AdvancedCommunication Reference Manual

v5r0

Generated by Doxygen 1.3.9.1

Tue Mar 9 12:21:52 2010

Contents

1 Advanced Communication package Documentation 1

2 AdvancedCommunication Namespace Documentation 3

3 AdvancedCommunication Data Structure Documentation 3

4 AdvancedCommunication File Documentation 23

1 Advanced Communication package Documentation

Author:
EGO SW Group

Date:
11/01/2010

Note:
created

1.1 Introduction

The Advance Communication package contains the base library to allow hardware devices to communicate with a server. Depending on the protocol used a specific configuration must be implemented.

1.2 Software architecture

A single factory interface class called ACComm embbed several communication protocols like serial, ethernet, modbus, canbus, To each different communication protocol is associated a subclasses of ACCom :

- ACRSComm: implements a communication via through the Serial protocol (SERIAL_PORT).
- ACEthComm: implements a communication through the Ethernet protocol (ETHERNET).
- ACMBComm: communication using ModBus protocol (MB_COMM).
- ACCanComm: communication using the CanBus protocol (CAN_COMM).

The interface ACComm defines 4 main methods:

- **Open** : Does not take arguments and insures the openning of the channel
- **Close** : Does not take arguments and insures the closing of the channel
- **Read** : Takes a predifined number of arguments depending of the protocol used
- **Write** : Takes a predifined number of arguments depending of the protocol used

1.3 Channel communication configuration

To create a channel object, one must provide 3 types of information depending on the protocol used:

- Protocol = {{SERIAL_PORT},{ETHERNET},{MB_COMM},{CAN_COMM}}.
- Address = {{/dev/ttyS0},{IP},{IP},{}}
- Settings = {{baudrate,vtime,flow control},{port,read timeout, write timeout},{},{netID,#conn trial,sleep}}

Once a channel is Open for a given protocol, the use of the Write and Read methods can be done in different authorized manners, depending on the type of device internal settings. The default usage will be always:

- Write(message,NULL);
- Read(message,NULL);

More parameters can be included in Write and Read which are described in the different methods implementations.

1.4 Example code for Ethernet communication

1.4.1 Channel creation

The configuration for the Ethernet communication requires:

The code name of the protocol:

string Protocol = "ETHERNET";

The IP address:

string Address = "192.168.103.32";

The communication port:

string Port = "2001";

The read time-out (in seconds):

string Read_Delay = "1";

The write time-out (in seconds):

string Write_Delay = "1";

The settings string reads:

string Settings = Port + "," + Read_Delay + "," + Write_Delay;

ACComm *Channel;

Channel = ACComm::Create (Protocol, Address, Settings);

1.4.2 Open Channel

if (Channel->Open () < 0)

{

```
CfgMsgAddError ("Channel open failed at address %s\n",
Address.c_str ());
}
```

1.4.3 Write Channel

```
string message;
message = "PR" + "\r\n";
CfgMsgAddFmt(CFG_INFO,ADVCOMMDEBUG,NULL, "Write message: %s", message.c_str());
if (Channel->Write (message,NULL) < (int)message.length())
{
CfgMsgAddWarning ("Write message failure");
}
```

1.4.4 Read Channel

```
string message;
CfgMsgAddFmt(CFG_INFO,ADVCOMMDEBUG,NULL, "Read level value");
if (!Channel->Read (message,NULL))
{
CfgMsgAddWarning ("Read message failure");
}
```

1.4.5 Close Channel

```
if (Channel->Close () < 0)
{
CfgMsgAddError ("Channel close failed at address %s\n",
Address.c_str ());
}
```

2 AdvancedCommunication Namespace Documentation

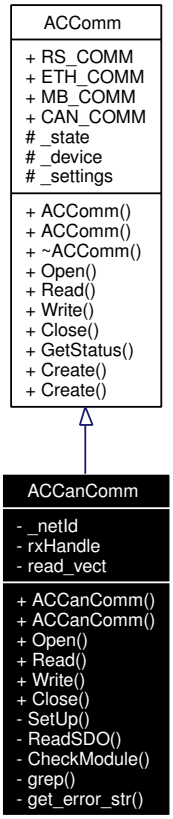
2.1 std Namespace Reference

3 AdvancedCommunication Data Structure Documentation

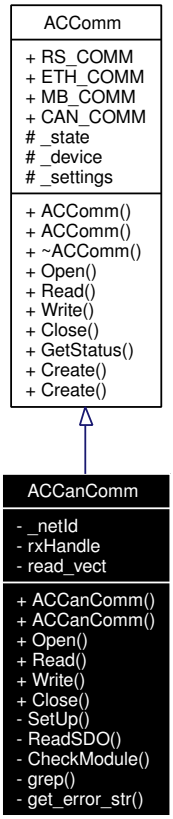
3.1 ACCanComm Class Reference

`#include <ACCanComm.hpp>`

Inheritance diagram for ACCanComm:



Collaboration diagram for ACCanComm:



Public Member Functions

- ACCanComm (string device, string settings)
- ACCanComm (const ACCanComm &channel)
- virtual int Open ()
- virtual int Read (string &message,...)
- virtual int Write (string &message,...)
- virtual int Close ()

Private Member Functions

- int SetUp ()
- int ReadSDO (int deviceNumber, int inputNumber, short *CBM_AI4Value)
- int CheckModule ()
- int grep (string &filename, string keyword)
- char * get_error_str (char *str_buf, long ntstatus)

Private Attributes

- int _netId
- NTCAN_HANDLE rxHandle
- vector< int > read_vect

3.1.1 Constructor & Destructor Documentation

3.1.1.1 ACCanComm::ACCanComm (string *device*, string *settings*) [inline]

Constructor

3.1.1.2 ACCanComm::ACCanComm (const ACCanComm & *channel*) [inline]

Constructor

3.1.2 Member Function Documentation

3.1.2.1 int ACCanComm::CheckModule () [private]

CANBUS CheckModule methods verifies that the machine is suitably configured to operate CanBus protocol

Here is the call graph for this function:

3.1.2.2 int ACCanComm::Close () [virtual]

Close the Communication Channel.

The Close method allows to close the channel

Implements ACComm.

Here is the call graph for this function:

3.1.2.3 char * ACCanComm::get_error_str (char * *str_buf*, long *ntstatus*) [private]

The get_error_str methods returns a string associated with CANBUS errors

Parameters:

str_buf The error string returned

ntstatus The error number

3.1.2.4 int ACCanComm::grep (string & *filename*, string *keyword*) [private]

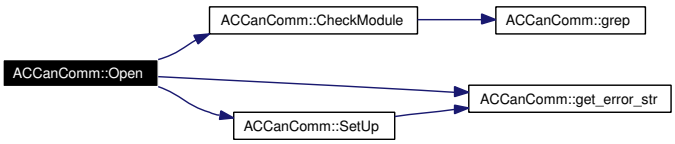
grep utility function

3.1.2.5 int ACCanComm::Open () [virtual]

The Open method allows to open the channel

Implements ACComm.

Here is the call graph for this function:



3.1.2.6 int ACCanComm::Read (string & *message*, ...) [virtual]

The Read method allows to read in the channel

Parameters:

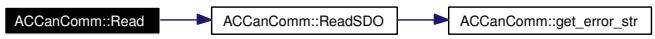
message The message to be read

... The parametrization must be either of the following:

- NULL : no default mode (will return a failure)
- "SDO",devicenumber (int), inputnumber (int) : reads a value using the ReadSDO function. Requires the device number and the (analog) input number

Implements ACComm.

Here is the call graph for this function:



3.1.2.7 int ACCanComm::ReadSDO (int *deviceNumber*, int *inputNumber*, short * *CBM_AI4Value*) [private]

ReadSDO methods performs a read using a single point-to-point SDO transfer

Parameters:

deviceNumber The device number

inputNumber The analogic input number

CBM_AI4Value The returned value

Here is the call graph for this function:

3.1.2.8 int ACCanComm::SetUp () [private]

This methods inits the CANBUS channel

Here is the call graph for this function:

3.1.2.9 int ACCanComm::Write (string & *message*, ...) [virtual]

The Write method allows to write in the channel

Parameters:

message The message to be read

...

Implements ACComm.

3.1.3 Field Documentation

3.1.3.1 int ACCanComm::_netId [private]

CANBUS Net Identifier (located in /etc/esd-plugin file)

3.1.3.2 vector<int> ACCanComm::read_vect [private]

The vector of read device Ids

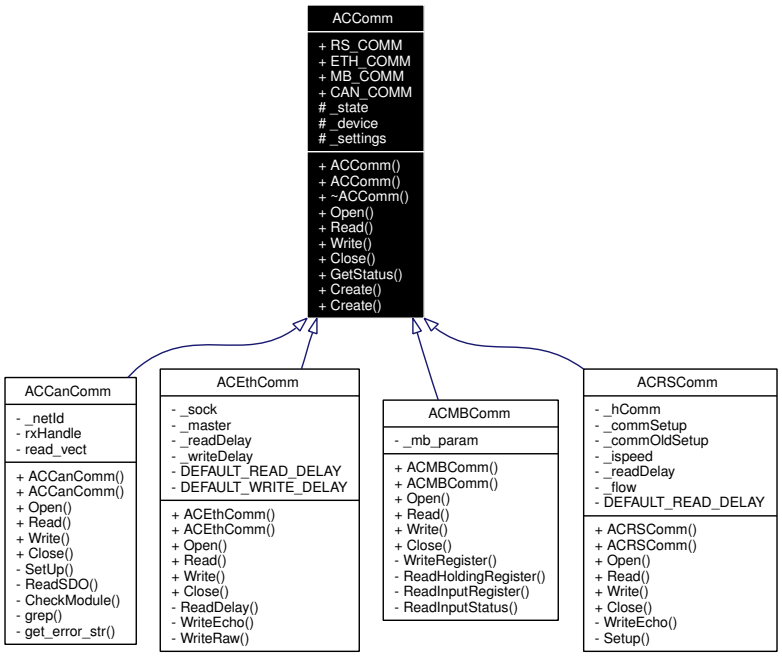
3.1.3.3 NTCAN_HANDLE ACCanComm::rxHandle [private]

The handle associated with the CANBUS channel

3.2 ACComm Class Reference

`#include <ACComm.hpp>`

Inheritance diagram for ACComm:



Public Types

- enum { OPEN, CLOSED }

Public Member Functions

- ACComm (string device, string settings)
- ACComm (const ACComm &channel)
- virtual ~ACComm ()
- virtual int Open ()=0
- virtual int Read (string &message,...)=0
- virtual int Write (string &message,...)=0
- virtual int Close ()=0
- virtual int GetStatus () const

Static Public Member Functions

- ACComm * Create (string channelType, string device, string settings)
- ACComm * Create (const ACComm *prefChannel)

Static Public Attributes

- const string RS_COMM = "SERIAL_PORT"
- const string ETH_COMM = "ETHERNET"
- const string MB_COMM = "MB_COMM"
- const string CAN_COMM = "CAN_COMM"

Protected Attributes

- int _state
- string _device
- string _settings

3.2.1 Member Enumeration Documentation

3.2.1.1 anonymous enum

The state values of the channel communication

Enumeration values:

OPEN

CLOSED

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ACComm::ACComm (string *device*, string *settings*)

Constructor

Parameters:

device The device name or address

settings The setting string as set in the configuration file

3.2.2.2 ACComm::ACComm (const ACComm & *channel*) [inline]

Copy Constructor

Parameters:

channel The object to be copyied

3.2.2.3 virtual ACComm::~ACComm () [inline, virtual]

Destructor

3.2.3 Member Function Documentation

3.2.3.1 virtual int ACComm::Close () [pure virtual]

The Close method allows to close the channel communication port

Implemented in ACCanComm, ACEthComm, ACMBComm, and ACRSComm.

3.2.3.2 ACComm * ACComm::Create (const ACComm * *prefChannel*) [static]

Copy factory instantiator

Parameters:

prefChannel The communication object to be instantiated

3.2.3.3 ACComm * ACComm::Create (string *channelType*, string *device*, string *settings*) [static]

The Create method is the factory instantiator of the communication object

Parameters:

channelType The type of channel to be instantiated (SERIAL,ETHERNET MB_COMM,CAN_-COMM)

device The device name or address as set in the configuration file

settings The settings string as set in the configuration file

3.2.3.4 virtual int ACComm::GetStatus () const [inline, virtual]

The GetStatus returns the channel status (OPEN or CLOSE)

3.2.3.5 virtual int ACComm::Open () [pure virtual]

The Open method allows to open the channel communication port

Implemented in ACCanComm, ACEthComm, ACMBComm, and ACRSComm.

3.2.3.6 virtual int ACComm::Read (string & *message*, ...) [pure virtual]

The Read method allows to read in the channel communication port

Parameters:

message The message to be read

... Various parameters depending on the type of protocol used

Implemented in ACCanComm, ACEthComm, ACMBComm, and ACRSComm.

3.2.3.7 virtual int ACComm::Write (string & *message*, ...) [pure virtual]

The Write method allows to write in the channel communication port

Parameters:

message The message to be written

... Various parameters depending on the type of protocol used

Implemented in ACCanComm, ACEthComm, ACMBComm, and ACRSComm.

3.2.4 Field Documentation

3.2.4.1 string ACComm::_device [protected]

The internal device name of the channel communication

3.2.4.2 string ACComm::_settings [protected]

The internal settings of the channel communication

3.2.4.3 int ACComm::_state [protected]

The internal state of the channel communication

3.2.4.4 const string ACComm::CAN_COMM = "CAN_COMM" [static]

The canbus protocol code name

3.2.4.5 const string ACComm::ETH_COMM = "ETHERNET" [static]

The ethernet protocol code name

3.2.4.6 const string ACComm::MB_COMM = "MB_COMM" [static]

The modbus protocol code name

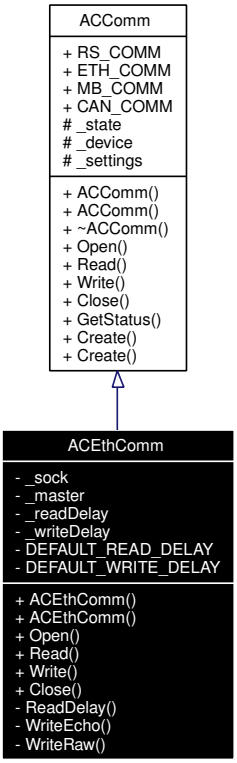
3.2.4.7 const string ACComm::RS_COMM = "SERIAL_PORT" [static]

The serial port protocol code name

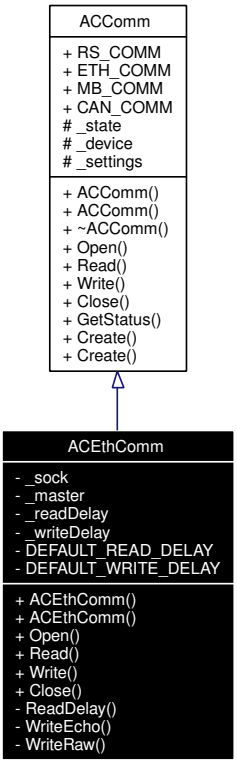
3.3 ACEthComm Class Reference

`#include <ACEthComm.hpp>`

Inheritance diagram for ACEthComm:



Collaboration diagram for ACEthComm:



Public Member Functions

- ACEthComm (string device, string settings)
- ACEthComm (const ACEthComm &channel)
- virtual int Open ()
- virtual int Read (string &message,...)
- virtual int Write (string &message,...)
- virtual int Close ()

Private Member Functions

- virtual int ReadDelay (string &message, int usec)
- virtual int WriteEcho (string message)
- virtual int WriteRaw (string message)

Private Attributes

- int _sock
- fd_set _master
- float _readDelay
- float _writeDelay

Static Private Attributes

- const float DEFAULT_READ_DELAY = 1
- const float DEFAULT_WRITE_DELAY = 1

3.3.1 Constructor & Destructor Documentation

3.3.1.1 ACEthComm::ACEthComm (string *device*, string *settings*) [inline]

Constructor

3.3.1.2 ACEthComm::ACEthComm (const ACEthComm & *channel*) [inline]

Constructor

3.3.2 Member Function Documentation

3.3.2.1 int ACEthComm::Close () [virtual]

The Close method allows to close the channel

Implements ACComm.

3.3.2.2 int ACEthComm::Open () [virtual]

The Open method allows to open the channel

Implements ACComm.

3.3.2.3 int ACEthComm::Read (string & *message*, ...) [virtual]

The Read method allows to read in the channel

Parameters:

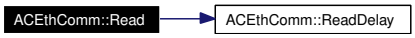
message The message to be read

... The parametrization must be either of the following:

- NULL : default mode
- "DELAY",delay (usec) : using a specific delay before reading

Implements ACComm.

Here is the call graph for this function:



3.3.2.4 int ACEthComm::ReadDelay (string & *message*, int *usec*) [private, virtual]

The ReadDelay reads with a delay

Parameters:

message The read message string

usec Delay in microseconds

3.3.2.5 int ACEthComm::Write (string & *message*, ...) [virtual]

The Write method allows to write in the channel

Parameters:

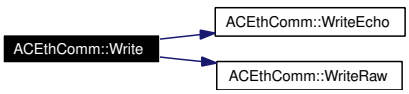
message The message to be read

... The parametrization must be either of the following:

- NULL : default mode (includes NULL terminal character)
- "RAW" : does not include NULL terminal character
- "ECHO" : echo mode writes character by character, each write followed by a single character read.
  The message string must be finished with '/r' character

Implements ACComm.

Here is the call graph for this function:



3.3.2.6 int ACEthComm::WriteEcho (string *message*) [private, virtual]

The WriteEcho methods writes to the channel character by character, each write followed by a single character read.

Parameters:

message The written message string

3.3.2.7 int ACEthComm::WriteRaw (string *message*) [private, virtual]

The WriteRaw methods writes to the channel without additional NULL character added.

Parameters:

message The written message string

3.3.3 Field Documentation

3.3.3.1 fd_set ACEthComm::_master [private]

The internal file descriptor set

3.3.3.2 float ACEthComm::_readDelay [private]

The internal read delay

3.3.3.3 int ACEthComm::_sock [private]

The initial socket file descriptor

3.3.3.4 float ACEthComm::_writeDelay [private]

The internal write delay

3.3.3.5 const float ACEthComm::DEFAULT_READ_DELAY = 1 [static, private]

The default read delay

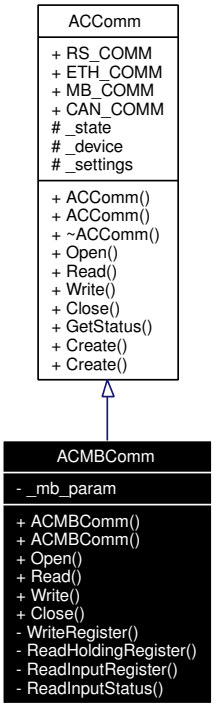
3.3.3.6 const float ACEthComm::DEFAULT_WRITE_DELAY = 1 [static, private]

The default write delay

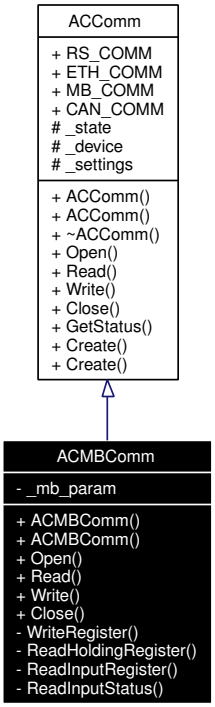
3.4 ACMBComm Class Reference

`#include <ACMBComm.hpp>`

Inheritance diagram for ACMBComm:



Collaboration diagram for ACMBComm:



Public Member Functions

- ACMBComm (string device, string settings)

- ACMBComm (const ACMBComm &channel)
- virtual int Open ()
- virtual int Read (string &message,...)
- virtual int Write (string &message,...)
- virtual int Close ()

Private Member Functions

- virtual int WriteRegister (int mb_addr, int mb_slave, int mb_value)
- virtual int ReadHoldingRegister (string &message, int mb_num, int mb_addr, int mb_slave, uint16_t *mb_value)
- virtual int ReadInputRegister (string &message, int mb_num, int mb_addr, int mb_slave, uint16_t *mb_value)
- virtual int ReadInputStatus (string &message, int mb_num, int mb_addr, int mb_slave, uint8_t *mb_value)

Private Attributes

- modbus_param_t _mb_param

3.4.1 Constructor & Destructor Documentation

3.4.1.1 ACMBComm::ACMBComm (string *device*, string *settings*) [inline]

Constructor

3.4.1.2 ACMBComm::ACMBComm (const ACMBComm & *channel*) [inline]

Constructor

3.4.2 Member Function Documentation

3.4.2.1 int ACMBComm::Close () [virtual]

The Close method allows to close the channel

Implements ACComm.

3.4.2.2 int ACMBComm::Open () [virtual]

The Open method allows to open the channel

Implements ACComm.

3.4.2.3 int ACMBComm::Read (string & *message*, ...) [virtual]

The Read method allows to read in the channel

Parameters:

message The message to be read

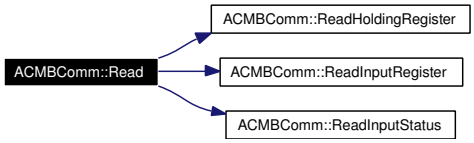
... The parametrization must be either of the following:

- NULL : no default mode (will return a failure)

- "HOLDING",mb_num (int), mb_address (int), mb_slave (int), mb_value (uint16_t*): reads a value array using the read_holding_registers function
- "INPUT",mb_num (int), mb_address (int), mb_slave (int), mb_value (uint16_t*): reads a value array using the read_input_registers function
- "STATUS",mb_num (int), mb_address (int), mb_slave (int), mb_value (uint8_t*): reads a value array using the read_input_status function

Implements ACComm.

Here is the call graph for this function:



3.4.2.4 int ACMBComm::ReadHoldingRegister (string & *message*, int *mb_num*, int *mb_addr*, int *mb_slave*, uint16_t * *mb_value*) [private, virtual]

The ReadHoldingRegister method uses read_holding_registers function to read the value from a single register.

Parameters:

message The returned value (in string format)

mb_num The number of values to be read

mb_addr The register address

mb_slave The slave flag (for example: 0xFF)

mb_value The array of value

3.4.2.5 int ACMBComm::ReadInputRegister (string & *message*, int *mb_num*, int *mb_addr*, int *mb_slave*, uint16_t * *mb_value*) [private, virtual]

The ReadInputRegister method uses read_input_registers function to read the value from a single register.

Parameters:

message The returned value (in string format)

mb_num The number of values to be read

mb_addr The register address

mb_slave The slave flag (for example: 0xFF)

mb_value The array of value

3.4.2.6 int ACMBComm::ReadInputStatus (string & *message*, int *mb_num*, int *mb_addr*, int *mb_slave*, uint8_t * *mb_value*) [private, virtual]

The ReadInputStatus method uses read_input_status function to read the value from a single register.

Parameters:

message The returned value (in string format)

mb_num The number of values to be read

mb_addr The register address

mb_slave The slave flag (for example: 0xFF)

mb_value The array of value

3.4.2.7 int ACMBComm::Write (string & *message*, ...) [virtual]

The Write method allows to write in the channel

Parameters:

message The message to be read

... The parametrization must be either of the following:

- NULL : no default mode (will return a failure)
- "SINGLE",mb_address (int), mb_slave (int), mb_value (int): writes a single value using the preset_single_register function

Implements ACComm.

Here is the call graph for this function:

3.4.2.8 int ACMBComm::WriteRegister (int *mb_addr*, int *mb_slave*, int *mb_value*) [private, virtual]

The WriteRegister method uses preset_single_register function to write the value for a single register.

Parameters:

mb_addr The register address

mb_slave The slave flag (for example: 0xFF)

mb_value The value to be written

3.4.3 Field Documentation

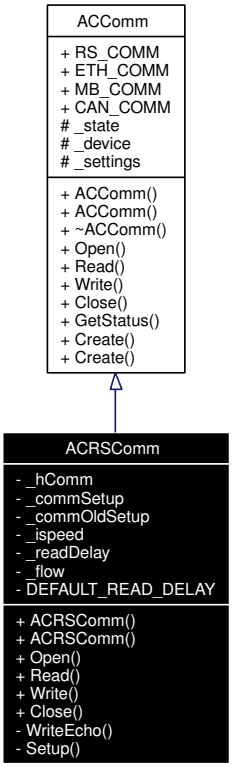
3.4.3.1 modbus_param_t ACMBComm::_mb_param [private]

The MODBUS channel object

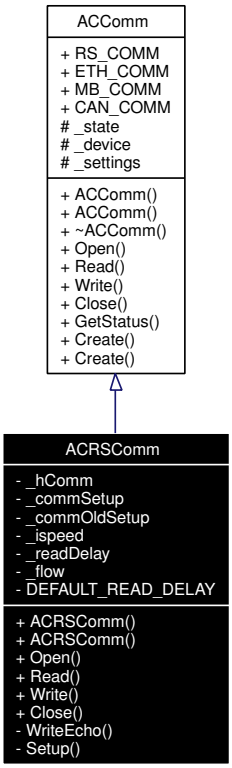
3.5 ACRSComm Class Reference

`#include <ACRSComm.hpp>`

Inheritance diagram for ACRSComm:



Collaboration diagram for ACRSComm:



Public Member Functions

- ACRSComm (string device, string settings)
- ACRSComm (const ACRSComm &channel)
- virtual int Open ()
- virtual int Read (string &message,...)
- virtual int Write (string &message,...)
- virtual int Close ()

Private Member Functions

- virtual int WriteEcho (string message)
- int Setup ()

Private Attributes

- int _hComm
- termios _commSetup
- termios _commOldSetup
- int _ispeed
- int _readDelay
- char _flow [32]

Static Private Attributes

- const int DEFAULT_READ_DELAY = 20

3.5.1 Constructor & Destructor Documentation

3.5.1.1 ACRSComm::ACRSComm (string *device*, string *settings*) [inline]

Constructor

3.5.1.2 ACRSComm::ACRSComm (const ACRSComm & *channel*) [inline]

Constructor

3.5.2 Member Function Documentation

3.5.2.1 int ACRSComm::Close () [virtual]

The Close method allows to close the channel

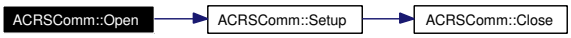
Implements ACComm.

3.5.2.2 int ACRSComm::Open () [virtual]

The Open method allows to open the channel

Implements ACComm.

Here is the call graph for this function:



3.5.2.3 int ACRSComm::Read (string & *message*, ...) [virtual]

The Read method allows to read in the channel

Parameters:

message The message to be read

... The parametrization must be either of the following:

- NULL : default mode

Implements ACComm.

3.5.2.4 int ACRSComm::Setup () [private]

The Setup method initialize the RS232 properties

Here is the call graph for this function:

3.5.2.5 int ACRSComm::Write (string & *message*, ...) [virtual]

The Write method allows to write in the channel

Parameters:

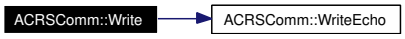
message The message to be read

... The parametrization must be either of the following:

- NULL : default mode
- "ECHO" : echo mode writes character by character, each write followed by a single character read.
  The message string must be finished with '/r' character

Implements ACComm.

Here is the call graph for this function:



3.5.2.6 int ACRSComm::WriteEcho (string *message*) [private, virtual]

The WriteEcho methods writes to the channel character by character, each write followed by a single character read.

Parameters:

message The written message string

3.5.3 Field Documentation

3.5.3.1 struct termios ACRSComm::_commOldSetup [private]

former communication setup

3.5.3.2 struct termios ACRSComm::_commSetup [private]

current communication setup

3.5.3.3 char ACRSComm::_flow[32] [private]

The internal flow type can be :

- NONE : default flow control
- XONXOFF

3.5.3.4 int ACRSComm::_hComm [private]

Internal file descriptor

3.5.3.5 int ACRSComm::_ispeed [private]

The internal baud rate (can be set by the server configuration)

3.5.3.6 int ACRSComm::_readDelay [private]

The internal read delay known as vtime parameter (can be set by the server configuration)

3.5.3.7 const int ACRSComm::DEFAULT_READ_DELAY = 20 [static, private]

The default read delay

4 AdvancedCommunication File Documentation

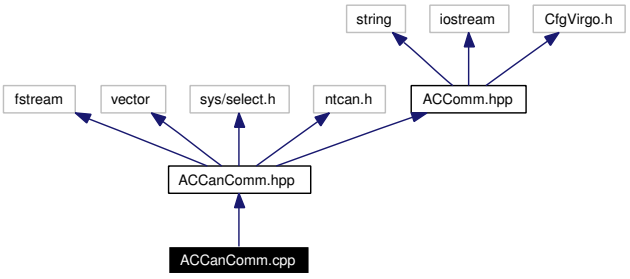
4.1 ACCanComm.cpp File Reference

4.1.1 Detailed Description

Class ACCanComm This class provides a set of operations to perform Canbus tcp/ip socket based communication over ethernet Only CBM-AI4 of EtherCan is supported (used by IMMS)

```
#include "ACCanComm.hpp"
```

Include dependency graph for ACCanComm.cpp:



4.2 ACCanComm.hpp File Reference

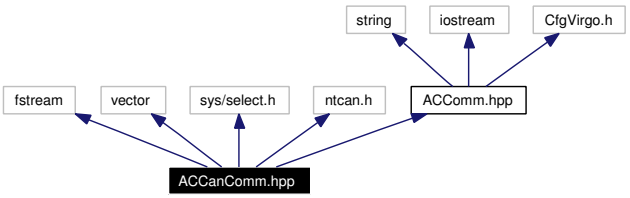
4.2.1 Detailed Description

Class ACCanComm

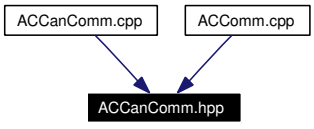
This class provides a set of operations to perform communication with an analog CBM-API over EtherCan

```
#include <fstream>
#include <vector>
#include <sys/select.h>
#include "ntcan.h"
#include "ACComm.hpp"
```

Include dependency graph for ACCanComm.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace std

Data Structures

- class ACCanComm

Defines

- #define NTCAN_FAILURE -1
- #define CAN_SLEEP 1000
- #define CANDEBUG 1

Variables

- const int DI08_SDO_ANSWER_READ_ID = 0x300
- const int DI08_SDO_REQUEST_SEND = 0x301
- const int SDO_READ_FAILED = 0x10000
- const int SDO_READ_SUCCEEDED = 0
- const int SDO_ANSWER_READ_ID = 0x580
- const int SDO_REQUEST_SEND = 0x600
- const int SDO_READ_CODE = 0x40
- const int SDO_ERROR_CODE = 0x80
- const int SDO_LSB_DATA_INDEX = 4
- const int SDO_MSB_DATA_INDEX = 5
- const int ANALOGICAL_INPUT_ID_LSB = 0x01
- const int ANALOGICAL_INPUT_ID_MSB = 0x64
- const unsigned long MODE = 0
- const long TXQUEUESIZE = 1
- const long RXQUEUESIZE = 1
- const long TXTIMEOUT = 5000
- const long RXTIMEOUT = 5000
- const unsigned long BAUDRATE = 2
- const int START_NODE [ ]

4.2.2 Define Documentation

4.2.2.1 #define CAN_SLEEP 1000

4.2.2.2 #define CANDEBUG 1

4.2.2.3 #define NTCAN_FAILURE -1

4.2.3 Variable Documentation

4.2.3.1 const int ANALOGICAL_INPUT_ID_LSB = 0x01 [static]

4.2.3.2 const int ANALOGICAL_INPUT_ID_MSB = 0x64 [static]

4.2.3.3 const unsigned long BAUDRATE = 2 [static]

4.2.3.4 const int DI08_SDO_ANSWER_READ_ID = 0x300 [static]

4.2.3.5 const int DI08_SDO_REQUEST_SEND = 0x301 [static]

4.2.3.6 const unsigned long MODE = 0 [static]

4.2.3.7 const long RXQUEUESIZE = 1 [static]

4.2.3.8 const long RXTIMEOUT = 5000 [static]

4.2.3.9 const int SDO_ANSWER_READ_ID = 0x580 [static]

4.2.3.10 const int SDO_ERROR_CODE = 0x80 [static]

4.2.3.11 const int SDO_LSB_DATA_INDEX = 4 [static]

4.2.3.12 const int SDO_MSB_DATA_INDEX = 5 [static]

4.2.3.13 const int SDO_READ_CODE = 0x40 [static]

4.2.3.14 const int SDO_READ_FAILED = 0x10000

4.2.3.15 const int SDO_READ_SUCCEEDED = 0

4.2.3.16 const int SDO_REQUEST_SEND = 0x600 [static]

4.2.3.17 const int START_NODE[ ] [static]

Initial value:

```
{
  0x1,0x0,130,129
}
```

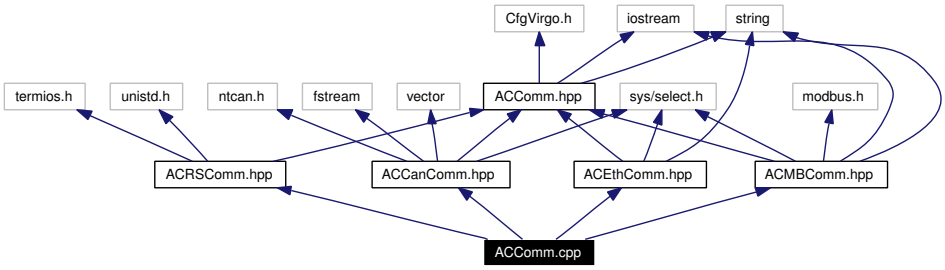
4.2.3.18 const long TXQUEUESIZE = 1 [static]

4.2.3.19 const long TXTIMEOUT = 5000 [static]

4.3 ACComm.cpp File Reference

```
#include "ACRSComm.hpp"
#include "ACEthComm.hpp"
#include "ACCanComm.hpp"
#include "ACMBComm.hpp"
```

Include dependency graph for ACComm.cpp:



4.4 ACComm.hpp File Reference

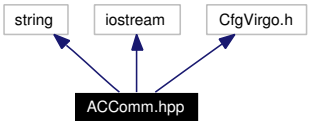
4.4.1 Detailed Description

Class ACComm

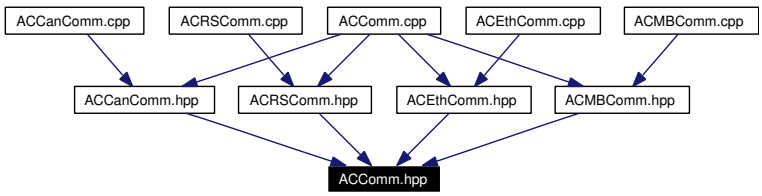
This class provides a superclass of communication channels.

```
#include <string>
#include <iostream>
#include "CfgVirgo.h"
```

Include dependency graph for ACComm.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class ACComm

Defines

- #define ADVCOMMDEBUG 2
- #define STRLENGTH 1024

4.4.2 Define Documentation

4.4.2.1 #define ADVCOMMDEBUG 2

4.4.2.2 #define STRLENGTH 1024

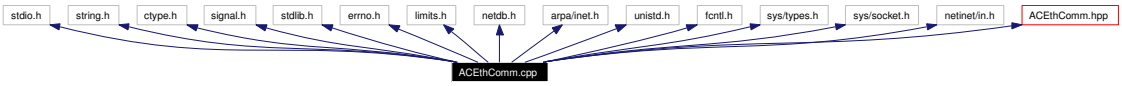
4.5 ACEthComm.cpp File Reference

4.5.1 Detailed Description

Class ACEthComm This class provides a set of operations to perform tcp/ip socket based communication

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <signal.h>
#include <stdlib.h>
#include <errno.h>
#include <limits.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "ACEthComm.hpp"
```

Include dependency graph for ACEthComm.cpp:



4.6 ACEthComm.hpp File Reference

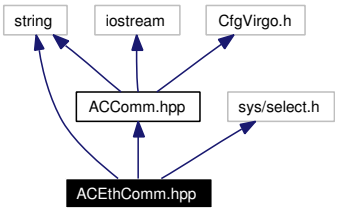
4.6.1 Detailed Description

Class ACEthComm

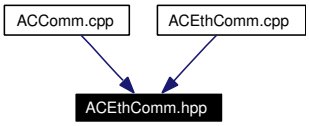
This class provides a set of operations to perform communication via a Tcp/Ip socket.

```
#include "ACComm.hpp"
#include <string>
#include <sys/select.h>
```

Include dependency graph for ACEthComm.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class ACEthComm

Defines

- #define _ACETHCOMM_H_

4.6.2 Define Documentation

4.6.2.1 #define _ACETHCOMM_H_

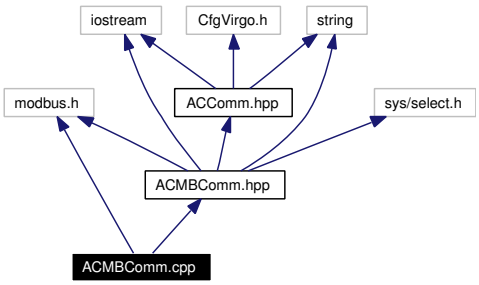
4.7 ACMBComm.cpp File Reference

4.7.1 Detailed Description

Class ACMBComm This class provides a set of operations to perform modbus tcp/ip socket based communication

```
#include <modbus.h>
#include "ACMBComm.hpp"
```

Include dependency graph for ACMBComm.cpp:



Defines

- #define Success 1
- #define Failure 0

4.7.2 Define Documentation

4.7.2.1 #define Failure 0

4.7.2.2 #define Success 1

4.8 ACMBComm.hpp File Reference

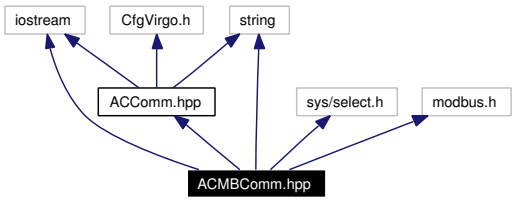
4.8.1 Detailed Description

Class ACMBComm

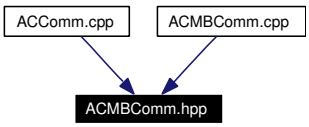
This class provides a set of operations to perform communication via a Tcp/Ip socket.

```
#include <iostream>
#include <string>
#include <sys/select.h>
#include <modbus.h>
#include "ACComm.hpp"
```

Include dependency graph for ACMBComm.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class ACMBComm

Defines

- #define _ACMBCOMM_H_

4.8.2 Define Documentation

4.8.2.1 #define _ACMBCOMM_H_

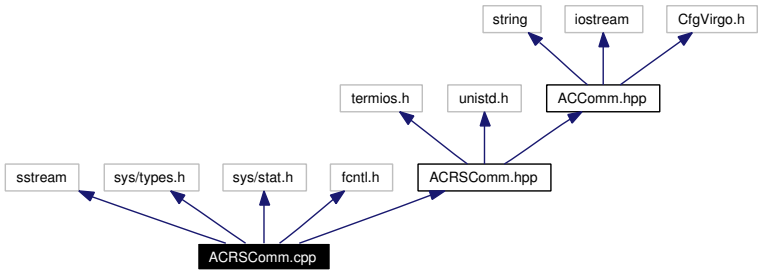
4.9 ACRSComm.cpp File Reference

4.9.1 Detailed Description

Class ACRSComm This class provides a superclass of communication channels

```
#include <sstream>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "ACRSComm.hpp"
```

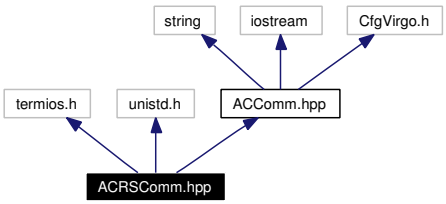
Include dependency graph for ACRSComm.cpp:



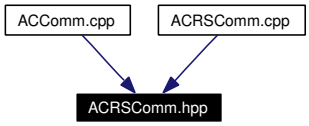
4.10 ACRSComm.hpp File Reference

```
#include <termios.h>
#include <unistd.h>
#include "ACComm.hpp"
```

Include dependency graph for ACRSComm.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class ACRSComm

Defines

- #define _ACRSCOMM_H

4.10.1 Define Documentation

4.10.1.1 #define _ACRSCOMM_H

&file ACRSComm.hpp Class ACRSComm This class provides a set of operations to perform communications via a serial port.