM5 = "/dev/ttyS4"

specifies the fifth serial port

Definition at line 28 of file serport.cpp.

const char∗ ctb::COM6

specifies the sixth serial port

Definition at line 29 of file serport.cpp.

const char∗ ctb::COM6 = "/dev/ttyS5"

specifies the sixth serial port

Definition at line 29 of file serport.cpp.

const char∗ ctb::COM7

specifies the seventh serial port

Definition at line 30 of file serport.cpp.

const char∗ ctb::COM7 = "/dev/ttyS6"

specifies the seventh serial port

Definition at line 30 of file serport.cpp.

const char∗ ctb::COM8

specifies the eighth serial port

Definition at line 31 of file serport.cpp.

const char∗ ctb::COM8 = "/dev/ttyS7"

specifies the eighth serial port

Definition at line 31 of file serport.cpp.

const char∗ ctb::COM9

specifies the ninth serial port

Definition at line 32 of file serport.cpp.

const char∗ ctb::COM9 = "/dev/ttyS8"

specifies the ninth serial port

Definition at line 32 of file serport.cpp.

const char∗ ctb::GPIB1

defines the os specific name for the first gpib controller

Definition at line 23 of file gpib.cpp.

const char∗ ctb::GPIB1 = "gpib1"

defines the os specific name for the first gpib controller

Definition at line 23 of file gpib.cpp.

const char∗ ctb::GPIB2

defines the os specific name for the second gpib controller

Definition at line 24 of file gpib.cpp.

const char∗ ctb::GPIB2 = "gpib2"

defines the os specific name for the second gpib controller

Definition at line 24 of file gpib.cpp.

gpibErr_t ctb::gpibErrors[ ] `[static]`

Initial value:

```
{
      {0,"EDVR","DOS Error"},
      {1,"ECIC","Specified GPIB Interface Board is Not Active Controller"},
      {2,"ENOL","No present listing device"},
      {3,"EADR","GPIB Board has not been addressed properly"},
      {4,"EARG","Invalid argument"},
      {5,"ESAC","Specified GPIB Interface Board is not System Controller"},
      {6,"EABO","I/O operation aborted (time-out)"},
      {7,"ENEB","Non-existent GPIB board"},
      {10,"EOIP","Routine not allowed during asynchronous I/O operation"},
      {11,"ECAP","No capability for operation"},
      {12,"EFSO","File System Error"},
      {14,"EBUS","Command byte transfer error"},
      {15,"ESTB","Serial poll status byte lost"},
      {16,"ESQR","SRQ stuck in ON position"},
      {20,"ETAB","Table problem"},
      {247,"EINT","No interrupt configured on board"},
      {248,"EWMD","Windows is not in Enhanced mode"},
      {249,"EVDD","GPIB driver is not installed"},
      {250,"EOVR","Buffer Overflow"},
      {251,"ESML","Two library calls running simultaneously"},
      {252,"ECFG","Board type does not match GPIB.CFG"},
      {253,"ETMR","No Windows timers available"},
      {254,"ESLC","No Windows selectors available"},
      {255,"EBRK","Control-Break pressed"}
   }
```

Definition at line 32 of file gpib.cpp.

Referenced by ctb::GpibDevice::GetErrorString().

0.3 libctb Class Documentation

0.3.1 ctb::Fifo Class Reference

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

0.3.1.1 Detailed Description

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

Definition at line 25 of file fifo.h.

Public Member Functions

- 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.*

- Fifo (size_t size)

  the constructor initialize a fifo with the given size.

- 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.

- virtual ∼Fifo ()

  the destructor destroys all internal memory.

Protected Attributes

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

0.3.1.2 Constructor & Destructor Documentation

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

the constructor initialize a fifo with the given size.

Parameters:

size size of the fifo

Definition at line 14 of file fifo.cpp.

References m_begin, m_end, m_rdptr, m_size, and m_wrptr.

ctb::Fifo::∼Fifo () `[virtual]`

the destructor destroys all internal memory.

Definition at line 22 of file fifo.cpp.

References m_begin.

0.3.1.3 Member Function Documentation

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.

Definition at line 27 of file fifo.cpp.

References m_begin, m_rdptr, and m_wrptr.

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

Definition at line 32 of file fifo.cpp.

References m_begin, m_end, m_rdptr, and m_wrptr.

size_t ctb::Fifo::items ()

query the fifo for it's available bytes.

Returns:

count of readable bytes, storing in the fifo

Definition at line 44 of file fifo.cpp.

References m_rdptr, m_size, and m_wrptr.

Referenced by ctb::SerialPort::Read(), and ctb::GpibDevice::Read().

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

Definition at line 69 of file fifo.cpp.

References m_begin, m_end, m_rdptr, and m_wrptr.

Referenced by ctb::IOBase::PutBack().

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

Definition at line 91 of file fifo.cpp.

References m_begin, m_end, m_rdptr, and m_wrptr.

Referenced by ctb::SerialPort::Read(), and ctb::GpibDevice::Read().

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

Definition at line 111 of file fifo.cpp.

References m_begin, m_end, m_rdptr, and m_wrptr.

0.3.1.4 Member Data Documentation

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

the start of the internal fifo buffer

Definition at line 31 of file fifo.h.

Referenced by clear(), Fifo(), get(), put(), read(), write(), and ∼Fifo().

char∗ ctb::Fifo::m_end `[protected]`

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

Definition at line 36 of file fifo.h.

Referenced by Fifo(), get(), put(), read(), and write().

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

the current read position

Definition at line 38 of file fifo.h.

Referenced by clear(), Fifo(), get(), items(), put(), read(), and write().

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

the size of the fifo

Definition at line 29 of file fifo.h.

Referenced by Fifo(), and items().

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

the current write position

Definition at line 40 of file fifo.h.

Referenced by clear(), Fifo(), get(), items(), put(), read(), and write().

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

- fifo.h
- fifo.cpp

0.3.2 ctb::Gpib_DCS Struct Reference

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

0.3.2.1 Detailed Description

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

Definition at line 76 of file gpib.h.

Public Member Functions

- char ∗ GetSettings ()

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

- 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

- ∼Gpib_DCS ()

Public Attributes

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

0.3.2.2 Constructor & Destructor Documentation

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

to avoid memory leak warnings generated by swig

Definition at line 107 of file gpib.h.

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!

Definition at line 113 of file gpib.h.

References ctb::GpibTimeout1ms, m_address1, m_address2, m_eosChar, m_eosMode, m_eot, and m_timeout.

0.3.2.3 Member Function Documentation

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

Definition at line 59 of file gpib.cpp.

References m_address1, m_address2, m_buf, and m_timeout.

Referenced by ctb::GpibDevice::GetSettingsAsString().

0.3.2.4 Member Data Documentation

int ctb::Gpib_DCS::m_address1

primary address of GPIB device

Definition at line 79 of file gpib.h.

Referenced by GetSettings(), Gpib_DCS(), ctb::GpibDevice::Ioctl(), ctb::GpibDevice::Open(), and ctb::GpibDevice::OpenDevice().

int ctb::Gpib_DCS::m_address2

secondary address of GPIB device

Definition at line 81 of file gpib.h.

Referenced by GetSettings(), Gpib_DCS(), and ctb::GpibDevice::OpenDevice().

char ctb::Gpib_DCS::m_buf[32]

buffer for internal use

Definition at line 105 of file gpib.h.

Referenced by GetSettings().

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)

Definition at line 94 of file gpib.h.

Referenced by Gpib_DCS(), ctb::GpibDevice::Ioctl(), and ctb::GpibDevice::OpenDevice().

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).

Definition at line 103 of file gpib.h.

Referenced by Gpib_DCS(), ctb::GpibDevice::Ioctl(), and ctb::GpibDevice::OpenDevice().

bool ctb::Gpib_DCS::m_eot

EOT enable

Definition at line 85 of file gpib.h.

Referenced by Gpib_DCS(), and ctb::GpibDevice::OpenDevice().

GpibTimeout ctb::Gpib_DCS::m_timeout

I/O timeout

Definition at line 83 of file gpib.h.

Referenced by GetSettings(), Gpib_DCS(), and ctb::GpibDevice::OpenDevice().

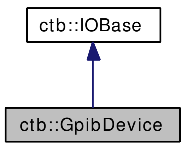
The documentation for this struct was generated from the following files:

- gpib.h
- gpib.cpp

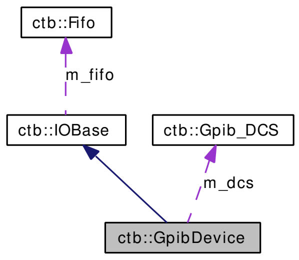
0.3.3 ctb::GpibDevice Class Reference

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

Inheritance diagram for ctb::GpibDevice:



Collaboration diagram for ctb::GpibDevice:



0.3.3.1 Detailed Description

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

Definition at line 222 of file gpib.h.

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.

- int Close ()
- 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.

- GpibDevice ()
- 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 Open (const char ∗devname, void ∗dcs=0L)
- int Open (const char ∗devname, int address)

Opens a GPIB device in a user likely way. Insteed of using the Device Control Struct just input your parameter in a more intuitive manner.

- int PutBack (char ch)

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

- int Read (char ∗buf, size_t len)
- 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, 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.

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

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 Types

- enum { fifoSize = 256 }

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_count
- Gpib_DCS m_dcs

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

- int m_error
- 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.

- 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...

0.3.3.2 Member Enumeration Documentation

anonymous enum `[protected, inherited]`

Enumerator:

fifoSize fifosize of the putback fifo

Definition at line 71 of file iobase.h.

0.3.3.3 Member Function Documentation

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.

Definition at line 286 of file gpib.h.

int ctb::IOBase::Close () `[inline, inherited]`

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.

Definition at line 123 of file iobase.h.

References ctb::IOBase::CloseDevice().

Referenced by ∼GpibDevice(), and ctb::SerialPort::∼SerialPort().

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.

Definition at line 73 of file gpib.cpp.

References m_board, and m_hd.

Referenced by OpenDevice().

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...

Definition at line 228 of file gpib.cpp.

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

Definition at line 293 of file gpib.h.

References GetErrorString().

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

Definition at line 302 of file gpib.h.

References GetErrorString().

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.

Definition at line 86 of file gpib.cpp.

References ctb::gpibErrors.

Referenced by GetErrorDescription(), and GetErrorNotation().

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'

Definition at line 310 of file gpib.h.

References ctb::Gpib_DCS::GetSettings(), and m_dcs.

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

This is only for internal usage.

Definition at line 102 of file gpib.cpp.

References m_hd.

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

This is only for internal usage.

Definition at line 108 of file gpib.cpp.

References m_hd.

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.

Definition at line 113 of file gpib.cpp.

References ctb::CTB_GPIB_GET_EOS_CHAR, ctb::CTB_GPIB_GET_EOS_MODE, ctb::CTB_-GPIB_GETERR, ctb::CTB_GPIB_GETLINES, ctb::CTB_GPIB_GETRSP, ctb::CTB_GPIB_-GETSTA, ctb::CTB_GPIB_GTL, ctb::CTB_GPIB_REN, ctb::CTB_GPIB_RESET_BUS, ctb::CTB_GPIB_SET_EOS_CHAR, ctb::CTB_GPIB_SET_EOS_MODE, ctb::CTB_GPIB_-SETTIMEOUT, ctb::CTB_RESET, ctb::GpibTimeout1000s, ctb::GpibTimeout100ms, ctb::Gpib-Timeout100s, ctb::GpibTimeout10ms, ctb::GpibTimeout10s, ctb::GpibTimeout1ms, ctb::Gpib-Timeout1s, ctb::GpibTimeout300ms, ctb::GpibTimeout300s, ctb::GpibTimeout30ms, ctb::Gpib-Timeout30s, ctb::GpibTimeout3ms, ctb::GpibTimeout3s, ctb::GpibTimeoutNone, ctb::Gpib_-DCS::m_address1, m_board, m_dcs, ctb::Gpib_DCS::m_eosChar, ctb::Gpib_DCS::m_eosMode, m_error, m_hd, and m_state.

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.

Definition at line 339 of file gpib.h.

References m_hd.

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

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).

Definition at line 163 of file iobase.h.

References ctb::IOBase::OpenDevice().

int ctb::GpibDevice::Open (const char ∗ *devname*, int *address*)

Opens a GPIB device in a user likely way. Insteed of using the Device Control Struct just input your parameter in a more intuitive manner.

Parameters:

devname the name of the GPIB controler like GPIB1 or GPIB2

address the address of the connected device (1...31)

Returns:

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

Definition at line 258 of file gpib.cpp.

References ctb::Gpib_DCS::m_address1, m_dcs, and OpenDevice().

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.

Definition at line 266 of file gpib.cpp.

References CloseDevice(), ctb::GpibTimeout1000s, ctb::GpibTimeout10us, ctb::Gpib_DCS::m_-address1, ctb::Gpib_DCS::m_address2, m_board, m_count, m_dcs, ctb::Gpib_DCS::m_eos-Char, ctb::Gpib_DCS::m_eosMode, ctb::Gpib_DCS::m_eot, m_error, m_hd, m_state, and ctb::Gpib_DCS::m_timeout.

Referenced by Open().

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

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

Definition at line 176 of file iobase.h.

References ctb::IOBase::m_fifo, and ctb::Fifo::put().

Referenced by ctb::IOBase::ReadUntilEOS().

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.

Definition at line 318 of file gpib.cpp.

References ctb::Fifo::items(), m_count, m_error, ctb::IOBase::m_fifo, m_hd, m_state, and ctb::Fifo::read().

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

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

Definition at line 77 of file iobase.cpp.

References ctb::IOBase::PutBack(), ctb::IOBase::Read(), ctb::sleepms(), and ctb::Timer::start().

int ctb::IOBase::Readv (char ∗ *buf,* size_t *len,* int ∗ *timeout_flag,* bool *nice* = `false`**)** `[inherited]`

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)

Definition at line 51 of file iobase.cpp.

References ctb::IOBase::Read(), and ctb::sleepms().

int ctb::IOBase::Readv (char ∗ *buf,* size_t *len,* unsigned int *timeout_in_ms*) `[inherited]`

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 never returns if there a not enough bytes to read.

Returns:

the number of data bytes successfully read

Definition at line 19 of file iobase.cpp.

References ctb::IOBase::Read(), and ctb::sleepms().

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.

Definition at line 330 of file gpib.cpp.

References m_count, m_error, m_hd, and m_state.

int ctb::IOBase::Writev (char ∗ *buf*, size_t *len*, int ∗ *timeout_flag*, bool *nice* = `false`**)** `[inherited]`

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)

Definition at line 188 of file iobase.cpp.

References ctb::sleepms(), and ctb::IOBase::Write().

int ctb::IOBase::Writev (char ∗ *buf*, size_t *len*, unsigned int *timeout_in_ms*) `[inherited]`

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.

Definition at line 158 of file iobase.cpp.

References ctb::sleepms(), ctb::Timer::start(), and ctb::IOBase::Write().

0.3.3.4 Member Data Documentation

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

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

Definition at line 230 of file gpib.h.

Referenced by CloseDevice(), GpibDevice(), Ioctl(), and OpenDevice().

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

the count of data read or written

Definition at line 245 of file gpib.h.

Referenced by GpibDevice(), OpenDevice(), Read(), and Write().

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...

Definition at line 250 of file gpib.h.

Referenced by GetSettingsAsString(), Ioctl(), Open(), and OpenDevice().

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

the internal GPIB error number

Definition at line 243 of file gpib.h.

Referenced by GpibDevice(), Ioctl(), OpenDevice(), Read(), and Write().

Fifo∗ ctb::IOBase::m_fifo `[protected, inherited]`

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.

Definition at line 70 of file iobase.h.

Referenced by ctb::IOBase::IOBase(), ctb::IOBase::PutBack(), ctb::SerialPort::Read(), Read(), and ctb::IOBase::∼IOBase().

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

the file descriptor of the connected gpib device

Definition at line 235 of file gpib.h.

Referenced by CloseDevice(), GpibDevice(), Ibrd(), Ibwrt(), Ioctl(), IsOpen(), OpenDevice(), Read(), and Write().

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

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

Definition at line 241 of file gpib.h.

Referenced by GpibDevice(), Ioctl(), OpenDevice(), Read(), and Write().

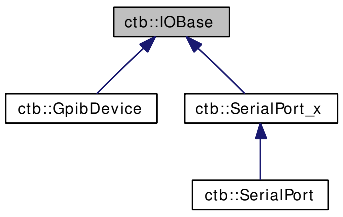
The documentation for this class was generated from the following files:

- gpib.h
- gpib.cpp

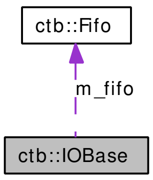
0.3.4 ctb::IOBase Class Reference

`#include <iobase.h>`

Inheritance diagram for ctb::IOBase:



Collaboration diagram for ctb::IOBase:



0.3.4.1 Detailed Description

An 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.

Definition at line 61 of file iobase.h.

Public Member Functions

- virtual const char ∗ ClassName ()

  A little helper function to detect the class name.

- int Close ()
- IOBase ()
- 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 over-readed 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, 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.

- 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.

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

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.

0.3.4.2 Member Enumeration Documentation

anonymous enum `[protected]`

Enumerator:

fifoSize fifosize of the putback fifo

Definition at line 71 of file iobase.h.

0.3.4.3 Constructor & Destructor Documentation

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

Default constructor

Definition at line 103 of file iobase.h.

References fifoSize, and m_fifo.

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

Default destructor

Definition at line 110 of file iobase.h.

References m_fifo.

0.3.4.4 Member Function Documentation

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.

Definition at line 117 of file iobase.h.

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.

Definition at line 123 of file iobase.h.

References CloseDevice().

Referenced by ctb::GpibDevice::~GpibDevice(), and ctb::SerialPort::~SerialPort().

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.

Referenced by Close().

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), IOBase-Ioctls

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.

Definition at line 142 of file iobase.h.

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.

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).

Definition at line 163 of file iobase.h.

References OpenDevice().

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.

Referenced by ctb::SerialPort_x::Open(), and Open().

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

Definition at line 176 of file iobase.h.

References m_fifo, and ctb::Fifo::put().

Referenced by ReadUntilEOS().

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.

Referenced by ReadUntilEOS(), and Readv().

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

Definition at line 77 of file iobase.cpp.

References PutBack(), Read(), ctb::sleepms(), and ctb::Timer::start().

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)

Definition at line 51 of file iobase.cpp.

References Read(), and ctb::sleepms().

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 never returns if there a not enough bytes to read.

Returns:

the number of data bytes successfully read

Definition at line 19 of file iobase.cpp.

References Read(), and ctb::sleepms().

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.

Referenced by Writev().

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)

Definition at line 188 of file iobase.cpp.

References ctb::sleepms(), and Write().

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.

Definition at line 158 of file iobase.cpp.

References ctb::sleepms(), ctb::Timer::start(), and Write().

0.3.4.5 Member Data Documentation

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.

Definition at line 70 of file iobase.h.

Referenced by IOBase(), PutBack(), ctb::SerialPort::Read(), ctb::GpibDevice::Read(), and ∼IOBase().

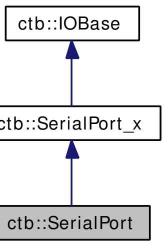
The documentation for this class was generated from the following files:

- iobase.h
- iobase.cpp

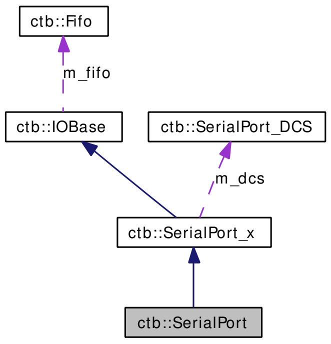
0.3.5 ctb::SerialPort Class Reference

```
#include <serport.h>
```

Inheritance diagram for ctb::SerialPort:



Collaboration diagram for ctb::SerialPort:



0.3.5.1 Detailed Description

the linux version

Definition at line 23 of file linux/serport.h.

Public Types

- enum FlowControl { NoFlowControl, RtsCtsFlowControl, XonXoffFlowControl }

  Specifies the flow control.

Public Member Functions

- int ChangeLineState (SerialLineState flags)

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

- 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.

- int Close ()
- 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.

- virtual char ∗ GetSettingsAsString ()

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

- 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 Open (const char ∗devname, void ∗dcs=0L)
- int Open (const int portnumber, int baudrate, const char ∗protocol="8N1", FlowControl flowControl=NoFlowControl)

  Opens the serial port with the given number.

- int Open (const char ∗portname, int baudrate, const char ∗protocol="8N1", FlowControl flowControl=NoFlowControl)

  Opens a serial port in a user likely way. Insteed of using the Device Control Struct just input your parameter in a more intuitive manner.

- 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.

- int Read (char ∗buf, size_t len)
- 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, 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.

- 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 SendBreak (int duration)

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

- SerialPort ()
- 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 SetLineState (SerialLineState flags)

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

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

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 Types

- enum { fifoSize = 256 }

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

- 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.

- 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.

- 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.

- termios t save_t

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

0.3.5.2 Member Enumeration Documentation

anonymous enum `[protected, inherited]`

Enumerator:

fifoSize fifosize of the putback fifo

Definition at line 71 of file iobase.h.

enum ctb::SerialPort_x::FlowControl `[inherited]`

Specifies the flow control.

Enumerator:

NoFlowControl No flow control at all

RtsCtsFlowControl Enable RTS/CTS hardware flow control

XonXoffFlowControl Enable XON/XOFF protocol

Definition at line 287 of file serportx.h.

0.3.5.3 Member Function Documentation

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

Definition at line 56 of file serport.cpp.

Referenced by OpenDevice(), and SetBaudrateStandard().

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.

Definition at line 101 of file serport.cpp.

References fd.

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

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.

Definition at line 304 of file serportx.h.

int ctb::IOBase::Close () `[inline, inherited]`

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.

Definition at line 123 of file iobase.h.

References ctb::IOBase::CloseDevice().

Referenced by ctb::GpibDevice::∼GpibDevice(), and ∼SerialPort().

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.

Definition at line 79 of file serport.cpp.

References fd.

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.

Definition at line 109 of file serport.cpp.

References fd.

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.

Definition at line 114 of file serport.cpp.

References fd, and ctb::LinestateNull.

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

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

Returns:

the settings as a string like '8N1 115200'

Definition at line 335 of file serportx.h.

References ctb::SerialPort_DCS::GetSettings(), and ctb::SerialPort_x::m_dcs.

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.

Definition at line 137 of file serport.cpp.

References ctb::CTB_RESET, ctb::CTB_SER_GETBRK, ctb::CTB_SER_GETEINFO, ctb::CTB_SER_GETFRM, ctb::CTB_SER_GETINQUE, ctb::CTB_SER_GETOVR, ctb::CTB_-SER_GETPAR, ctb::CTB_SER_SETPAR, fd, last_info, SendBreak(), and SetParityBit().

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.

Definition at line 190 of file serport.cpp.

References fd.

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

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

Definition at line 86 of file serportx.cpp.

Referenced by OpenDevice(), and SetBaudrate().

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

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).

Definition at line 163 of file iobase.h.

References ctb::IOBase::OpenDevice().

int ctb::SerialPort_x::Open (const int *portnumber*, int *baudrate*, const char ∗ *protocol* = `"8N1"`**, FlowControl *flowControl* =** `NoFlowControl`**)** `[inherited]`

Opens the serial port with the given number.

Note:

The port numbering starts with 1 (COM1 for windows and /dev/ttyS0 for Linux. Please note, that USB to RS232 converter in Linux are named as /dev/ttyUSBx and from there have to opened with their device name!

Parameters:

number of the serial port count from 1

baudrate any baudrate, also an unusual one, if your serial device support them

protocol a string with the number of databits (5...8), the parity setting (N=None,O=Odd,E=Even,M=Mark,S=Space), also in lower case, and the count of stopbits (1...2)

flowControl one of NoFlowControl, RtsCtsFlowControl or XonXoffFlowControl.

Returns:

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

Definition at line 63 of file serportx.cpp.

References ctb::SerialPort_x::Open().

int ctb::SerialPort_x::Open (const char ∗ *portname*, int *baudrate*, const char ∗ *protocol* = `"8N1"`**, FlowControl *flowControl* =** `NoFlowControl`**)** `[inherited]`

Opens a serial port in a user likely way. Insteed of using the Device Control Struct just input your parameter in a more intuitive manner.

Parameters:

portname the name of the serial port

baudrate any baudrate, also an unusual one, if your serial device support them

protocol a string with the number of databits (5...8), the parity setting (N=None,O=Odd,E=Even,M=Mark,S=Space), also in lower case, and the count of stopbits (1...2)

flowControl one of NoFlowControl, RtsCtsFlowControl or XonXoffFlowControl.

Returns:

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

Definition at line 7 of file serportx.cpp.

References ctb::SerialPort_DCS::baud, ctb::SerialPort_x::m_dcs, ctb::IOBase::OpenDevice(), ctb::SerialPort_DCS::parity, ctb::ParityEven, ctb::ParityMark, ctb::ParityNone, ctb::ParityOdd, ctb::ParitySpace, ctb::SerialPort_DCS::rtscts, ctb::SerialPort_x::RtsCtsFlowControl, ctb::SerialPort_DCS::stopbits, ctb::SerialPort_DCS::wordlen, ctb::SerialPort_DCS::xonxoff, and ctb::SerialPort_x::XonXoffFlowControl.

Referenced by ctb::GetAvailablePorts(), and ctb::SerialPort_x::Open().

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.

Definition at line 195 of file serport.cpp.

References AdaptBaudrate(), ctb::SerialPort_DCS::baud, fd, ctb::SerialPort_x::IsStandardRate(), last_info, ctb::SerialPort_x::m_dcs, ctb::SerialPort_x::m_devname, ctb::SerialPort_DCS::parity, ctb::ParityEven, ctb::ParityMark, ctb::ParityNone, ctb::ParityOdd, ctb::ParitySpace, ctb::SerialPort_DCS::rtscts, save_t, SetBaudrateAny(), ctb::SerialPort_DCS::stopbits, ctb::SerialPort_DCS::wordlen, and ctb::SerialPort_DCS::xonxoff.

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

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

Definition at line 176 of file iobase.h.

References ctb::IOBase::m_fifo, and ctb::Fifo::put().

Referenced by ctb::IOBase::ReadUntilEOS().

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.

Definition at line 310 of file serport.cpp.

References fd, ctb::Fifo::items(), ctb::IOBase::m_fifo, and ctb::Fifo::read().

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

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

Definition at line 77 of file iobase.cpp.

References ctb::IOBase::PutBack(), ctb::IOBase::Read(), ctb::sleepms(), and ctb::Timer::start().

int ctb::IOBase::Readv (char * *buf*, size_t *len*, int * *timeout_flag*, bool *nice* = `false`**)** `[inherited]`

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)

Definition at line 51 of file iobase.cpp.

References ctb::IOBase::Read(), and ctb::sleepms().

int ctb::IOBase::Readv (char * *buf*, size_t *len*, unsigned int *timeout_in_ms*) `[inherited]`

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 never returns if there a not enough bytes to read.

Returns:

the number of data bytes successfully read

Definition at line 19 of file iobase.cpp.

References ctb::IOBase::Read(), and ctb::sleepms().

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.

Definition at line 323 of file serport.cpp.

References fd.

Referenced by Ioctl().

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.

Definition at line 391 of file serport.cpp.

References ctb::SerialPort_x::IsStandardRate(), SetBaudrateAny(), and SetBaudrateStandard().

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

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

Definition at line 360 of file serport.cpp.

References fd.

Referenced by OpenDevice(), and SetBaudrate().

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

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

Definition at line 375 of file serport.cpp.

References AdaptBaudrate(), ctb::SerialPort_DCS::baud, fd, and ctb::SerialPort_x::m_dcs.

Referenced by SetBaudrate().

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.

Definition at line 399 of file serport.cpp.

References fd.

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.

Definition at line 404 of file serport.cpp.

References fd.

Referenced by Ioctl().

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.

Definition at line 433 of file serport.cpp.

References fd.

int ctb::IOBase::Writev (char * *buf*, size_t *len*, int * *timeout_flag*, bool *nice* = `false`**)** `[inherited]`

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)

Definition at line 188 of file iobase.cpp.

References ctb::sleepms(), and ctb::IOBase::Write().

int ctb::IOBase::Writev (char ∗ *buf*, size_t *len*, unsigned int *timeout_in_ms*) `[inherited]`

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.

Definition at line 158 of file iobase.cpp.

References ctb::sleepms(), ctb::Timer::start(), and ctb::IOBase::Write().

0.3.5.4 Member Data Documentation

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

under Linux, the serial ports are normal file descriptor

Definition at line 29 of file linux/serport.h.

Referenced by ChangeLineState(), CloseDevice(), ClrLineState(), GetLineState(), Ioctl(), IsOpen(), OpenDevice(), Read(), SendBreak(), SerialPort(), SetBaudrateAny(), SetBaudrateStandard(), SetLineState(), SetParityBit(), and Write().

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.

Definition at line 43 of file linux/serport.h.

Referenced by Ioctl(), and OpenDevice().

SerialPort_DCS ctb::SerialPort_x::m_dcs `[protected, inherited]`

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

Definition at line 273 of file serportx.h.

Referenced by ctb::SerialPort_x::GetSettingsAsString(), ctb::SerialPort_x::Open(), OpenDevice(), and SetBaudrateStandard().

char ctb::SerialPort_x::m_devname[SERIALPORT_NAME_LEN] `[protected, inherited]`

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

Definition at line 278 of file serportx.h.

Referenced by OpenDevice(), and ctb::SerialPort_x::SerialPort_x().

Fifo∗ ctb::IOBase::m_fifo `[protected, inherited]`

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.

Definition at line 70 of file iobase.h.

Referenced by ctb::IOBase::IOBase(), ctb::IOBase::PutBack(), Read(), ctb::GpibDevice::Read(), and ctb::IOBase::∼IOBase().

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.

Definition at line 35 of file linux/serport.h.

Referenced by OpenDevice().

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

- linux/serport.h
- serport.cpp

0.3.6 ctb::SerialPort_DCS Struct Reference

`#include <serportx.h>`

0.3.6.1 Detailed Description

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

Definition at line 140 of file serportx.h.

Public Member Functions

- char ∗ GetSettings ()

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

- SerialPort_DCS ()
- ∼SerialPort_DCS ()

Public Attributes

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

0.3.6.2 Member Function Documentation

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

Definition at line 171 of file serportx.h.

References baud, buf, parity, stopbits, and wordlen.

Referenced by ctb::SerialPort_x::GetSettingsAsString().

0.3.6.3 Member Data Documentation

int ctb::SerialPort_DCS::baud

the baudrate

Definition at line 143 of file serportx.h.

Referenced by GetSettings(), ctb::SerialPort_x::Open(), ctb::SerialPort::OpenDevice(), SerialPort_DCS(), and ctb::SerialPort::SetBaudrateStandard().

char ctb::SerialPort_DCS::buf[16]

buffer for internal use

Definition at line 155 of file serportx.h.

Referenced by GetSettings().

Parity ctb::SerialPort_DCS::parity

the parity

Definition at line 145 of file serportx.h.

Referenced by GetSettings(), ctb::SerialPort_x::Open(), ctb::SerialPort::OpenDevice(), and SerialPort_DCS().

bool ctb::SerialPort_DCS::rtscts

rtscts flow control

Definition at line 151 of file serportx.h.

Referenced by ctb::SerialPort_x::Open(), ctb::SerialPort::OpenDevice(), and SerialPort_DCS().

unsigned char ctb::SerialPort_DCS::stopbits

count of stopbits

Definition at line 149 of file serportx.h.

Referenced by GetSettings(), ctb::SerialPort_x::Open(), ctb::SerialPort::OpenDevice(), and SerialPort_DCS().

unsigned char ctb::SerialPort_DCS::wordlen

the wordlen

Definition at line 147 of file serportx.h.

Referenced by GetSettings(), ctb::SerialPort_x::Open(), ctb::SerialPort::OpenDevice(), and SerialPort_DCS().

bool ctb::SerialPort_DCS::xonxoff

XON/XOFF flow control

Definition at line 153 of file serportx.h.

Referenced by ctb::SerialPort_x::Open(), ctb::SerialPort::OpenDevice(), and SerialPort_DCS().

The documentation for this struct was generated from the following file:

- serportx.h

0.3.7 ctb::SerialPort_EINFO Struct Reference

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

0.3.7.1 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.

Definition at line 191 of file serportx.h.

Public Member Functions

- SerialPort_EINFO ()

Public Attributes

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

0.3.7.2 Member Data Documentation

int ctb::SerialPort_EINFO::brk

number of breaks

Definition at line 194 of file serportx.h.

Referenced by SerialPort_EINFO().

int ctb::SerialPort_EINFO::frame

number of framing errors

Definition at line 196 of file serportx.h.

Referenced by SerialPort_EINFO().

int ctb::SerialPort_EINFO::overrun

number of overrun errors

Definition at line 198 of file serportx.h.

Referenced by SerialPort_EINFO().

int ctb::SerialPort_EINFO::parity

number of parity errors

Definition at line 200 of file serportx.h.

Referenced by SerialPort_EINFO().

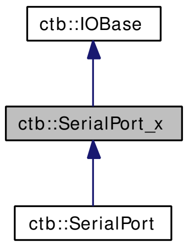
The documentation for this struct was generated from the following file:

- serportx.h

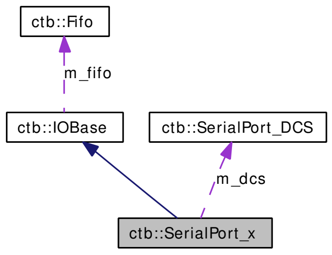
0.3.8 ctb::SerialPort_x Class Reference

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

Inheritance diagram for ctb::SerialPort_x:



Collaboration diagram for ctb::SerialPort_x:



0.3.8.1 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.

Definition at line 266 of file serportx.h.

Public Types

- enum FlowControl { NoFlowControl, RtsCtsFlowControl, XonXoffFlowControl }

  Specifies the flow control.

Public Member Functions

- virtual int ChangeLineState (SerialLineState flags)=0

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

- 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.

- int Close ()
- 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 IsOpen ()=0
- int Open (const char ∗devname, void ∗dcs=0L)
- int Open (const int portnumber, int baudrate, const char ∗protocol="8N1", FlowControl flowControl=NoFlowControl)

  Opens the serial port with the given number.

- int Open (const char ∗portname, int baudrate, const char ∗protocol="8N1", FlowControl flowControl=NoFlowControl)

  Opens a serial port in a user likely way. Insteed of using the Device Control Struct just input your parameter in a more intuitive manner.

- 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, 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.

- 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.

- virtual int SendBreak (int duration)=0

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

- SerialPort_x ()
- 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.

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

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 Types

- enum { fifoSize = 256 }

Protected Member Functions

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

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.

- 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.

0.3.8.2 Member Enumeration Documentation

anonymous enum `[protected, inherited]`

Enumerator:

fifoSize fifosize of the putback fifo

Definition at line 71 of file iobase.h.

enum ctb::SerialPort_x::FlowControl

Specifies the flow control.

Enumerator:

NoFlowControl No flow control at all

RtsCtsFlowControl Enable RTS/CTS hardware flow control

XonXoffFlowControl Enable XON/XOFF protocol

Definition at line 287 of file serportx.h.

0.3.8.3 Member Function Documentation

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.

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.

Definition at line 304 of file serportx.h.

int ctb::IOBase::Close () `[inline, inherited]`

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.

Definition at line 123 of file iobase.h.

References ctb::IOBase::CloseDevice().

Referenced by ctb::GpibDevice::∼GpibDevice(), and ctb::SerialPort::∼SerialPort().

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

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.

Referenced by ctb::IOBase::Close().

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.

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.

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'

Definition at line 335 of file serportx.h.

References ctb::SerialPort_DCS::GetSettings(), and m_dcs.

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.

Definition at line 356 of file serportx.h.

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

Returns the current state of the device.

Returns:

1 if device is valid and open, otherwise 0

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

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

Definition at line 86 of file serportx.cpp.

Referenced by ctb::SerialPort::OpenDevice(), and ctb::SerialPort::SetBaudrate().

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

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).

Definition at line 163 of file iobase.h.

References ctb::IOBase::OpenDevice().

int ctb::SerialPort_x::Open (const int *portnumber*, int *baudrate*, const char ∗ *protocol* = `"8N1"`, FlowControl *flowControl* = `NoFlowControl`)

Opens the serial port with the given number.

Note:

The port numbering starts with 1 (COM1 for windows and /dev/ttyS0 for Linux. Please note, that USB to RS232 converter in Linux are named as /dev/ttyUSBx and from there have to opened with their device name!

Parameters:

number of the serial port count from 1

baudrate any baudrate, also an unusual one, if your serial device support them

protocol a string with the number of databits (5...8), the parity setting (N=None,O=Odd,E=Even,M=Mark,S=Space), also in lower case, and the count of stopbits (1...2)

flowControl one of NoFlowControl, RtsCtsFlowControl or XonXoffFlowControl.

Returns:

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

Definition at line 63 of file serportx.cpp.

References Open().

int ctb::SerialPort_x::Open (const char ∗ *portname*, int *baudrate*, const char ∗ *protocol* = `"8N1"`**, FlowControl *flowControl* =** `NoFlowControl`**)**

Opens a serial port in a user likely way. Insteed of using the Device Control Struct just input your parameter in a more intuitive manner.

Parameters:

portname the name of the serial port

baudrate any baudrate, also an unusual one, if your serial device support them

protocol a string with the number of databits (5...8), the parity setting (N=None,O=Odd,E=Even,M=Mark,S=Space), also in lower case, and the count of stopbits (1...2)

flowControl one of NoFlowControl, RtsCtsFlowControl or XonXoffFlowControl.

Returns:

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

Definition at line 7 of file serportx.cpp.

References ctb::SerialPort_DCS::baud, m_dcs, ctb::IOBase::OpenDevice(), ctb::SerialPort_-DCS::parity, ctb::ParityEven, ctb::ParityMark, ctb::ParityNone, ctb::ParityOdd, ctb::ParitySpace, ctb::SerialPort_DCS::rtscts, RtsCtsFlowControl, ctb::SerialPort_DCS::stopbits, ctb::SerialPort_-DCS::wordlen, ctb::SerialPort_DCS::xonxoff, and XonXoffFlowControl.

Referenced by ctb::GetAvailablePorts(), and Open().

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

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.

Referenced by Open(), and ctb::IOBase::Open().

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

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

Definition at line 176 of file iobase.h.

References ctb::IOBase::m_fifo, and ctb::Fifo::put().

Referenced by ctb::IOBase::ReadUntilEOS().

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

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.

Referenced by ctb::IOBase::ReadUntilEOS(), and ctb::IOBase::Readv().

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

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

Definition at line 77 of file iobase.cpp.

References ctb::IOBase::PutBack(), ctb::IOBase::Read(), ctb::sleepms(), and ctb::Timer::start().

int ctb::IOBase::Readv (char ∗ *buf*, size_t *len*, int ∗ *timeout_flag*, bool *nice* = `false`**)** `[inherited]`

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)

Definition at line 51 of file iobase.cpp.

References ctb::IOBase::Read(), and ctb::sleepms().

int ctb::IOBase::Readv (char ∗ *buf*, size_t *len*, unsigned int *timeout_in_ms*) `[inherited]`

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 never returns if there a not enough bytes to read.

Returns:

the number of data bytes successfully read

Definition at line 19 of file iobase.cpp.

References ctb::IOBase::Read(), and ctb::sleepms().

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.

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.

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.

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.

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

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.

Referenced by ctb::IOBase::Writev().

int ctb::IOBase::Writev (char ∗ *buf*, size_t *len*, int ∗ *timeout_flag*, bool *nice* = `false`**)** `[inherited]`

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)

Definition at line 188 of file iobase.cpp.

References ctb::sleepms(), and ctb::IOBase::Write().

int ctb::IOBase::Writev (char ∗ *buf*, size_t *len*, unsigned int *timeout_in_ms*) `[inherited]`

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.

Definition at line 158 of file iobase.cpp.

References ctb::sleepms(), ctb::Timer::start(), and ctb::IOBase::Write().

0.3.8.4 Member Data Documentation

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

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

Definition at line 273 of file serportx.h.

Referenced by GetSettingsAsString(), Open(), ctb::SerialPort::OpenDevice(), and ctb::SerialPort::SetBaudrateStandard().

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

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

Definition at line 278 of file serportx.h.

Referenced by ctb::SerialPort::OpenDevice(), and SerialPort_x().

Fifo∗ ctb::IOBase::m_fifo `[protected, inherited]`

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.

Definition at line 70 of file iobase.h.

Referenced by ctb::IOBase::IOBase(), ctb::IOBase::PutBack(), ctb::SerialPort::Read(), ctb::GpibDevice::Read(), and ctb::IOBase::∼IOBase().

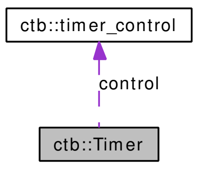
The documentation for this class was generated from the following files:

- serportx.h
- serportx.cpp

0.3.9 ctb::Timer Class Reference

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

Collaboration diagram for ctb::Timer:



0.3.9.1 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.

Definition at line 65 of file linux/timer.h.

Public Member Functions

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

Protected Attributes

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

0.3.9.2 Constructor & Destructor Documentation

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.

Definition at line 44 of file timer.cpp.

References control, ctb::timer_control::exitflag, ctb::timer_control::exitfnc, stopped, and ctb::timer_control::usecs.

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.

Definition at line 54 of file timer.cpp.

References stop(), and stopped.

0.3.9.3 Member Function Documentation

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.

Definition at line 63 of file timer.cpp.

References control, stopped, tid, and ctb::timer_fnc().

Referenced by ctb::IOBase::ReadUntilEOS(), and ctb::IOBase::Writev().

int ctb::Timer::stop ()

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

Definition at line 83 of file timer.cpp.

References control, ctb::timer_control::exitflag, stopped, and tid.

Referenced by ∼Timer().

0.3.9.4 Member Data Documentation

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.

Definition at line 73 of file linux/timer.h.

Referenced by start(), stop(), and Timer().

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.

Definition at line 80 of file linux/timer.h.

Referenced by start(), stop(), Timer(), and ∼Timer().

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

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

Definition at line 85 of file linux/timer.h.

Referenced by start(), and stop().

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

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

Definition at line 90 of file linux/timer.h.

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

- linux/timer.h
- timer.cpp

0.3.10 ctb::timer_control Struct Reference

`#include <timer.h>`

0.3.10.1 Detailed Description

A data struct, using from class timer.

Definition at line 24 of file linux/timer.h.

Public Attributes

- int ∗ exitflag
- void ∗(∗ exitfnc)(void ∗)
- unsigned int usecs

0.3.10.2 Member Data Documentation

int∗ ctb::timer_control::exitflag

covers the adress of the exitflag

Definition at line 33 of file linux/timer.h.

Referenced by ctb::Timer::stop(), ctb::Timer::Timer(), and ctb::timer_fnc().

void∗(∗ ctb::timer_control::exitfnc)(void ∗)

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

Referenced by ctb::Timer::Timer(), and ctb::timer_fnc().

unsigned int ctb::timer_control::usecs

under linux, we used usec internally

Definition at line 29 of file linux/timer.h.

Referenced by ctb::Timer::Timer(), and ctb::timer_fnc().

The documentation for this struct was generated from the following file:

- linux/timer.h

0.4 libctb File Documentation

0.4.1 fifo.h File Reference

0.4.1.1 Detailed Description

Definition in file fifo.h.

Namespaces

- namespace ctb

Classes

- class ctb::Fifo

0.4.2 gpib.h File Reference

0.4.2.1 Detailed Description

Definition in file gpib.h.

Namespaces

- namespace ctb

Classes

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

Enumerations

- enum ctb::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 }
- enum ctb::GpibTimeout {

  ctb::GpibTimeoutNone = 0, ctb::GpibTimeout10us, ctb::GpibTimeout30us, ctb::Gpib-Timeout100us,

  ctb::GpibTimeout300us, ctb::GpibTimeout1ms, ctb::GpibTimeout3ms, ctb::Gpib-Timeout10ms,

  ctb::GpibTimeout30ms, ctb::GpibTimeout100ms, ctb::GpibTimeout300ms, ctb::Gpib-Timeout1s,

  ctb::GpibTimeout3s, ctb::GpibTimeout10s, ctb::GpibTimeout30s, ctb::GpibTimeout100s,

  ctb::GpibTimeout300s, ctb::GpibTimeout1000s }

Variables

- const char * ctb::GPIB1
- const char * ctb::GPIB2

0.4.3 iobase.h File Reference

0.4.3.1 Detailed Description

Definition in file iobase.h.

Namespaces

- namespace ctb

Classes

- class ctb::IOBase

Enumerations

- enum { **CTB_COMMON** = 0x0000, **CTB_SERIAL** = 0x0100, **CTB_GPIB** = 0x0200, **CTB_-TIMEOUT_INFINITY** = 0xFFFFFFFF }
- enum ctb::IOBaseIoctls { ctb::CTB_RESET = CTB_COMMON }

0.4.4 portscan.h File Reference

0.4.4.1 Detailed Description

Definition in file portscan.h.

Namespaces

- namespace ctb

Functions

- bool ctb::GetAvailablePorts (std::vector< std::string > &result, bool checkInUse=true)

  returns all available COM ports as an array of strings.

0.4.5 serportx.h File Reference

0.4.5.1 Detailed Description

Definition in file serportx.h.

Namespaces

- namespace ctb

Classes

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

Defines

- #define SERIALPORT_NAME_LEN 32

Enumerations

- enum ctb::Parity {
  ctb::ParityNone, ctb::ParityOdd, ctb::ParityEven, ctb::ParityMark,
  ctb::ParitySpace }

  Defines the different modes of parity checking. Under Linux, the struct termios will be set to provide the wanted behaviour.

- enum ctb::SerialLineState {
  ctb::LinestateDcd = 0x040, ctb::LinestateCts = 0x020, ctb::LinestateDsr = 0x100, ctb::LinestateDtr = 0x002,
  ctb::LinestateRing = 0x080, ctb::LinestateRts = 0x004, ctb::LinestateNull = 0x000 }
- enum ctb::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::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::COM2
- const char ∗ ctb::COM20
- 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

0.4.5.2 Define Documentation

#define SERIALPORT_NAME_LEN 32

defines the maximum length of the os depending serial port names

Definition at line 28 of file serportx.h.

Index