attys-comm

Generated by Doxygen 1.8.17

1 Hierarchical Index

1.1 Class Hierarchy

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

2 Class Index

2.1 Class List

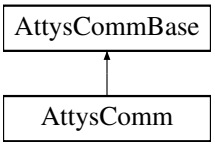
Here are the classes, structs, unions and interfaces with brief descriptions:

3 Class Documentation

3.1 AttysComm Class Reference

`#include <AttysComm.h>`

Inheritance diagram for AttysComm:



Public Member Functions

- AttysComm (void *_btAddr=NULL, int _btAddrLen=0)
- virtual void connect ()
- virtual void closeSocket ()
- virtual void run ()
- virtual void quit ()
- virtual void sendSyncCommand (const char *message, int waitForOK)
- virtual void sendInit ()
- virtual void start ()
- virtual void receptionTimeout ()
- unsigned char * getBluetoothBinaryAdress ()
- void getBluetoothAdressString (char *s)

Additional Inherited Members

3.1.1 Detailed Description

AttysComm contains all the neccessary comms to talk to the Attys on Linux, Windows and Mac.
AttysComm class contains the device specific definitions and implements the abstract classes of AttysCommBase. See AttysCommBase for the definitions there. Instances of this class are automatically created by AttysScan and the user can ignore definitions here. All relevant user functions are in AttysCommBase. Use this class only if you have a fixed bluetooth address (Linux/Win) or a fixed bluetooth device (Mac) and won't need to scan for a bluetooth device.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 AttysComm() `AttysComm::AttysComm ( void * _btAddr = NULL, int _btAddrLen = 0 ) [inline]`

Constructor: Win/Linux: takes the bluetooth device structure and its length as an argument. For Mac: just a pointer to the bluetooth device (needs typecast to *_btAddr) and provide no length.

3.1.3 Member Function Documentation

3.1.3.1 closeSocket() `virtual void AttysComm::closeSocket ( ) [virtual]`

Closes the socket safely.

Implements AttysCommBase.

3.1.3.2 connect() `virtual void AttysComm::connect ( ) [virtual]`

Connects to the Attys by opening the socket. Throws char* exception if it fails.

Implements AttysCommBase.

3.1.3.3 getBluetoothAdressString() `void AttysComm::getBluetoothAdressString ( char * s ) [virtual]`

returns the MAC address as a string.

Implements AttysCommBase.

3.1.3.4 getBluetoothBinaryAdress() `unsigned char* AttysComm::getBluetoothBinaryAdress ( ) [virtual]`

Returns an array of 14 bytes of the bluetooth address.

Implements AttysCommBase.

3.1.3.5 quit() `virtual void AttysComm::quit ( ) [inline], [virtual]`

Call this from the main activity to shut down the connection.

3.1.3.6 receptionTimeout() `virtual void AttysComm::receptionTimeout ( ) [virtual]`

Called from the watchdog after a timeout. Do not call this directly.

Implements AttysCommBase.

3.1.3.7 run() `virtual void AttysComm::run ( ) [virtual]`

Thread which does the data acquisition. Do not call directly.

Implements AttysCommBase.

3.1.3.8 sendInit() `virtual void AttysComm::sendInit ( ) [virtual]`

Sends the init sequence to the Attys. Do not use unless you know exactly what you are doing.

Implements AttysCommBase.

3.1.3.9 sendSyncCommand()

```
virtual void AttysComm::sendSyncCommand (
        const char * message,
        int waitForOK ) [virtual]
```

Sends a command to the Attys. Do not use unless you know exactly what you are doing.

Implements AttysCommBase.

3.1.3.10 start() `virtual void AttysComm::start ( ) [virtual]`

Starts the data acquisition by starting the main thread. and sending possibly init commands.

Reimplemented from AttysCommBase.

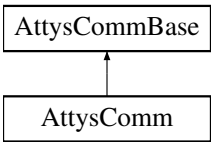
The documentation for this class was generated from the following file:

- AttysComm.h

3.2 AttysCommBase Class Reference

`#include <AttysCommBase.h>`

Inheritance diagram for AttysCommBase:



Public Member Functions

- AttysCommBase ()
- virtual ~AttysCommBase ()
- void setAdc_samplingrate_index (int idx)
- int getSamplingRateInHz ()
- int getAdc_samplingrate_index ()
- float getADCFullScaleRange (int channel)
- void setAdc0_gain_index (int idx)
- void setAdc1_gain_index (int idx)
- void setBiasCurrent (int currIndex)
- int getBiasCurrent ()
- void enableCurrents (int pos_ch1, int neg_ch1, int pos_ch2)
- float getAccelFullScaleRange ()
- void setAccel_full_scale_index (int idx)
- float getMagFullScaleRange ()
- int getIsCharging ()
- virtual void connect ()=0
- virtual void start ()
- virtual void closeSocket ()=0
- int hasActiveConnection ()
- sample_p getSampleFromBuffer ()
- int hasSampleAvailable ()
- void resetRingbuffer ()
- void registerCallback (AttysCommListener *f)
- void unregisterCallback ()
- void registerMessageCallback (AttysCommMessage *f)
- void unregisterMessageCallback ()
- void quit ()
- virtual unsigned char * getBluetoothBinaryAdress ()=0
- virtual void getBluetoothAdressString (char *s)=0
- char * getAttysName ()
- void setAttysName (char *s)

Static Public Member Functions

- static float phys2temperature (float volt)

Public Attributes

- const std::string CHANNEL_DESCRIPTION [NCHANNELS]
- const std::string CHANNEL_SHORT_DESCRIPTION [NCHANNELS]
- const std::string CHANNEL_UNITS [NCHANNELS]
- const int ADC_GAIN_FACTOR [7] = { 6, 1, 2, 3, 4, 8, 12 }
- const float ADC_REF = 2.42F
- const float oneG = 9.80665F
- const float ACCEL_FULL_SCALE [4] = { 2 ∗ oneG, 4 ∗ oneG, 8 ∗ oneG, 16 ∗ oneG }
- const float MAG_FULL_SCALE = 4800.0E-6F

Static Public Attributes

- static const int NCHANNELS = 10
- static const int nMem = 1000 ∗ 10
- static const int INDEX_Acceleration_X = 0
- static const int INDEX_Acceleration_Y = 1
- static const int INDEX_Acceleration_Z = 2
- static const int INDEX_Magnetic_field_X = 3
- static const int INDEX_Magnetic_field_Y = 4
- static const int INDEX_Magnetic_field_Z = 5
- static const int INDEX_Analogue_channel_1 = 6
- static const int INDEX_Analogue_channel_2 = 7
- static const int INDEX_GPIO0 = 8
- static const int INDEX_GPIO1 = 9
- static const int ADC_RATE_125HZ = 0
- static const int ADC_RATE_250HZ = 1
- static const int ADC_RATE_500HZ = 2
- static const int ADC_DEFAULT_RATE = ADC_RATE_250HZ
- static const int ADC_SAMPLINGRATE [4]
- static const int ADC_GAIN_6 = 0
- static const int ADC_GAIN_1 = 1
- static const int ADC_GAIN_2 = 2
- static const int ADC_GAIN_3 = 3
- static const int ADC_GAIN_4 = 4
- static const int ADC_GAIN_8 = 5
- static const int ADC_GAIN_12 = 6
- static const int ADC_CURRENT_6NA = 0
- static const int ADC_CURRENT_22NA = 1
- static const int ADC_CURRENT_6UA = 2
- static const int ADC_CURRENT_22UA = 3
- static const int ADC_MUX_NORMAL = 0
- static const int ADC_MUX_SHORT = 1
- static const int ADC_MUX_SUPPLY = 3
- static const int ADC_MUX_TEMPERATURE = 4
- static const int ADC_MUX_TEST_SIGNAL = 5
- static const int ADC_MUX_ECG_EINTHOVEN = 6
- static const int ACCEL_2G = 0
- static const int ACCEL_4G = 1
- static const int ACCEL_8G = 2
- static const int ACCEL_16G = 3
- static const int MESSAGE_CONNECTED = 0
- static const int MESSAGE_ERROR = 1
- static const int MESSAGE_TIMEOUT = 7
- static const int MESSAGE_RECONNECTED = 8
- static const int MESSAGE_RECEIVING_DATA = 9

3.2.1 Detailed Description

Platform independent definitions for the Attys

3.2.2 Constructor & Destructor Documentation

3.2.2.1 AttysCommBase() `AttysCommBase::AttysCommBase ( )`

Constructor which is overloaded by AttysComm.

3.2.2.2 ~AttysCommBase() `virtual AttysCommBase::~AttysCommBase ( ) [virtual]`

Destructor which releases memory and closes any open connection.

3.2.3 Member Function Documentation

3.2.3.1 closeSocket() `virtual void AttysCommBase::closeSocket ( ) [pure virtual]`

Closes the socket safely.

Implemented in AttysComm.

3.2.3.2 connect() `virtual void AttysCommBase::connect ( ) [pure virtual]`

Connects to the Attys by opening the socket. Throws char* exception if it fails.

Implemented in AttysComm.

3.2.3.3 enableCurrents()

```
void AttysCommBase::enableCurrents (
        int pos_ch1,
        int neg_ch1,
        int pos_ch2 ) [inline]
```

Switches bias currents on

3.2.3.4 getAccelFullScaleRange() `float AttysCommBase::getAccelFullScaleRange ( ) [inline]`

Returns the accelerometer current full scale reading in m/s^2.

3.2.3.5 getAdc_samplingrate_index() `int AttysCommBase::getAdc_samplingrate_index ( ) [inline]`

Gets the sampling rate in form for the index.

3.2.3.6 getADCFullScaleRange() `float AttysCommBase::getADCFullScaleRange ( int channel ) [inline]`

Gets the ADC full range. This depends on the gain setting of the ADC.

3.2.3.7 getAttysName() `char* AttysCommBase::getAttysName ( ) [inline]`

Returns the name of the Attys

3.2.3.8 getBiasCurrent() `int AttysCommBase::getBiasCurrent ( ) [inline]`

Gets the bias current as in index.

3.2.3.9 getBluetoothAdressString() `virtual void AttysCommBase::getBluetoothAdressString ( char * s ) [pure virtual]`

returns the MAC address as a string.

Implemented in AttysComm.

3.2.3.10 getBluetoothBinaryAdress() `virtual unsigned char* AttysCommBase::getBluetoothBinaryAdress ( ) [pure virtual]`

Returns an array of 14 bytes of the bluetooth address.

Implemented in AttysComm.

3.2.3.11 getIsCharging() `int AttysCommBase::getIsCharging ( ) [inline]`

Charging indicator. Returns one if charging.

3.2.3.12 getMagFullScaleRange() `float AttysCommBase::getMagFullScaleRange ( ) [inline]`

Returns the full scale magnetometer in Tesla.

3.2.3.13 getSampleFromBuffer() `sample_p AttysCommBase::getSampleFromBuffer ( )`

Gets a sample from the ringbuffer. This is a float* array of all channels.

3.2.3.14 getSamplingRateInHz() `int AttysCommBase::getSamplingRateInHz ( ) [inline]`

Gets the sampling rate in Hz (not index number).

3.2.3.15 hasActiveConnection() `int AttysCommBase::hasActiveConnection ( ) [inline]`

Returns one if the connection is active.

3.2.3.16 hasSampleAvailable() `int AttysCommBase::hasSampleAvailable ( ) [inline]`

Is set to one if samples are available in the ringbuffer.

3.2.3.17 phys2temperature() `static float AttysCommBase::phys2temperature ( float volt ) [inline], [static]`

Temperature

3.2.3.18 quit() `void AttysCommBase::quit ( )`

Call this from the main activity to shut down the connection.

3.2.3.19 registerCallback() `void AttysCommBase::registerCallback ( AttysCommListener * f ) [inline]`

Register a realtime callback function which is called whenever a sample has arrived. AttysCommListener is an abstract class which needs to implement hasSample().

3.2.3.20 registerMessageCallback() `void AttysCommBase::registerMessageCallback ( AttysCommMessage * f ) [inline]`

Callback which is called whenever a special error/event has occurred.

3.2.3.21 resetRingbuffer() `void AttysCommBase::resetRingbuffer ( ) [inline]`

Resets the ringbuffer to zero content.

3.2.3.22 setAccel_full_scale_index() `void AttysCommBase::setAccel_full_scale_index ( int idx ) [inline]`

Sets the accelerometer full scale range using the index.

3.2.3.23 setAdc0_gain_index() `void AttysCommBase::setAdc0_gain_index ( int idx ) [inline]`

Gets the gain index for ADC1.

3.2.3.24 setAdc1_gain_index() `void AttysCommBase::setAdc1_gain_index ( int idx ) [inline]`

Gets the gain index for ADC2.

3.2.3.25 setAdc_samplingrate_index() `void AttysCommBase::setAdc_samplingrate_index ( int idx ) [inline]`

Sets the sampling rate using the sampling rate index numbers.

3.2.3.26 setAttysName() `void AttysCommBase::setAttysName ( char * s ) [inline]`

Sets the name of the Attys

3.2.3.27 setBiasCurrent() `void AttysCommBase::setBiasCurrent ( int currIndex ) [inline]`

Sets the bias current which can be switched on.

3.2.3.28 start() `virtual void AttysCommBase::start ( ) [inline], [virtual]`

Starts the data acquisition by starting the main thread. and sending possibly init commands.

Reimplemented in AttysComm.

3.2.3.29 unregisterCallback() `void AttysCommBase::unregisterCallback ( ) [inline]`

Unregister the realtime sample callback.

3.2.3.30 unregisterMessageCallback() `void AttysCommBase::unregisterMessageCallback ( ) [inline]`

Unregister the error/event callback.

3.2.4 Member Data Documentation

3.2.4.1 ACCEL_16G `const int AttysCommBase::ACCEL_16G = 3 [static]`

Setting full scale range of the accelerometer to 16G.

3.2.4.2 ACCEL_2G `const int AttysCommBase::ACCEL_2G = 0 [static]`

Setting full scale range of the accelerometer to 2G.

3.2.4.3 ACCEL_4G

```
const int AttysCommBase::ACCEL_4G = 1  [static]
```

Setting full scale range of the accelerometer to 4G.

3.2.4.4 ACCEL_8G

```
const int AttysCommBase::ACCEL_8G = 2  [static]
```

Setting full scale range of the accelerometer to 8G.

3.2.4.5 ACCEL_FULL_SCALE

```
const float AttysCommBase::ACCEL_FULL_SCALE[4] = { 2 * oneG, 4 * oneG, 8 * oneG, 16 * oneG }
```

Mapping of the index to the full scale accelerations.

3.2.4.6 ADC_CURRENT_22NA

```
const int AttysCommBase::ADC_CURRENT_22NA = 1  [static]
```

Bias current of 22nA.

3.2.4.7 ADC_CURRENT_22UA

```
const int AttysCommBase::ADC_CURRENT_22UA = 3  [static]
```

Bias current of 22uA.

3.2.4.8 ADC_CURRENT_6NA

```
const int AttysCommBase::ADC_CURRENT_6NA = 0  [static]
```

Bias current of 6nA.

3.2.4.9 ADC_CURRENT_6UA

```
const int AttysCommBase::ADC_CURRENT_6UA = 2  [static]
```

Bias current of 6uA.

3.2.4.10 ADC_DEFAULT_RATE

```
const int AttysCommBase::ADC_DEFAULT_RATE = ADC_RATE_250HZ  [static]
```

Constant defining the default sampling rate (250Hz).

3.2.4.11 ADC_GAIN_1

```
const int AttysCommBase::ADC_GAIN_1 = 1  [static]
```

Gain index setting it to gain 6.

3.2.4.12 ADC_GAIN_12

```
const int AttysCommBase::ADC_GAIN_12 = 6  [static]
```

Gain index setting it to gain 6.

3.2.4.13 ADC_GAIN_2

```
const int AttysCommBase::ADC_GAIN_2 = 2  [static]
```

Gain index setting it to gain 2.

3.2.4.14 ADC_GAIN_3 `const int AttysCommBase::ADC_GAIN_3 = 3 [static]`

Gain index setting it to gain 3.

3.2.4.15 ADC_GAIN_4 `const int AttysCommBase::ADC_GAIN_4 = 4 [static]`

Gain index setting it to gain 4.

3.2.4.16 ADC_GAIN_6 `const int AttysCommBase::ADC_GAIN_6 = 0 [static]`

Gain index setting it to gain 6.

3.2.4.17 ADC_GAIN_8 `const int AttysCommBase::ADC_GAIN_8 = 5 [static]`

Gain index setting it to gain 5.

3.2.4.18 ADC_GAIN_FACTOR `const int AttysCommBase::ADC_GAIN_FACTOR[7] = { 6, 1, 2, 3, 4, 8, 12 }`

Mmapping between index and actual gain.

3.2.4.19 ADC_MUX_ECG_EINTHOVEN `const int AttysCommBase::ADC_MUX_ECG_EINTHOVEN = 6 [static]`

Muliplexer routing: both positive ADC inputs are connected together.

3.2.4.20 ADC_MUX_NORMAL `const int AttysCommBase::ADC_MUX_NORMAL = 0 [static]`

Muliplexer routing is normal: ADC1 and ADC2 are connected to the sigma/delta.

3.2.4.21 ADC_MUX_SHORT `const int AttysCommBase::ADC_MUX_SHORT = 1 [static]`

Muliplexer routing: inputs are short circuited.

3.2.4.22 ADC_MUX_SUPPLY `const int AttysCommBase::ADC_MUX_SUPPLY = 3 [static]`

Muliplexer routing: inputs are connected to power supply.

3.2.4.23 ADC_MUX_TEMPERATURE `const int AttysCommBase::ADC_MUX_TEMPERATURE = 4 [static]`

Muliplexer routing: ADC measures internal temperature.

3.2.4.24 ADC_MUX_TEST_SIGNAL `const int AttysCommBase::ADC_MUX_TEST_SIGNAL = 5 [static]`

Muliplexer routing: ADC measures test signal.

3.2.4.25 ADC_RATE_125HZ `const int AttysCommBase::ADC_RATE_125HZ = 0 [static]`

Constant defining sampling rate of 125Hz.

3.2.4.26 ADC_RATE_250HZ `const int AttysCommBase::ADC_RATE_250HZ = 1 [static]`

Constant defining sampling rate of 250Hz.

3.2.4.27 ADC_RATE_500HZ `const int AttysCommBase::ADC_RATE_500HZ = 2 [static]`

Constant defining sampling rate of 500Hz (Attys2 only).

3.2.4.28 ADC_REF `const float AttysCommBase::ADC_REF = 2.42F`

The voltage reference of the ADC in volts.

3.2.4.29 ADC_SAMPLINGRATE `const int AttysCommBase::ADC_SAMPLINGRATE[4] [static]`

Array of the sampling rates mapping the index to the actual sampling rate.

3.2.4.30 CHANNEL_DESCRIPTION `const std::string AttysCommBase::CHANNEL_DESCRIPTION[NCHANNELS]`

Initial value:

```
= {
                "Acceleration X",
                "Acceleration Y",
                "Acceleration Z",
                "Magnetic field X",
                "Magnetic field Y",
                "Magnetic field Z",
                "Analogue channel 1",
                "Analogue channel 2"
                "DIN channel 0",
                "DIN channel 1",
                "Charging status"
        }
```

Long descriptions of the channels.

3.2.4.31 CHANNEL_SHORT_DESCRIPTION `const std::string AttysCommBase::CHANNEL_SHORT_DESCR↩ IPTION[NCHANNELS]`

Initial value:

```
= {
                "Acc X",
                "Acc Y",
                "Acc Z",
                "Mag X",
                "Mag Y",
                "Mag Z",
                "ADC 1",
                "ADC 2",
                "DIN 0",
                "DIN 1",
        }
```

Short descriptions of the channels.

3.2.4.32 CHANNEL_UNITS `const std::string AttysCommBase::CHANNEL_UNITS[NCHANNELS]`

Initial value:

```
= {
                "m/s^2",
                "m/s^2",
                "m/s^2",
                "T",
                "T",
                "T",
                "V",
                "V",
                " ",
                " ",
        }
```

Units of the channels.

3.2.4.33 INDEX_Acceleration_X `const int AttysCommBase::INDEX_Acceleration_X = 0 [static]`

Channel index for X Acceleration.

3.2.4.34 INDEX_Acceleration_Y `const int AttysCommBase::INDEX_Acceleration_Y = 1 [static]`

Channel index for Y Acceleration.

3.2.4.35 INDEX_Acceleration_Z `const int AttysCommBase::INDEX_Acceleration_Z = 2 [static]`

Channel index for Z Acceleration.

3.2.4.36 INDEX_Analogue_channel_1 `const int AttysCommBase::INDEX_Analogue_channel_1 = 6 [static]`

Index of analogue channel 1.

3.2.4.37 INDEX_Analogue_channel_2 `const int AttysCommBase::INDEX_Analogue_channel_2 = 7 [static]`

Index of analogue channel 2.

3.2.4.38 INDEX_GPIO0 `const int AttysCommBase::INDEX_GPIO0 = 8 [static]`

Index of the internal GPIO pin 1.

3.2.4.39 INDEX_GPIO1 `const int AttysCommBase::INDEX_GPIO1 = 9 [static]`

Index of the internal GPIO pin 2.

3.2.4.40 INDEX_Magnetic_field_X `const int AttysCommBase::INDEX_Magnetic_field_X = 3 [static]`

Magnetic field in X direction.

3.2.4.41 INDEX_Magnetic_field_Y

```
const int AttysCommBase::INDEX_Magnetic_field_Y = 4  [static]
```

Magnetic field in Y direction.

3.2.4.42 INDEX_Magnetic_field_Z

```
const int AttysCommBase::INDEX_Magnetic_field_Z = 5  [static]
```

Magnetic field in Z direction.

3.2.4.43 MAG_FULL_SCALE

```
const float AttysCommBase::MAG_FULL_SCALE = 4800.0E-6F
```

Full scale range of the magnetometer in Tesla.

3.2.4.44 MESSAGE_CONNECTED

```
const int AttysCommBase::MESSAGE_CONNECTED = 0  [static]
```

Message callback: Connected.

3.2.4.45 MESSAGE_ERROR

```
const int AttysCommBase::MESSAGE_ERROR = 1  [static]
```

Message callback: Generic error.

3.2.4.46 MESSAGE_RECEIVING_DATA

```
const int AttysCommBase::MESSAGE_RECEIVING_DATA = 9  [static]
```

Message callback: Receiving data. One off once the connection has been set up.

3.2.4.47 MESSAGE_RECONNECTED

```
const int AttysCommBase::MESSAGE_RECONNECTED = 8  [static]
```

Message callback: Managed to reconnect.

3.2.4.48 MESSAGE_TIMEOUT

```
const int AttysCommBase::MESSAGE_TIMEOUT = 7  [static]
```

Message callback: Reception timeout detected by the watchdog.

3.2.4.49 NCHANNELS

```
const int AttysCommBase::NCHANNELS = 10  [static]
```

Total number of channels per samples.

3.2.4.50 nMem

```
const int AttysCommBase::nMem = 1000 * 10  [static]
```

Number of entries in the ringbuffer. Buffer for 10secs at 1kHz.

3.2.4.51 oneG

```
const float AttysCommBase::oneG = 9.80665F
```

One g in m/s^2.

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

- AttysCommBase.h

3.3 AttysCommListener Struct Reference

`#include <AttysCommBase.h>`

Public Member Functions

- virtual void hasSample (double, sample_p)=0

3.3.1 Detailed Description

Callback after a sample has arrived. The main class can for example inherit class and implement hasSample.

3.3.2 Member Function Documentation

3.3.2.1 hasSample()

```
virtual void AttysCommListener::hasSample (
        double ,
        sample_p  )  [pure virtual]
```

Provides the timestamp and an array of all channels. This is an abstract method and needs to be overloaded with a real method doing the work.

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

- AttysCommBase.h

3.4 AttysCommMessage Struct Reference

`#include <AttysCommBase.h>`

Public Member Functions

- virtual void hasMessage (int, const char *)=0

3.4.1 Detailed Description

Callback after an error has occurred. This callback is in particular useful after a broken connection has been re-established.

3.4.2 Member Function Documentation

3.4.2.1 hasMessage()

```
virtual void AttysCommMessage::hasMessage (
        int ,
        const char *  )  [pure virtual]
```

Provides the error number and a text message about the error.

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

- AttysCommBase.h

3.5 AttysScan Class Reference

```
#include <AttysScan.h>
```

Public Member Functions

- int scan (int maxAttys=1)
- void registerCallback (AttysScanListener *f)
- void unregisterCallback ()
- AttysComm * getAttysComm (int i)
- char * getAttysName (int i)
- int getNAttysDevices ()

Static Public Attributes

- static const int SCAN_CONNECTED = 0
- static const int SCAN_SEARCHING = 1
- static const int SCAN_NODEV = 2
- static const int SCAN_SOCKETERR = 3
- static const int SCAN_CONNECTING = 4
- static const int SCAN_CONNECTERR = 5
- static const int MAX_ATTYS_DEVS = 4

3.5.1 Detailed Description

Scans for Attys and creates instances of AttysComm for every detected/paired Attys. There is no need to create instances of AttysComm yourself. This is done by this class automatically.

3.5.2 Member Function Documentation

3.5.2.1 getAttysComm()

```
AttysComm* AttysScan::getAttysComm (
        int i )  [inline]
```

Obtains the pointer to a valid AttysComm class which has been successfully detected while scanning.

3.5.2.2 getAttysName() char* AttysScan::getAttysName (int *i*) [inline]

Gets the Attys name as reported by the bluetooth manager

3.5.2.3 getNAttysDevices() int AttysScan::getNAttysDevices ( ) [inline]

Returns the number of Attys devices

3.5.2.4 registerCallback() void AttysScan::registerCallback (AttysScanListener * *f*) [inline]

Register callback which reports the scanning progress for example for a splash screen.

3.5.2.5 scan() int AttysScan::scan (int *maxAttys = 1*)

Scans for the specified number of devices and connects to them. By default only for one Attys. returns 0 on success

3.5.2.6 unregisterCallback() void AttysScan::unregisterCallback ( ) [inline]

Unregisters the callback

3.5.3 Member Data Documentation

3.5.3.1 MAX_ATTYS_DEVS const int AttysScan::MAX_ATTYS_DEVS = 4 [static]

Max number of Attys Devices

3.5.3.2 SCAN_CONNECTED const int AttysScan::SCAN_CONNECTED = 0 [static]

Message index that the connection to an attys has been successful.

3.5.3.3 SCAN_CONNECTERR const int AttysScan::SCAN_CONNECTERR = 5 [static]

Connection error during scanning

3.5.3.4 SCAN_CONNECTING const int AttysScan::SCAN_CONNECTING = 4 [static]

In the process of connecting

3.5.3.5 SCAN_NODEV `const int AttysScan::SCAN_NODEV = 2 [static]`

Message index that no Attys has been detected

3.5.3.6 SCAN_SEARCHING `const int AttysScan::SCAN_SEARCHING = 1 [static]`

Message index that AttysScan is actively scanning

3.5.3.7 SCAN_SOCKETERR `const int AttysScan::SCAN_SOCKETERR = 3 [static]`

Message that the socket could not be opened

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

- AttysScan.h

3.6 AttysScanListener Struct Reference

`#include <AttysScan.h>`

3.6.1 Detailed Description

Callback which reports the status of the scanner

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

- AttysScan.h

Index