attys-comm

Generated by Doxygen 1.8.13

Contents

1 AttysCOMM

The C++ & Python API for the Attys bluetooth data acquisition board: http://www.attys.tech

The library is cross platform: It's for Linux, Windows and Mac.

Installation instructions

Linux

```
cmake .
make
sudo make install
```

This will generate: a dynamic library libattyscomm∗.so and a static one called libattyscomm_static.a.

Windows:

Under windows only the static library is generated which should be used for your code development.

```
cmake -G "Visual Studio 15 2017 Win64" .
```

and then start Visual C++ and compile it.

MacOS

For pure commandline install:

```
cmake .
make
make install
```

This will generate: a dynamic library libattyscomm.∗dylib and a static one called libattyscomm_static.a.

If you want to debug/develop the library in Xcode:

```
cmake -G Xcode
```

Usage

A small test program is in the `examples` directory which scans for an Attys and then prints the incoming data to stdout. Type `cmake .`, `make` and then `./attystest` to run it.

Here is a step by guide how to code it:

1. scan for Attys

   ```
   int ret = attysScan.scan();
   ```

2. Check the number of Attys detected

   ```
   attysScan.getNAttysDevices()
   ```

3. If devices have been detected you can get them via `getAttysComm(0,1,2,etc)`.
4. Set the parameters, for example:

   ```
   attysScan.getAttysComm(0)->setAdc_samplingrate_index(AttysComm::ADC_RATE_250HZ);
   ```

5. Register a callback (optional)

   ```
   attysCallback = new AttysCallback(this);
   attysScan.getAttysComm(0)->registerCallback(attysCallback);
   ```

   You need to overload the virtual function of the callback in your program.

6. Start data acquisition

   ```
   attysScan.getAttysComm(0)->start();
   ```

   If you have registered a callback then it's all set!

If you have no callback registered then you need to retreive the samples from the ringbuffer in steps 7.-9.:

1. Check if ringbuffer contains data and wait till true

   ```
   attysScan.getAttysComm(n)->hasSampleAvilabale();
   ```

2. Get samples from buffer

   ```
   float* values = attysScan.getAttysComm(n)->getSampleFromBuffer();
   ```

3. go back to 7)
4. Ending the program:

   ```
   attysScan.getAttysComm(n)->quit();
   ```

Python (SWIG)

This libary is fast and multi threaded. It performs the data acquisition in the background while python can then do the postprocessing.

Pre-compiled packages for Linux, Windows and MacOS are available.

Linux

Python package (pip):

Make sure you have the bluetooth development libraries:

```
sudo apt-get install libbluetooth-dev
```

and then install with:

```
pip3 install pyattyscomm
```

From source

You need to have swig-3.x installed. Then run:

```
cmake .
make
make install
./setup.py install
```

and then you can load the module `pyattyscomm` system-wide!

Windows

Python package (pip):

In the python console type:

```
pip install pyattyscomm
```

From source

Install `swig` and re-run the C++ installation. Make sure to select "Release" in Visual Studio as python is usually not installed with its debug libraries. After compilation you get:

- `Release\_pyattyscomm.exp`
- `Release\_pyattyscomm.pyd`
- `pyattyscomm.py`

Install them with:

```
python setup.py install
```

MacOS

Python package (pip):

```
pip3 install pyattyscomm
```

From source

You need to have swig-3.x installed for homebrew. Then run:

```
cmake .
make
make install
./setup.py install
```

and then you can load the module `pyattyscomm` system-wide!

How to use

The python API is identical to the C++ one. All the definitions are in AttysComm.h and AttysScan.h.

Here is an example:

```
# load the module
import pyattyscomm

# Gets the AttysScan class which scans for Attys via bluetooth
s = pyattyscomm.AttysScan()

# Scan for Attys
s.scan()

# get the 1st Attys
c = s.getAttysComm(0)

# if an attys has been found c points to it. Otherwise it's None.

# Start data acquisition in the background
c.start()

# Now we just read data at our convenience in a loop or timer or thread

while (not c.hasSampleAvilabale()):
        # do something else or nothing
        a = a + 1
    # getting a sample
    sample = c.getSampleFromBuffer()

    # do something with the sample
    print(sample)
```

Demo programs

There are demo programs which show you how to read/plot data with pyattyscomm:

```
readdata_demo.py
realtime_plot_demo.py
realtime_two_channel_plot.py
```

Enjoy!

http://www.attys.tech

2 Hierarchical Index

2.1 Class Hierarchy

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

3 Class Index

3.1 Class List

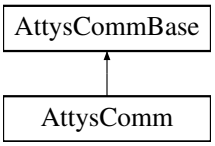
Here are the classes, structs, unions and interfaces with brief descriptions:

4 Class Documentation

4.1 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 start ()
- unsigned char * getBluetoothBinaryAdress ()
- void getBluetoothAdressString (char *s)

Additional Inherited Members

4.1.1 Detailed Description

AttysComm contains all the neccessary comms to talk to the Attys on both Linux and Windows.

1) Instantiate the class AttyScan and do a scan It finds all paired Attys and creates separate AttysComm classes 2) These classes are in in the array attysComm in AttysScan and the number of them in nAttysDevices. 3) All attys↩ Comm are Threads so just start the data acquisition with start(), for example attysComm[0]->start() for the 1st Attys 4) Get the data either via the RingBuffer functions or register a callback to get the data as it arrives. AttysComm class which contains the device specific definitions and implements the abstract classes of AttysCommBase. See AttysCommBase for the definitions there.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 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 device.

4.1.3 Member Function Documentation

4.1.3.1 closeSocket()

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

closes socket safely

Implements AttysCommBase.

4.1.3.2 connect()

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

connects to the Attys by opening the socket throws exception if it fails.

Implements AttysCommBase.

4.1.3.3 getBluetoothAdressString()

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

returns the Mac address as a string

Implements AttysCommBase.

4.1.3.4 getBluetoothBinaryAdress()

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

returns an array of 14 bytes of the bluetooth address

Implements AttysCommBase.

4.1.3.5 start()

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

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

Reimplemented from AttysCommBase.

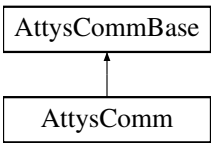
The documentation for this class was generated from the following file:

- AttysComm.h

4.2 AttysCommBase Class Reference

```
#include <AttysCommBase.h>
```

Inheritance diagram for AttysCommBase:



Public Member Functions

- 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

Static Public Member Functions

- static float phys2temperature (float volt)

Public Attributes

- const std::string CHANNEL_DESCRIPTION [NCHANNELS]
- const std::string CHANNEL_SHORT_DESCRIPTION [NCHANNELS]
- std::string const CHANNEL_UNITS [NCHANNELS]
- const int ADC_SAMPLINGRATE [4] = { 125, 250, 500, 1000 }
- 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_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

4.2.1 Detailed Description

Platform independent definitions for the Attys

4.2.2 Member Function Documentation

4.2.2.1 closeSocket()

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

closes socket safely

Implemented in AttysComm.

4.2.2.2 connect()

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

connects to the Attys by opening the socket throws exception if it fails.

Implemented in AttysComm.

4.2.2.3 enableCurrents()

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

Switches bias currents on

4.2.2.4 getAccelFullScaleRange()

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

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

4.2.2.5 getAdc_samplingrate_index()

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

Gets the sampling rate in form for the index.

4.2.2.6 getADCFullScaleRange()

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

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

4.2.2.7 getBiasCurrent()

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

Gets the bias current as in index.

4.2.2.8 getBluetoothAdressString()

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

returns the Mac address as a string

Implemented in AttysComm.

4.2.2.9 getBluetoothBinaryAdress()

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

returns an array of 14 bytes of the bluetooth address

Implemented in AttysComm.

4.2.2.10 getIsCharging()

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

Charging indicator. Returns one if charging.

4.2.2.11 getMagFullScaleRange()

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

Returns the full scale magnetometer in Tesla

4.2.2.12 getSampleFromBuffer()

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

Gets a sample from the ringbuffer. This is a C array of all samples.

4.2.2.13 getSamplingRateInHz()

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

Gets the sampling rate in Hz (not index number)

4.2.2.14 hasActiveConnection()

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

Returns 1 if the connection is active.

4.2.2.15 hasSampleAvailable()

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

Is one if samples are available in the ringbuffer

4.2.2.16 phys2temperature()

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

Temperature

4.2.2.17 quit()

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

Call this from the main activity to shutdown the connection

4.2.2.18 registerCallback()

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

Realtime callback function which is called whenever a sample has arrived. Implemented as an interface.

4.2.2.19 registerMessageCallback()

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

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

4.2.2.20 resetRingbuffer()

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

Resets the ringbuffer to zero content

4.2.2.21 setAccel_full_scale_index()

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

Sets the accelerometer full scale range using the index.

4.2.2.22 setAdc0_gain_index()

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

Gets the gain index for ADC1

4.2.2.23 setAdc1_gain_index()

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

Gets the gain index for ADC2

4.2.2.24 setAdc_samplingrate_index()

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

Sets the sampling rate using the sampling rate index numbers.

4.2.2.25 setBiasCurrent()

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

Sets the bias current which can be switched on.

4.2.2.26 start()

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

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

Reimplemented in AttysComm.

4.2.2.27 unregisterCallback()

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

Unregister the sample callback

4.2.2.28 unregisterMessageCallback()

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

Unregister the message callback.

4.2.3 Member Data Documentation

4.2.3.1 ACCEL_16G

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

Setting full scale range of the accelerometer to 16G

4.2.3.2 ACCEL_2G

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

Setting full scale range of the accelerometer to 2G

4.2.3.3 ACCEL_4G

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

Setting full scale range of the accelerometer to 4G

4.2.3.4 ACCEL_8G

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

Setting full scale range of the accelerometer to 8G

4.2.3.5 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

4.2.3.6 ADC_CURRENT_22NA

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

Bias current of 22nA

4.2.3.7 ADC_CURRENT_22UA

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

Bias current of 22uA

4.2.3.8 ADC_CURRENT_6NA

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

Bias current of 6nA

4.2.3.9 ADC_CURRENT_6UA

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

Bias current of 6uA

4.2.3.10 ADC_DEFAULT_RATE

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

Constant defining the default sampling rate (250Hz)

4.2.3.11 ADC_GAIN_1

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

Gain index setting it to gain 6.

4.2.3.12 ADC_GAIN_12

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

Gain index setting it to gain 6.

4.2.3.13 ADC_GAIN_2

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

Gain index setting it to gain 2.

4.2.3.14 ADC_GAIN_3

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

Gain index setting it to gain 3.

4.2.3.15 ADC_GAIN_4

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

Gain index setting it to gain 4.

4.2.3.16 ADC_GAIN_6

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

Gain index setting it to gain 6.

4.2.3.17 ADC_GAIN_8

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

Gain index setting it to gain 5.

4.2.3.18 ADC_GAIN_FACTOR

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

Mmapping between index and actual gain.

4.2.3.19 ADC_MUX_ECG_EINTHOVEN

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

Muliplexer routing: both positive ADC inputs are connected together

4.2.3.20 ADC_MUX_NORMAL

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

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

4.2.3.21 ADC_MUX_SHORT

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

Muliplexer routing: inputs are short circuited

4.2.3.22 ADC_MUX_SUPPLY

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

Muliplexer routing: inputs are connected to power supply

4.2.3.23 ADC_MUX_TEMPERATURE

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

Muliplexer routing: ADC measures internal temperature

4.2.3.24 ADC_MUX_TEST_SIGNAL

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

Muliplexer routing: ADC measures test signal

4.2.3.25 ADC_RATE_125HZ

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

Constant defining sampling rate of 125Hz

4.2.3.26 ADC_RATE_250HZ

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

Constant defining sampling rate of 250Hz

4.2.3.27 ADC_RATE_500Hz

```
const int AttysCommBase::ADC_RATE_500Hz = 2  [static]
```

Constant defining sampling rate of 500Hz

4.2.3.28 ADC_REF

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

The voltage reference of the ADC in volts

4.2.3.29 ADC_SAMPLINGRATE

```
const int AttysCommBase::ADC_SAMPLINGRATE[4] = { 125, 250, 500, 1000 }
```

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

4.2.3.30 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 in text form

4.2.3.31 CHANNEL_SHORT_DESCRIPTION

```
const std::string AttysCommBase::CHANNEL_SHORT_DESCRIPTION[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 in text form

4.2.3.32 CHANNEL_UNITS

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

Initial value:

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

Units of the channels

4.2.3.33 INDEX_Acceleration_X

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

Channel index for X Acceleration

4.2.3.34 INDEX_Acceleration_Y

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

Channel index for Y Acceleration

4.2.3.35 INDEX_Acceleration_Z

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

Channel index for Z Acceleration

4.2.3.36 INDEX_Analogue_channel_1

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

Index of analogue channel 1

4.2.3.37 INDEX_Analogue_channel_2

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

Index of analogue channel 2

4.2.3.38 INDEX_GPIO0

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

Index of the internal GPIO pin 1

4.2.3.39 INDEX_GPIO1

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

Index of the internal GPIO pin 2

4.2.3.40 INDEX_Magnetic_field_X

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

Magnetic field in X direction

4.2.3.41 INDEX_Magnetic_field_Y

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

Magnetic field in Y direction

4.2.3.42 INDEX_Magnetic_field_Z

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

Magnetic field in Z direction

4.2.3.43 MAG_FULL_SCALE

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

Full scale range of the magnetometer in Tesla

4.2.3.44 MESSAGE_CONNECTED

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

Message callback: Connected.

4.2.3.45 MESSAGE_ERROR

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

Message callback: Generic error.

4.2.3.46 MESSAGE_RECEIVING_DATA

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

Message callback: Receiving data.

4.2.3.47 MESSAGE_RECONNECTED

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

Message callback: Managed to reconnect.

4.2.3.48 MESSAGE_TIMEOUT

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

Message callback: Reception timeout detected by the watchdog.

4.2.3.49 NCHANNELS

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

Total number of channels per samples.

4.2.3.50 nMem

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

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

4.2.3.51 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

4.3 AttysCommListener Struct Reference

```
#include <AttysCommBase.h>
```

4.3.1 Detailed Description

callback when a sample has arrived

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

- AttysCommBase.h

4.4 AttysCommMessage Struct Reference

```
#include <AttysCommBase.h>
```

4.4.1 Detailed Description

callback when an error has occurred

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

- AttysCommBase.h

4.5 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

4.5.1 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.

4.5.2 Member Function Documentation

4.5.2.1 getAttysComm()

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

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

4.5.2.2 getAttysName()

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

Gets the Attys name as reported by the bluetooth manager

4.5.2.3 getNAttysDevices()

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

Returns the number of Attys devices

4.5.2.4 registerCallback()

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

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

4.5.2.5 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

4.5.2.6 unregisterCallback()

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

Unregisters the callback

4.5.3 Member Data Documentation

4.5.3.1 MAX_ATTYS_DEVS

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

Max number of Attys Devices

4.5.3.2 SCAN_CONNECTED

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

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

4.5.3.3 SCAN_CONNECTERR

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

Connection error during scanning

4.5.3.4 SCAN_CONNECTING

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

In the process of connecting

4.5.3.5 SCAN_NODEV

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

Message index that no Attys has been detected

4.5.3.6 SCAN_SEARCHING

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

Message index that AttysScan is actively scanning

4.5.3.7 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

4.6 AttysScanListener Struct Reference

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

4.6.1 Detailed Description

Callback which reports the status of the scanner

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

- AttysScan.h

Index