libwxctb Reference Manual

0.13

Generated by Doxygen 1.4.7

Thu Jan 10 09:33:54 2008

Contents

1 ctb overview

2 libwxctb Hierarchical Index

2.1 libwxctb Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 libwxctb Class Index

3.1 libwxctb Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 libwxctb File Index

4.1 libwxctb File List

Here is a list of all documented files with brief descriptions:

5 libwxctb Class Documentation

5.1 fifo Class Reference

`#include <fifo.h>`

Public Member Functions

- fifo (size_t size)

  the constructor initialize a fifo with the given size.

- virtual ~fifo ()

  the destructor destroys all internal memory.

- virtual void clear ()

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

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

- virtual int get (char *ch)

  fetch the next available byte from the fifo.

- size_t items ()
  query the fifo for it's available bytes.
- virtual int put (char ch)
  put a character into the fifo.
- virtual int read (char *data, int count)
  read a given count of bytes out of the fifo.
- virtual int write (char *data, int count)
  write a given count of bytes into the fifo.

Protected Attributes

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

5.1.1 Detailed Description

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

5.1.2 Constructor & Destructor Documentation

5.1.2.1 fifo::fifo (size_t *size*)

the constructor initialize a fifo with the given size.

Parameters:

size size of the fifo

5.1.2.2 fifo::~fifo () [virtual]

the destructor destroys all internal memory.

5.1.3 Member Function Documentation

5.1.3.1 void 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.

5.1.3.2 int 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

5.1.3.3 size_t fifo::items ()

query the fifo for it's available bytes.

Returns:

count of readable bytes, storing in the fifo

5.1.3.4 int fifo::put (char *ch*) [virtual]

put a character into the fifo.

Parameters:

ch the character to put in

Returns:

1 if successful, 0 otherwise

5.1.3.5 int 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

5.1.3.6 int 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

5.1.4 Member Data Documentation

5.1.4.1 char* fifo::m_begin [protected]

the start of the internal fifo buffer

5.1.4.2 char* fifo::m_end [protected]

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

5.1.4.3 char* fifo::m_rdptr [protected]

the current read position

5.1.4.4 size_t fifo::m_size [protected]

the size of the fifo

5.1.4.5 char* fifo::m_wrptr [protected]

the current write position

5.2 timer Class Reference

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

`#include <timer.h>`

Public Member Functions

- timer (unsigned int msec, int *exitflag, void *(*exitfnc)(void *))
- ~timer ()
- int start ()
- int stop ()

Protected Attributes

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

5.2.1 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.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 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.

5.2.2.2 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.

5.2.3 Member Function Documentation

5.2.3.1 int 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.

5.2.3.2 int timer::stop ()

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

5.2.4 Member Data Documentation

5.2.4.1 timer_control 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.

5.2.4.2 int 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.

5.2.4.3 pthread_t timer::tid [protected]

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

5.2.4.4 unsigned int timer::timer_secs [protected]

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

5.3 timer_control Struct Reference

A data struct, using from class timer.

`#include <timer.h>`

Public Attributes

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

5.3.1 Detailed Description

A data struct, using from class timer.

5.3.2 Member Data Documentation

5.3.2.1 int* timer_control::exitflag

covers the adress of the exitflag

5.3.2.2 void*(* timer_control::exitfnc)(void *)

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

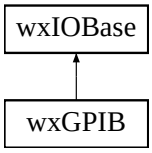
5.3.2.3 unsigned int timer_control::usecs

under linux, we used usec internally

5.4 wxGPIB Class Reference

`#include <gpib.h>`

Inheritance diagram for wxGPIB::



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 wxIOBase pointer.

- virtual const char * GetErrorDescription (int error)

  returns a more detail description of the given error number.

- virtual const char * GetErrorNotation (int error)

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

- virtual char * GetSettingsAsString ()

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

- int Ibrd (char *buf, size_t len)

  This is only for internal usage.

- int Ibwrt (char *buf, size_t len)

  This is only for internal usage.

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

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

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

Static Public Member Functions

- static int FindListeners (int board=0)

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

Protected Member Functions

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

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

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

Protected Attributes

- int m_board

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

- int m_hd

  the file descriptor of the connected gpib device

- int m_state

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

- int m_error
- int m_count
- wxGPIB_DCS m_dcs

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

5.4.1 Detailed Description

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

5.4.2 Member Function Documentation

5.4.2.1 const char* wxGPIB::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 wxIOBase pointer.

Returns:

name of the class.

Reimplemented from wxIOBase.

5.4.2.2 int wxGPIB::CloseDevice () [protected, virtual]

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

Returns:

zero on success, otherwise -1.

Implements wxIOBase.

5.4.2.3 int wxGPIB::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 wxGPIB_x 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...

5.4.2.4 virtual const char* wxGPIB::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

5.4.2.5 virtual const char* wxGPIB::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

5.4.2.6 const char * wxGPIB::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.

5.4.2.7 virtual char* wxGPIB::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'

5.4.2.8 int wxGPIB::Ibrd (char * *buf*, size_t *len*)

This is only for internal usage.

5.4.2.9 int wxGPIB::Ibwrt (char * *buf*, size_t *len*)

This is only for internal usage.

5.4.2.10 int wxGPIB::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 wxGPIBIoctls 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 wxIOBase.

5.4.2.11 int wxGPIB::IsOpen () [inline, virtual]

Returns the current state of the device.

Returns:

1 if device is valid and open, otherwise 0

Implements wxIOBase.

5.4.2.12 int wxGPIB::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 wxGPIB_DCS data struct.

Parameters:

devname the name of the GPIB device, wxGPIB1 means the first GPIB controller, wxGPIB2 the second (if available).

dcs untyped pointer of advanced device parameters,

See also:

struct wxGPIB_DCS (data struct for the gpib device)

Returns:

zero on success, otherwise -1

Implements wxIOBase.

5.4.2.13 int wxGPIB::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 wxIOBase.

5.4.2.14 int wxGPIB::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 wxIOBase.

5.4.3 Member Data Documentation

5.4.3.1 int wxGPIB::m_board [protected]

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

5.4.3.2 int wxGPIB::m_count [protected]

the count of data read or written

5.4.3.3 wxGPIB_DCS wxGPIB::m_dcs [protected]

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

5.4.3.4 int wxGPIB::m_error [protected]

the internal GPIB error number

5.4.3.5 int wxGPIB::m_hd [protected]

the file descriptor of the connected gpib device

5.4.3.6 int wxGPIB::m_state [protected]

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

5.5 wxGPIB_DCS Struct Reference

`#include <gpib.h>`

Public Member Functions

- ~wxGPIB_DCS ()
- wxGPIB_DCS ()

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

- char * GetSettings ()

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

Public Attributes

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

5.5.1 Detailed Description

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

5.5.2 Constructor & Destructor Documentation

5.5.2.1 wxGPIB_DCS::~wxGPIB_DCS () [inline]

to avoid memory leak warnings generated by swig

5.5.2.2 wxGPIB_DCS::wxGPIB_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!

5.5.3 Member Function Documentation

5.5.3.1 char * wxGPIB_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

5.5.4 Member Data Documentation

5.5.4.1 int wxGPIB_DCS::m_address1

primary address of GPIB device

5.5.4.2 int wxGPIB_DCS::m_address2

secondary address of GPIB device

5.5.4.3 char wxGPIB_DCS::m_buf[32]

buffer for internal use

5.5.4.4 unsigned char wxGPIB_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)

5.5.4.5 unsigned char wxGPIB_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).

5.5.4.6 bool wxGPIB_DCS::m_eot

EOT enable

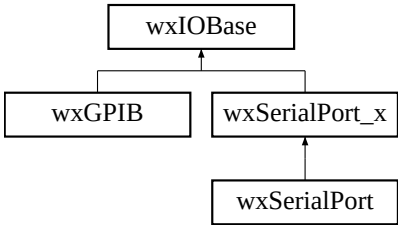
5.5.4.7 wxGPIB_Timeout wxGPIB_DCS::m_timeout

I/O timeout

5.6 wxIOBase Class Reference

`#include <iobase.h>`

Inheritance diagram for wxIOBase::



Public Member Functions

- wxIOBase ()
- virtual ~wxIOBase ()
- virtual const char * ClassName ()

  A little helper function to detect the class name.

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

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

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

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

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

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

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

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

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

Protected Types

- fifoSize = 256

  fifosize of the putback fifo

- 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.

5.6.1 Detailed Description

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

5.6.2 Member Enumeration Documentation

5.6.2.1 anonymous enum [protected]

Enumerator:

fifoSize fifosize of the putback fifo

5.6.3 Constructor & Destructor Documentation

5.6.3.1 wxIOBase::wxIOBase () [inline]

Default constructor

5.6.3.2 virtual wxIOBase::~wxIOBase () [inline, virtual]

Default destructor

5.6.4 Member Function Documentation

5.6.4.1 virtual const char* wxIOBase::ClassName () [inline, virtual]

A little helper function to detect the class name.

Returns:

the name of the class

Reimplemented in wxGPIB, and wxSerialPort_x.

5.6.4.2 int wxIOBase::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.

5.6.4.3 virtual int wxIOBase::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 wxGPIB, and wxSerialPort.

5.6.4.4 virtual int wxIOBase::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), wxIOBaseIoctls

args typeless parameter pointer for the command above.

Returns:

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

Reimplemented in wxGPIB, wxSerialPort_x, and wxSerialPort.

5.6.4.5 virtual int wxIOBase::IsOpen () [pure virtual]

Returns the current state of the device.

Returns:

1 if device is valid and open, otherwise 0

Implemented in wxGPIB, and wxSerialPort.

5.6.4.6 int wxIOBase::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).

5.6.4.7 virtual int wxIOBase::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 wxGPIB, and wxSerialPort.

5.6.4.8 int wxIOBase::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

5.6.4.9 virtual int wxIOBase::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 wxGPIB, and wxSerialPort.

5.6.4.10 int wxIOBase::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

5.6.4.11 int wxIOBase::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)

5.6.4.12 int wxIOBase::Readv (char * *buf*, size_t *len*, unsigned int *timeout_in_ms*)

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

Parameters:

buf starting address of the buffer

len count bytes, we want to read

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

Returns:

the number of data bytes successfully read

5.6.4.13 virtual int wxIOBase::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 wxGPIB, and wxSerialPort.

5.6.4.14 int wxIOBase::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)

5.6.4.15 int wxIOBase::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.

5.6.5 Member Data Documentation

5.6.5.1 fifo* wxIOBase::m_fifo [protected]

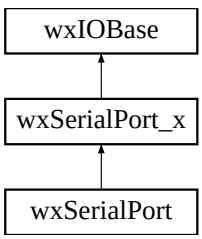
internal fifo (first in, first out queue) to put back already readed bytes into the reading stream. After put back a single byte or sequence of characters, you can read them again with the next Read call.

5.7 wxSerialPort Class Reference

the linux version

`#include <serport.h>`

Inheritance diagram for wxSerialPort::



Public Member Functions

- int ChangeLineState (wxSerialLineState flags)

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

- int ClrLineState (wxSerialLineState flags)

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

- int GetLineState ()

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

- int Ioctl (int cmd, void *args)

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

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

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

- int SetBaudRate (wxBaud baudrate)

  Set the baudrate.

- int SetLineState (wxSerialLineState flags)

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

- int Write (char *buf, size_t len)

Protected Member Functions

- speed_t AdaptBaudrate (wxBaud 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)

Protected Attributes

- int fd

  under Linux, the serial ports are normal file descriptor

- termios t save_t

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

- serial_icounter_struct save_info last_info

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

5.7.1 Detailed Description

the linux version

5.7.2 Member Function Documentation

5.7.2.1 speed_t wxSerialPort::AdaptBaudrate (wxBaud *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

5.7.2.2 int wxSerialPort::ChangeLineState (wxSerialLineState *flags*) [virtual]

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

Parameters:

flags valid line flags are wxSERIAL_LINESTATE_DSR and/or wxSERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implements wxSerialPort_x.

5.7.2.3 int wxSerialPort::CloseDevice () [protected, virtual]

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

Returns:

zero on success, otherwise -1.

Implements wxIOBase.

5.7.2.4 int wxSerialPort::ClrLineState (wxSerialLineState *flags*) [virtual]

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

Parameters:

flags valid line flags are wxSERIAL_LINESTATE_DSR and/or wxSERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implements wxSerialPort_x.

5.7.2.5 int wxSerialPort::GetLineState () [virtual]

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

Returns:

returns the appropriate bits on sucess, otherwise -1

Implements wxSerialPort_x.

5.7.2.6 int wxSerialPort::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 wxSerialPortIoctls 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 wxSerialPort_x.

5.7.2.7 int wxSerialPort::IsOpen () [virtual]

Returns the current state of the device.

Returns:

1 if device is valid and open, otherwise 0

Implements wxIOBase.

5.7.2.8 int wxSerialPort::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 wxIOBase.

5.7.2.9 int wxSerialPort::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 wxIOBase.

5.7.2.10 int wxSerialPort::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 wxSerialPort_x.

5.7.2.11 int wxSerialPort::SetBaudRate (wxBaud *baudrate*) [virtual]

Set the baudrate.

Parameters:

baudrate the new baudrate

Returns:

zero on success, -1 if an error occurs

Implements wxSerialPort_x.

5.7.2.12 int wxSerialPort::SetLineState (wxSerialLineState *flags*) [virtual]

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

Parameters:

flags valid line flags are wxSERIAL_LINESTATE_DSR and/or wxSERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implements wxSerialPort_x.

5.7.2.13 int wxSerialPort::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 wxIOBase.

5.7.3 Member Data Documentation

5.7.3.1 int wxSerialPort::fd [protected]

under Linux, the serial ports are normal file descriptor

5.7.3.2 struct serial_icounter_struct save_info wxSerialPort::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.

5.7.3.3 struct termios t wxSerialPort::save_t [protected]

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

5.8 wxSerialPort_DCS Struct Reference

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

Public Member Functions

- char * GetSettings ()
  returns the internal settings of the DCS as a human readable string like '8N1 115200'.

Public Attributes

- wxBaud baud
- wxParity parity
- unsigned char wordlen
- unsigned char stopbits
- bool rtscts
- bool xonxoff
- char buf [16]

5.8.1 Detailed Description

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

5.8.2 Member Function Documentation

5.8.2.1 char* wxSerialPort_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

5.8.3 Member Data Documentation

5.8.3.1 wxBaud wxSerialPort_DCS::baud

the baudrate

5.8.3.2 char wxSerialPort_DCS::buf[16]

buffer for internal use

5.8.3.3 wxParity wxSerialPort_DCS::parity

the parity

5.8.3.4 bool wxSerialPort_DCS::rtscts

rtscts flow control

5.8.3.5 unsigned char wxSerialPort_DCS::stopbits

count of stopbits

5.8.3.6 unsigned char wxSerialPort_DCS::wordlen

the wordlen

5.8.3.7 bool wxSerialPort_DCS::xonxoff

XON/XOFF flow control

5.9 wxSerialPort_EINFO Struct Reference

`#include <serportx.h>`

Public Attributes

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

5.9.1 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.

5.9.2 Member Data Documentation

5.9.2.1 int wxSerialPort_EINFO::brk

number of breaks

5.9.2.2 int wxSerialPort_EINFO::frame

number of framing errors

5.9.2.3 int wxSerialPort_EINFO::overrun

number of overrun errors

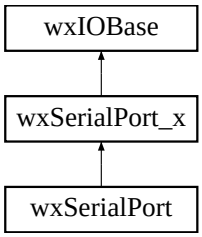
5.9.2.4 int wxSerialPort_EINFO::parity

number of parity errors

5.10 wxSerialPort_x Class Reference

`#include <serportx.h>`

Inheritance diagram for wxSerialPort_x::



Public Member Functions

- const char * ClassName ()

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

- virtual int ChangeLineState (wxSerialLineState flags)=0

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

- virtual int ClrLineState (wxSerialLineState flags)=0

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

- virtual int GetLineState ()=0

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

- virtual char * GetSettingsAsString ()

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

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

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

- virtual int SendBreak (int duration)=0

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

- virtual int SetBaudRate (wxBaud baudrate)=0

  Set the baudrate.

- virtual int SetLineState (wxSerialLineState flags)=0

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

Protected Attributes

- wxSerialPort_DCS m_dcs

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

- char m_devname [WXSERIALPORT_NAME_LEN]

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

5.10.1 Detailed Description

wxSerialPort_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.

5.10.2 Member Function Documentation

5.10.2.1 virtual int wxSerialPort_x::ChangeLineState (wxSerialLineState *flags*) [pure virtual]

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

Parameters:

flags valid line flags are wxSERIAL_LINESTATE_DSR and/or wxSERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implemented in wxSerialPort.

5.10.2.2 const char* wxSerialPort_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 wxIOBase pointer.

Returns:

name of the class.

Reimplemented from wxIOBase.

5.10.2.3 virtual int wxSerialPort_x::ClrLineState (wxSerialLineState *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 wxSERIAL_LINESTATE_DSR and/or wxSERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implemented in wxSerialPort.

5.10.2.4 virtual int wxSerialPort_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 wxSerialPort.

5.10.2.5 virtual char* wxSerialPort_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'

5.10.2.6 virtual int wxSerialPort_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 wxSerialPortIoctls 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 wxIOBase.

Reimplemented in wxSerialPort.

5.10.2.7 virtual int wxSerialPort_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 wxSerialPort.

5.10.2.8 virtual int wxSerialPort_x::SetBaudRate (wxBaud *baudrate*) [pure virtual]

Set the baudrate.

Parameters:

baudrate the new baudrate

Returns:

zero on success, -1 if an error occurs

Implemented in wxSerialPort.

5.10.2.9 virtual int wxSerialPort_x::SetLineState (wxSerialLineState *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 wxSERIAL_LINESTATE_DSR and/or wxSERIAL_LINESTATE_RTS

Returns:

zero on success, -1 if an error occurs

Implemented in wxSerialPort.

5.10.3 Member Data Documentation

5.10.3.1 wxSerialPort_DCS wxSerialPort_x::m_dcs [protected]

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

5.10.3.2 char wxSerialPort_x::m_devname[WXSERIALPORT_NAME_LEN] [protected]

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

6 libwxctb File Documentation

6.1 gpib.h File Reference

Classes

- struct wxGPIB_DCS
- class wxGPIB

Defines

- #define wxGPIB1 "gpib1"
- #define wxGPIB2 "gpib2"

Enumerations

- enum wxGPIB_Timeout {
  wxGPIB_TO_NONE = 0, wxGPIB_TO_10us, wxGPIB_TO_30us, wxGPIB_TO_100us,
  wxGPIB_TO_300us, wxGPIB_TO_1ms, wxGPIB_TO_3ms, wxGPIB_TO_10ms,
  wxGPIB_TO_30ms, wxGPIB_TO_100ms, wxGPIB_TO_300ms, wxGPIB_TO_1s,
  wxGPIB_TO_3s, wxGPIB_TO_10s, wxGPIB_TO_30s, wxGPIB_TO_100s,
  wxGPIB_TO_300s, wxGPIB_TO_1000s }
- enum wxGPIBIoctls {
  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 }

6.1.1 Detailed Description

6.1.2 Define Documentation

6.1.2.1 #define wxGPIB1 "gpib1"

defines the os specific name for the first gpib controller

6.1.2.2 #define wxGPIB2 "gpib2"

defines the os specific name for the second gpib controller

6.1.3 Enumeration Type Documentation

6.1.3.1 enum wxGPIB_Timeout

NI488.2 API defines the following valid timeouts.

Enumerator:

wxGPIB_TO_NONE no timeout (infinity)
wxGPIB_TO_10us 10 micro seconds
wxGPIB_TO_30us 30 micro seconds
wxGPIB_TO_100us 100 micro seconds
wxGPIB_TO_300us 300 micro seconds
wxGPIB_TO_1ms 1 milli second
wxGPIB_TO_3ms 3 milli seconds
wxGPIB_TO_10ms 10 milli seconds
wxGPIB_TO_30ms 30 milli seconds
wxGPIB_TO_100ms 0.1 seconds
wxGPIB_TO_300ms 0.3 seconds

wxGPIB_TO_1s 1 second

wxGPIB_TO_3s 3 seconds

wxGPIB_TO_10s 10 seconds

wxGPIB_TO_30s 30 seconds

wxGPIB_TO_100s 100 seconds

wxGPIB_TO_300s 300 seconds (5 minutes)

wxGPIB_TO_1000s 1000 seconds

6.1.3.2 enum wxGPIBIoctls

The following Ioctl calls are only valid for the wxGPIB 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).

6.2 serportx.h File Reference

Classes

- struct wxSerialPort_DCS
- struct wxSerialPort_EINFO
- class wxSerialPort_x

Defines

- #define WXSERIALPORT_NAME_LEN 32

Enumerations

- enum wxBaud {
  wxBAUD_150 = 150, wxBAUD_300 = 300, wxBAUD_600 = 600, wxBAUD_1200 = 1200,
  wxBAUD_2400 = 2400, wxBAUD_4800 = 4800, wxBAUD_9600 = 9600, wxBAUD_19200 = 19200,
  wxBAUD_38400 = 38400, wxBAUD_57600 = 57600, wxBAUD_115200 = 115200, wxBAUD_230400 = 230400,
  wxBAUD_460800 = 460800, wxBAUD_921600 = 921600 }
- enum wxParity {
  wxPARITY_NONE, wxPARITY_ODD, wxPARITY_EVEN, wxPARITY_MARK,
  wxPARITY_SPACE }
- enum wxSerialLineState {
  wxSERIAL_LINESTATE_DCD = 0x040, wxSERIAL_LINESTATE_CTS = 0x020, wxSERIAL_LINESTATE_DSR = 0x100, wxSERIAL_LINESTATE_DTR = 0x002,
  wxSERIAL_LINESTATE_RING = 0x080, wxSERIAL_LINESTATE_RTS = 0x004, wxSERIAL_LINESTATE_NULL = 0x000 }
- enum wxSerialPortIoctls {
  CTB_SER_GETEINFO = CTB_SERIAL, CTB_SER_GETBRK, CTB_SER_GETFRM, CTB_SER_GETOVR,
  CTB_SER_GETPAR, CTB_SER_GETINQUE }

6.2.1 Detailed Description

6.2.2 Define Documentation

6.2.2.1 #define WXSERIALPORT_NAME_LEN 32

defines the maximum length of the os depending serial port names

6.2.3 Enumeration Type Documentation

6.2.3.1 enum wxBaud

wxBaud covers the valid baudrates. Until wxBAUD_38400 (means a baudrate of 38400 baud) should be supported by every PC. In some circumstances, greater baudrates require a serial FIFO. But this should be built in, in the latest PCs.

Enumerator:

wxBAUD_150 150 baud

wxBAUD_300 300 baud

wxBAUD_600 600 baud

wxBAUD_1200 1200 baud

wxBAUD_2400 2400 baud

wxBAUD_4800 4800 baud
wxBAUD_9600 9600 baud
wxBAUD_19200 19200 baud
wxBAUD_38400 38400 baud
wxBAUD_57600 57600 baud
wxBAUD_115200 115200 baud
wxBAUD_230400 230400 baud
wxBAUD_460800 460800 baud
wxBAUD_921600 921600 baud

6.2.3.2 enum wxParity

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

Enumerator:

wxPARITY_NONE no parity check
wxPARITY_ODD odd parity check
wxPARITY_EVEN even parity check
wxPARITY_MARK mark (not implemented yet)
wxPARITY_SPACE space (not implemented yet)

6.2.3.3 enum wxSerialLineState

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:

wxSERIAL_LINESTATE_DCD Data Carrier Detect (read only)
wxSERIAL_LINESTATE_CTS Clear To Send (read only)
wxSERIAL_LINESTATE_DSR Data Set Ready (read only)
wxSERIAL_LINESTATE_DTR Data Terminal Ready (write only)
wxSERIAL_LINESTATE_RING Ring Detect (read only)
wxSERIAL_LINESTATE_RTS Request To Send (write only)
wxSERIAL_LINESTATE_NULL no active line state, use this for clear

6.2.3.4 enum wxSerialPortIoctls

The following Ioctl calls are only valid for the wxSerialPort 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 wxSerialPort_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.

Index