

Goby Underwater Autonomy Project

Series: 1.1, revision: 162, released on 2013-01-15 17:01:49 -0500

Generated by Doxygen 1.7.6.1

Tue Jan 15 2013 22:02:16

Contents

1	Goby Underwater Autonomy Project	1
1.1	Resources	1
1.2	Developer manual	1
1.3	Publications	1
1.4	Download and Install Goby	1
1.5	Building Examples	2
1.6	Authors	2
2	goby-acomms: Overview of Acoustic Communications Libraries	2
2.1	Quick Start	2
2.2	Overview	3
2.2.1	Analogy to established networking systems	3
2.2.2	Acoustic Communications are slow	5
2.2.3	Efficiency to make messages small is good	5
2.2.4	Total throughput unrealistic: prioritize data	5
2.2.5	Despite all this, simplicity is good	5
2.2.6	Component model	6
2.3	libdccl: Encoding and decoding	7
2.4	libqueue: Priority based message queuing	7
2.5	libmodemdriver: Modem driver	7
2.6	libamac: Medium Access Control (MAC)	8
2.7	Software concepts used in goby-acomms	8
2.7.1	Signal / Slot model for asynchronous events	8
2.7.2	Google Protocol Buffers	9
2.8	UML models	10
3	goby-acomms: libdccl (Dynamic Compact Control Language)	13
3.1	Designing a message	13
3.2	DCCL tag structure	15
3.3	Interacting with the DCCLCodec	17
3.4	Encryption	19
3.5	Details of encoding/decoding scheme	19
3.6	DCCL XML Tags Reference	22

3.6.1	<all>	22
3.6.2	<array_length>	22
3.6.3	<bool>	23
3.6.4	<dest_id>	23
3.6.5	<destination_var>	23
3.6.6	<enum>	24
3.6.7	<float>	24
3.6.8	<format>	24
3.6.9	<header>	25
3.6.10	<hex>	25
3.6.11	<id>	26
3.6.12	<int>	26
3.6.13	<layout>	26
3.6.14	<max_length>	27
3.6.15	<max>	27
3.6.16	<max_delta>	28
3.6.17	<message_set>	28
3.6.18	<message_var>	28
3.6.19	<message>	29
3.6.20	<min>	29
3.6.21	<name>	30
3.6.22	<num_bytes>	30
3.6.23	<on_receipt>	31
3.6.24	<precision>	31
3.6.25	<publish>	31
3.6.26	<publish_var>	32
3.6.27	<repeat>	32
3.6.28	<size>	32
3.6.29	<src_id>	33
3.6.30	<src_var>	33
3.6.31	<static>	34
3.6.32	<string>	34
3.6.33	<time>	34
3.6.34	<trigger_var>	35

3.6.35	<trigger_time>	35
3.6.36	<trigger>	35
3.6.37	<value>	36
4	goby-acomms: libqueue (Message Priority Queuing)	36
4.1	Understanding dynamic priority queuing	37
4.2	Queuing tag structure	38
4.3	Interacting with the QueueManager	38
4.3.1	Instantiate and configure	38
4.3.2	Signals and (application layer) slots	38
4.3.3	Operation	39
4.4	Queuing XML Tags Reference	39
4.4.1	<ack>	39
4.4.2	<blackout_time>	40
4.4.3	<max_queue>	40
4.4.4	<newest_first>	40
4.4.5	<value_base>	41
4.4.6	<ttd>	41
4.4.7	<queuing>	42
5	goby-acomms: libmodemdriver (Driver to interact with modem firmware)	42
5.1	Abstract class: ModemDriverBase	42
5.2	WHOI Micro-Modem Driver: MMDriver	43
5.3	Writing a new driver	49
6	goby-acomms: libamac (Medium Access Control)	54
6.1	Supported MAC schemes	54
6.2	Interacting with the goby::acomms::MACManager	54
7	goby-util: Overview of Utility Libraries	57
7.1	Overview	57
7.2	Logging	57
7.2.1	Configurable extension of std::ostream - liblogger	58
7.3	TCP and Serial port communications - liblinebasedcomms	60
8	Module Index	60

8.1	Modules	60
9	Namespace Index	60
9.1	Namespace List	60
10	Class Index	61
10.1	Class Hierarchy	61
11	Class Index	62
11.1	Class List	62
12	File Index	64
12.1	File List	64
13	Module Documentation	69
13.1	API classes for the acoustic communications libraries.	69
14	Namespace Documentation	69
14.1	goby Namespace Reference	69
14.1.1	Detailed Description	70
14.2	goby::acomms Namespace Reference	70
14.2.1	Detailed Description	73
14.2.2	Typedef Documentation	73
14.2.3	Enumeration Type Documentation	74
14.2.4	Function Documentation	75
14.2.5	Variable Documentation	75
14.3	goby::acomms::protobuf Namespace Reference	75
14.3.1	Detailed Description	76
14.4	goby::util Namespace Reference	76
14.4.1	Detailed Description	79
14.4.2	Function Documentation	79
14.5	goby::util::tcolor Namespace Reference	83
14.5.1	Detailed Description	84
14.5.2	Function Documentation	84
15	Class Documentation	84
15.1	goby::acomms::ABCDriver Class Reference	85

15.1.1 Detailed Description	85
15.1.2 Member Function Documentation	85
15.2 goby::acomms::DCCLCodec Class Reference	86
15.2.1 Detailed Description	89
15.2.2 Constructor & Destructor Documentation	89
15.2.3 Member Function Documentation	90
15.3 goby::acomms::DCCLException Class Reference	95
15.3.1 Detailed Description	95
15.4 goby::acomms::DCCLMessageVal Class Reference	95
15.4.1 Detailed Description	97
15.4.2 Member Function Documentation	97
15.5 goby::acomms::MACManager Class Reference	100
15.5.1 Detailed Description	101
15.5.2 Constructor & Destructor Documentation	101
15.5.3 Member Function Documentation	102
15.5.4 Member Data Documentation	102
15.6 goby::acomms::MMDriver Class Reference	103
15.6.1 Detailed Description	104
15.6.2 Constructor & Destructor Documentation	104
15.6.3 Member Function Documentation	104
15.7 goby::acomms::ModemDriverBase Class Reference	105
15.7.1 Detailed Description	107
15.7.2 Constructor & Destructor Documentation	107
15.7.3 Member Function Documentation	107
15.7.4 Member Data Documentation	110
15.8 goby::acomms::QueueManager Class Reference	112
15.8.1 Detailed Description	114
15.8.2 Constructor & Destructor Documentation	114
15.8.3 Member Function Documentation	114
15.8.4 Member Data Documentation	116
15.9 goby::ConfigException Class Reference	118
15.9.1 Detailed Description	118
15.10 goby::Exception Class Reference	118
15.10.1 Detailed Description	118

15.11goby::util::Colors Struct Reference	119
15.11.1 Detailed Description	119
15.12goby::util::FlexNCurses Class Reference	119
15.12.1 Detailed Description	120
15.13goby::util::FlexOstream Class Reference	120
15.13.1 Detailed Description	121
15.13.2 Member Function Documentation	121
15.13.3 Friends And Related Function Documentation	122
15.14goby::util::FlexOStreamBuf Class Reference	122
15.14.1 Detailed Description	123
15.15goby::util::LineBasedInterface Class Reference	123
15.15.1 Detailed Description	124
15.15.2 Member Function Documentation	124
15.16goby::util::Logger Struct Reference	125
15.16.1 Detailed Description	125
15.17goby::util::SerialClient Class Reference	125
15.17.1 Detailed Description	126
15.17.2 Constructor & Destructor Documentation	126
15.18goby::util::TCPClient Class Reference	126
15.18.1 Detailed Description	127
15.18.2 Constructor & Destructor Documentation	127
15.19goby::util::TCPServer Class Reference	127
15.19.1 Detailed Description	128
15.19.2 Constructor & Destructor Documentation	128
15.20goby::util::TermColor Class Reference	128
15.20.1 Detailed Description	129
15.21Group Class Reference	129
15.21.1 Detailed Description	130
15.22GroupSetter Class Reference	130
15.22.1 Detailed Description	130
16 File Documentation	131
16.1 moos/libmoos_util/moos_protobuf_helpers.h File Reference	131
16.1.1 Detailed Description	131

16.1.2	Function Documentation	131
17	Example Documentation	132
17.1	acomms/chat/chat.cpp	132
17.2	libamac/amac_simple/amac_simple.cpp	137
17.3	libdccl/dccl_simple/dccl_simple.cpp	138
17.4	libdccl/plusnet/plusnet.cpp	140
17.5	libdccl/test/test.cpp	152
17.6	libdccl/two_message/two_message.cpp	156
17.7	libmodemdriver/driver_simple/driver_simple.cpp	160
17.8	libqueue/queue_simple/queue_simple.cpp	162

1 Goby Underwater Autonomy Project

The Goby Underwater Autonomy Project aims to create a unified framework for multiple scientific autonomous marine vehicle collaboration, seamlessly incorporating acoustic, ethernet, wifi, and serial communications. Presently the main thrust of the project is developing a set of robust acoustic networking libraries. Goby is licensed under the GNU General Public License v3 <<http://www.gnu.org/licenses/gpl.html>>.

1.1 Resources

- Home page, code, bug tracking, and answers: <<https://launchpad.net/goby>>.
- User Manual: ([pdf](#)).
- Developers' Manual: ([html](#)) ([pdf](#)).
- Wiki: <<http://gobysoft.com/wiki>>.

1.2 Developer manual

- [goby-acomms: Overview of Acoustic Communications Libraries](#) - tackle the extremely rate limited acoustic networking problem. These libraries were designed together but can operate independently for a developer looking integrate a specific component (e.g. just encoding/decoding) without committing to the entire goby-acomms stack.
- [goby-util: Overview of Utility Libraries](#) - provide utility functions for tasks such as logging, scientific calculations, string parsing, and serial device i/o. Goby also relies on the boost libraries <<http://www.boost.org/>> for many utility tasks to fill in areas where the C++ Standard Library is insufficient or unelegant.

1.3 Publications

- [The Dynamic Compact Control Language: A Compact - Marshalling Scheme for Acoustic Communications.](#) IEEE OCEANS'10 / Sydney.

1.4 Download and Install Goby

Please visit [<http://gobysoft.com/wiki/InstallingGoby>](http://gobysoft.com/wiki/InstallingGoby) for help on obtaining and installing Goby.

1.5 Building Examples

Please visit [<http://gobysoft.com/wiki/Examples>](http://gobysoft.com/wiki/Examples) for to learn about the available code examples for Goby.

1.6 Authors

Goby is developed by the Goby Developers group (<https://launchpad.net/~goby-dev>). The lead developer is Toby Schneider (<http://gobysoft.com>)

2 goby-acomms: Overview of Acoustic Communications Libraries

Table of Contents for [goby-acomms: Overview of Acoustic Communications Libraries](#).

- [Quick Start](#)
- [Overview](#)
 - [Analogy to established networking systems](#)
 - [Acoustic Communications are slow](#)
 - [Efficiency to make messages small is good](#)
 - [Total throughput unrealistic: prioritize data](#)
 - [Component model](#)
- [libdccl: Encoding and decoding \(Detailed documentation \)](#)
- [libqueue: Priority based message queuing \(Detailed documentation \)](#)
- [libmodemdriver: Modem driver \(Detailed documentation \)](#)
- [libamac: Medium Access Control \(MAC\) \(Detailed documentation \)](#)
- [Software concepts used in goby-acomms](#)

- [Signal / Slot model for asynchronous events](#)
- [Google Protocol Buffers](#)
- [UML models](#)

2.1 Quick Start

To get started using the goby-acomms libraries as quickly as possible:

1. If you haven't yet, follow instructions on [installing Goby](#).
2. Identify which components you need:
 - Encoding and decoding from C++ types to bit-packed messages: [libdccl](#).
 - Queuing of DCCL and CCL messages with priority based message selection: [libqueue](#).
 - A driver for interacting with the acoustic modem firmware. Presently the - WHOI Micro-Modem <<http://acomms.who.i.edu/>> is supported: [libmodemdriver](#).
 - Time division multiple access (TDMA) medium access control (MAC): [liba-mac](#).
3. Look at the "simple" code examples that accompany each library ([dccl_simple.cpp](#), [queue_simple.cpp](#), [driver_simple.cpp](#), [amac_simple.cpp](#)). Then look at the example that uses all the libraries together: [chat.cpp](#). The full list of examples is given in [this table](#).
4. Refer to the rest of the documentation as needed.

Please visit <<https://answers.launchpad.net/goby>> with any questions.

2.2 Overview

2.2.1 Analogy to established networking systems

To start on some (hopefully) common ground, let's begin with an analogy to Open - Systems Initiative (OSI) networking layers in this [table](#). For a complete description of the OSI layers see <<http://www.itu.int/rec/T-REC-X.200-199407-->

I/en>.

OSI Layer	Goby library	API class(es)	Example(s)
Application	Not yet part of Goby		MOOS Application: pAcommsHandler
Presentation	libdccl: Encoding and decoding	goby::acomms::D-CCLCodec	dccl_simple.cpp two_message.cpp plusnet.cpp test.cpp chat.cpp
Session	Not used, sessions are established passively.		
Transport	libqueue: Priority based message queuing	goby::acomms::QueueManager	queue_simple.cpp chat.cpp
Network	Does not yet exist. All transmissions are considered single hop, currently. Addressing routing over multiple hops is an open and pressing research problem.		
Data Link	libmodemdriver: Modem driver	classes derived from goby::acomms::ModemDriver-Base; e.g. goby::acomms::MMDriver	driver_simple.cpp chat.cpp
	libamac: Medium Access Control (MAC)	goby::acomms::MACManager	amac_simple.cpp chat.cpp
Physical	Not part of Goby		Modem Firmware, e.g. WHOI Micro-Modem Firmware (NMEA 0183 on RS-232) (see Interface Guide)

2.2.2 Acoustic Communications are slow

Do not take the previous analogy too literally; some things we are doing here for acoustic communications (hereafter, acommms) are unconventional from the approach of networking on electromagnetic carriers (hereafter, EM networking). The difference is a vast spread in the expected throughput of a standard internet hardware carrier and acoustic communications. For example, an optical fiber can put through greater than 10 Tbps over greater than 100 km, whereas the WHOI acoustic Micro-Modem can (at best) do 5000 bps over several km. This is a difference of thirteen orders of magnitude for the bit-rate distance product!

2.2.3 Efficiency to make messages small is good

Extremely low throughput means that essentially every efficiency in bit packing messages to the smallest size possible is desirable. The traditional approach of layering (e.g. TCP/IP) creates inefficiencies as each layer wraps the message of the higher layer with its own header. See RFC3439 section 3 ("Layering Considered Harmful") for an interesting discussion of this issue <<http://tools.ietf.org/html/rfc3439#page-7>>. Thus, the "layers" of goby-acommms are more tightly interrelated than TCP/IP, for example. Higher layers depend on lower layers to carry out functions such as error checking and do not replicate this functionality.

2.2.4 Total throughput unrealistic: prioritize data

The second major difference stemming from this bandwidth constraint is that *total throughput is often an unrealistic goal*. The quality of the acoustic channel varies widely from place to place, and even from hour to hour as changes in the sea affect propagation of sound. This means that it is also difficult to predict what one's throughput will be at any given time. These two considerations manifest themselves in the goby-acommms design as a priority based queueing system for the transport layer. Messages are placed in different queues based on their priority (which is determined by the designer of the message). This means that

- the channel is always utilized (low priority data are sent when the channel quality is high)
- important messages are not swamped by low priority data

In contrast, TCP/IP considers all packets equally. Packets made from a spam email are given the same consideration as a high priority email from the President. This is a tradeoff in efficiency versus simplicity that makes sense for EM networking, but does not for acoustic communications.

2.2.5 Despite all this, simplicity is good

The "law of diminishing returns" means that at some point, if we try to optimize excessively, we will end up making the system more complex without substantial gain. Thus, goby-acommms makes some concessions for the sake of simplicity:

- Numerical message fields are bounded by powers of 10, rather than 2. Humans deal much better with decimal than binary.
- User data splitting (and subsequent stitching) is not done. This is a key component of TCP/IP, but with the number of dropped packets one can expect with acomms, at the moment this does not seem like a good idea. The user is expected to provide data that is smaller or equal to the packet size of the physical layer (e.g. 32 - 256 bytes for the WHOI Micro-Modem).

2.2.6 Component model

A relatively simple component model for the goby-acomms libraries showing the [interface classes](#) :

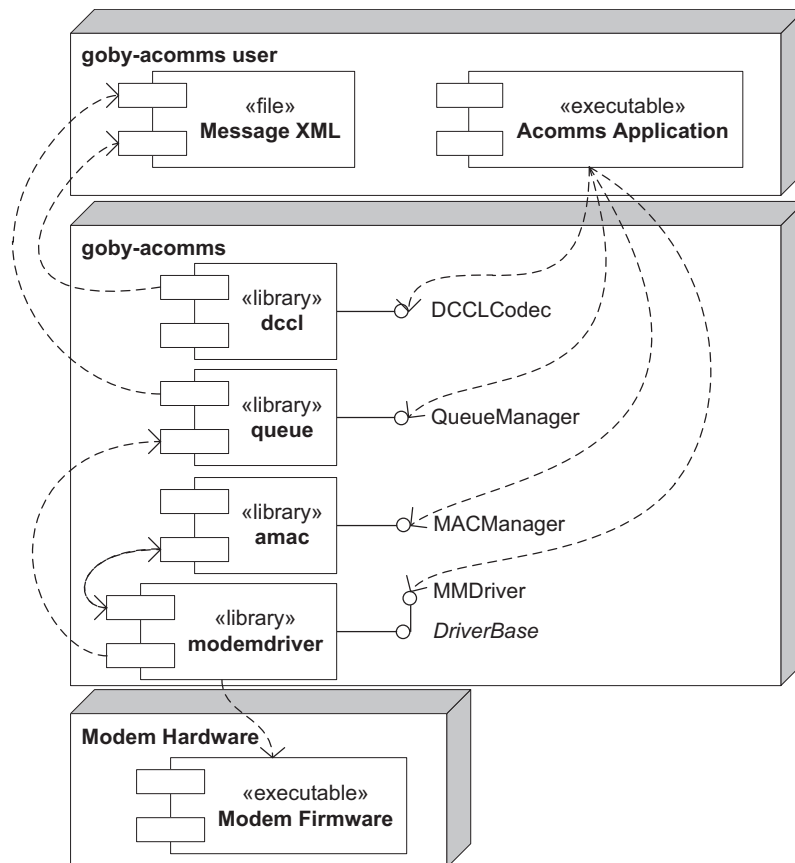


Figure 1: Basic overview of goby-acomms libraries.

For a more detailed model, see the [UML models](#) section.

2.3 libdccl: Encoding and decoding

Dynamic Compact Control Language (DCCL) provides a structure for defining messages to be sent through an acoustic modem. The messages are configured in XML and are intended to be easily reconfigurable, unlike the original CCL framework used in the REMUS vehicles and others (for information on CCL, see <<http://acomms.whoii.edu/ccl/>>). DCCL can operate within a CCL network, as the most significant byte (or CCL ID) is 0x20.

DCCL messages are packed based on boundaries determined with knowledge of the XML file. They are not self-describing as this would be prohibitively expensive in terms of data use. Thus, the sender and receiver must have a copy of the same XML file for decoding a given message. Also, each message is defined by an ID that must be unique with a network.

Detailed documentation for [goby-acomms: libdccl \(Dynamic Compact Control - Language\)](#).

2.4 libqueue: Priority based message queuing

The goby-acomms queuing library (libqueue) interacts with both the application level process that handles decoding (either through libdccl or other CCL codecs) and the modem driver process that talks directly to the modem.

On the application side, libqueue provides the ability for the application level process to push (CCL or DCCL encoded) messages to various queues and receive messages from a remote sender that correspond to messages in the same queue (e.g. you have a queue for STATUS_MESSAGE that you can push messages to you and also receive other STATUS_MESSAGES on). The push feature is called by the application level process and received messages are signaled to all previous bound slots (see [Signal / Slot model for asynchronous events](#)).

On the driver side, libqueue provides the modem driver with data upon request. It chooses the data to send based on dynamic priorities (and several other configuration parameters). It will also pack several messages from the user into a single frame from the modem to fully utilize space (e.g. if the modem frame is 32 bytes and the user's data are in 16 byte DCCL messages, libqueue will pack two user frames for each modem frame). This packing and unpacking is transparent to the application side user. Note, however, that libqueue will *not* split a user's data into frames (like TCP/IP). If this functionality is desired, it must be provided at the application layer. Acoustic communications are too unpredictable to reliably stitch together frames.

Detailed documentation for [goby-acomms: libqueue \(Message Priority Queuing\)](#).

2.5 libmodemdriver: Modem driver

The goby-acomms Modem driver library (libmodemdriver) provides an interface from the rest of goby-acomms to the acoustic modem firmware. While currently the only driver available is for the WHOI Micro-Modem, this library is written in such a way that drivers for any acoustic modem that interfaces over a serial connection and can provide (or pro-

vide abstractions for) sending data directly to another modem on the link should be able to be written. Any one who is interested in writing a modem driver for another acoustic modem should get in touch with the goby project <<https://launchpad.net/goby>> and see [Writing a new driver](#).

Detailed documentation for [goby-acomms: libmodemdriver](#) (Driver to interact with modem firmware).

2.6 libamac: Medium Access Control (MAC)

The goby-acomms MAC library (libamac) handles access to the shared medium, in our case the acoustic channel. We assume that we have a single (frequency) band for transmission so that if vehicles transmit simultaneously, collisions will occur between messaging. Therefore, we use time division multiple access (TDMA) schemes, or "slotting". Networks with multiple frequency bands will have to employ a different MAC scheme or augment libamac for the frequency division multiple access (FDMA) scenario.

The MAC library provides two basic types of TDMA:

- Decentralized: Each node initiates its own transaction at the appropriate time in the TDMA cycle. This requires reasonably well synchronized clocks (any skew must be included in the time of the slot as a guard, so skews of less than 0.1 seconds are generally acceptable.). Two variants on the decentralized type are provided:
 - Fixed: The cycle of which nodes send when is fixed and can only be changed by updating all the nodes manually.
 - Auto-discovery: Basic peer discovery and subsequent expiry of peers after a long time of silence. Allows for more flexibility of operations.
- Centralized (also called "polling"): For legacy support, "polling" is also provided. This is a TDMA enforced by a central computer (the "poller"). The "poller" sends a request for data from a list of nodes in sequential order. The advantage of polling is that synchronous clocks are not needed and the MAC scheme can be changed on short notice by the topside operator. Clearly this only works with modem hardware capable of third-party mediation of transmission (such as the WHOI Micro-Modem).

Detailed documentation for [goby-acomms: libamac](#) (Medium Access Control).

2.7 Software concepts used in goby-acomms

2.7.1 Signal / Slot model for asynchronous events

The layers of goby-acomms use a signal / slot system for asynchronous events such as receipt of an acoustic message. Each signal can be connected ([goby::acomms::connect\(\)](#)) to one or more slots, which are functions or member functions matching the signature of the signal. When the signal is emitted, the slots are called in order

they were connected. To ensure synchronous behavior and thread-safety throughout goby-acomms, signals are only emitted during a call to a given library's API class `do_work` method (i.e. `goby::acomms::ModemDriverBase::do_work()`, `goby::acomms::QueueManager::do_work()`, `goby::acomms::MACManager::do_work()`).

For example, if I want to receive data from libqueue, I need to connect to the signal `QueueManager::signal_receive`. Thus, I need to define a function or class method with the same signature:

```
void receive_data(const goby::acomms::protobuf::ModemDataTransmission& msg);
```

At startup, I then connect the signal to the slot:

```
goby::acomms::connect(&q_manager.signal_receive, &receive_data);
```

If instead, I was using a member function such as

```
class MyApplication
{
public:
    void receive_data(const goby::acomms::protobuf::ModemDataTransmission&
msg);
};
```

I would call `connect` (probably in the constructor for `MyApplication`) passing the pointer to the class:

```
MyApplication::MyApplication()
{
    goby::acomms::connect(&q_manager.signal_receive, this, &
MyApplication::receive_data);
}
```

The `Boost.Signals` library is used without modification, so for details see http://www.boost.org/doc/libs/1_46_0/doc/html/signals.html. Member function binding is provided by `Boost.Bind` http://www.boost.org/doc/libs/1_46_0/libs/bind/bind.html

2.7.2 Google Protocol Buffers

`Google Protocol Buffers` are used as a convenient way of generating data structures (basic classes with accessors, mutators). They can also be serialized efficiently, though this is not generally used within goby-acomms. Protocol buffers messages are defined in `.proto` files that have a C-like syntax:

```
message MyMessage
{
    optional uint32 a = 1;
    required string b = 2;
    repeated double c = 3;
}
```


The identifier "optional" means a proper MyMessage object may or may not contain that field. "required" means that a proper MyMessage always contains such a field. "repeated" means a MyMessage can contain a vector of this field (0 to n entries). The sequence number "= 1" must be unique for each field and determines the serialized format on the wire. For our purposes it is otherwise insignificant. See <http://code.google.com/apis/protocolbuffers/docs/proto.html> for full details.

The .proto file is pre-compiled into a C++ class that is loosely speaking (see <http://code.google.com/apis/protocolbuffers/docs/reference/cpp-generated.html> for precise details):

```
class MyMessage : public google::protobuf::Message
{
public:
    MyMessage ();

    // set
    void set_a(unsigned a);
    void set_b(const std::string& b);
    void add_c(double c);

    // get
    unsigned a();
    std::string b();
    double c(int index);
    const RepeatedField<double>& c(); // RepeatedField ~= std::vector

    // has
    bool has_a();
    bool has_b();
    int c_size();

    // clear
    void clear_a();
    void clear_b();
    void clear_c();

private:
    unsigned a_;
    std::string b_;
    RepeatedField<double> c_; // RepeatedField ~= std::vector
}
```

Clearly the .proto representation is more compact and amenable to easy modification. All the Protocol Buffers messages used in goby-acomms are placed in the `goby::acomms::protobuf` namespace for easy identification. This doxygen documentation does not understand Protocol Buffers language so you will need to look at the source code directly for the .proto (e.g. [modem_message.proto](#)).

2.8 UML models

Model that describes the static structure of goby-acomms as a whole:

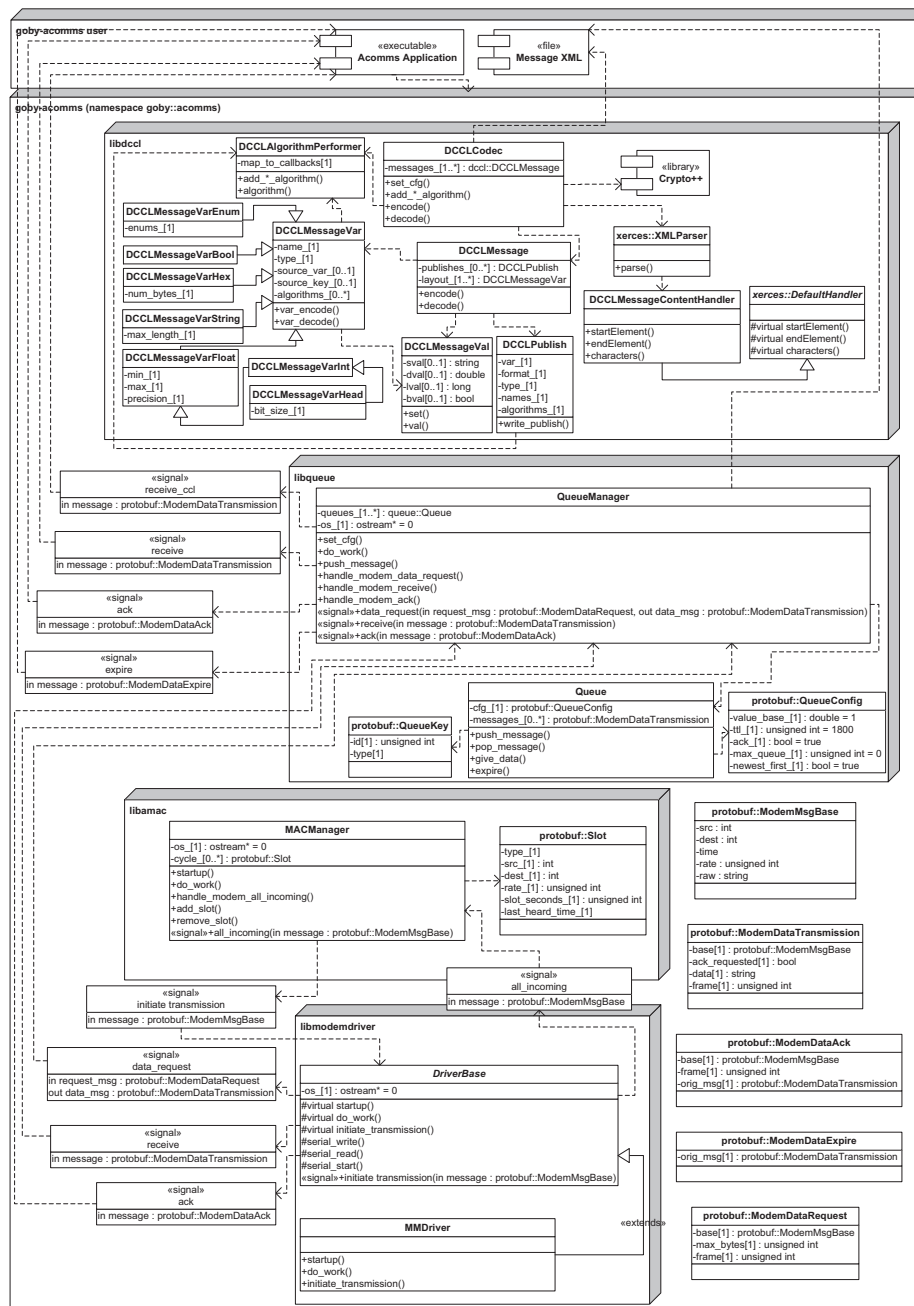


Figure 2: UML Structure Model of goby-acomms

Model that gives the sequence for sending a message with goby-acomms:

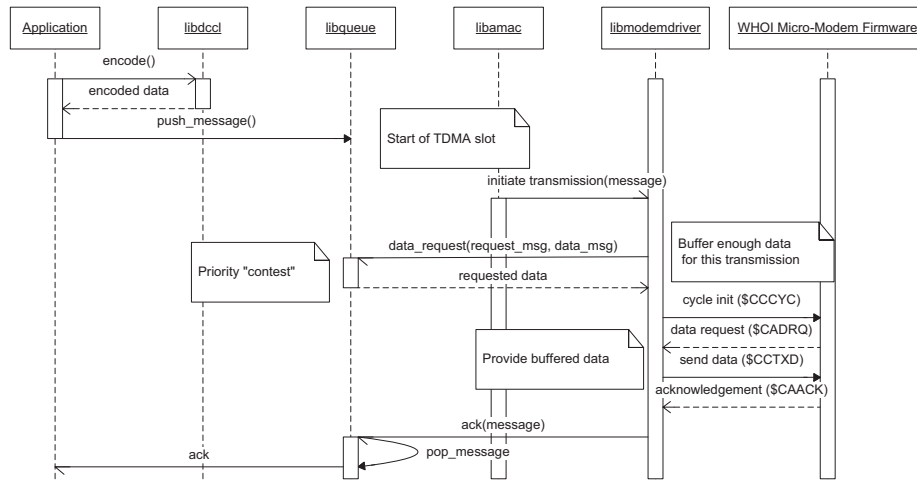


Figure 3: UML model that gives the sequence of calls required in sending a message using goby-acomms. The WHOI Micro-Modem is used as example firmware but the specific libmodemdriver-firmware interaction will depend on the acoustic modem used.

Model that shows the commands needed to start and keep goby-acomms running:

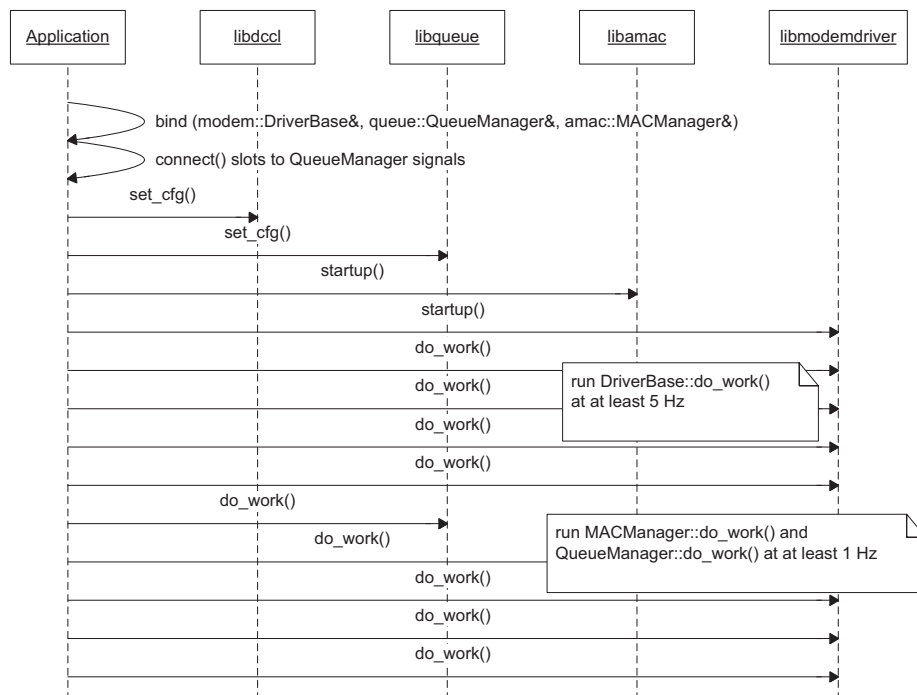


Figure 4: UML model that illustrates the set of commands needed to start up goby-acomms and keep it running.

3 goby-acomms: libdccl (Dynamic Compact Control Language)

Table of contents for libdccl:

- [Designing a message](#)
- [DCCL tag structure](#)
- [Interacting with the DCCLCodec](#)
- [Details of encoding/decoding scheme](#)
- [Encryption](#)
- [DCCL XML Tags Reference](#)

Return to [goby-acomms: Overview of Acoustic Communications Libraries](#).

3.1 Designing a message

Scenario 1: Send a string command to a vehicle:

We need to send an ASCII string command to an underwater vehicle. We thus make the `<layout>` section of the message contain a single message variable, a `<string>`. We know that a string uses a byte for each character and DCCL uses six header bytes (CCL id, DCCL id, time, src, dest, flags), making the `<max_length>` of our string 26 (32-6). We need some sort of name for our string to use internally when encoding and decoding this message, so we'll use `<name>` of "s_key" to stand for "string key".

We want to have the ability to use the lowest rate WHOI Micro-Modem message size, so we pick `<size>` to be 32. We have no other DCCL messages currently in the system so we start with an `<id>` of 1. Since this is a simple case we choose a "Simple" for our `<name>`.

See `dccl_simple.cpp` for the final result (`simple.xml`) and for an example of how to use this message.

Scenario 2: Send a more realistic command and receive a status message from the vehicle:

We want to be able to command our vehicle (to which we have assigned an ID number of "2") to go to a specific point on a local XY grid (meters from some known latitude / longitude), but no more than 10 kilometers from the datum. We also want to be able to turn the lights on or off, and send a short string for other new instructions. Finally, we need to be able to command a speed. Our vehicle can move no faster than 3 m/s, but its control is precise enough to handle hundredths of a m/s (wow!). It's probably easiest to make a table with our conditions:

message variable name	description	type	bounds
dest_id	id number of the vehicle we are commanding	integer	built into the header [0, 31]
goto_x	meters east to transit from datum	integer	[0, 10000]
goto_y	meters north to transit from datum	integer	[0, 10000]
lights_on	turn on the lights?	boolean	
new_instructions	string instructions	string	no longer than 10 characters
goto_speed	transit speed (m/s)	float	[0.00, 3.00]

Taking all this into account, we form the `<layout>` section of the first message (named GoToCommand) in the file two_message.xml (see two_message.cpp).

We choose `<id>` of 2 to avoid conflicting with the message from Scenario 1 (simple.xml) and a `<size>` of 32 bytes to again allow sending in the WHOI Micro-Modem rate 0 packet.

Now, for the second message in two_message.xml. We want to receive the vehicle's present position and its current health, which can either be "good", "low_battery" or "abort". We make a similar table to before:

message variable name	description	type	bounds
nav_x	current vehicle position (meters east of the datum)	integer	[0, 10000]
nav_y	current vehicle position (meters north of the datum)	integer	[0, 10000]
health	vehicle state	enumeration	good, low_battery, or abort

The resulting message, along with an example of how to use it, can be seen here: two_message.cpp.

You can run `analyze_dccl_xml` to view more information on your messages:

```
> analyze_dccl_xml /path/to/two_message.xml
```

When I ran the above command I got (omitting parts of the header we're not using to save space):

```
creating DCCLCodec using xml file: [examples/two_message/two_message.xml] and schema: [../../message_schema.xml]
schema must be specified with an absolute path or a relative path to the xml file location (not provided)
parsed file ok!
#####
detailed message summary:
#####
*****
```

```

message 2: {GoToCommand}
requested size {bytes} [bits]: {32} [256]
actual size {bytes} [bits]: {21} [167]
>>>> HEADER <<<<
destination (int):
size [bits]: [5]
[min, max] = [0,31]
>>>> LAYOUT (message_vars) <<<<
type (static):
size [bits]: [0]
value: "goto"
goto_x (int):
size [bits]: [14]
[min, max] = [0,10000]
goto_y (int):
size [bits]: [14]
[min, max] = [0,10000]
lights_on (bool):
size [bits]: [2]
new_instructions (string):
size [bits]: [80]
goto_speed (float):
size [bits]: [9]
[min, max] = [0,3]
precision: {2}
*****
*****
message 3: {VehicleStatus}
requested size {bytes} [bits]: {32} [256]
actual size {bytes} [bits]: {11} [84]
>>>> LAYOUT (message_vars) <<<<
nav_x (float):
size [bits]: [17]
[min, max] = [0,10000]
precision: {1}
nav_y (float):
size [bits]: [17]
[min, max] = [0,10000]
precision: {1}
health (enum):
size [bits]: [2]
values:{good,low_battery,abort}
*****

```

Besides validity checking, the most useful feature of `analyze_dccl_xml` is the calculation of the size (in bits) of each message variable. This lets you see which fields in the message are too big. To make fields smaller, tighten up bounds (depending on the type, increase `<min>`, decrease `<max>`, decrease `<precision>`, decrease `<max_length>`, decrease `<num_bytes>`, or decrease the number of `<enum>` `<value>` options).

3.2 DCCL tag structure

This section gives a brief outline of the tag structure of an XML file for defining a DCCL message. See [DCCL XML Tags Reference](#) for a full description of each tag.

DCCL root tags:

- `<?xml version="1.0" encoding="UTF-8"?>`: specifies that the file is XML; must be the first line of every message XML file.
- `<message_set>`
 - `<message>`

Children of `<message>` needed for normal `goby::acomms::DCCLCodec::encode` and `goby::acomms::DCCLCodec::decode`:

- `<message>`
 - `<name>`
 - `<id>`
 - `<size>`
 - `<repeat>`
 - `<header>`
 - * `<time>`
 - `<name>`
 - * `<src_id>`
 - `<name>`
 - * `<dest_id>`
 - `<name>`
 - `<layout>`
 - * `<static>`
 - `<name>`
 - `<value>`
 - * `<bool>`
 - `<name>`
 - `<array_length>`
 - * `<int>`
 - `<name>`
 - `<max>`
 - `<min>`
 - `<array_length>`
 - `<max_delta>`
 - * `<float>`
 - `<name>`
 - `<max>`
 - `<min>`
 - `<precision>`
 - `<array_length>`
 - `<max_delta>`
 - * `<string>`

- `<name>`
- `<max_length>`
- `<array_length>`
- * `<enum>`
 - `<name>`
 - `<value>`
 - `<array_length>`
- * `<hex>`
 - `<name>`
 - `<num_bytes>`
 - `<array_length>`

Children of `<message>` needed (in addition to those above) for publish/subscribe architecture methods `goby::acomms::DCCLCodec::pubsub_encode` and `goby::acomms::DCCLCodec::pubsub_decode` (these tags are ignored for calls to `goby::acomms::DCCLCodec::encode` and `goby::acomms::DCCLCodec::decode`):

- `<message>`
 - `<header>`
 - * `<src_id>`, `<dest_id>`, or `<time>`
 - `<src_var>`
 - `<layout>`
 - * `<int>`, `<hex>`, `<string>`, `<float>`, `<enum>`, or `<bool>`
 - `<src_var>`
 - `<trigger>`
 - `<trigger_var>`
 - `<trigger_time>`
 - `<on_receipt>`
 - * `<publish>`
 - `<publish_var>`
 - `<format>`
 - `<message_var>`
 - `<all>`

3.3 Interacting with the DCCLCodec

Using the `goby::acomms::DCCLCodec` is a fairly straightforward endeavor. First you need to instantiate a copy of this object with the configuration you want (including the XML files you want to be able to use):

```
goby::acomms::DCCLCodec dccl(&std::clog);
goby::acomms::protobuf::DCCLConfig cfg;
cfg.add_message_file()->set_path("path/to/file.xml");
cfg.set_modem_id(1); // unique id 1-31 for each platform
dccl.set_cfg(cfg);
```


Then, to encode a message, fill up `std::maps` of `dccl::MessageVal` where the key of the map (i.e. the `it->first` if `it` is the map iterator) is the `<name>` of each message variable, and the value (`it->second`) is the quantity you wish to encode. All reasonable type conversions will be made by DCCL using `dccl::MessageVal` (doubles to `<int>`, for example), but the most predictable (and fastest) results will be gained by using the following mapping between DCCL message variable types and C++ types:

DCCL Message Variable Type	C++ Type	Example
<code><int></code>	<code>long</code>	421
<code><float></code>	<code>double</code>	42.1
<code><hex></code>	<code>std::string</code>	"abc23" or "ABC23" (case does not matter)
<code><enum></code>	<code>std::string</code>	"ON" (case matters, this will not match <code><value>on</value></code> , but will match <code><value>O-N</value></code>)
<code><string></code>	<code>std::string</code>	"i am hungry" (case matters, string cannot contain any null characters in the middle, i.e. <code>'\0'</code>)
<code><bool></code>	<code>bool</code>	true

After filling up the `std::maps`, pass pointers to the maps to `goby::acomms::DCCLCodec::encode` along with a reference to a string in which to store the result:

```
std::map<std::string, dccl::MessageVal> vals;

// code to insert values into map
// ...
//

// store the result here, using the string as a byte container
std::string bytes;

dccl.encode(id, bytes, vals);
```

`bytes` will now contain the encoded message in the form of a byte string (each char will contain a single byte of the message).

You may now send this message through whatever channel you would like, or pass it to the `goby::acomms::QueueManager` to queue for sending later.

To decode a message (stored in `bytes` as a byte string), simply pass `hex` as a reference along with pointers to the maps to store the results. Based on the maps you provide, values will be cast and stored as the best fit.

```
std::map<std::string, std::string> vals;

dccl.decode(1, bytes, vals);
```

For line by line interaction with the `goby::acomms::DCCLCodec` and for advanced use, investigate the code examples given in the Examples column of this [table](#).

3.4 Encryption

Encryption of all messages can be enabled by providing a secret passphrase to the `goby::acomms::protobuf::DCCLConfig` object passed to `goby::acomms::DCCLCodec::set_cfg()`. All parties to the communication must have the same secret key.

DCCL provides AES (Rijndael) encryption for the body (`<layout>`) of the message. The header, which is sent in plain text, is hashed to form an initialization vector (IV), and the passphrase is hashed using SHA-256 to form the cipher key.

AES is considered secure and is used for United States top secret information.

3.5 Details of encoding/decoding scheme

We may want to know the actual layout of the binary/hex message. For the first of the two messages in `two_message.xml`, we can run `analyze_dccl_xml` to find the sizes of each message variable. The calculated sizes are used to determine the boundaries (which are by bit, not by byte) when the message is packed. Each field is placed in the order it is declared in the XML file such that the message is as follows (where left to right is the same as reading the hex string from left to right):

```
[[header][0 {1}][goto_x {14}][goto_y {14}][lights_on {2}][new_instructions {80}][goto_speed {9}]]
```

where `[0 {1}]` means zero fill the message to the closest whole byte (15 bytes = 120 bits minus 119 for other fields = 1). Byte boundaries are dissolved and encoded as a string "ABCDEF..." where the most significant byte (MSB, or leftmost 8 bits) is 0xAB, second MSB is 0xCD, etc. Encoding and decoding are done by functions available in `tes_utils.h`. You will notice that the resulting size is 15 bytes which is short of the 32 bytes specified for the `<size>`. This is because the `<size>` is a maximum size before a warning is generated, not the actual size always returned by the `DCCLCodec`. This allows `libqueue` to pack multiple messages together to form a modem message frame and thus fit a nearly optimal amount of data into each modem packet. For example, you could now fit one of the `GoToCommand` messages *and* another message up to 17 bytes in a single 32 byte WHOI Micro-Modem rate 0 frame.

The encoding of each `message_var` is done as an unsigned integer, with the exception of strings, which are store as ASCII. The value 0 (all bits zero) always indicates "not specified" or "Not a Number" (nan). This means that the user did not specify any value for this field, specified a value causing overflow (`<int>` or `<float>` greater than `<max>` or less than `<min>`), or provided a value for an `<enum>` that did not match any of the enumerate's `<value>` options. Along with this rule, the method for encoding and decoding is summarized below:

message_var	size (bits)	encode	decode
static	0	not sent	not sent
bool	2	$val_{enc} = (val == \text{to_lower}(\text{"true"}) \text{ OR } val \neq 0) ? 2 : 1$	$val = (val_{enc} == 2) ? \text{true} : \text{false}$
enum	$\text{ceil}(\log_2(\text{total_enums} + 1))$	$val_{enc} = 1 + \text{index to array of enum values based on order they were declared}$	$val = \text{value at index } val_{enc} - 1$
string	$\text{max_length} \cdot 8$	string is filled at end with zeros ('0') to max_length then encoded using ASCII byte values	ASCII, ignoring null termination chars (if any)
int	$\text{ceil}(\log_2(\text{max} - \text{min} + 2))$	$val_{enc} = \text{round}(val - \text{min}, 0) + 1$	$val = val_{enc} + \text{min} - 1$
float	$\text{ceil}(\log_2((\text{max} - \text{min}) \cdot 10^{\text{precision}} + 2))$	$val_{enc} = \text{round}((val - \text{min}) \cdot 10^{\text{precision}}, 0) + 1$	$val = (val_{enc} - 1) / 10^{\text{precision}} + \text{min}$
hex	$\text{num_bytes} \cdot 8$	$val_{enc} = val$	$val = val_{enc}$

where val is the original (and decoded) value, val_{enc} is the encoded value, min , max , max_length , precision are the contents of the `<min>`, `<max>`, `<max_length>`, and `<precision>` tags, respectively, and $\text{round}(x, 0)$ means round x to the nearest integer.

An example. Say you have the following XML file:

```
...
<id>1</id>
<header>
  <src_id>
    <name>Src</name>
  </src_id>
  <dest_id>
    <name>Dest</name>
  </dest_id>
</header>
<layout>
  <bool>
    <name>B</name>
  </bool>
  <enum>
    <name>E</name>
    <value>cat</value>
    <value>dog</value>
    <value>mouse</value>
  </enum>
  <string>
    <name>S</name>
    <max_length>4</max_length>
  </string>
```

```

<int>
  <name>I</name>
  <max>100</max>
  <min>-50</min>
</int>
<float>
  <name>F</name>
  <max>100</max>
  <min>-50</min>
  <precision>2</precision>
</float>
</layout>
...

```

The header is always the same size and is given by (sizes shown in bits):

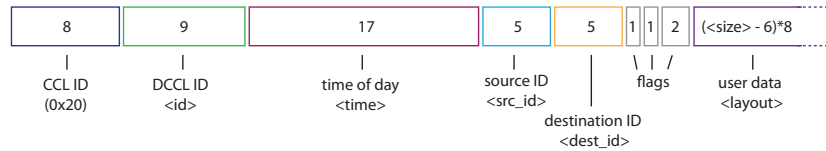


Figure 5: Header for all DCCL messages. Sizes shown in bits

Next, the size of each *message_var* (in the *<layout>* section) and its encoded value in decimal and binary are calculated for an example set of inputs:

message_var	example <i>val</i>	size (bits)	<i>val_{enc}</i> (decimal)	<i>val_{enc}</i> (binary)
B	true	2	2	10
E	cat	2	1	01
S	FAT	32	1178686464	01000110 01000001 01010100 00000000
I	34	8	85	01010101
F	-22.49	14	2752	00101011000000

and thus the whole message (zero padded from the most significant bits to the closest byte) sent would be

00000010 01010001 10010000 01010101 00000000 00010101 01001010 11000000

or

0x0251905500154AC0

plus the header, which in this case is 0x2000AA300230, so the full message sent is

0x2000AA3002300251905500154AC0

3.6 DCCL XML Tags Reference

3.6.1 <all>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <on_receipt>
      <publish>
        ...
        <all />
      </publish>
    </on_receipt>
  </message>
</message_set>
```

Description: Equivalent to [<message_var>](#) for all the *message_vars* in the message. This is a shortcut when you want to publish all the data in a human readable string. [optional, one allowed].

3.6.2 <array_length>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <int>
        <array_length>5</array_length>
      </int>
      <string>
        <array_length>5</array_length>
      </string>
      <float>
        <array_length>5</array_length>
      </float>
      <bool>
        <array_length>5</array_length>
      </bool>
      <hex>
        <array_length>5</array_length>
      </hex>
      <enum>
        <array_length>5</array_length>
      </enum>
    </layout>
  </message>
</message_set>
```

Description: If larger than 1, this makes an array of values instead of a single value.

3.6.3 <bool>

```
\b Syntax:
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      ...
      <bool algorithm="">
        <src_var></src_var>
        <name></name>
      </bool>
    </layout>
  </message>
</message_set>
```

Description: a boolean (true or false) *message_var* The optional parameter *algorithm* allows you to perform certain algorithms on the data before encoding. See `libdccl/examples/test/test.cpp` [optional, one or more allowed].

3.6.4 <dest_id>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <header>
      <dest_id>
        <name></name>
        <src_var></src_var>
      </dest_id>
    </header>
  </message>
</message_set>
```

Description: Allows setting a name other than the default ("_dest_id") and a [<src_var>](#) for the destination id field of the header.

3.6.5 <destination_var>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    ...
    <!-- OUT_MESSAGE: destination=abcd,... -->
    <destination_var key="destination">OUT_MESSAGE</destination_var>
  </message>
</message_set>
```

Description: deprecated. Use [<dest_id>](#) instead.

architecture variable to find where this message should be sent. Specify attribute "key=" to specify a substring to look for within the value of this architecture variable. For example, if `COMMAND` contained the string `Destination=3` and you want this message sent to modem_id 3, then you should set `key=Destination` to properly parse that string. [optional: default is 0 (broadcast), one allowed].

3.6.6 <enum>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      ...
      <enum algorithm="">
        <src_var></src_var>
        <name></name>
        <value></value>
        <value></value>
        <value></value>
      </enum>
    </layout>
  </message>
</message_set>
```

Description: an enumeration *message_var* [optional, one or more allowed].

3.6.7 <float>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      ...
      <float algorithm="">
        <src_var></src_var>
        <name></name>
        <max></max>
        <min></min>
        <precision></precision>
      </float>
    </layout>
  </message>
</message_set>
```

Description: a decimal valued real number *message_var* [optional, one or more allowed].

3.6.8 <format>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <on_receipt>
      <publish>
        ...
        <format>A=%1%,B=%2%</format>
      </publish>
    </on_receipt>
  </message>
</message_set>
```

Description: a string conforming to the format string syntax of the `boost::format` library. This field will specify the format of the string published to the architecture variable defined in `<publish_var>`. At its simplest, it is a string of incrementing numbers surrounded by `%`. Or, instead, you may also use a `printf` style string, using `%d` for int `message_var`, `%lf` for floats, and `%s` for strings, bools, enums and hex. [optional: default is `name1=%1%,name2=%2%,name3=%3%`, where `name1` is the name of the first `<message_var>` field to follow, `name2` is the second, etc. exception: default is `%1%` if only a single `<message_var>` defined. one allowed].

3.6.9 <header>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <header>
      <time></time>
      <src_id></src_id>
      <dest_id></dest_id>
    </header>
    <layout>
      ...
    </layout>
  </message>
</message_set>
```

Description: holds tags allowing some parts of the DCCL header to be referenced by a new name (other than the defaults: `"_time"`, `"_src_id"`, `"_dest_id"`) at encode and decode time. See `<src_id>`, `<dest_id>`, `<time>`.

3.6.10 <hex>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      ...
      <hex algorithm="">
        <src_var></src_var>
        <name></name>
      </hex>
    </layout>
  </message>
</message_set>
```



```

        <num_bytes></num_bytes>
    </hex>
</layout>
</message>
</message_set>

```

Description: a message variable represented pre-encoded hexadecimal to add to the message. This field is useful if another source is encoding part or all of a DCCL message. [optional, one or more allowed].

3.6.11 <id>

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    ...
    <id>23</id>
  </message>
</message_set>

```

Description: an unsigned nine bit integer (0-511)g that identifies this message within a network. very similar to the CCL identifier, but for DCCL messages. The CCL identifier occupies the most significant byte (MSB) of the message followed by this id which takes the part of the second MSB (two flags, multmessage and broadcast, use the remainder of the second MSB). *This must be unique within a network.* [mandatory, one allowed]

3.6.12 <int>

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      ...
      <int algorithm="">
        <src_var></src_var>
        <name></name>
        <max></max>
        <min></min>
      </int>
    </layout>
  </message>
</message_set>

```

Description: an integer *message_var* [optional, one or more allowed].

3.6.13 <layout>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    ...
    <layout>
      <int></int>
      <string></string>
      <float></float>
      <bool></bool>
      <hex></hex>
      <static></static>
      <enum></enum>
    </layout>
  </message>
</message_set>
```

Description: defines the message structure itself (what fields [the message variables or *message_vars*] the message contains and how they are to be encoded). [mandatory, one allowed].

3.6.14 <max_length>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <string>
        ...
        <max_length>10</max_length>
      </string>
    </layout>
  </message>
</message_set>
```

Description: the length of the string value in this field. Longer strings are truncated. <max_length>4</max_length> means "ABCDEFGH" is sent as "AB-CD". [mandatory, one allowed].

3.6.15 <max>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <int>
        <max>100</max>
      </int>
      <float>
        <max>100</max>
      </float>
    </layout>
  </message>
</message_set>
```

Description: the maximum value this field can take. [mandatory, one allowed].

3.6.16 <max_delta>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <int>
        <max_delta>10</max_delta>
      </int>
      <float>
        <max_delta>100</max_delta>
      </float>
    </layout>
  </message>
</message_set>
```

Description: if specified, delta-difference encoding is done of the [<repeat>](#) ed message or the values in the array (for [<array_length>](#) > 1). The first value is used as a key for the remaining values which are sent as a difference to this key. The number specified here is the maximum expected difference between the first value (key) and any of the remaining values in the message. [optional, if omitted, delta-difference encoding is not performed].

3.6.17 <message_set>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message></message>
  <message></message>
  <message></message>
</message_set>
```

Description: the root element. All XML files must have a single root element. Since we are define a set of messages (one or more per file), this is a logical choice of name for the root element. [mandatory, one allowed].

3.6.18 <message_var>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <on_receipt>
      <publish>
        ...
      <message_var></message_var>
    </on_receipt>
  </message>
</message_set>
```

```

        <message_var></message_var>
    </publish>
</on_receipt>
</message>
</message_set>

```

Description: the name ([<name>](#) above) of a *message_var* contained in this message (i.e. an [<int>](#), [<bool>](#), etc.) the values of these fields upon receipt of a message will be used to populate the format string and the result will be published to [<publish_var>](#). The optional parameter *algorithm* allows you to perform certain algorithms on the data after receipt before publishing. [mandatory unless [<all>](#) used, one or more allowed].

3.6.19 <message>

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name></name>
    <id></id>
    <size></size>
    <layout></layout>
    <destination_var key=""></destination_var>
    <trigger>publish</trigger>
    <trigger_var mandatory_content=""></trigger_var>
    <!-- OR -->
    <trigger>time</trigger>
    <trigger_time></trigger_time>
    <on_receipt></on_receipt>
    <queuing></queuing>
  </message>
  <message>
    ...
  </message>
</message_set>

```

Description: defines the start of a message. [mandatory, one or more allowed].

3.6.20 <min>

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <int>
        <min>-100</min>
      </int>
      <float>
        <min>-100</min>
      </float>
    </layout>
  </message>
</message_set>

```

Description: the minimum value this field can take. [mandatory, one allowed].

3.6.21 <name>

Syntax:

```
<message_set>
  <message>
    ...
    <name>STATUS_REPORT</name>
  </message>
</message_set>
```

or

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <int>
        <name>parameter1</name>
      </int>
      <string>
        <name>parameter2</name>
      </string>
      <float>
        <name>parameter3</name>
      </float>
      <bool>
        <name>parameter4</name>
      </bool>
      <hex>
        <name>parameter5</name>
      </hex>
      <static>
        <name>parameter6</name>
      </static>
      <enum>
        <name>parameter7</name>
      </enum>
    </layout>
  </message>
</message_set>
```

Description: (as child of [<message>](#)): a human readable name for the message. [mandatory, one allowed]

(as child of [<int>](#), [<hex>](#), [<string>](#), [<float>](#), [<enum>](#), or [<bool>](#)): the name of this *message_var*. [mandatory, one allowed].

3.6.22 <num_bytes>

the number of bytes for this field. The string provided should be twice as many characters as [<num_bytes>](#) since each character of a hexadecimal string is one nibble (4 bits or 1/2 byte). [mandatory, one allowed].

3.6.23 `<on_receipt>`**Syntax:**

```
<?xml version="1.0" encoding="UTF-8"?>
```

Description: contains the various `<publish>` options for publishing parts of a message upon receipt when using the publish-subscribe architecture method (goby::acomms::DCCLCodec::encode_to_publish). [optional, one allowed].

3.6.24 `<precision>`**Syntax:**

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <float>
        <precision>3</precision>
      </float>
    </layout>
  </message>
</message_set>
```

Description: an integer that specifies the number of decimal digits to preserve. - Negatives are allowed. For example, `<precision>2</precision>` rounds 1042.1234 to 1042.12; `<precision>-1</precision>` rounds 1042.1234 to 1.04e3. [mandatory, one allowed].

3.6.25 `<publish>`**Syntax:**

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <on_receipt>
      <publish>
        <publish_var></publish_var>
        <format></format>
        <message_var></message_var>
      </publish>
    </on_receipt>
  </message>
</message_set>
```

Description: defines a single output value upon receipt of a message. Any number of publishes containing any subset of the *message_vars* can be specified. [mandatory, one or more allowed].

3.6.26 <publish_var>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <on_receipt>
      <publish>
        <publish_var type="string">OUT_STATUS_REPORT</publish_var>
      </publish>
    </on_receipt>
  </message>
</message_set>
```

Description: the name of the architecture variable to publish to. If desired, a format string is allowed here as well (e.g. %1%_NAV_X will fill %1% with the first *message_var*). See the <format> tag description for more info. [mandatory, one allowed].

3.6.27 <repeat>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    ...
    <repeat/>
  </message>
</message_set>
```

Description: Including this empty tag tells DCCL to make as many copies of the message structure defined in <layout> as will fit in the message <size>. No message will be sent until the message is full. For example, if the message is 32 bytes and the layout is 8 bytes, three copies of the message will be stored before sending ($32 - 6 - 3 \times 8 = 0$). That is, three messages will be triggered, packed and sent as a single DCCL message. [optional, if omitted only a single copy is made]. *If <repeat> is specified, <array_length> must omitted for all message}. That is, you cannot have repeated messages that contain arrays.*

3.6.28 <size>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <size>32</size>
  </message>
</message_set>
```

Description: the size of the message in bytes. There are eight bits (binary digits) to a byte. Use N here for messages passed to the Micro-Modem where N is the desired

Micro-Modem frame size (N=32, 64, or 256 depending on the rate). If the [<layout>](#) of the message exceeds this size, DCCL will exit on startup with information about sizes, from which you can remove or reduce the size of certain *message_vars*.

3.6.29 [<src_id>](#)

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <header>
      <src_id>
        <name></name>
        <src_var></src_var>
      </src_id>
    </header>
  </message>
</message_set>
```

Description: Allows setting a name other than the default ("_src_id") and a [<src_var>](#) for the source id field of the header.

3.6.30 [<src_var>](#)

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <!-- OUT_MESSAGE: ...,parameter1=123,parameter2=abc,parameter3=3.42,
        parameter4=true,parameter5=ON
        SOME_OTHER_HEX: 24bbc231 -->
      <int>
        <src_var key="parameter1">OUT_MESSAGE</src_var>
      </int>
      <string>
        <src_var key="parameter2">OUT_MESSAGE</src_var>
      </string>
      <float>
        <src_var key="parameter3">OUT_MESSAGE</src_var>
      </float>
      <bool>
        <src_var key="parameter4">OUT_MESSAGE</src_var>
      </bool>
      <hex>
        <src_var>SOME_OTHER_HEX</src_var>
      </hex>
      <enum>
        <src_var key="parameter5">OUT_MESSAGE</src_var>
      </enum>
    </layout>
  </message>
</message_set>
```


Description: the architecture variable from which to pull the value of this field. [optional if <trigger>publish</trigger>: default is trigger_var; mandatory if <trigger>time</trigger>, one allowed].

3.6.31 <static>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <static algorithm="">
        <name></name>
        <value></value>
      </static>
    </layout>
  </message>
</message_set>
```

Description: a *message_var* that is not actually sent with the message but can be used to include in received messages (*publishes*). [optional, one or more allowed].

3.6.32 <string>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <string algorithm="">
        <src_var></src_var>
        <name></name>
        <max_length></max_length>
      </string>
    </layout>
  </message>
</message_set>
```

Description: an ASCII string *message_var* [optional, one or more allowed].

3.6.33 <time>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <header>
      <time>
        <name></name>
        <src_var></src_var>
      </time>
    </header>
  </message>
</message_set>
```

```

    </header>
  </message>
</message_set>

```

Description: Allows setting a name other than the default ("`_time`") and a `<src_var>` for the time field of the header. Note that the value of the `<src_var>` should be a UNIX timestamp (seconds since 1/1/1970 00:00:00). In the DCCL encoding, this reduced to seconds since the start of the day, with precision of one second. Upon decoding, assuming the message arrives within twelve hours of its creation, it is properly restored to a full UNIX time.

3.6.34 `<trigger_var>`

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <trigger>publish</trigger>
    <trigger_var mandatory_content="">OUT_MESSAGE</trigger_var>
  </message>
</message_set>

```

Description: used if `<trigger>publish</trigger>`, this field gives the architecture variable that publishes to will trigger the creation of this message [mandatory if and only if `<trigger>publish</trigger>`]. optional attribute `mandatory_content` specifies a string that must be a substring of the contents of the trigger variable in order to trigger the creation of a message. For example, if you wanted to create a certain message every time `COMMAND` contained the string `CommandType=GoTo...` but no other time, you would specify `mandatory_content="CommandType=GoTo"` within this tag.

3.6.35 `<trigger_time>`

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <trigger>time</trigger>
    <trigger_time>60</trigger_time>
  </message>
</message_set>

```

Description: used if `<trigger>time</trigger>`, this field gives the time interval used to create this message. For example, a value of `<trigger_time>10</trigger_time>` would mean a message should be created every ten seconds. [mandatory if and only if `<trigger>time</trigger>`].

3.6.36 `<trigger>`

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <trigger>publish</trigger>
    <trigger_var mandatory_content=""></trigger_var>
    <!-- OR -->
    <trigger>time</trigger>
    <trigger_time></trigger_time>
  </message>
</message_set>
```

Description: how the message is created. Currently this field must take the value "publish" (meaning a message is created on a publish event to a certain architecture variable) or "time" (a message is created on a certain time interval). [mandatory, one allowed]

3.6.37 <value>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <static>
        <value>my static value</value>
      </static>
    </layout>
  </message>
</message_set>
```

or

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <layout>
      <enum>
        <value>ON</value>
        <value>OFF</value>
        <value>IN_BETWEEN</value>
      </enum>
    </layout>
  </message>
</message_set>
```

Description: (as child of [<static>](#)): the value of this static variable. [mandatory, one allowed].

(as child of [<enum>](#)): a possible value (string) the enum can take. Any number of values can be specified. [mandatory, one or more allowed].

4 goby-acomms: libqueue (Message Priority Queuing)

Table of Contents for libqueue:

- [Understanding dynamic priority queuing](#)
- [Queuing tag structure](#)
- [Interacting with the QueueManager](#)
 - [Instantiate and configure](#)
 - [Signals and \(application layer\) slots](#)
 - [Operation](#)
- [Queuing XML Tags Reference](#)

Return to [goby-acomms: Overview of Acoustic Communications Libraries](#).

4.1 Understanding dynamic priority queuing

Each queue has a base value (V_{base}) and a time-to-live (t_{tl}) that create the priority ($P(t)$) at any given time (t):

$$P(t) = V_{base} \frac{(t - t_{last})}{t_{tl}}$$

where t_{last} is the time of the last send from this queue.

This means for every queue, the user has control over two variables (V_{base} and t_{tl}). V_{base} is intended to capture how important the message type is in general. Higher base values mean the message is of higher importance. The t_{tl} governs the number of seconds the message lives from creation until it is destroyed by libqueue. The t_{tl} also factors into the priority calculation since all things being equal (same V_{base}), it is preferable to send more time sensitive messages first. So in these two parameters, the user can capture both overall value (i.e. V_{base}) and latency tolerance (t_{tl}) of the message queue.

The following graph illustrates the priority growth over time of three queues with different t_{tl} and V_{base} . A message is sent every 100 seconds and the queue that is chosen is marked on the graph.

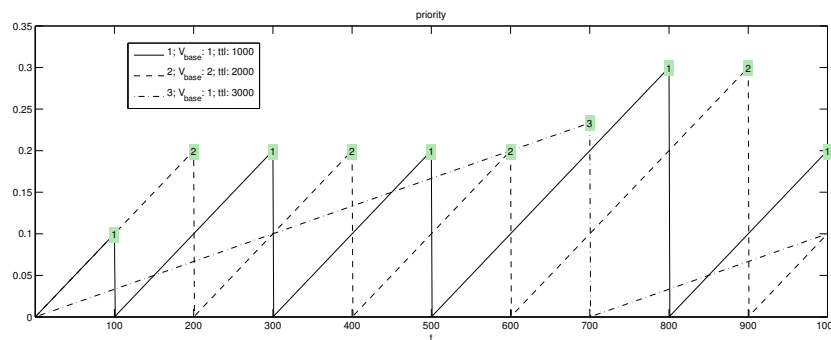


Figure 6: Graph of the growth of queueing priorities for libqueue for three different queues. A message is sent every 100 seconds from the %queue with the highest priority (numbered on the graph).

4.2 Queuing tag structure

This section gives a brief outline of the tag structure of an XML file for defining the queuing for a DCCL message. The tags fit in the same file used for DCCL encoding and decoding; see [DCCL tag structure](#) for more information.

- `<?xml version="1.0" encoding="UTF-8"?>`: specifies that the file is XML; must be the first line of every message XML file.
- `<message_set>`
 - `<message>`
 - * `<queuing>`
 - `<ack>`
 - `<blackout_time>`
 - `<max_queue>`
 - `<newest_first>`
 - `<ttd>`
 - `<value_base>`

4.3 Interacting with the QueueManager

4.3.1 Instantiate and configure

The `goby::acomms::QueueManager` is configured similarly to the `goby::acomms::DCCLCodec`. You need to add queues to the `goby::acomms::protobuf::QueueManagerConfig` which is done by feeding it either XML files or `goby::acomms::protobuf::QueueManagerConfig` objects. You also need to set a unique identification number in the range 1-31 for this platform (the "modem_id"). This is analogous to a MAC address.

```
goby::acomms::QueueManager q_manager(&std::clog);
goby::acomms::protobuf::QueueManagerConfig cfg;
cfg.add_message_file()->set_path("path/to/file.xml");
cfg.set_modem_id(1); // unique id 1-31 for each platform
q_manager.set_cfg(cfg);
```

4.3.2 Signals and (application layer) slots

Then, you need to do a few more initialization chores:

- Connect (using `goby::acomms::connect()`) QueueManager signals to your application layer slots (functions or member functions that match the signal's signature). You do not need to connect a slot to a given signal if you do not need its functionality. See [Signal / Slot model for asynchronous events](#) for more on using signals and slots:
 - Received DCCL data: `goby::acomms::QueueManager::signal_receive`
 - Received CCL data: `goby::acomms::QueueManager::signal_receive_ccl`

- Received acknowledgements: [goby::acomms::QueueManager::signal_ack](#)
- Expired messages (ttl exceeded): [goby::acomms::QueueManager::signal_expire](#)
- Additional advanced features
 - Connect a slot to learn every time a queue size changes due to a new message being pushed or a message being sent: [goby::acomms::QueueManager::signal_queue_size_change](#)
 - Request that a queue be *on_demand*, that is, request data from the application layer every time the modem layer requests data (DCCL messages only). This effectively bypasses the queue and forwards the modem's data request to the application layer. Use this for sending highly time sensitive data which needs to be encoded immediately prior to sending. Set the [ON_DEMAND manipulator](#) for that particular XML file in [goby::acomms::protobuf::QueueManagerConfig](#). You must also connect a slot that will be executed each time data is request to the signal [goby::acomms::QueueManager::signal_data_on_demand](#).

4.3.3 Operation

At this point the [goby::acomms::QueueManager](#) is ready to use. At the application layer, new messages are pushed to the queues for sending using [goby::acomms::QueueManager::push_message](#). Each queue is identified by a unique [goby::acomms::protobuf::QueueKey](#), which is simply the identification number of the queue (`<id>` for DCCL queues or the decimal representation of the first byte of a CCL message) and the queue type ([goby::acomms::protobuf::QUEUE_DCCL](#) or [goby::acomms::protobuf::QUEUE_CCL](#)).

At the driver layer, messages are requested using [goby::acomms::QueueManager::handle_modem_data_request](#), incoming messages are published using [goby::acomms::QueueManager::handle_modem_receive](#), and acknowledgements are given using [goby::acomms::QueueManager::handle_modem_ack](#). If using the goby-acomms drivers (i.e. some class derived from [goby::acomms::ModemDriverBase](#)), simply call [goby::acomms::bind](#) ([ModemDriverBase&](#), [QueueManager&](#)) and these methods (slots) will be invoked automatically from the proper driver signals.

You must run [goby::acomms::QueueManager::do_work\(\)](#) regularly (faster than 1 Hz; 10 Hertz is good) to process expired messages ([goby::acomms::QueueManager::signal_expire](#)). All other signals are emitted in response to a driver level signal (and thus are called during a call to [goby::acomms::ModemDriverBase::do_work\(\)](#)).

4.4 Queuing XML Tags Reference

4.4.1 <ack>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<message_set>
  <message>
    <queuing>
      ...
      <ack>true</ack>
    </queuing>
  </message>
</message_set>

```

Description: boolean flag (1=true, 0=false) whether to request an acoustic acknowledgment on all sent messages from this field. If omitted, default of 1 (true, request ack) is used. Note that if a message has a destination of 0 (broadcast), an ack will never be requested regardless of the value set here.

4.4.2 <blackout_time>

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <queuing>
      ...
      <blackout_time>0</blackout_time>
    </queuing>
  </message>
</message_set>

```

Description: time in seconds after sending a message from this queue for which no more messages will be sent. Use this field to stop an always full queue from hogging the channel. If omitted, default of 0 (no blackout) is used.

4.4.3 <max_queue>

Syntax:

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <queuing>
      ...
      <max_queue>1</max_queue>
    </queuing>
  </message>
</message_set>

```

Description: number of messages allowed in the queue before discarding messages. If [<newest_first>](#) is set to true, the oldest message in the queue is discarded to make room for the new message. Otherwise, any new messages are discarded until the space in the queue opens up.

4.4.4 <newest_first>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <queuing>
      ...
      <newest_first>true</newest_first>
    </queuing>
  </message>
</message_set>
```

Description: boolean flag (1=true=FILO, 0=false=FIFO) whether to send newest messages in the queue first (FILO) or not (FIFO).

4.4.5 <value_base>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <queuing>
      ...
      <value_base>10</value_base>
    </queuing>
  </message>
</message_set>
```

Description: base priority value for this message queue. priorities are calculated on a request for data by the modem (to send a message). The queue with the highest priority (and isn't in blackout) is chosen. The actual priority (P) is calculated by $P(t) = V_{base} \frac{(t - t_{last})}{ttl}$ where V_{base} is the value set here, t is the current time (in seconds), t_{last} is the time of the last send from this queue, and ttl is the <ttl>. Essentially, a message with low <ttl> will become effective quickly again after a sent message (the priority line grows faster). See [Understanding dynamic priority queuing](#) for further discussion.

4.4.6 <ttl>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <queuing>
      ...
      <ttl>120</ttl>
    </queuing>
  </message>
</message_set>
```

Description: the time in seconds a message lives after its creation before being discarded. This time-to-live also factors into the growth in priority of a queue. see <value_base> for the main discussion on this. 0 is a special value indicating infinite life (i.e. <ttl>0</ttl> is effectively the same as <ttl> ∞</ttl>).

4.4.7 <queuing>

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <queuing>
      <ack></ack>
      <blackout_time></blackout_time>
      <max_queue></max_queue>
      <newest_first></newest_first>
      <priority_base></priority_base>
      <priority_time_const></priority_time_const>
    </queuing>
  </message>
</message_set>
```

Description: Represents the XML embodiment of the [goby::acomms::protobuf::Queue-Config](#).

5 goby-acomms: libmodemdriver (Driver to interact with modem firmware)

Table of contents for libmodemdriver:

- [Abstract class: ModemDriverBase](#)
- [WHOI Micro-Modem Driver: MMDriver](#)
- [Writing a new driver](#)

Return to [goby-acomms: Overview of Acoustic Communications Libraries](#).

5.1 Abstract class: ModemDriverBase

[goby::acomms::ModemDriverBase](#) defines the core functionality for an acoustic modem. It provides

- **A serial or serial-like (over TCP or UDP) reader.** This is an instantiation of an appropriate derivative of the [goby::util::LineBasedInterface](#) class which reads the physical interface (serial, TCP or UDP) to the acoustic modem. - The data (assumed to be ASCII lines offset by a delimiter such as NMEA0183 or the Hayes command set [AT]) are read into a buffer for use by the [goby::acomms::ModemDriverBase](#) derived class (e.g. [goby::acomms::MMDriver](#)). - The type of interface is configured using a [goby::acomms::protobuf::Driver-Config](#). The modem is accessed by the derived class using [goby::acomms::ModemDriverBase::modem_start](#), [goby::acomms::ModemDriverBase::modem_read](#), [goby::acomms::ModemDriverBase::modem_write](#), and [goby::acomms::ModemDriverBase::modem_close](#).

- **Signals** to be called at the appropriate time by the derived class (`goby::acomms::ModemDriverBase::signal_receive`, `goby::acomms::ModemDriverBase::signal_data_request`, `goby::acomms::ModemDriverBase::range_reply`, `goby::acomms::ModemDriverBase::signal_ack`, `goby::acomms::ModemDriverBase::signal_all_incoming`, `goby::acomms::ModemDriverBase::signal_all_outgoing`). At the application layer, either bind the modem driver to a `goby::acomms::QueueManager` (`goby::acomms::bind(goby::acomms::ModemDriverBase&, goby::acomms::QueueManager&)`) or connect custom function pointers or objects to the driver layer signals. The `goby::acomms::ModemDriverBase::signal_all_incoming` is used by the `goby::acomms::MACManager` to "discover" vehicles so this signal should be connected to the `goby::acomms::MACManager` (using `goby::acomms::bind(goby::acomms::MACManager&, goby::acomms::ModemDriverBase&)`). The `goby::acomms::ModemDriverBase::signal_all_incoming` and `goby::acomms::ModemDriverBase::signal_all_outgoing` signals are for use by the application layer if desired to monitor the functionality of the modem.
- **Virtual functions** for starting the driver (`goby::acomms::ModemDriverBase::startup`), running the driver (`goby::acomms::ModemDriverBase::do_work`), and initiating the transmission of a message (`goby::acomms::ModemDriverBase::handle_initiate_transmission`). If the modem supports it, ranging (via an acoustic ping) can also be initiated (`goby::acomms::ModemDriverBase::handle_initiate_ranging`). These last two slots are typically bound to the corresponding signals from the `goby::acomms::MACManager`.

5.2 WHOI Micro-Modem Driver: MMDriver

The `goby::acomms::MMDriver` extends the `goby::acomms::ModemDriverBase` for the WHOI Micro-Modem acoustic modem. It is tested to work with revision 0.93.0.30 of the Micro-Modem firmware, but is known to work with older firmware (at least 0.92.0.85), as well as newer (WHOI firmware 0.93.0.35 to 0.93.0.51 has a critical bug, however, where XST should be set to 0). The following commands of the WHOI Micro-Modem are implemented:

Modem to Control Computer (\$CA / \$SN):

- \$CAACK - acknowledgment of sent message. Will be transformed into a `goby::acomms::protobuf::ModemDataAck` and signaled on `goby::acomms::ModemDriverBase::signal_ack`.
- \$CADRQ - data request. Data will be provided that has been buffered previously on the necessary number of calls of the signal `goby::acomms::ModemDriverBase::signal_data_request`. See \$CCCYC for this buffering behavior.
- \$CARXD - received hexadecimal data. Will be transformed into a `goby::acomms::protobuf::ModemDataTransmission` and signaled on `goby::acomms::ModemDriverBase::signal_receive`.
- \$CAREV - revision number and heartbeat. Used to check for correct clock time and modem reboots.
- \$CAERR - error message. The error message is logged to the `std::ostream` provided to `goby::acomms::MMDriver` at instantiation.

- \$CACYC - response to cycle initialization request. If data as has not yet been buffered because the cycle (\$CCCYC) was initiated elsewhere, the signal `goby::acomms::ModemDriverBase::signal_data_request` will be called N times, where N is the number of frames for the given rate (rate 0 = 1 frame, rate 2 = 3 frames, rate 3 = 2 frames, rate 5 = 8 frames).
- \$CAMPR - response to two-way modem ping. Transformed into `goby::acomms::protobuf::ModemRangingReply` and signaled on `goby::acomms::ModemDriverBase::signal_range_reply`.
- \$SNTTA - response to REMUS LBL ping. Transformed into `goby::acomms::protobuf::ModemRangingReply` and signaled on `goby::acomms::ModemDriverBase::signal_range_reply`.
- \$CATOA - one way synchronous ranging. Requires an accurate PPS signal on the WHOI Micro-Modem, and setting of NVRAM parameters "SN-V,1" and "TOA,1". See <http://acomms.who.edu/documents/-Synchronous%20Navigation%20With%20MicroModem%20RevD.-pdf> for details of configuration and usage. Transformed into `goby::acomms::protobuf::ModemRangingReply` and signaled on `goby::acomms::ModemDriverBase::signal_range_reply`.

Control Computer to Modem (\$CC). Also implemented is the NMEA acknowledge (e.g. \$CACK for \$CCCLK):

- \$CCTXD - transmit data. Sent using data buffered by calls of the signal `goby::acomms::ModemDriverBase::signal_data_request`. See \$CCCYC for this buffering behavior. Sent in response to \$CADRQ.
- \$CCCYC - initiate a cycle. Sent on response to a call of `goby::acomms::MM-Driver::initiate_transmission`. If the \$CCCYC is local (that is, the ADR1 is the same as this vehicle's modem id), the signal `goby::acomms::ModemDriverBase::signal_data_request` will be called N times, where N is the number of frames for the given rate (rate 0 = 1 frame, rate 2 = 3 frames, rate 3 = 2 frames, rate 5 = 8 frames). If the \$CCCYC is for a third party request (ADR1 is *not* this platform), no buffering is done. This buffering will happen on the ADR1 node after it hears the \$CACYC. The reason we buffer on the \$CCCYC if we can instead of always on the \$CACYC is that we can avoid sending the \$CCCYC at all if we have no data.
- \$CCCLK - set the clock. The clock is set on startup until a suitably value within 1 second of the computer time is reported back. If the modem reboots (\$CARE-V,...,INIT), the clock is set again.
- \$CCCFG - configure NVRAM value. All values passed to the extension - `MicroModemConfig::nvram_cfg` of `goby::acomms::protobuf::DriverConfig` will be passed to \$CCCFG. Thus, to send "\$CCCFG,SNR,1", put "SNR,1" in `MicroModemConfig::nvram_cfg`. That is:

```
goby::acomms::protobuf::DriverConfig cfg;
cfg.AddExtension(MicroModemConfig::nvram_cfg, "SNR,1");
mm_driver.startup(cfg);
```

NVRAM parameters SRC and ALL will be set automatically as needed. Set [Micro-ModemConfig::reset_nvram](#) to true to send "\$CCCFG,ALL,0". Setting `modem_id` in [goby::acomms::protobuf::DriverConfig](#) will set SRC to the same value.

- \$CCMPC - ping another modem. Sent when [goby::acomms::ModemDriverBase::handle_initiate_ranging](#) is called with [goby::acomms::protobuf::ModemRangingReply::type == goby::acomms::protobuf::RANGING_MODEM_TWO_WAY_PING](#). The destination of the ping is [goby::acomms::protobuf::ModemRangingReply::ModemMsgBase::dest](#).
- \$CCPDT - ping REMUS digital transponder. Sent when [goby::acomms::ModemDriverBase::handle_initiate_ranging](#) is called with [goby::acomms::protobuf::ModemRangingReply::type == goby::acomms::protobuf::REMUS_LBL_RANGING](#).

Mapping between [modem_message.proto](#) messages and NMEA fields (see <http://acomms.whoi.edu/documents/uModem%20Software%20Interface%20Guide.pdf> for NMEA fields of the WHOI Micro-Modem):

Modem to Control Computer (\$CA / \$SN):

NMEA talker	Mapping
\$CAACK	Mapping <code>goby::acomms::protobuf::ModemData-Ack.base().src()</code> = SRC <code>goby::acomms::protobuf::ModemData-Ack.base().dest()</code> = DEST <code>goby::acomms::protobuf::ModemData-Ack.frame()</code> = Frame#-1 (Goby starts counting at frame 0, WHOI starts with frame 1)
\$CADRQ	Data request is anticipated from the \$CCCYC or \$CACYC and buffered. Thus it is not translated into any of the <code>modem_message.proto</code> messages.
\$CARXD	<code>goby::acomms::protobuf::ModemData-Transmission.base().src()</code> = SRC <code>goby::acomms::protobuf::ModemData-Transmission.base().dest()</code> = DEST <code>goby::acomms::protobuf::ModemData-Transmission.ack_requested()</code> = ACK <code>goby::acomms::protobuf::ModemData-Transmission.frame()</code> = F# <code>goby::acomms::protobuf::ModemData-Transmission.data()</code> = <code>goby::acomms::hex_decode(HH...HH)</code>
\$CACYC	If we did not send \$CCCYC, buffer data for \$CADRQ by sending this message n = 0 to Nframes times: <code>goby::acomms::protobuf::ModemData-Request.base().src()</code> = ADR1 <code>goby::acomms::protobuf::ModemData-Request.base().dest()</code> = ADR2 <code>goby::acomms::protobuf::ModemData-Request.max_bytes()</code> = 32 for Packet Type == 0, 64 for Packet Type == 2, 256 for Packet Type == 3 or 5 <code>goby::acomms::protobuf::ModemData-Request.frame()</code> = n
\$CAREV	Not translated into any of the <code>modem_message.proto</code> messages.
\$CAERR	Not translated into any of the <code>modem_message.proto</code> messages.
\$CAMPR	<code>goby::acomms::protobuf::Modem-RangingReply.base().dest()</code> = SRC (SRC and DEST flipped to be SRC and DEST of \$CCMPC) <code>goby::acomms::protobuf::Modem-RangingReply.base().src()</code> = DEST

Control Computer to Modem (\$CC):

\$CCTXD	SRC = <code>goby::acomms::protobuf::ModemDataTransmission.base().src()</code> DEST = <code>goby::acomms::protobuf::ModemDataTransmission.base().dest()</code> A = <code>goby::acomms::protobuf::ModemDataTransmission.ack_requested()</code> HH...HH = <code>goby::acomms::hex_encode(goby::acomms::protobuf::ModemDataTransmission.data())</code>
\$CCCYC	CMD = 0 (deprecated field) ADR1 = <code>goby::acomms::protobuf::ModemDataInit.base().src()</code> ADR2 = <code>goby::acomms::protobuf::ModemDataInit.base().dest()</code> Packet Type = <code>goby::acomms::protobuf::ModemDataInit.base().rate()</code> ACK = 0 (deprecated field) Nframes = <code>goby::acomms::protobuf::ModemDataInit.num_frames()</code> If ADR1 == modem_id, buffer data for later \$CADRQ by sending this message n = 0 to Nframes times: <code>goby::acomms::protobuf::ModemDataRequest.base().src() = ADR1</code> <code>goby::acomms::protobuf::ModemDataRequest.base().dest() = ADR2</code> <code>goby::acomms::protobuf::ModemDataRequest.max_bytes() = 32</code> for Packet Type == 0, 64 for Packet Type == 2, 256 for Packet Type == 3 or 5 <code>goby::acomms::protobuf::ModemDataRequest.frame() = n</code>
\$CCCLK	Not translated from any of the <code>modem_message.proto</code> messages. (taken from the system time using the <code>boost::date_time</code> library)
\$CCCFG	Not translated from any of the <code>modem_message.proto</code> messages. (taken from values passed to the extension <code>MicroModemConfig::nvram_cfg</code> of <code>goby::acomms::protobuf::DriverConfig</code>)
\$CCCFQ	Not translated from any of the <code>modem_message.proto</code> messages. \$CCCFQ,ALL sent at startup.
\$CCMPC	<code>protobuf::MODEM_TWO_WAY_PING</code> == <code>goby::acomms::protobuf::ModemRangingRequest.type()</code>
Generated on Tue Jan 15 2013 22:02:09 for Goby Underwater Autonomy Project by Doxygen	SRC = <code>goby::acomms::protobuf::ModemRangingRequest.base().src()</code> DEST = <code>goby::acomms::protobuf::ModemRangingRequest.base().dest()</code>
\$CCPDT	<code>protobuf::REMUS_LBL_RANGING</code> == <code>goby::acomms::protobuf::Modem-</code>

5.3 Writing a new driver

All of goby-acomms is designed to be agnostic of which physical modem is used. - Different modems can be supported by subclassing `goby::acomms::ModemDriverBase`.

These are the requirements of the acoustic modem:

- it communicates using a line based text duplex connection using either serial or TCP (either client or server). NMEA0183 and AT (Hayes) protocols fulfill this requirement, for example.
- it is capable of sending and verifying the accuracy (using a cyclic redundancy check or similar error checking) of fixed size datagrams (note that modems capable of variable sized datagrams also fit into this category).

Optionally, it can also support

- Acoustic acknowledgment of proper message receipt.
- Ranging to another acoustic modem or LBL beacons using time of flight measurements
- User selectable bit rates

The steps to writing a new driver include:

- Fully understand the basic usage of the new acoustic modem manually using minicom or other terminal emulator. Have a copy of the modem software interface manual handy.
- Figure out what type of configuration the modem will need. For example, the WHOI Micro-Modem is configured using string values (e.g. "SNV,1"). Extend `goby::acomms::protobuf::DriverConfig` to accomodate these configuration options. You will need to claim a group of extension field numbers that do not overlap with any of the drivers. The WHOI Micro-Modem driver `goby::acomms::MMDriver` uses extension field numbers 1000-1100 (see `mm_driver.proto`). You can read more about extensions in the official Google Protobuf documentation here: <http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions>.

For example, if I was writing a new driver for the ABC Modem that needs to be configured using a few boolean flags, I might create a new message `abc_driver.proto`:

```
import "goby/protobuf/driver_base.proto"; // load up message DriverBaseConfig

message ABCDriverConfig
{
    extend goby.acomms.protobuf.DriverConfig
    {
        optional bool enable_foo = 1201 [ default = true ];
        optional bool enable_bar = 1202 [ default = false ];
    }
}
```


make a note in [driver_base.proto](#) claiming extension numbers 1201 and 1202 (and others you may expect to need in the future). Extension field numbers can go up to 536,870,911 so don't worry about running out.

- Subclass [goby::acomms::ModemDriverBase](#) and overload the pure virtual methods ([goby::acomms::ModemDriverBase::handle_initiate_ranging](#) is optional). - Your interface should look like this:

```
namespace goby
{
    namespace acomms
    {
        class ABCDriver : public ModemDriverBase
        {
        public:
            ABCDriver(std::ostream* log = 0);
            void startup(const protobuf::DriverConfig& cfg);
            void shutdown();
            void do_work();
            void handle_initiate_transmission(protobuf::ModemMsgBase* m);

        private:
            protobuf::DriverConfig driver_cfg_; // configuration given to you
            at launch
            std::ostream* log_; // place to log all human readable debugging
            messages
            // rest is up to you!
        };
    }
}
```

- Fill in the methods. You are responsible for emitting the [goby::acomms::ModemDriverBase](#) signals at the appropriate times. Read on and all should be clear.

- Minimally your constructor should store a local copy of the ostream logger:

```
goby::acomms::ABCDriver::ABCDriver() : ModemDriverBase(log),
                                       log_(log)
{
    // other initialization you can do before you have your
    goby::acomms::DriverConfig configuration object
}
```

- At startup() you get your configuration from the application (pAcomms-Handler or other)

```
void goby::acomms::ABCDriver::startup(const protobuf::DriverConfig& cfg)
{
    driver_cfg_ = cfg;
    // check 'driver_cfg_' to your satisfaction and then start the modem
    physical interface
    if(!driver_cfg_.has_serial_baud())
        driver_cfg_.set_serial_baud(DEFAULT_BAUD);

    // log_ is allowed to be 0 (NULL), so always check it first
    if(log_) *log_ << group("modem_out") << "ABCDriver configuration good.
        Starting modem..." << std::endl;
    ModemDriverBase::modem_start(driver_cfg_);

    // set your local modem id (MAC address)
```

```

driver_cfg_.modem_id();

{
    std::stringstream raw;
    raw << "CONF,MAC:" << driver_cfg_.modem_id() << "\r\n";
    signal_and_write(raw.str());
}

// now set our special configuration values
{
    std::stringstream raw;
    raw << "CONF,FOO:" << driver_cfg_.GetExtension(
        ABCDriverConfig::enable_foo) << "\r\n";
    signal_and_write(raw.str());
}
{
    std::stringstream raw;
    raw << "CONF,BAR:" << driver_cfg_.GetExtension(
        ABCDriverConfig::enable_bar) << "\r\n";
    signal_and_write(raw.str());
}
} // startup

```

- At `shutdown()` you should make yourself ready to `startup()` again if necessary and stop the modem:

```

void goby::acomms::ABCDriver::shutdown()
{
    // put the modem in a low power state?
    // ...
    ModemDriverBase::modem_close();
} // shutdown

```

- `handle_initiate_transmission()` is called when you are expected to initiate a transmission. It *does not* contain data, you are required to request data using the `goby::acomms::ModemDriverBase::signal_data_request` signal. - Once you have data, you are responsible for sending it. I think a bit of code will make this clearer:

```

void goby::acomms::ABCDriver::handle_initiate_transmission(
    protobuf::ModemMsgBase* base_msg)
{
    if(log_)
    {
        // base_msg->rate() can be 0 (lowest), 1, 2, 3, 4, or 5 (lowest). Map
        // these integers onto real bit-rates
        // in a meaningful way (on the WHOI Micro-Modem 0 ~= 80 bps, 5 ~= 5000
        // bps).
        *log_ << group("modem_out") << "We were asked to transmit from " <<
        base_msg->src()
            << " to " << base_msg->dest()
            << " at bitrate code " << base_msg->rate() << std::endl;
    }

    protobuf::ModemDataRequest request_msg; // used to request data from
        libqueue
    protobuf::ModemDataTransmission data_msg; // used to store the requested
        data

```

```

// set up request_msg
request_msg.mutable_base()->set_src(base_msg->src());
request_msg.mutable_base()->set_dest(base_msg->dest());
// let's say ABC modem uses 500 byte packet
request_msg.set_max_bytes(500);

ModemDriverBase::signal_data_request(request_msg, &data_msg);

// do nothing with an empty message
if(data_msg.data().empty()) return;

if(log_)
{
    *log_ << group("modem_out") << "Sending these data now: " << data_msg
    << std::endl;
}

// let's say we can send at three bitrates with ABC modem: map these onto
// 0-5
const unsigned BITRATE [] = { 100, 1000, 10000, 10000, 10000, 10000};

// I'm making up a syntax for the wire protocol...
std::stringstream raw;
raw << "SEND,TO:" << data_msg.base().dest()
    << ",FROM:" << data_msg.base().src()
    << ",HEX:" << hex_encode(data_msg.data())
    << ",BITRATE:" << BITRATE[base_msg->rate()]
    << ",ACK:TRUE"
    << "\r\n";

// let anyone who is interested know
signal_and_write(raw.str(), base_msg);
} // handle_initiate_transmission

```

- Finally, you can use `do_work()` to do continuous work. You can count on it being called at 5 Hz or more (in `pAcommsHandler`, it is called on the `MOOS AppTick`). Here's where you want to read the modem incoming stream.

```

void goby::acomms::ABCDriver::do_work()
{
    std::string in;
    while(modem_read(&in))
    {
        std::map<std::string, std::string> parsed;

        // breaks `in`: "RECV,TO:3,FROM:6,HEX:ABCD015910"
        // into `parsed`: "KEY"=>"RECV", "TO"=>"3", "FROM"=>"6",
        "HEX"=>"ABCD015910"
        try
        {
            boost::trim(in); // get whitespace off from either end
            parse_in(in, &parsed);

            protobuf::ModemMsgBase base_msg;
            base_msg.set_raw(in);

            using google::protobuf::int32;
            base_msg.set_src(goby::util::as<int32>(parsed["FROM"]));
            base_msg.set_dest(goby::util::as<int32>(parsed["TO"]));
            base_msg.set_time(goby::util::as<std::string>(
                goby::util::goby_time()));

            if(log_) *log_ << group("modem_in") << in << std::endl;

```

```

ModemDriverBase::signal_all_incoming(base_msg);

if(parsed["KEY"] == "RCV")
{
    protobuf::ModemDataTransmission data_msg;
    data_msg.mutable_base()->CopyFrom(base_msg);
    data_msg.set_data(hex_decode(parsed["HEX"]));
    if(log_) *log_ << group("modem_in") << "received: " << data_msg
<< std::endl;
    ModemDriverBase::signal_receive(data_msg);
}
else if(parsed["KEY"] == "ACKN")
{
    protobuf::ModemDataAck ack_msg;
    ack_msg.mutable_base()->CopyFrom(base_msg);
    ModemDriverBase::signal_ack(ack_msg);
}
}
catch(std::exception& e)
{
    if(log_) *log_ << warn << "Bad line: " << in << std::endl;
    if(log_) *log_ << warn << "Exception: " << e.what() << std::endl;
}
}
} // do_work

```

- Add your driver header to `goby/src/acomms/modem_driver.h`
- Modify `libmodemdriver/examples/driver_simple/driver_simple.cpp` to work with your new driver.
- Add your driver to the `pAcommsHandler_config.proto` `DriverType` enumeration.
- Add your driver to the `pAcommsHandler.cpp` driver object creation.

The full ABC Modem example driver exists in `acomms/libmodemdriver/abc_driver.h` and `acomms/libmodemdriver/abc_driver.cpp`. A simulator for the ABC Modem exists that uses TCP to mimic a very basic set of modem commands (send data and acknowledgment). To use the ABC Modem using the `driver_simple` example, run this set of commands ('socat' is available in most package managers or at <http://www.dest-unreach.org/socat/>):

```

1. run abc_modem_simulator running on same port (as TCP server)
> abc_modem_simulator 54321
2. create fake tty terminals connected to TCP as client to port 54321
> socat -d -d -v pty,raw,echo=0,link=/tmp/ttyFAKE1 TCP:localhost:54321
> socat -d -d -v pty,raw,echo=0,link=/tmp/ttyFAKE2 TCP:localhost:54321
3. start up driver_simple
> driver_simple /tmp/ttyFAKE1 1 ABCDriver
// wait a few seconds to avoid collisions
> driver_simple /tmp/ttyFAKE2 2 ABCDriver

```

Notes:

- See `goby::acomms::MMDriver` for an example real implementation.
- When a message is sent to `goby::acomms::BROADCAST_ID` (0), it should be broadcast if the modem supports such functionality. Otherwise, the driver should

throw an `goby::acomms::driver_exception` indicating that it does not support broadcast allowing the user to reconfigure their MAC / addressing scheme.

6 goby-acomms: libamac (Medium Access Control)

Table of Contents for libamac:

- [Supported MAC schemes](#)
- [Interacting with the `goby::acomms::MACManager`](#)

Return to [goby-acomms: Overview of Acoustic Communications Libraries](#).

6.1 Supported MAC schemes

The Medium Access Control schemes provided by libamac are based on Time Division Multiple Access (TDMA) where different communicators share the same bandwidth but transmit at different times to avoid conflicts. Time is divided into slots and each vehicle is given a slot to transmit on. The set of slots comprising all the vehicles is referred to here as a cycle, which repeats itself when it reaches the end. The two variations on this scheme provided by libamac are:

1. Decentralized: Each vehicle initiates its own transmission at the start of its slot.
 - Auto-discovery (`goby::acomms::protobuf::MAC_AUTO_DECENTRALIZED`): Each vehicle has a single slot in the cycle on which it transmits. Collisions are avoided by each vehicle following the same rules about slot placement within the time window (based on real time of day). This scheme requires that each vehicle have reasonably accurate clocks (perhaps better than +/- 0.5 seconds).
 - Fixed (`goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED`): Slots are fixed at runtime and can be updated manually using `goby::acomms::MACManager::add_slot()` and `goby::acomms::MACManager::remove_slot()`. Each vehicle can have more than one slot in the cycle. The cycles must agree across all platforms; the network designer is responsible for this (unlike in auto-discovery where this happens within goby-acomms).
2. Centralized Polling (`goby::acomms::protobuf::MAC_POLLED` on the master, `goby::acomms::protobuf::MAC_NONE` on all other nodes): The TDMA cycle is set up and operated by a centralized master modem ("poller"), which is usually the modem connected to the vehicle operator's topside. The poller initiates each transmission and thus the vehicles are not required to maintain synchronous clocks.

6.2 Interacting with the `goby::acomms::MACManager`

To use the `goby::acomms::MACManager`, you need to instantiate it (optionally with a `std::ostream` pointer to a location to log to):

```
goby::acomms::MACManager mac(&std::clog);
```

Then you need to provide a slot for initiated transmissions for the signal `goby::acomms::MACManager::signal_initiate_transmission`. This signal will be called when the `goby::acomms::MACManager` determines it is time to send a message. If using libmodem-driver, simply call `goby::acomms::bind(goby::acomms::MACManager&, goby::acomms::ModemDriverBase&)` to bind this callback to the modem driver.

Next you need to decide which type of MAC to use: decentralized auto-discovery, decentralized fixed or centralized polling and set the type of the `goby::acomms::protobuf::MACConfig` with the corresponding `goby::acomms::protobuf::MACType`. We also need to give `goby::acomms::MACManager` the vehicle's modem id (like all the other components of `goby-acomms`):

```
using namespace goby::acomms;
protobuf::MACConfig mac_cfg;
mac_cfg.set_type(protobuf::MAC_FIXED_DECENTRALIZED);
mac_cfg.set_modem_id(1);
```

The usage of the `goby::acomms::MACManager` depends now on the type:

- `goby::acomms::protobuf::MAC_AUTO_DECENTRALIZED`: Set the rest of the parameters (modem rate (integer from 0-5), slot time (seconds), cycles before removing a "dead" vehicle from the cycle):

```
mac_cfg.set_rate(0);
mac_cfg.set_slot_seconds(10);
mac_cfg.set_expire_cycles(2);
```

If using a derivative of `goby::acomms::ModemDriverBase`, vehicles will automatically be discovered as they are heard from and added to the cycle (via the slot `goby::acomms::MACManager::handle_modem_all_incoming` connected to `goby::acomms::ModemDriverBase::signal_all_incoming`). If using a different driver, you need to inform the `goby::acomms::MACManager` each time you hear from a vehicle by calling `goby::acomms::MACManager::handle_modem_all_incoming` with all foreign messages (doesn't need to contain data). If a vehicle isn't heard from for a certain number of cycles (set by `goby::acomms::MACManager::set_expire_cycles`), it will be removed from the cycle to increase throughput for the remaining vehicles.

- `goby::acomms::protobuf::MAC_POLLED`: On the vehicles, you do not need to run the `goby::acomms::MACManager` at all, or simply give it the "do nothing" `goby::acomms::protobuf::MAC_NONE` type. All the MAC is done on the top-side (the centralized poller). On the poller, you need to manually set up a list of vehicles to be polled by adding an `goby::acomms::protobuf::Slot` (in the initial `goby::acomms::protobuf::MACConfig` object or at runtime via `goby::acomms::MACManager::add_slot`) for each vehicle to be polled. You can poll the same vehicle multiple times, just add more `goby::acomms::protobuf::Slot` objects corresponding to that vehicle. Each slot has a source, destination, rate, type (data or ping [not yet implemented]), and length (in seconds). If the source is the poller, you can set the destination to `goby::acomms::QUERY_DESTINATION_ID` (=-1) to let libqueue determine the next destination (based on the highest priority message to send). All `goby::acomms::protobuf::Slot` objects for vehicles must have a specified destination (the `goby::acomms::BROADCAST_ID` is a good choice or the id of the poller). For example:

```
// poll ourselves (for commands, perhaps)

goby::acomms::protobuf::Slot slot;
slot.set_src(1);
slot.set_dest(goby::acomms::QUERY_DESTINATION_ID);
slot.set_rate(0);
slot.set_type(goby::acomms::protobuf::SLOT_DATA);
slot.set_seconds(10);
mac_cfg.add_slot(10); // 1->-1@0 wait 10

// reuse slot
slot.set_src(3);
slot.set_dest(goby::acomms::BROADCAST_ID);
mac_cfg.add_slot(slot); // 3->0@0 wait 10

slot.set_rate(5);
mac_cfg.add_slot(slot); // 3->0@5 wait 10

slot.set_src(4);
slot.set_rate(0);
mac_cfg.add_slot(slot); // 4->0@0 wait 10
```

You can remove vehicles by a call to `goby::acomms::MACManager::remove_slot` or clear out the entire cycle and start over with `goby::acomms::MACManager::clear_all_slots`.

- `goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED`: Configured in much the same way as `goby::acomms::protobuf::MAC_POLLED`. However, all vehicles must now be running `goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED` and share the same cycle (set of slots). Also, since each vehicle initiates its own transaction, you can use `goby::acomms::QUERY_DESTINATION_ID` throughout. The same example cycle as for the polled situation above would look like this for this type of decentralized MAC:

```
goby::acomms::protobuf::Slot slot;
slot.set_src(1);
slot.set_dest(goby::acomms::QUERY_DESTINATION_ID);
slot.set_rate(0);
slot.set_type(goby::acomms::protobuf::SLOT_DATA);
slot.set_seconds(10);
mac_cfg.add_slot(10); // 1->-1@0 wait 10

slot.set_src(3);
mac_cfg.add_slot(slot); // 3->-1@0 wait 10

slot.set_rate(5);
mac_cfg.add_slot(slot); // 3->-1@5 wait 10

slot.set_src(4);
slot.set_rate(0);
mac_cfg.add_slot(slot); // 4->-1@0 wait 10
```

Then, for either MAC scheme, start the `goby::acomms::MACManager` running (`goby::acomms::MACManager::startup` with the `goby::acomms::protobuf::MACConfig` object), and call `goby::acomms::MACManager::do_work()` periodically (5 Hz is ok, 10 Hz is better).

7 goby-util: Overview of Utility Libraries

Table of Contents for [goby-util: Overview of Utility Libraries](#).

- [Overview](#)
- [Logging](#)
 - [Configurable extension of std::ostream - liblogger](#)
- [TCP and Serial port communications - liblinebasedcomms](#)

7.1 Overview

The goby-util libraries are intended to provide functions and classes for handling "utility" tasks, such as logging, string manipulation, scientific calculations, etc. Wherever possible, a high quality open source peer reviewed solution is used (such as the C++ STL, boost). However, many of these libraries are very full-featured complex and are simplified here for Goby specific tasks.

7.2 Logging

Because Goby is designed first and foremost as an engineering testbed and scientific research architecture, comprehensive logging is extremely important for debugging both at runtime and post-mission. Thus, Goby provides a logging utility ([goby::util::FlexOstream](#)) based on C++ STL streams that provides highly configurable runtime (i.e. terminal window) and/or post-mission (text log file) logging. The syntax inside the code should be familiar to any C++ programmer, as any std::ostream functions can be used. The [goby-acomms](#) API classes all have a constructor which can take a pointer to std::ostream object that will be used to log. Thus, for runtime debugging one might instantiate them with std::cout:

```
goby::acomms::DCCLCodec dccl(&std::cout);
```

In which case you get output (to std::cout, aka the terminal window) that looks like:

```
[ 2011-Mar-01 04:06:35.169817 ]           {dccl_enc}: cryptography enabled with given passphrase
[ 2011-Mar-01 04:06:35.170610 ]           {dccl_enc}: starting encode for TEST
[ 2011-Mar-01 04:06:35.170683 ]           {dccl_enc}: B: bool: true
...
```

The timestamp (in Universal Coordinated Time) is given, with a group name (dccl_enc = DCCL Encoder) and finally the message. These groups are provided by using the manipulator "group". Text in the stream is a member of the given group until the next flush (std::endl or std::flush). For example:

```
// prints [ 2011-Mar-01 04:06:35.169817 ]           {my_group}: my message
std::cout << group("my_group") << "my message" << std::endl; // endl flushes my_group
```

Several other manipulators are provided:

- "debug" indicates that the buffer output is insignificant except for debugging (not useful for normal runtime)

- "warn" prints the buffer until the next flush as a warning.
- "die" is a fatal warning that calls "exit" with a non-zero code (indicating a fatal error). "die" should be used very sparingly.

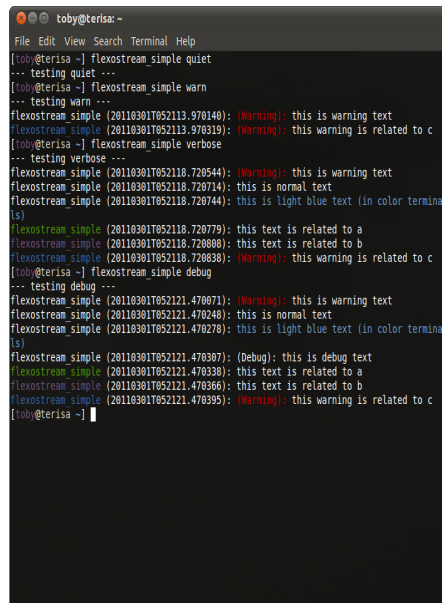
7.2.1 Configurable extension of `std::ostream` - liblogger

`goby::util::FlexOstream` extends `std::ostream` to provide a number of extra logging features. This is generally the preferred logger (instead of `std::cout`, etc.) for goby applications. Use `goby::util::glogger()` in the same way you would use `std::cout` or a `std::ofstream` object. These features include:

- Often it is desirable to log simultaneously to a text file (`std::ofstream`) and the terminal window (`std::cout`). `goby::util::FlexOstream` allows you to attach any number of streams to it, which are all written to with a single call to operator<< on the `goby::util::FlexOstream` object.
- Color support for ANSI terminals (`std::cout` and `std::cerr` stream objects only)
- Multiple verbosity settings for each attached stream: QUIET (display nothing to this stream), WARN (display only warnings), VERBOSE (display warnings and normal text, but not debug text), DEBUG (display warnings, normal text, and debug messages), GUI (display all messages in an NCurses terminal GUI window, splitting groups into different displays)
- Optional thread safe access using a simple lock / unlock syntax.

The best way to get used to `goby::util::glogger()` is to compile and play with the `flexostream_simple.cpp` example.

A handful of examples:



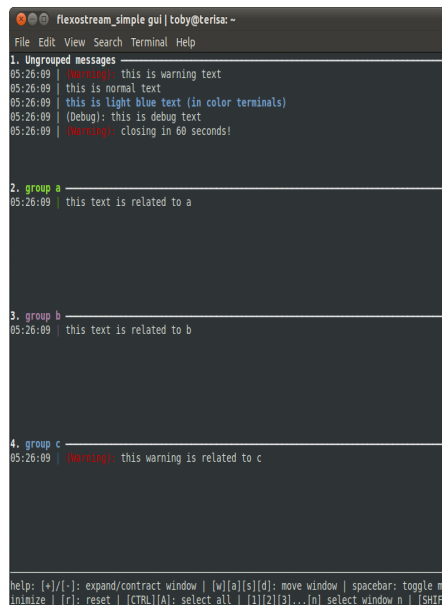
```

toby@terisa: ~
File Edit View Search Terminal Help
[toby@terisa ~] flexostream_simple quiet
... testing quiet ...
[toby@terisa ~] flexostream_simple warn
... testing warn ...
flexostream_simple (20110301T052113.970140): (Warning): this is warning text
flexostream_simple (20110301T052113.970319): (Warning): this warning is related to c
[toby@terisa ~] flexostream_simple verbose
... testing verbose ...
flexostream_simple (20110301T052118.720544): (Warning): this is warning text
flexostream_simple (20110301T052118.720714): this is normal text
flexostream_simple (20110301T052118.720744): this is light blue text (in color terminals)
flexostream_simple (20110301T052118.720779): this text is related to a
flexostream_simple (20110301T052118.720808): this text is related to b
flexostream_simple (20110301T052118.720838): (Warning): this warning is related to c
[toby@terisa ~] flexostream_simple debug
... testing debug ...
flexostream_simple (20110301T052121.470071): (Warning): this is warning text
flexostream_simple (20110301T052121.470240): this is normal text
flexostream_simple (20110301T052121.470270): this is light blue text (in color terminals)
flexostream_simple (20110301T052121.470307): (Debug): this is debug text
flexostream_simple (20110301T052121.470338): this text is related to a
flexostream_simple (20110301T052121.470366): this text is related to b
flexostream_simple (20110301T052121.470395): (Warning): this warning is related to c
[toby@terisa ~]

```

Figure 7: Example of the `goby::util::logger()` output at different verbosity settings to the terminal window

Graphical user interface logger mode:



```

flexostream_simple gui | toby@terisa: ~
File Edit View Search Terminal Help
1. Ungrouped messages
05:26:09 | (Warning): this is warning text
05:26:09 | this is normal text
05:26:09 | this is light blue text (in color terminals)
05:26:09 | (Debug): this is debug text
05:26:09 | (Warning): closing in 60 seconds!

2. group a
05:26:09 | this text is related to a

3. group b
05:26:09 | this text is related to b

4. group c
05:26:09 | (Warning): this warning is related to c

help: [+]/[-]: expand/contract window | [w][a][s][d]: move window | spacebar: toggle m
inimize | [r]: reset | [CTRL][A]: select all | [1][2][3]...[n] select window n | [SHIF

```

Figure 8: Example of the `goby::util::logger()` in NCurses GUI mode

Simultaneous terminal window and file logging:

```
flexostream_simple quiet|warn|verbose|debug|gui test.txt
```

test.txt:

```
[ 2011-Mar-01 05:33:26.224050 ] {}: (Warning): this is warning text
[ 2011-Mar-01 05:33:26.224277 ] {}: this is normal text
[ 2011-Mar-01 05:33:26.224320 ] {}: this is light blue text (in color terminals)
[ 2011-Mar-01 05:33:26.224362 ] {}: (Debug): this is debug text
[ 2011-Mar-01 05:33:26.224388 ] {a}: this text is related to a
[ 2011-Mar-01 05:33:26.224429 ] {b}: this text is related to b
[ 2011-Mar-01 05:33:26.224471 ] {c}: (Warning): this warning is related to c
```

7.3 TCP and Serial port communications - liblinebasedcomms

liblinebasedcomms provides a common interface ([goby::util::LineBasedInterface](#)) for line-based (defined as blocks of text offset by a common delimiter such as "\\r\\n" or "\\n") text communications over a TCP or serial connection. liblinebasedcomms using the boost::asio library to perform the actual communications.

You should create the proper subclass for your needs:

- Serial communications: [goby::util::SerialClient](#)
- TCP Client: [goby::util::TCPClient](#)
- TCP Server: [goby::util::TCPServer](#) - all incoming messages (as read by [goby::util::LineBasedInterface::readline](#)) are interleaved in the order they are received from all connected clients. Outgoing messages are sent to all connected clients unless using [goby::util::LineBasedInterface::write](#) (const protobuf::Datagram &msg) and msg.dest() is set to a specific endpoint (ip:port, e.g. "192.168.1.101:5123").

8 Module Index

8.1 Modules

Here is a list of all modules:

API classes for the acoustic communications libraries.

69

9 Namespace Index

9.1 Namespace List

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

[goby](#)

The global namespace for the Goby project

69

goby::acomms	
Objects pertaining to acoustic communications (acomms)	70
goby::acomms::protobuf	
Contains Google Protocol Buffers messages and helper functions. - See specific .proto files for definition of the actual messages (e.g. modem_message.proto)	75
goby::util	
Utility objects for performing functions such as logging, non- acoustic communication (ethernet / serial), time, scientific, string manipulation, etc	76
goby::util::tcolor	
Contains functions for adding color to Terminal window streams	83

10 Class Index

10.1 Class Hierarchy

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

goby::acomms::DCCLCodec	86
goby::acomms::DCCLMessageVal	95
goby::acomms::MACManager	100
goby::acomms::ModemDriverBase	105
goby::acomms::ABCDriver	85
goby::acomms::MMDriver	103
goby::acomms::QueueManager	112
goby::Exception	118
goby::acomms::DCCLException	95
goby::ConfigException	118
goby::util::Colors	119
goby::util::FlexNCurses	119
goby::util::FlexOstream	120
goby::util::FlexOStreamBuf	122

goby::util::LineBasedInterface	123
goby::util::TCPServer	127
goby::util::Logger	125
goby::util::SerialClient	125
goby::util::TCPClient	126
goby::util::TermColor	128
Group	129
GroupSetter	130

11 Class Index

11.1 Class List

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

goby::acomms::ABCDriver	
API to the imaginary ABC modem (as an example how to write drivers)	85
goby::acomms::DCCLCodec	
API to the Dynamic CCL Codec	86
goby::acomms::DCCLException	
Exception class for libdccl	95
goby::acomms::DCCLMessageVal	
Defines a DCCL value	95
goby::acomms::MACManager	
API to the goby-acomms MAC library	100
goby::acomms::MMDriver	
API to the WHOI Micro-Modem driver	103
goby::acomms::ModemDriverBase	
Abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems	105
goby::acomms::QueueManager	
API to the goby-acomms Queuing Library	112

<code>goby::ConfigException</code>	Indicates a problem with the runtime command line or .cfg file configuration (or --help was given)	118
<code>goby::Exception</code>	Simple exception class for goby applications	118
<code>goby::util::Colors</code>	Represents the eight available terminal colors (and bold variants)	119
<code>goby::util::FlexNCurses</code>	Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content	119
<code>goby::util::FlexOstream</code>	Forms the basis of the Goby logger: std::ostream derived class for holding the <code>FlexOStreamBuf</code>	120
<code>goby::util::FlexOStreamBuf</code>	Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by <code>goby::util::FlexOstream</code> for the Goby <code>Logger</code> (glogger)	122
<code>goby::util::LineBasedInterface</code>	Basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)	123
<code>goby::util::Logger</code>	Holds static objects of the Goby <code>Logger</code>	125
<code>goby::util::SerialClient</code>	Basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control	125
<code>goby::util::TCPClient</code>	Basic TCP client for line by line text based communications to a remote TCP server	126
<code>goby::util::TCPServer</code>	Basic TCP server for line by line text based communications to a one or more remote TCP clients	127
<code>goby::util::TermColor</code>	Converts between string, escape code, and enumeration representations of the terminal colors	128
<code>Group</code>	Defines a group of messages to be sent to the Goby logger. For Verbosity == verbose streams, all entries appear interleaved, but each group is offset with a different color. For Verbosity == gui streams, all groups have a separate subwindow	129

GroupSetter

Helper class for enabling the group(std::string) manipulator

130**12 File Index****12.1 File List**

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

acomms.h	??
util.h	??
version.h	??
acomms/acomms_constants.h	??
acomms/acomms_helpers.h	??
acomms/amac.h	??
acomms/bind.h	??
acomms/connect.h	??
acomms/dccl.h	??
acomms/modem_driver.h	??
acomms/queue.h	??
acomms/libamac/mac_manager.cpp	??
acomms/libamac/mac_manager.h	??
acomms/libdccl/dccl.cpp	??
acomms/libdccl/dccl.h	??
acomms/libdccl/dccl_constants.h	??
acomms/libdccl/dccl_exception.h	??
acomms/libdccl/message.cpp	??
acomms/libdccl/message.h	??
acomms/libdccl/message_algorithms.cpp	??
acomms/libdccl/message_algorithms.h	??

acomms/libdccl/message_publish.cpp	??
acomms/libdccl/message_publish.h	??
acomms/libdccl/message_val.cpp	??
acomms/libdccl/message_val.h	??
acomms/libdccl/message_var.cpp	??
acomms/libdccl/message_var.h	??
acomms/libdccl/message_var_bool.h	??
acomms/libdccl/message_var_enum.h	??
acomms/libdccl/message_var_float.cpp	??
acomms/libdccl/message_var_float.h	??
acomms/libdccl/message_var_head.h	??
acomms/libdccl/message_var_hex.h	??
acomms/libdccl/message_var_int.h	??
acomms/libdccl/message_var_static.h	??
acomms/libdccl/message_var_string.h	??
acomms/libdccl/message_xml_callbacks.cpp	??
acomms/libdccl/message_xml_callbacks.h	??
acomms/libdccl/tools/analyze_dccl_xml/analyze_dccl_xml.cpp	??
acomms/libmodemdriver/abc_driver.cpp	??
acomms/libmodemdriver/abc_driver.h	??
acomms/libmodemdriver/driver_base.cpp	??
acomms/libmodemdriver/driver_base.h	??
acomms/libmodemdriver/driver_exception.h	??
acomms/libmodemdriver/mm_driver.cpp	??
acomms/libmodemdriver/mm_driver.h	??
acomms/libmodemdriver/tools/abc_modem_simulator/abc_modem_simulator.cpp	??
acomms/libqueue/queue.cpp	??

acomms/libqueue/queue.h	??
acomms/libqueue/queue_constants.h	??
acomms/libqueue/queue_exception.h	??
acomms/libqueue/queue_manager.cpp	??
acomms/libqueue/queue_manager.h	??
acomms/libqueue/queue_xml_callbacks.cpp	??
acomms/libqueue/queue_xml_callbacks.h	??
acomms/xml/message_schema.xsd.h	??
acomms/xml/tags.h	??
acomms/xml/xerces_strings.h	??
acomms/xml/xml_parser.h	??
core/core.h	??
core/core_constants.h	??
core/libcore/configuration_reader.cpp	??
core/libcore/configuration_reader.h	??
core/libcore/exception.h	??
moos/iCommander/command_gui.cpp	??
moos/iCommander/command_gui.h	??
moos/iCommander/commander_cdk.cpp	??
moos/iCommander/commander_cdk.h	??
moos/iCommander/iCommander.cpp	??
moos/iCommander/iCommander.h	??
moos/iCommander/iCommander_config.proto	??
moos/iCommander/iCommanderMain.cpp	??
moos/libmoos_util/dynamic_moos_vars.h	??
moos/libmoos_util/modem_id_convert.cpp	??
moos/libmoos_util/modem_id_convert.h	??

moos/libmoos_util/ moos_protobuf_helpers.h	
Helpers for MOOS applications for serializing and parsed Google - Protocol buffers messages	131
moos/libmoos_util/tes_moos_app.cpp	??
moos/libmoos_util/tes_moos_app.h	??
moos/libmoos_util/tes_moos_app.proto	??
moos/pAcommsHandler/pAcommsHandler.cpp	??
moos/pAcommsHandler/pAcommsHandler.h	??
moos/pAcommsHandler/pAcommsHandler_config.proto	??
moos/pAcommsHandler/pAcommsHandlerMain.cpp	??
moos/pREMUSCodec/pREMUSCodec.cpp	??
moos/pREMUSCodec/pREMUSCodec.h	??
moos/pREMUSCodec/pREMUSCodec_config.proto	??
moos/pREMUSCodec/pREMUSCodecMain.cpp	??
moos/pREMUSCodec/WhoiUtil.cpp	??
moos/pREMUSCodec/WhoiUtil.h	??
protobuf/abc_driver.proto	??
protobuf/acomms_proto_helpers.h	??
protobuf/amac.proto	??
protobuf/app_base_config.proto	??
protobuf/cmoosmsg_mimic.proto	??
protobuf/config.proto	??
protobuf/dccl.proto	??
protobuf/driver_base.proto	??
protobuf/header.proto	??
protobuf/interprocess_notification.proto	??
protobuf/linebasedcomms.proto	??
protobuf/mm_driver.proto	??

protobuf/modem_message.proto	??
protobuf/option_extensions.proto	??
protobuf/queue.proto	??
protobuf/xml_config.proto	??
util/binary.h	??
util/linebasedcomms.h	??
util/logger.h	??
util/sci.h	??
util/string.h	??
util/time.h	??
util/liblinebasedcomms/client_base.h	??
util/liblinebasedcomms/connection.h	??
util/liblinebasedcomms/interface.cpp	??
util/liblinebasedcomms/interface.h	??
util/liblinebasedcomms/nmea_sentence.cpp	??
util/liblinebasedcomms/nmea_sentence.h	??
util/liblinebasedcomms/serial_client.cpp	??
util/liblinebasedcomms/serial_client.h	??
util/liblinebasedcomms/tcp_client.cpp	??
util/liblinebasedcomms/tcp_client.h	??
util/liblinebasedcomms/tcp_server.cpp	??
util/liblinebasedcomms/tcp_server.h	??
util/liblogger/flex_ncurses.cpp	??
util/liblogger/flex_ncurses.h	??
util/liblogger/flex_ostream.cpp	??
util/liblogger/flex_ostream.h	??
util/liblogger/flex_ostreambuf.cpp	??

<code>util/liblogger/flex_ostreambuf.h</code>	??
<code>util/liblogger/logger_manipulators.cpp</code>	??
<code>util/liblogger/logger_manipulators.h</code>	??
<code>util/liblogger/term_color.cpp</code>	??
<code>util/liblogger/term_color.h</code>	??
<code>util/tools/serial2tcp_server/serial2tcp_server.cpp</code>	??

13 Module Documentation

13.1 API classes for the acoustic communications libraries.

Classes

- class [goby::acomms::ABCDriver](#)
provides an API to the imaginary ABC modem (as an example how to write drivers)
- class [goby::acomms::MACManager](#)
provides an API to the goby-acomms MAC library.
- class [goby::acomms::DCCLCodec](#)
provides an API to the Dynamic CCL Codec.
- class [goby::acomms::ModemDriverBase](#)
provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.
- class [goby::acomms::MMDriver](#)
provides an API to the WHOI Micro-Modem driver
- class [goby::acomms::QueueManager](#)
provides an API to the goby-acomms Queuing Library.

14 Namespace Documentation

14.1 goby Namespace Reference

The global namespace for the Goby project.

Namespaces

- namespace [acomms](#)
Objects pertaining to acoustic communications (acomms)
- namespace [util](#)
Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc.

Classes

- class [Exception](#)
simple exception class for goby applications
- class [ConfigException](#)
indicates a problem with the runtime command line or .cfg file configuration (or --help was given)

Variables

- const std::string **VERSION_STRING** = "@GOBY_VERSION@"
- const std::string **VERSION_DATE** = "@GOBY_VERSION_DATE@"

14.1.1 Detailed Description

The global namespace for the Goby project. All objects related to the Goby Underwater Autonomy Project.

copyright 2009 t. schneider tes@mit.edu

14.2 goby::acomms Namespace Reference

Objects pertaining to acoustic communications (acomms)

Namespaces

- namespace [protobuf](#)
Contains Google Protocol Buffers messages and helper functions. See specific .proto files for definition of the actual messages (e.g. [modem_message.proto](#)).

Classes

- class [MACManager](#)
provides an API to the goby-acomms MAC library.
- class [DCCLCodec](#)
provides an API to the Dynamic CCL Codec.
- class [DCCLException](#)
[Exception](#) class for libdccl.
- class [DCCLMessageVal](#)
defines a DCCL value
- class [ABCDriver](#)
provides an API to the imaginary ABC modem (as an example how to write drivers)
- class [ModemDriverBase](#)

provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.

- class [MMDriver](#)

provides an API to the WHOI Micro-Modem driver

- class [QueueManager](#)

provides an API to the goby-acomms Queuing Library.

Typedefs

- typedef boost::function< void([DCCLMessageVal](#) &)> [AlgFunction1](#)
boost::function for a function taking a single [DCCLMessageVal](#) reference. Used for algorithm callbacks.
- typedef boost::function< void([DCCLMessageVal](#) &, const std::vector< [DCCLMessageVal](#) > &)> [AlgFunction2](#)
boost::function for a function taking a [dccl::MessageVal](#) reference, and the [MessageVal](#) of a second part of the message. Used for algorithm callbacks.

Enumerations

- enum [DCCLHeaderPart](#) { [HEAD_CCL_ID](#) = 0, [HEAD_DCCL_ID](#) = 1, [HEAD_TIME](#) = 2, [HEAD_SRC_ID](#) = 3, [HEAD_DEST_ID](#) = 4, [HEAD_MULTIMESSAGE_FLAG](#) = 5, [HEAD_BROADCAST_FLAG](#) = 6, [HEAD_UNUSED](#) = 7 }
- enum [DCCLHeaderBits](#) { [HEAD_CCL_ID_SIZE](#) = 8, [HEAD_DCCL_ID_SIZE](#) = 9, [HEAD_TIME_SIZE](#) = 17, [HEAD_SRC_ID_SIZE](#) = 5, [HEAD_DEST_ID_SIZE](#) = 5, [HEAD_FLAG_SIZE](#) = 1, [HEAD_UNUSED_SIZE](#) = 2 }
- enum [DCCLType](#) { [dccl_static](#), [dccl_bool](#), [dccl_int](#), [dccl_float](#), [dccl_enum](#), [dccl_string](#), [dccl_hex](#) }
Enumeration of DCCL types used for sending messages. [dccl_enum](#) and [dccl_string](#) primarily map to [cpp_string](#), [dccl_bool](#) to [cpp_bool](#), [dccl_int](#) to [cpp_long](#), [dccl_float](#) to [cpp_double](#).
- enum [DCCLCppType](#) { [cpp_notype](#), [cpp_bool](#), [cpp_string](#), [cpp_long](#), [cpp_double](#) }
Enumeration of C++ types used in DCCL.
- enum { [POWER2_BITS_IN_BYTE](#) = 3 }
- enum { [POWER2_NIBS_IN_BYTE](#) = 1 }

Functions

- std::string [to_str](#) ([DCCLHeaderPart](#) p)
- void [hex_decode](#) (const std::string &in, std::string *out)
- std::string [hex_decode](#) (const std::string &in)
- void [hex_encode](#) (const std::string &in, std::string *out)
- std::string [hex_encode](#) (const std::string &in)
- std::ostream & [operator<<](#) (std::ostream &out, const google::protobuf::Message &msg)
- void [bind](#) ([ModemDriverBase](#) &driver, [QueueManager](#) &queue_manager)

binds the driver link-layer callbacks to the [QueueManager](#)

- void [bind](#) ([MACManager](#) &mac, [ModemDriverBase](#) &driver)
- void [bind](#) ([ModemDriverBase](#) &driver, [QueueManager](#) &queue_manager, [MACManager](#) &mac)

bind all three (shortcut to calling the other three bind functions)

- template<typename Signal , typename Slot >
void [connect](#) (Signal *signal, Slot slot)

connect a signal to a function slot (non-member of a class)

- template<typename Signal , typename Obj , typename A1 >
void [connect](#) (Signal *signal, Obj *obj, void(Obj::*mem_func)(A1))

connect a signal to a member function with one argument

- template<typename Signal , typename Obj , typename A1 , typename A2 >
void [connect](#) (Signal *signal, Obj *obj, void(Obj::*mem_func)(A1, A2))

connect a signal to a member function with two arguments

- template<typename Signal , typename Obj , typename A1 , typename A2 , typename A3 >
void [connect](#) (Signal *signal, Obj *obj, void(Obj::*mem_func)(A1, A2, A3))

connect a signal to a member function with three arguments

- template<typename Value >
std::ostream & [operator](#)<< (std::ostream &out, const std::map< std::string, - Value > &m)

use this for displaying a human readable version

- template<typename Value >
std::ostream & [operator](#)<< (std::ostream &out, const std::multimap< std::string, Value > &m)
- std::ostream & [operator](#)<< (std::ostream &out, const std::set< unsigned > &s)

use this for displaying a human readable version of this STL object

- std::ostream & [operator](#)<< (std::ostream &out, const std::set< std::string > &s)

use this for displaying a human readable version of this STL object

- std::ostream & [operator](#)<< (std::ostream &out, const [DCCLCodec](#) &d)

outputs information about all available messages (same as std::string summary())

- unsigned [bits2bytes](#) (unsigned bits)
- unsigned [bytes2bits](#) (unsigned bytes)
- unsigned [bytes2nibs](#) (unsigned bytes)
- unsigned [nibs2bytes](#) (unsigned nibs)
- std::string [type_to_string](#) ([DCCLType](#) type)
- std::string [type_to_string](#) ([DCCLCppType](#) type)
- void [bitset2string](#) (const boost::dynamic_bitset< unsigned char > &body_bits, std::string &body)
- void [string2bitset](#) (boost::dynamic_bitset< unsigned char > &body_bits, const std::string &body)
- std::ostream & [operator](#)<< (std::ostream &out, const [DCCLMessage](#) &message)
- std::ostream & [operator](#)<< (std::ostream &out, const [DCCLPublish](#) &publish)
- std::ostream & [operator](#)<< (std::ostream &os, const [acomms::DCCLMessage-Val](#) &mv)

- `std::ostream & operator<< (std::ostream &os, const std::vector< acomms::DCCLMessageVal > &vm)`
- `std::ostream & operator<< (std::ostream &out, const DCCLMessageVar &m)`
- `std::ostream & operator<< (std::ostream &os, const Queue &oq)`
- `std::ostream & operator<< (std::ostream &out, const QueueManager &d)`
outputs information about all available messages (same as std::string summary())

Variables

- `const unsigned BITS_IN_BYTE = 8`
- `const unsigned NIBS_IN_BYTE = 2`
- `const int BROADCAST_ID = 0`
special modem id for the broadcast destination - no one is assigned this address. Analogous to 192.168.1.255 on a 192.168.1.0 subnet
- `const int QUERY_DESTINATION_ID = -1`
special modem id used internally to goby-acomms for indicating that the MAC layer (libamac) is agnostic to the next destination. The next destination is thus set by the data provider (typically libqueue).
- `const unsigned char DCCL_CCL_HEADER = 32`
- `const double NaN = std::numeric_limits<double>::quiet_NaN()`
- `const unsigned DCCL_NUM_HEADER_BYTES = 6`
- `const unsigned DCCL_NUM_HEADER_PARTS = 8`
- `const std::string DCCL_HEADER_NAMES []`
- `const unsigned MIN_ID = 1`
- `const unsigned MAX_ID = 1 << HEAD_DCCL_ID_SIZE`
- `const unsigned MULTIMESSAGE_MASK = 1 << 7`
- `const unsigned BROADCAST_MASK = 1 << 6`
- `const unsigned VAR_ID_MASK = 0xFF ^ MULTIMESSAGE_MASK ^ BROADCAST_MASK`
- `const unsigned USER_FRAME_NEXT_SIZE_BYTES = 1`
- `const boost::posix_time::time_duration ON_DEMAND_SKEW = boost::posix_time::seconds(1)`

14.2.1 Detailed Description

Objects pertaining to acoustic communications (acomms)

14.2.2 Typedef Documentation

14.2.2.1 `typedef boost::function<void (DCCLMessageVal&>> goby::acomms::AlgFunction1`

`boost::function` for a function taking a single [DCCLMessageVal](#) reference. Used for algorithm callbacks.

Think of this as a generalized version of a function pointer (void (*)(DCCLMessageVal&)). See http://www.boost.org/doc/libs/1_34_0/doc/html/function.html for more on boost::function.

Definition at line 36 of file message_algorithms.h.

14.2.2.2 `typedef boost::function<void (DCCLMessageVal&, const std::vector<DCCLMessageVal>&)> goby::acomms::AlgFunction2`

boost::function for a function taking a dccl::MessageVal reference, and the MessageVal of a second part of the message. Used for algorithm callbacks.

Think of this as a generalized version of a function pointer (void (*)(DCCLMessageVal&, const DCCLMessageVal&)). See http://www.boost.org/doc/libs/1_34_0/doc/html/function.html for more on boost::function.

Definition at line 45 of file message_algorithms.h.

14.2.3 Enumeration Type Documentation

14.2.3.1 enum goby::acomms::DCCLCppType

Enumeration of C++ types used in DCCL.

Enumerator:

cpp_notype not one of the C++ types used in DCCL
cpp_bool C++ bool
cpp_string C++ std::string
cpp_long C++ long
cpp_double C++ double

Definition at line 46 of file dccl_constants.h.

14.2.3.2 enum goby::acomms::DCCLType

Enumeration of DCCL types used for sending messages. dccl_enum and dccl_string primarily map to cpp_string, dccl_bool to cpp_bool, dccl_int to cpp_long, dccl_float to cpp_double.

Enumerator:

dccl_static <static>
dccl_bool <bool>
dccl_int <int>
dccl_float <float>
dccl_enum <enum>
dccl_string <string>
dccl_hex <hex>

Definition at line 37 of file dccl_constants.h.

14.2.4 Function Documentation

14.2.4.1 void **goby::acomms::bind** (MACManager & *mac*, ModemDriverBase & *driver*)
[inline]

binds the MAC initiate transmission callback to the driver and the driver parsed message callback to the MAC

Definition at line 53 of file bind.h.

14.2.5 Variable Documentation

14.2.5.1 const std::string **goby::acomms::DCCL_HEADER_NAMES** []

Initial value:

```
{ "_ccl_id",
                                     "_id",
                                     "_time",
                                     "_src_id",
                                     "_dest_id",
                                     "_multimessage_flag",
                                     "_broadcast_flag",
                                     "_unused",
}
```

Definition at line 59 of file acomms_constants.h.

14.3 goby::acomms::protobuf Namespace Reference

Contains Google Protocol Buffers messages and helper functions. See specific .proto files for definition of the actual messages (e.g. [modem_message.proto](#)).

Functions

- bool **operator**< (const std::map< int, protobuf::Slot >::iterator &a, const std::map< int, protobuf::Slot >::iterator &b)
- bool **operator**== (const protobuf::Slot &a, const protobuf::Slot &b)
Are two Slots equal?
- bool **operator**< (const goby::acomms::protobuf::QueueKey &a, const goby::acomms::protobuf::QueueKey &b)
- bool **operator**> (const goby::acomms::protobuf::QueueKey &a, const goby::acomms::protobuf::QueueKey &b)
- bool **operator**>= (const goby::acomms::protobuf::QueueKey &a, const goby::acomms::protobuf::QueueKey &b)
- bool **operator**<= (const goby::acomms::protobuf::QueueKey &a, const goby::acomms::protobuf::QueueKey &b)
- bool **operator**== (const goby::acomms::protobuf::QueueKey &a, const goby::acomms::protobuf::QueueKey &b)

14.3.1 Detailed Description

Contains Google Protocol Buffers messages and helper functions. See specific .proto files for definition of the actual messages (e.g. [modem_message.proto](#)).

14.4 goby::util Namespace Reference

Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc.

Namespaces

- namespace [tcolor](#)
Contains functions for adding color to Terminal window streams.

Classes

- class [LineBasedInterface](#)
basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)
- class [SerialClient](#)
provides a basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control
- class [TCPClient](#)
provides a basic TCP client for line by line text based communications to a remote TCP server
- class [TCPServer](#)
provides a basic TCP server for line by line text based communications to a one or more remote TCP clients
- class [FlexNCurses](#)
Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.
- class [FlexOstream](#)
Forms the basis of the Goby logger: std::ostream derived class for holding the [FlexOStreamBuf](#).
- struct [Logger](#)
Holds static objects of the Goby [Logger](#).
- class [FlexOStreamBuf](#)
Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by [goby::util::FlexOstream](#) for the Goby [Logger](#) (glogger)
- struct [Colors](#)
Represents the eight available terminal colors (and bold variants)
- class [TermColor](#)
Converts between string, escape code, and enumeration representations of the terminal colors.

Functions

Binary encoding

- bool [char_array2hex_string](#) (const unsigned char *c, std::string &s, const unsigned int n)
converts a char (byte) array into a hex string
- bool [hex_string2char_array](#) (unsigned char *c, const std::string &s, const unsigned int n)
turns a string of hex chars ABCDEF into a character array reading each byte 0xAB, 0xCD, 0xEF, etc.
- std::string [long2binary_string](#) (unsigned long l, unsigned short bits)
return a string represented the binary value of 'l' for 'bits' number of bits which reads MSB -> LSB
- std::string [binary_string2hex_string](#) (const std::string &bs)
converts a binary string ("1000101010101010") into a hex string ("8AAA")
- std::string [dyn_bitset2hex_string](#) (const boost::dynamic_bitset< unsigned char > &bits, unsigned trim_to_bytes_size=0)
converts a boost::dynamic_bitset (similar to std::bitset but without compile time size requirements) into a hex string
- std::string [hex_string2binary_string](#) (const std::string &bs)
converts a hex string ("8AAA") into a binary string ("1000101010101010")
- boost::dynamic_bitset < unsigned char > [hex_string2dyn_bitset](#) (const std::string &hs, unsigned bits_size=0)
converts a hex string ("8AAA") into a dynamic_bitset
- template<typename T >
bool [hex_string2number](#) (const std::string &s, T &t)
attempts to convert a hex string into a numerical representation (of type T)
- template<typename T >
bool [number2hex_string](#) (std::string &s, const T &t, unsigned int width=2)
converts a decimal number of type T into a hex string
- template<typename T >
std::string [number2hex_string](#) (const T &t, unsigned int width=2)
converts a decimal number of type T into a hex string assuming success

Logger

- [FlexOstream](#) & [glogger](#) (logger_lock::LockAction lock_action=logger_lock::none)
Access the Goby logger through this function.
- std::ostream & **operator**<< ([FlexOstream](#) &out, char c)
- std::ostream & **operator**<< ([FlexOstream](#) &out, signed char c)
- std::ostream & **operator**<< ([FlexOstream](#) &out, unsigned char c)
- std::ostream & **operator**<< ([FlexOstream](#) &out, const char *s)
- std::ostream & **operator**<< ([FlexOstream](#) &out, const signed char *s)
- std::ostream & **operator**<< ([FlexOstream](#) &out, const unsigned char *s)

Science

- double [unbiased_round](#) (double r, double dec)

- double [mackenzie_soundspeed](#) (double T, double S, double D)

String

- template<typename To, typename From >
To [as](#) (From from, typename boost::enable_if< boost::is_fundamental< To >
>::type *dummy=0)
non-throwing lexical cast (e.g. assert(as<double>("3.2") == 3.2)). For fundamental types (double, int, etc.)
- template<typename To, typename From >
To [as](#) (From from, typename boost::disable_if< boost::is_fundamental< To >
>::type *dummy=0)
non-throwing lexical cast (e.g. assert(as<double>("3.2") == 3.2)) for non-fundamental types (MyClass(), etc.)
- template<>
bool [as](#)< bool, std::string > (std::string from, void *dummy)
specialization of as() for string -> bool
- template<>
std::string [as](#)< std::string, bool > (bool from, void *dummy)
specialization of as() for bool -> string
- void [stripblanks](#) (std::string &s)
remove all blanks from string s
- bool [val_from_string](#) (std::string &out, const std::string &str, const std::string &key)
- template<typename T >
bool [val_from_string](#) (T &out, const std::string &str, const std::string &key)
- bool [val_from_string](#) (bool &out, const std::string &str, const std::string &key)
specialization of val_from_string for boolean 'out'

Time

- int64_t [microtime](#) ()
- boost::posix_time::ptime [goby_time](#) ()
Always use for current time within the Goby project.
- std::string [goby_time_as_string](#) (const boost::posix_time::ptime &t=[goby_time](#)())
Simple string representation of [goby_time](#)()
- std::string [goby_file_timestamp](#) ()
ISO string representation of [goby_time](#)()
- double [ptime2unix_double](#) (boost::posix_time::ptime given_time)
convert from boost date_time ptime to the number of seconds (including fractional) since 1/1/1970 0:00 UTC ("UNIX Time")
- boost::posix_time::ptime [unix_double2ptime](#) (double given_time)
convert to boost date_time ptime from the number of seconds (including fractional) since 1/1/1970 0:00 UTC ("UNIX Time"): good to the microsecond
- boost::posix_time::ptime [time_t2ptime](#) (std::time_t t)
convert to ptime from time_t from [time.h](#) (whole seconds since UNIX)
- std::time_t [ptime2time_t](#) (boost::posix_time::ptime t)
convert from ptime to time_t from [time.h](#) (whole seconds since UNIX)
- double [time_duration2double](#) (boost::posix_time::time_duration time_of_day)
time duration to double number of seconds: good to the microsecond

Variables

- const std::string **esc_red** = "\33[31m"
- const std::string **esc_lt_red** = "\33[91m"
- const std::string **esc_green** = "\33[32m"
- const std::string **esc_lt_green** = "\33[92m"
- const std::string **esc_yellow** = "\33[33m"
- const std::string **esc_lt_yellow** = "\33[93m"
- const std::string **esc_blue** = "\33[34m"
- const std::string **esc_lt_blue** = "\33[94m"
- const std::string **esc_magenta** = "\33[35m"
- const std::string **esc_lt_magenta** = "\33[95m"
- const std::string **esc_cyan** = "\33[36m"
- const std::string **esc_lt_cyan** = "\33[96m"
- const std::string **esc_white** = "\33[37m"
- const std::string **esc_lt_white** = "\33[97m"
- const std::string **esc_nocolor** = "\33[0m"

14.4.1 Detailed Description

Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc.

14.4.2 Function Documentation

14.4.2.1 `template<typename To , typename From > To goby::util::as( From from, typename boost::enable_if< boost::is_fundamental< To > >::type * dummy = 0 )`

non-throwing lexical cast (e.g. `assert(as<double>("3.2") == 3.2)`). For fundamental types (double, int, etc.)

Parameters

<i>from</i>	value to cast from
-------------	--------------------

Returns

to value to cast to

Exceptions

<i>none</i>

Examples:

[acomms/chat/chat.cpp](#).

Definition at line 47 of file string.h.

14.4.2.2 `template<typename To , typename From > To goby::util::as ( From from, typename boost::disable_if< boost::is_fundamental< To > >::type * dummy = 0 )`

non-throwing lexical cast (e.g. `assert(as<double>("3.2") == 3.2)`) for non-fundamental types (`MyClass()`, etc.)

Parameters

<i>from</i>	value to cast from
-------------	--------------------

Returns

to value to cast to

Exceptions

<i>none</i>

Definition at line 64 of file `string.h`.

14.4.2.3 `bool goby::util::char_array2hex_string ( const unsigned char * c, std::string & s, const unsigned int n ) [inline]`

converts a char (byte) array into a hex string

Parameters

<i>c</i>	pointer to array of char
<i>s</i>	reference to string to put char into as hex
<i>n</i>	length of <i>c</i> the first two hex chars in <i>s</i> are the 0 index in <i>c</i>

Definition at line 47 of file `binary.h`.

14.4.2.4 `goby::util::FlexOstream & goby::util::glogger ( logger_lock::LockAction lock_action = logger_lock::none )`

Access the Goby logger through this function.

For normal (non thread safe use), do not pass any parameters: `glogger() << "some text" << std::endl;`

To group messages, pass the `group(group_name)` manipulator, where `group_name` is a previously defined group (by call to `glogger().add_group(Group)`). For example: `glogger() << group("incoming") << "received message foo" << std::endl;`

For thread safe use, use `glogger(lock)` and then insert the "unlock" manipulator when relinquishing the lock. The "unlock" manipulator MUST be inserted before the next call to `glogger(lock)`. **Nothing** must throw exceptions between `glogger(lock)` and `unlock`. For example: `glogger(lock) << "my thread is the best" << std::endl << unlock;`

Parameters

<i>lock_action</i>	logger_lock::lock to lock access to the logger (for thread safety) or logger_lock::none for no mutex action (typical)
--------------------	---

Returns

reference to Goby logger (std::ostream derived class [FlexOstream](#))

Definition at line 24 of file flex_ostream.cpp.

14.4.2.5 `std::string goby::util::hex_string2binary_string ( const std::string & bs )`
`[inline]`

converts a hex string ("8AAA") into a binary string ("1000101010101010")

only works on whole byte string (even number of nibbles)

Definition at line 118 of file binary.h.

14.4.2.6 `template<typename T> bool goby::util::hex_string2number ( const std::string & s, T & t )`

attempts to convert a hex string into a numerical representation (of type T)

Returns

true if conversion succeeds, false otherwise

Definition at line 149 of file binary.h.

14.4.2.7 `double goby::util::mackenzie_soundspeed ( double T, double S, double D )`
`[inline]`

K.V. Mackenzie, Nine-term equation for the sound speed in the oceans (1981) - J. Acoust. Soc. Am. 70(3), pp 807-812 <http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=JASMA-N000070000003000807000001&idtype=cvips&gifs=yes>

Parameters

<i>T</i>	temperature in degrees Celcius (see paper for applicable ranges)
<i>S</i>	salinity (unitless, calculated using Practical Salinity Scale) (see paper for applicable ranges)
<i>D</i>	depth in meters (see paper for applicable ranges)

Returns

speed of sound in meters per second

Definition at line 60 of file sci.h.

14.4.2.8 `template<typename T> bool goby::util::number2hex_string ( std::string & s, const T & t, unsigned int width = 2 )`

converts a decimal number of type T into a hex string

Parameters

<i>s</i>	string reference to store result in
<i>t</i>	decimal number to convert
<i>width</i>	desired width (in characters) of return string. Width should be twice the number of bytes

Returns

true if successful, false otherwise

Definition at line 164 of file binary.h.

14.4.2.9 `template<typename T> std::string goby::util::number2hex_string ( const T & t, unsigned int width = 2 )`

converts a decimal number of type T into a hex string assuming success

Parameters

<i>t</i>	decimal number to convert
<i>width</i>	desired width (in characters) of return string. Width should be twice the number of bytes

Returns

hex string

Definition at line 177 of file binary.h.

14.4.2.10 `double goby::util::unbiased_round ( double r, double dec ) [inline]`

round 'r' to 'dec' number of decimal places we want no upward bias so round 5 up if odd next to it, down if even

Parameters

<i>r</i>	value to round
<i>dec</i>	number of places past the decimal to round (e.g. dec=1 rounds to tenths)

Returns

r rounded

Definition at line 41 of file sci.h.

14.4.2.11 `bool goby::util::val_from_string ( std::string & out, const std::string & str, const std::string & key ) [inline]`

find 'key' in 'str' and if successful put it in out and return true deal with these basic forms:
str = foo=1,bar=2,pig=3 str = foo=1,bar={2,3,4,5},pig=3

Parameters

<i>out</i>	string to return value in
<i>str</i>	haystack to search
<i>key</i>	needle to find

Returns

true if key is in str, false otherwise

Definition at line 112 of file string.h.

14.4.2.12 `template<typename T> bool goby::util::val_from_string ( T & out, const std::string & str, const std::string & key ) [inline]`

variation of [val_from_string\(\)](#) for arbitrary return type

Returns

false if 'key' not in 'str' OR if 'out' is not of proper type T

Definition at line 174 of file string.h.

14.5 goby::util::tcolor Namespace Reference

Contains functions for adding color to Terminal window streams.

Functions

- `std::ostream & add_escape_code (std::ostream &os, const std::string &esc_code)`
- `std::ostream & red (std::ostream &os)`
All text following this manipulator is red. (e.g. std::cout << red << "text";)
- `std::ostream & lt_red (std::ostream &os)`
All text following this manipulator is light red (e.g. std::cout << lt_red << "text";)
- `std::ostream & green (std::ostream &os)`
All text following this manipulator is green (e.g. std::cout << green << "text";)
- `std::ostream & lt_green (std::ostream &os)`
All text following this manipulator is light green (e.g. std::cout << lt_green << "text";)
- `std::ostream & yellow (std::ostream &os)`
All text following this manipulator is yellow (e.g. std::cout << yellow << "text";)

- `std::ostream & lt_yellow (std::ostream &os)`
All text following this manipulator is light yellow (e.g. `std::cout << lt_yellow << "text";`)
- `std::ostream & blue (std::ostream &os)`
All text following this manipulator is blue (e.g. `std::cout << blue << "text";`)
- `std::ostream & lt_blue (std::ostream &os)`
All text following this manipulator is light blue (e.g. `std::cout << lt_blue << "text";`)
- `std::ostream & magenta (std::ostream &os)`
All text following this manipulator is magenta (e.g. `std::cout << magenta << "text";`)
- `std::ostream & lt_magenta (std::ostream &os)`
All text following this manipulator is light magenta (e.g. `std::cout << lt_magenta << "text";`)
- `std::ostream & cyan (std::ostream &os)`
All text following this manipulator is cyan (e.g. `std::cout << cyan << "text";`)
- `std::ostream & lt_cyan (std::ostream &os)`
All text following this manipulator is light cyan (e.g. `std::cout << lt_cyan << "text";`)
- `std::ostream & white (std::ostream &os)`
All text following this manipulator is white (e.g. `std::cout << white << "text";`)
- `std::ostream & lt_white (std::ostream &os)`
All text following this manipulator is bright white (e.g. `std::cout << lt_white << "text";`)
- `std::ostream & nocolor (std::ostream &os)`
All text following this manipulator is uncolored (e.g. `std::cout << green << "green" << nocolor << "uncolored";`)

14.5.1 Detailed Description

Contains functions for adding color to Terminal window streams.

14.5.2 Function Documentation

14.5.2.1 `std::ostream & goby::util::tcolor::add_escape_code ( std::ostream & os, const std::string & esc_code )`

Append the given escape code to the stream os

Parameters

<code>os</code>	ostream to append to
<code>esc_code</code>	escape code to append (e.g. <code>"\33[31m"</code>)

Definition at line 23 of file `term_color.cpp`.

15 Class Documentation

15.1 goby::acomms::ABCDriver Class Reference

provides an API to the imaginary ABC modem (as an example how to write drivers)

```
#include <acomms/libmodemdriver/abc_driver.h>
```

Inherits [goby::acomms::ModemDriverBase](#).

Public Member Functions

- [ABCDriver](#) (std::ostream *log=0)
Instantiate with an optional logger object.
- void [startup](#) (const protobuf::DriverConfig &cfg)
Starts the modem driver. Must be called before [do_work\(\)](#).
- void [shutdown](#) ()
Shuts down the modem driver.
- void [do_work](#) ()
Allows the modem driver to do its work.
- void [handle_initiate_transmission](#) (protobuf::ModemMsgBase *m)
Virtual initiate_transmission method. Typically connected to [MACManager::signal_initiate_transmission\(\)](#) using [bind\(\)](#).

15.1.1 Detailed Description

provides an API to the imaginary ABC modem (as an example how to write drivers)

Examples:

[libmodemdriver/driver_simple/driver_simple.cpp](#).

Definition at line 33 of file [abc_driver.h](#).

15.1.2 Member Function Documentation

15.1.2.1 void goby::acomms::ABCDriver::do_work () [virtual]

Allows the modem driver to do its work.

Should be called regularly to perform the work of the driver as the driver **does not** run in its own thread. This allows us to guarantee that no signals are called except inside this [do_work](#) method.

Implements [goby::acomms::ModemDriverBase](#).

Definition at line 119 of file [abc_driver.cpp](#).

15.1.2.2 `void goby::acomms::ABCDriver::handle_initiate_transmission (`
`protobuf::ModemMsgBase * m ) [virtual]`

Virtual initiate_transmission method. Typically connected to [MACManager::signal_initiate_transmission\(\)](#) using [bind\(\)](#).

Parameters

<i>m</i>	ModemMsgBase (defined in modem_message.proto) containing the details of the transmission to be started. This does <i>*not*</i> contain data, which will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request)
----------	--

Implements [goby::acomms::ModemDriverBase](#).

Definition at line 73 of file `abc_driver.cpp`.

15.1.2.3 `void goby::acomms::ABCDriver::startup ( const protobuf::DriverConfig & cfg )`
`[virtual]`

Starts the modem driver. Must be called before [do_work\(\)](#).

Parameters

<i>cfg</i>	Startup configuration for the driver and modem. DriverConfig is defined in driver_base.proto . Derived classes can define extensions (see http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions) to DriverConfig to handle modem specific configuration.
------------	---

Implements [goby::acomms::ModemDriverBase](#).

Definition at line 30 of file `abc_driver.cpp`.

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

- `acomms/libmodemdriver/abc_driver.h`
- `acomms/libmodemdriver/abc_driver.cpp`

15.2 goby::acomms::DCCLCodec Class Reference

provides an API to the Dynamic CCL Codec.

```
#include <goby/acomms/dccl.h>
```

Public Member Functions

- `std::vector< DCCLMessage > & messages ()`
- `const ManipulatorManager & manip_manager () const`

Constructors/Destructor

- `DCCLCodec` (`std::ostream *log=0`)
Instantiate optionally with a ostream logger (for human readable output)
- `~DCCLCodec` ()
destructor

Codec functions.

This is where the real work happens.

- `template<typename Key >`
`void encode` (`const Key &k`, `std::string &bytes`, `const std::map< std::string, DCCLMessageVal > &m`)
Encode a message.
- `template<typename Key >`
`void encode` (`const Key &k`, `std::string &bytes`, `const std::map< std::string, std::vector< DCCLMessageVal > > &m`)
Encode a message.
- `void decode` (`const std::string &bytes`, `std::map< std::string, DCCLMessageVal > &m`)
Decode a message.
- `void decode` (`const std::string &bytes`, `std::map< std::string, std::vector< DCCLMessageVal > > &m`)
Decode a message.

Informational Methods

- `template<typename Key >`
`std::string summary` (`const Key &k`) `const`
- `std::string summary` () `const`
long summary of a message for all loaded messages
- `template<typename Key >`
`std::string brief_summary` (`const Key &k`) `const`
brief summary of a message for a given Key (std::string name or unsigned id)
- `std::string brief_summary` () `const`
brief summary of a message for all loaded messages
- `unsigned message_count` ()
- `template<typename Key >`
`unsigned get_repeat` (`const Key &k`)
- `std::set< unsigned > all_message_ids` ()
- `std::set< std::string > all_message_names` ()
- `template<typename Key >`
`std::map< std::string, std::string > message_var_names` (`const Key &k`) `const`
- `std::string id2name` (`unsigned id`)
- `unsigned name2id` (`const std::string &name`)

Publish/subscribe architecture related methods

Methods written largely to support DCCL in the context of a publish/subscribe architecture (e.g., see MOOS (<http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php>)). The other methods provide a complete interface for encoding and decoding DCCL messages. However, the methods listed here extend the functionality to allow for

- *message creation triggering (a message is encoded on a certain event, either time based or publish based)*
- *encoding that will parse strings of the form: "key1=value,key2=value,key3=value"*
- *decoding to an arbitrarily formatted string (similar concept to printf)*

These methods will be useful if you are interested in any of the features mentioned above.

- `template<typename Key >`
`void pubsub_encode (const Key &k, protobuf::ModemDataTransmission *msg, const std::map< std::string, std::vector< DCCLMessageVal > > &pubsub_vals)`
Encode a message using `<src_var>` tags instead of `<name>` tags.
- `template<typename Key >`
`void pubsub_encode (const Key &k, protobuf::ModemDataTransmission *msg, const std::map< std::string, DCCLMessageVal > &pubsub_vals)`
Encode a message using `<src_var>` tags instead of `<name>` tags.
- `void pubsub_decode (const protobuf::ModemDataTransmission &msg, std::multimap< std::string, DCCLMessageVal > *pubsub_vals)`
Decode a message using formatting specified in `<publish>` tags.
- `template<typename Key >`
`std::set< std::string > get_pubsub_src_vars (const Key &k)`
what moos variables do i need to provide to create a message with a call to encode_using_src_vars
- `template<typename Key >`
`std::set< std::string > get_pubsub_all_vars (const Key &k)`
for a given message name, all architecture variables (sources, input, destination, trigger)
- `template<typename Key >`
`std::set< std::string > get_pubsub_encode_vars (const Key &k)`
all architecture variables needed for encoding (includes trigger)
- `template<typename Key >`
`std::set< std::string > get_pubsub_decode_vars (const Key &k)`
for a given message, all architecture variables for decoding (input)
- `template<typename Key >`
`std::string get_outgoing_hex_var (const Key &k)`
returns outgoing architecture hexadecimal variable
- `template<typename Key >`
`std::string get_incoming_hex_var (const Key &k)`
returns incoming architecture hexadecimal variable
- `bool is_publish_trigger (std::set< unsigned > &id, const std::string &key, const std::string &value)`
look if key / value are trigger for any loaded messages if so, store to id and return true
- `bool is_time_trigger (std::set< unsigned > &id)`
look if the time is right for trigger for any loaded messages if so, store to id and return true
- `bool is_incoming (unsigned &id, const std::string &key)`
see if this key is for an incoming message if so, return id for decoding

Initialization Methods.

These methods are intended to be called before doing any work with the class. However, they may be called at any time as desired.

- void [set_cfg](#) (const protobuf::DCCLConfig &cfg)
Set (and overwrite completely if present) the current configuration. (protobuf::DCCLConfig defined in [dccl.proto](#))
- void [merge_cfg](#) (const protobuf::DCCLConfig &cfg)
Set (and merge "repeat" fields) the current configuration. (protobuf::DCCLConfig defined in [dccl.proto](#))
- void [add_algorithm](#) (const std::string &name, [AlgFunction1](#) func)
Add an algorithm callback for a MessageVal. The return value is stored back into the input parameter (MessageVal). See [test.cpp](#) for an example.
- void [add_adv_algorithm](#) (const std::string &name, [AlgFunction2](#) func)
Add an advanced algorithm callback for any DCCL C++ type that may also require knowledge of all the other message variables and can optionally have additional parameters.
- static void [add_flex_groups](#) ([util::FlexOstream](#) *tout)
Registers the group names used for the FlexOstream logger.

15.2.1 Detailed Description

provides an API to the Dynamic CCL Codec.

See also

[dccl.proto](#) and [modem_message.proto](#) for definition of Google Protocol Buffers messages (namespace [goby::acomms::protobuf](#)).

Examples:

[acomms/chat/chat.cpp](#), [libdccl/dccl_simple/dccl_simple.cpp](#), [libdccl/plusnet/plusnet.cpp](#), [libdccl/test/test.cpp](#), and [libdccl/two_message/two_message.cpp](#).

Definition at line 87 of file [dccl.h](#).

15.2.2 Constructor & Destructor Documentation

15.2.2.1 goby::acomms::DCCLCodec::DCCLCodec (std::ostream * log = 0)

Instantiate optionally with a ostream logger (for human readable output)

Parameters

<i>log</i>	std::ostream object or FlexOstream to capture all humanly readable runtime and debug information (optional).
------------	--

Definition at line 47 of file dccl.cpp.

15.2.3 Member Function Documentation

15.2.3.1 void goby::acomms::DCCLCodec::add_adv_algorithm (const std::string & name, AlgFunction2 func)

Add an advanced algorithm callback for any DCCL C++ type that may also require knowledge of all the other message variables and can optionally have additional parameters.

Parameters

<i>name</i>	name of the algorithm (<... algorithm="name:param1:param2">)
<i>func</i>	has the form void name(MessageVal& val_to_edit, const std::vector<std::string> params, const std::map<std::string, MessageVal>& vals) (see AdvAlgFunction3). func can be a function pointer (&name) or any function object supported by boost::function (http://www.boost.org/doc/libs/1_34_0/doc/html/function.html).
<i>params</i>	(passed to func) a list of colon separated parameters passed by the user in the XML file. param[0] is the name.
<i>vals</i>	(passed to func) a map of <name> to current values for all message variables.

Definition at line 122 of file dccl.cpp.

15.2.3.2 void goby::acomms::DCCLCodec::add_algorithm (const std::string & name, AlgFunction1 func)

Add an algorithm callback for a MessageVal. The return value is stored back into the input parameter (MessageVal). See test.cpp for an example.

Parameters

<i>name</i>	name of the algorithm (<... algorithm="name">)
<i>func</i>	has the form void name(MessageVal& val_to_edit) (see AdvAlgFunction1). can be a function pointer (&name) or any function object supported by boost::function (http://www.boost.org/doc/libs/1_34_0/doc/html/function.html)

Definition at line 116 of file dccl.cpp.

15.2.3.3 std::set< unsigned > goby::acomms::DCCLCodec::all_message_ids ()

Returns

set of all message ids loaded

Definition at line 84 of file dccl.cpp.

15.2.3.4 `std::set< std::string > goby::acomms::DCCLCodec::all_message_names ( )`

Returns

set of all message names loaded

Definition at line 91 of file dccl.cpp.

15.2.3.5 `void goby::acomms::DCCLCodec::decode ( const std::string & bytes, std::map< std::string, DCCLMessageVal > & m ) [inline]`

Decode a message.

Parameters

<i>k</i>	can either be std::string (the name of the message) or unsigned (the id of the message)
<i>bytes</i>	the bytes to be decoded.
<i>m</i>	map of std::string (<name>) to MessageVal to store the values to be decoded. No fields can be arrays using this call. If fields are arrays, only the first value is returned.

Examples:

[libdccl/dccl_simple/dccl_simple.cpp](#).

Definition at line 175 of file dccl.h.

15.2.3.6 `void goby::acomms::DCCLCodec::decode ( const std::string & bytes, std::map< std::string, std::vector< DCCLMessageVal > > & m ) [inline]`

Decode a message.

Parameters

<i>k</i>	can either be std::string (the name of the message) or unsigned (the id of the message)
<i>bytes</i>	the bytes to be decoded.
<i>m</i>	map of std::string (<name>) to MessageVal to store the values to be decoded

Definition at line 192 of file dccl.h.

15.2.3.7 `template<typename Key > void goby::acomms::DCCLCodec::encode ( const Key & k, std::string & bytes, const std::map< std::string, DCCLMessageVal > & m ) [inline]`

Encode a message.

Parameters

<i>k</i>	can either be std::string (the name of the message) or unsigned (the id of the message)
<i>bytes</i>	location for the encoded bytes to be stored. this is suitable for sending to the Micro-Modem
<i>m</i>	map of std::string (<name>) to a vector of MessageVal representing the values to encode. No fields can be arrays using this call. If fields are arrays, all values but the first in the array will be set to NaN or blank.

Examples:

[libdccl/dccl_simple/dccl_simple.cpp](#).

Definition at line 147 of file dccl.h.

```
15.2.3.8 template<typename Key > void goby::acomms::DCCLCodec::encode
( const Key & k, std::string & bytes, const std::map< std::string, std::vector<
DCCLMessageVal >> & m ) [inline]
```

Encode a message.

Parameters

<i>k</i>	can either be std::string (the name of the message) or unsigned (the id of the message)
<i>bytes</i>	location for the encoded bytes to be stored. this is suitable for sending to the Micro-Modem
<i>m</i>	map of std::string (<name>) to a vector of MessageVal representing the values to encode. Fields can be arrays.

Definition at line 165 of file dccl.h.

```
15.2.3.9 template<typename Key > unsigned goby::acomms::DCCLCodec::get_repeat
( const Key & k ) [inline]
```

Returns

repeat value (number of copies of the message per encode)

Definition at line 223 of file dccl.h.

```
15.2.3.10 std::string goby::acomms::DCCLCodec::id2name ( unsigned id )
[inline]
```

Parameters

<i>id</i>	message id
-----------	------------

Returns

name of message

Definition at line 237 of file dccl.h.

15.2.3.11 `unsigned goby::acomms::DCCLCodec::message_count ( ) [inline]`

how many message are loaded?

Returns

number of messages loaded

Examples:

[libdccl/two_message/two_message.cpp](#).

Definition at line 219 of file dccl.h.

15.2.3.12 `template<typename Key > std::map<std::string, std::string>
goby::acomms::DCCLCodec::message_var_names (const Key & k) const
[inline]`

Returns

map of names to DCCL types needed to encode a given message

Definition at line 232 of file dccl.h.

15.2.3.13 `unsigned goby::acomms::DCCLCodec::name2id (const std::string & name)
[inline]`

Parameters

<i>name</i>	message name
-------------	--------------

Returns

id of message

Definition at line 240 of file dccl.h.

15.2.3.14 `void goby::acomms::DCCLCodec::pubsub_decode (const
protobuf::ModemDataTransmission & msg, std::multimap< std::string,
DCCLMessageVal > * pubsub_vals) [inline]`

Decode a message using formatting specified in [<publish>](#) tags.

Values will be received in two maps, one of strings and the other of doubles. The [<publish>](#) value will be placed either based on the "type" parameter of the [<publish_var>](#) tag (e.g. [<publish_var type="long">SOMEVAR</publish_var>](#) will be placed as a long). If no type parameter is given and the variable is numeric (e.g. "23242.23") it will be considered a double. If not numeric, it will be considered a string.

Parameters

<i>k</i>	can either be <code>std::string</code> (the name of the message) or unsigned (the id of the message)
<i>msg</i>	message to be decoded. (protobuf::ModemDataTransmission defined in modem_message.proto)
<i>vals</i>	pointer to <code>std::multimap</code> of publish variable name to <code>std::string</code> values.

Definition at line 341 of file `dccl.h`.

15.2.3.15 `template<typename Key > void goby::acomms::DCCLCodec::pubsub_encode ( const Key & k, protobuf::ModemDataTransmission * msg, const std::map< std::string, std::vector< DCCLMessageVal > > & pubsub_vals ) [inline]`

Encode a message using `<src_var>` tags instead of `<name>` tags.

Values can be passed in on one or more maps of names to values, similar to [DCC-LCodec::encode](#). Casts are made and string parsing of key=value comma delimited fields is performed. This differs substantially from the behavior of `encode` above. For example, take this message variable:

```
<int>
<name>myint</name>
<src_var>somevar</src_var>
</int>
```

Using this method you can pass `vals["somevar"] = "mystring=foo,blah=dog,myint=32"` or `vals["somevar"] = 32.0` and both cases will properly parse out 32 as the value for this field. In comparison, using the normal `encode` you would pass `vals["myint"] = 32`

Parameters

<i>k</i>	can either be <code>std::string</code> (the name of the message) or unsigned (the id of the message)
<i>msg</i>	location for the encoded message to be stored (protobuf::ModemDataTransmission defined in modem_message.proto)
<i>vals</i>	map of source variable name to MessageVal values.

Definition at line 278 of file `dccl.h`.

15.2.3.16 `template<typename Key > void goby::acomms::DCCLCodec::pubsub_encode ( const Key & k, protobuf::ModemDataTransmission * msg, const std::map< std::string, DCCLMessageVal > & pubsub_vals ) [inline]`

Encode a message using `<src_var>` tags instead of `<name>` tags.

Use this version if you do not have vectors of `src_var` values

Definition at line 319 of file `dccl.h`.

15.2.3.17 `template<typename Key > std::string goby::acomms::DCCLCodec::summary
(const Key & k) const [inline]`

long summary of a message for a given Key (std::string name or unsigned id)

Parameters

<code>k</code>	can either be std::string (the name of the message) or unsigned (the id of the message)
----------------	---

Definition at line 205 of file dccl.h.

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

- `acomms/libdccl/dccl.h`
- `acomms/libdccl/dccl.cpp`

15.3 goby::acomms::DCCLException Class Reference

[Exception](#) class for libdccl.

```
#include <acomms/libdccl/dccl_exception.h>
```

Inherits [goby::Exception](#).

Public Member Functions

- **DCCLException** (const std::string &s)

15.3.1 Detailed Description

[Exception](#) class for libdccl.

Definition at line 30 of file `dccl_exception.h`.

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

- `acomms/libdccl/dccl_exception.h`

15.4 goby::acomms::DCCLMessageVal Class Reference

defines a DCCL value

```
#include <acomms/libdccl/message_val.h>
```

Public Types

- enum { **MAX_DBL_PRECISION** = 15 }

Public Member Functions

Constructors/Destructor

- [DCCLMessageVal](#) ()
empty
- [DCCLMessageVal](#) (const std::string &s)
construct with string value
- [DCCLMessageVal](#) (const char *s)
construct with char value*
- [DCCLMessageVal](#) (double d, int p=MAX_DBL_PRECISION)
construct with double value, optionally giving the precision of the double (number of decimal places) which is used if a cast to std::string is required in the future.
- [DCCLMessageVal](#) (long l)
construct with long value
- [DCCLMessageVal](#) (int i)
construct with int value
- [DCCLMessageVal](#) (float f)
construct with float value
- [DCCLMessageVal](#) (bool b)
construct with bool value
- [DCCLMessageVal](#) (const std::vector< [DCCLMessageVal](#) > &vm)
construct with vector

Setters

- void [set](#) (std::string sval)
set the value with a string (overwrites previous value regardless of type)
- void [set](#) (double dval, int precision=MAX_DBL_PRECISION)
set the value with a double (overwrites previous value regardless of type)
- void [set](#) (long lval)
set the value with a long (overwrites previous value regardless of type)
- void [set](#) (bool bval)
set the value with a bool (overwrites previous value regardless of type)

Getters

- bool [get](#) (std::string &s) const
extract as std::string (all reasonable casts are done)
- bool [get](#) (bool &b) const
extract as bool (all reasonable casts are done)
- bool [get](#) (long &t) const
extract as long (all reasonable casts are done)
- bool [get](#) (double &d) const
extract as double (all reasonable casts are done)
- [operator double](#) () const
- [operator bool](#) () const
- [operator std::string](#) () const
- [operator long](#) () const

- [operator int](#) () const
- [operator unsigned](#) () const
- [operator float](#) () const
- [operator std::vector< DCCLMessageVal >](#) () const
- [DCCLCppType type](#) () const
what type is the original type of this DCCLMessageVal?
- bool [empty](#) () const
was this just constructed with DCCLMessageVal() ?
- unsigned [precision](#) () const

Comparison

- bool [operator==](#) (const [DCCLMessageVal](#) &mv) const
- bool [operator==](#) (const std::string &s) const
- bool [operator==](#) (double d) const
- bool [operator==](#) (long l) const
- bool [operator==](#) (bool b) const

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [DCCLMessageVal](#) &mv)

15.4.1 Detailed Description

defines a DCCL value

Examples:

[libdccl/test/test.cpp](#).

Definition at line 33 of file message_val.h.

15.4.2 Member Function Documentation

15.4.2.1 bool goby::acomms::DCCLMessageVal::get (std::string & s) const

extract as std::string (all reasonable casts are done)

Parameters

s	std::string to store value in
----------	-------------------------------

Returns

successfully extracted (and if necessary successfully cast to this type)

Examples:

[libdccl/test/test.cpp](#).

Definition at line 124 of file message_val.cpp.

15.4.2.2 bool goby::acomms::DCCLMessageVal::get (bool & *b*) const

extract as bool (all reasonable casts are done)

Parameters

<i>b</i>	bool to store value in
----------	------------------------

Returns

successfully extracted (and if necessary successfully cast to this type)

Definition at line 155 of file message_val.cpp.

15.4.2.3 bool goby::acomms::DCCLMessageVal::get (long & *t*) const

extract as long (all reasonable casts are done)

Parameters

<i>t</i>	long to store value in
----------	------------------------

Returns

successfully extracted (and if necessary successfully cast to this type)

Definition at line 187 of file message_val.cpp.

15.4.2.4 bool goby::acomms::DCCLMessageVal::get (double & *d*) const

extract as double (all reasonable casts are done)

Parameters

<i>d</i>	double to store value in
----------	--------------------------

Returns

successfully extracted (and if necessary successfully cast to this type)

Definition at line 226 of file message_val.cpp.

15.4.2.5 goby::acomms::DCCLMessageVal::operator bool () const

allows statements of the form

```
bool b = DCCLMessageVal("1");
```

Definition at line 273 of file message_val.cpp.

15.4.2.6 goby::acomms::DCCLMessageVal::operator double () const

allows statements of the form

```
double d = DCCLMessageVal("3.23");
```

Definition at line 265 of file message_val.cpp.

15.4.2.7 goby::acomms::DCCLMessageVal::operator float () const

allows statements of the form

```
float f = DCCLMessageVal("3.5");
```

Definition at line 304 of file message_val.cpp.

15.4.2.8 goby::acomms::DCCLMessageVal::operator int () const

allows statements of the form

```
int i = DCCLMessageVal(2);
```

Definition at line 294 of file message_val.cpp.

15.4.2.9 goby::acomms::DCCLMessageVal::operator long () const

allows statements of the form

```
long l = DCCLMessageVal(5);
```

Definition at line 287 of file message_val.cpp.

15.4.2.10 goby::acomms::DCCLMessageVal::operator std::string () const

allows statements of the form

```
std::string s = DCCLMessageVal(3);
```

Definition at line 280 of file message_val.cpp.

15.4.2.11 goby::acomms::DCCLMessageVal::operator unsigned () const

allows statements of the form

```
unsigned u = DCCLMessageVal(2);
```

Definition at line 299 of file message_val.cpp.

15.4.2.12 void goby::acomms::DCCLMessageVal::set (double dval, int precision = MAX_DBL_PRECISION)

set the value with a double (overwrites previous value regardless of type)

Parameters

<i>dval</i>	values to set
<i>precision</i>	decimal places of precision to preserve if this is cast to a string

Definition at line 116 of file message_val.cpp.

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

- acomms/libdccl/message_val.h
- acomms/libdccl/message_val.cpp

15.5 goby::acomms::MACManager Class Reference

provides an API to the goby-acomms MAC library.

```
#include <goby/acomms/amac.h>
```

Public Member Functions

Constructors/Destructor

- [MACManager](#) (std::ostream *log=0)
Default constructor.
- [~MACManager](#) ()

Control

- void [startup](#) (const protobuf::MACConfig &cfg)
Starts the MAC with given configuration.
- void [shutdown](#) ()
Shutdown the MAC.
- void [do_work](#) ()
Must be called regularly for the MAC to perform its work (10 Hertz or so is fine)

Modem Slots

- void [handle_modem_all_incoming](#) (const protobuf::ModemMsgBase &m)
Call every time a message is received from vehicle to "discover" this vehicle or reset the expire timer. Only needed when the type is amac::mac_auto_decentralized. - Typically connected to [ModemDriverBase::signal_all_incoming](#) using [bind\(\)](#).

Manipulate slots

- std::map< int, protobuf::Slot > ::iterator [add_slot](#) (const protobuf::Slot &s)
- bool [remove_slot](#) (const protobuf::Slot &s)
removes any slots in the cycle where [protobuf::operator==\(const protobuf::Slot&, const protobuf::Slot&\)](#) is true.
- void [clear_all_slots](#) ()
clears all slots from communications cycle.

Static Public Member Functions

Other

- static void [add_flex_groups](#) ([util::FlexOstream](#) *tout)
Adds groups for a FlexOstream logger.

Public Attributes

Modem Signals

- boost::signal< void([protobuf::ModemMsgBase](#) *m) [signal_initiate_transmission](#))
Signals when it is time for this platform to begin transmission of an acoustic message at the start of its TDMA slot. Typically connected to [ModemDriverBase::handle_initiate_transmission\(\)](#) using [bind\(\)](#).
- boost::signal< void([protobuf::ModemRangingRequest](#) *m) [signal_initiate_ranging](#))
Signals when it is time for this platform to begin an acoustic ranging transmission to another vehicle or ranging beacon(s). Typically connected to [ModemDriverBase::handle_initiate_ranging\(\)](#) using [bind\(\)](#).

15.5.1 Detailed Description

provides an API to the goby-acomms MAC library.

See also

[amac.proto](#) and [modem_message.proto](#) for definition of Google Protocol Buffers messages (namespace [goby::acomms::protobuf](#)).

Examples:

[acomms/chat/chat.cpp](#), and [libamac/amac_simple/amac_simple.cpp](#).

Definition at line 53 of file [mac_manager.h](#).

15.5.2 Constructor & Destructor Documentation

15.5.2.1 goby::acomms::MACManager::MACManager ([std::ostream](#) * *log* = 0)

Default constructor.

Parameters

<i>log</i>	std::ostream object or FlexOstream to capture all humanly readable runtime and debug information (optional).
------------	--

Definition at line 40 of file [mac_manager.cpp](#).

15.5.3 Member Function Documentation

15.5.3.1 `std::map< int, goby::acomms::protobuf::Slot >::iterator`
`goby::acomms::MACManager::add_slot ( const protobuf::Slot & s )`

Returns

iterator to newly added slot

Definition at line 400 of file `mac_manager.cpp`.

15.5.3.2 `void goby::acomms::MACManager::handle_modem_all_incoming ( const`
`protobuf::ModemMsgBase & m )`

Call every time a message is received from vehicle to "discover" this vehicle or reset the expire timer. Only needed when the type is `amac::mac_auto_decentralized`. Typically connected to [ModemDriverBase::signal_all_incoming](#) using `bind()`.

Parameters

<i>m</i>	the new incoming message (used to detect vehicles). (protobuf::ModemMsgBase defined in modem_message.proto)
----------	--

Definition at line 290 of file `mac_manager.cpp`.

15.5.3.3 `bool goby::acomms::MACManager::remove_slot ( const protobuf::Slot & s )`

removes any slots in the cycle where `protobuf::operator==(const protobuf::Slot&, const protobuf::Slot&)` is true.

Returns

true if one or more slots are removed

Definition at line 426 of file `mac_manager.cpp`.

15.5.3.4 `void goby::acomms::MACManager::startup ( const protobuf::MACConfig & cfg`
`)`

Starts the MAC with given configuration.

Parameters

<i>cfg</i>	Initial configuration values (protobuf::MACConfig defined in amac.proto)
------------	---

Definition at line 72 of file `mac_manager.cpp`.

15.5.4 Member Data Documentation

15.5.4.1 `boost::signal<void (protobuf::ModemRangingRequest* m)`
`goby::acomms::MACManager::signal_initiate_ranging)`

Signals when it is time for this platform to begin an acoustic ranging transmission to another vehicle or ranging beacon(s). Typically connected to [ModemDriverBase::handle_initiate_ranging\(\)](#) using `bind()`.

Parameters

<code>m</code>	parameters of the ranging request to be performed (protobuf::ModemRangingRequest defined in modem_message.proto).
----------------	--

Definition at line 103 of file `mac_manager.h`.

15.5.4.2 `boost::signal<void (protobuf::ModemMsgBase* m)`
`goby::acomms::MACManager::signal_initiate_transmission)`

Signals when it is time for this platform to begin transmission of an acoustic message at the start of its TDMA slot. Typically connected to [ModemDriverBase::handle_initiate_transmission\(\)](#) using `bind()`.

Parameters

<code>m</code>	a message containing details of the transmission to be initiated. (protobuf::ModemMsgBase defined in modem_message.proto)
----------------	--

Definition at line 98 of file `mac_manager.h`.

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

- `acomms/libamac/mac_manager.h`
- `acomms/libamac/mac_manager.cpp`

15.6 goby::acomms::MMDriver Class Reference

provides an API to the WHOI Micro-Modem driver

```
#include <goby/acomms/modem_driver.h>
```

Inherits [goby::acomms::ModemDriverBase](#).

Public Member Functions

- [MMDriver](#) (std::ostream *log=0)
Default constructor.
- [~MMDriver](#) ()
Destructor.
- void [startup](#) (const protobuf::DriverConfig &cfg)
Starts the driver.

- void [shutdown](#) ()
Shuts down the modem driver.
- void [do_work](#) ()
Must be called regularly for the driver to perform its work. 10 Hz is good, but less frequently is fine too. Signals will be emitted only during calls to this method.
- void [handle_initiate_transmission](#) (protobuf::ModemMsgBase *m)
Initiate a transmission to the modem.
- void [handle_initiate_ranging](#) (protobuf::ModemRangingRequest *m)
Initiate ranging ("ping") to the modem.

15.6.1 Detailed Description

provides an API to the WHOI Micro-Modem driver

Examples:

[acomms/chat/chat.cpp](#), and [libmodemdriver/driver_simple/driver_simple.cpp](#).

Definition at line 37 of file `mm_driver.h`.

15.6.2 Constructor & Destructor Documentation

15.6.2.1 goby::acomms::MMDriver::MMDriver (std::ostream * log = 0)

Default constructor.

Parameters

<i>log</i>	std::ostream object or FlexOstream to capture all humanly readable runtime and debug information (optional).
------------	--

Definition at line 52 of file `mm_driver.cpp`.

15.6.3 Member Function Documentation

15.6.3.1 void goby::acomms::MMDriver::startup (const protobuf::DriverConfig & cfg) [virtual]

Starts the driver.

Parameters

<i>cfg</i>	Configuration for the Micro-Modem driver. DriverConfig is defined in driver_base.proto , and various extensions specific to the WHOI Micro-Modem are defined in mm_driver.proto .
------------	---

Implements [goby::acomms::ModemDriverBase](#).

Definition at line 68 of file mm_driver.cpp.

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

- `acomms/libmodemdriver/mm_driver.h`
- `acomms/libmodemdriver/mm_driver.cpp`

15.7 goby::acomms::ModemDriverBase Class Reference

provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.

```
#include <goby/acomms/modem_driver.h>
```

Inherited by [goby::acomms::ABCDriver](#), and [goby::acomms::MMDriver](#).

Public Member Functions

- virtual [~ModemDriverBase](#) ()
Public Destructor.

Control

- virtual void [startup](#) (const protobuf::DriverConfig &cfg)=0
Starts the modem driver. Must be called before [do_work\(\)](#).
- virtual void [shutdown](#) ()=0
Shuts down the modem driver.
- virtual void [do_work](#) ()=0
Allows the modem driver to do its work.

MAC Slots

- virtual void [handle_initiate_transmission](#) (protobuf::ModemMsgBase *m)=0
Virtual `initiate_transmission` method. Typically connected to [MACManager::signal_initiate_transmission\(\)](#) using `bind()`.
- virtual void [handle_initiate_ranging](#) (protobuf::ModemRangingRequest *m)
Virtual `initiate_ranging` method. Typically connected to [MACManager::signal_initiate_ranging\(\)](#) using `bind()`.

Static Public Member Functions

Static helpers

- static void [add_flex_groups](#) (util::FlexOstream *tout)
Set the output groups for the modem driver if using the [util::FlexOstream](#) class for human readable debugging out. Setting groups allows the [util::FlexOstream](#) class to differentiate between different types of debugging messages.

Public Attributes

MAC / Queue Signals

- boost::signal< void(const protobuf::ModemDataTransmission &message)> [signal_receive](#)
Called when a binary data transmission is received from the modem.
- boost::signal< void(const protobuf::ModemDataRequest &msg_request, × protobuf::ModemDataTransmission *msg_data) [signal_data_request](#))
Called when the modem or modem driver needs data to send. msg_request contains details on the data request, and the returned data should be stored in msg_data.
- boost::signal< void(const protobuf::ModemRangingReply &message)> [signal_range_reply](#)
Called when the modem receives ranging information (time of flight to another vehicle or LBL ranging beacons)
- boost::signal< void(const protobuf::ModemDataAck &message)> [signal_ack](#)
Called when the modem receives an acknowledgment of proper receipt of a prior data transmission. The frame number of the acknowledgment must match the frame number of the original message. The modem driver is only responsible for the base (source, destination, timestamp) and acknowledged frame number in - ModemDataAck.
- boost::signal< void(const protobuf::ModemMsgBase &msg_data)> [signal_all_incoming](#)
Called after any message is received from the modem by the driver. Used by the [MACManager](#) for auto-discovery of vehicles. Also useful for higher level analysis and debugging of the transactions between the driver and the modem.
- boost::signal< void(const protobuf::ModemMsgBase &msg_data)> [signal_all_outgoing](#)
Called after any message is sent from the driver to the modem. Useful for higher level analysis and debugging of the transactions between the driver and the modem.

Protected Member Functions

Constructors/Destructor

- [ModemDriverBase](#) (std::ostream *log=0)
Constructor.

Write/read from the line-based interface to the modem

- void [modem_write](#) (const std::string &out)
write a line to the serial port.
- bool [modem_read](#) (std::string *in)
read a line from the serial port, including end-of-line character(s)
- void [modem_start](#) (const protobuf::DriverConfig &cfg)
start the physical connection to the modem (serial port, TCP, etc.). must be called before [ModemDriverBase::modem_read\(\)](#) or [ModemDriverBase::modem_write\(\)](#)
- void [modem_close](#) ()
closes the serial port. Use modem_start to reopen the port.

15.7.1 Detailed Description

provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.

See also

[driver_base.proto](#) and [modem_message.proto](#) for definition of Google Protocol - Buffers messages (namespace [goby::acomms::protobuf](#)).

Examples:

[libmodemdriver/driver_simple/driver_simple.cpp](#).

Definition at line 42 of file [driver_base.h](#).

15.7.2 Constructor & Destructor Documentation

15.7.2.1 `goby::acomms::ModemDriverBase::ModemDriverBase ( std::ostream * log = 0 ) [protected]`

Constructor.

Parameters

<i>log</i>	pointer to std::ostream to log human readable debugging and runtime information
------------	---

Definition at line 27 of file [driver_base.cpp](#).

15.7.3 Member Function Documentation

15.7.3.1 `void goby::acomms::ModemDriverBase::add_flex_groups ( util::FlexOstream * tout ) [static]`

Set the output groups for the modem driver if using the [util::FlexOstream](#) class for human readable debugging out. Setting groups allows the [util::FlexOstream](#) class to differentiate between different types of debugging messages.

Parameters

<i>tout</i>	pointer to util::FlexOstream stream object to add groups to.
-------------	--

Definition at line 110 of file [driver_base.cpp](#).

15.7.3.2 `virtual void goby::acomms::ModemDriverBase::do_work ( ) [pure virtual]`

Allows the modem driver to do its work.

Should be called regularly to perform the work of the driver as the driver **does not** run in its own thread. This allows us to guarantee that no signals are called except inside this `do_work` method.

Implemented in [goby::acomms::MMDriver](#), and [goby::acomms::ABCDriver](#).

Examples:

[libmodemdriver/driver_simple/driver_simple.cpp](#).

15.7.3.3 `virtual void goby::acomms::ModemDriverBase::handle_initiate_ranging ( protobuf::ModemRangingRequest * m ) [inline, virtual]`

Virtual `initiate_ranging` method. Typically connected to [MACManager::signal_initiate_ranging\(\)](#) using `bind()`.

Parameters

<i>m</i>	ModemRangingRequest (defined in modem_message.proto) containing the details of the ranging request to be started: source, destination, type, etc.
----------	--

Reimplemented in [goby::acomms::MMDriver](#).

Definition at line 71 of file `driver_base.h`.

15.7.3.4 `virtual void goby::acomms::ModemDriverBase::handle_initiate_transmission ( protobuf::ModemMsgBase * m ) [pure virtual]`

Virtual `initiate_transmission` method. Typically connected to [MACManager::signal_initiate_transmission\(\)](#) using `bind()`.

Parameters

<i>m</i>	ModemMsgBase (defined in modem_message.proto) containing the details of the transmission to be started. This does <i>*not*</i> contain data, which will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request)
----------	--

Implemented in [goby::acomms::MMDriver](#), and [goby::acomms::ABCDriver](#).

Examples:

[libmodemdriver/driver_simple/driver_simple.cpp](#).

15.7.3.5 `bool goby::acomms::ModemDriverBase::modem_read ( std::string * in ) [protected]`

read a line from the serial port, including end-of-line character(s)

Parameters

<i>in</i>	pointer to string to store line
-----------	---------------------------------

Returns

true if a line was available, false if no line available

Definition at line 49 of file driver_base.cpp.

15.7.3.6 void **goby::acomms::ModemDriverBase::modem_start** (const
protobuf::DriverConfig & *cfg*) [protected]

start the physical connection to the modem (serial port, TCP, etc.). must be called before [ModemDriverBase::modem_read\(\)](#) or [ModemDriverBase::modem_write\(\)](#)

Parameters

<i>cfg</i>	Configuration including the parameters for the physical connection. (protobuf::DriverConfig is defined in driver_base.proto).
------------	--

Exceptions

<i>driver_exception</i>	Problem opening the physical connection.
-------------------------	--

Definition at line 67 of file driver_base.cpp.

15.7.3.7 void **goby::acomms::ModemDriverBase::modem_write** (const std::string &
out) [protected]

write a line to the serial port.

Parameters

<i>out</i>	reference to string to write. Must already include any end-of-line character(s).
------------	--

Definition at line 37 of file driver_base.cpp.

15.7.3.8 virtual void **goby::acomms::ModemDriverBase::startup** (const
protobuf::DriverConfig & *cfg*) [pure virtual]

Starts the modem driver. Must be called before [do_work\(\)](#).

Parameters

<i>cfg</i>	Startup configuration for the driver and modem. Driver-Config is defined in driver_base.proto . Derived classes can define extensions (see http://code.google.com/apis/protocolbuffers/docs/proto.-html#extensions) to DriverConfig to handle modem specific configuration.
------------	--

Implemented in [goby::acomms::MMDriver](#), and [goby::acomms::ABCDriver](#).

Examples:

[libmodemdriver/driver_simple/driver_simple.cpp](#).

15.7.4 Member Data Documentation

15.7.4.1 `boost::signal<void (const protobuf::ModemDataAck& message)> goby::acomms::ModemDriverBase::signal_ack`

Called when the modem receives an acknowledgment of proper receipt of a prior data transmission. The frame number of the acknowledgment must match the frame number of the original message. The modem driver is only responsible for the base (source, destination, timestamp) and acknowledged frame number in `ModemDataAck`.

You should connect one or more slots (a function or member function) to this signal to handle acknowledgments. Use the [goby::acomms::connect](#) family of functions to do this. This signal will only be called during a call to `do_work`. `ModemDataAck` is defined in [modem_message.proto](#).

Definition at line 100 of file `driver_base.h`.

15.7.4.2 `boost::signal<void (const protobuf::ModemMsgBase& msg_data)> goby::acomms::ModemDriverBase::signal_all_incoming`

Called after any message is received from the modem by the driver. Used by the [MACManager](#) for auto-discovery of vehicles. Also useful for higher level analysis and debugging of the transactions between the driver and the modem.

If desired, you should connect one or more slots (a function or member function) to this signal to listen on incoming transactions. Use the [goby::acomms::connect](#) family of functions to do this. This signal will only be called during a call to `do_work`. `ModemMsgBase` is defined in [modem_message.proto](#).

Definition at line 106 of file `driver_base.h`.

15.7.4.3 `boost::signal<void (const protobuf::ModemMsgBase& msg_data)> goby::acomms::ModemDriverBase::signal_all_outgoing`

Called after any message is sent from the driver to the modem. Useful for higher level analysis and debugging of the transactions between the driver and the modem.

If desired, you should connect one or more slots (a function or member function) to this signal to listen on outgoing transactions. Use the [goby::acomms::connect](#) family of functions to do this. This signal will only be called during a call to `do_work`. `ModemMsgBase` is defined in [modem_message.proto](#).

Definition at line 112 of file `driver_base.h`.

15.7.4.4 `boost::signal<void (const protobuf::ModemDataRequest& msg_request, protobuf::ModemDataTransmission* msg_data) goby::acomms::ModemDriverBase::signal_data_request)`

Called when the modem or modem driver needs data to send. `msg_request` contains details on the data request, and the returned data should be stored in `msg_data`.

You should connect one or more slots (a function or member function) to this signal to handle data requests. Use the [goby::acomms::connect](#) family of functions to do this. This signal will only be called during a call to `do_work`. `ModemDataRequest` and `ModemDataTransmission` are defined in [modem_message.proto](#).

Examples:

[libmodemdriver/driver_simple/driver_simple.cpp](#).

Definition at line 88 of file `driver_base.h`.

15.7.4.5 `boost::signal<void (const protobuf::ModemRangingReply& message)> goby::acomms::ModemDriverBase::signal_range_reply`

Called when the modem receives ranging information (time of flight to another vehicle or LBL ranging beacons)

You should connect one or more slots (a function or member function) to this signal to handle ranging replies. Use the [goby::acomms::connect](#) family of functions to do this. This signal will only be called during a call to `do_work`. `ModemRangingReply` is defined in [modem_message.proto](#).

Definition at line 94 of file `driver_base.h`.

15.7.4.6 `boost::signal<void (const protobuf::ModemDataTransmission& message)> goby::acomms::ModemDriverBase::signal_receive`

Called when a binary data transmission is received from the modem.

You should connect one or more slots (a function or member function) to this signal to receive incoming messages. Use the [goby::acomms::connect](#) family of functions to do this. This signal will only be called during a call to `do_work`. `ModemDataTransmission` is defined in [modem_message.proto](#).

Examples:

[libmodemdriver/driver_simple/driver_simple.cpp](#).

Definition at line 82 of file `driver_base.h`.

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

- `acomms/libmodemdriver/driver_base.h`
- `acomms/libmodemdriver/driver_base.cpp`

15.8 goby::acomms::QueueManager Class Reference

provides an API to the goby-acomms Queuing Library.

```
#include <goby/acomms/queue.h>
```

Public Member Functions

Constructors/Destructor

- [QueueManager](#) (std::ostream *log=0)
Default constructor.
- [~QueueManager](#) ()
Destructor.

Initialization Methods

These methods are intended to be called before doing any work with the class.

- void [set_cfg](#) (const protobuf::QueueManagerConfig &cfg)
Set (and overwrite completely if present) the current configuration. (protobuf::QueueManagerConfig defined in [queue.proto](#))
- void [merge_cfg](#) (const protobuf::QueueManagerConfig &cfg)
Set (and merge "repeat" fields) the current configuration. (protobuf::QueueManagerConfig defined in [queue.proto](#))

Application level Push/Receive Methods

These methods are the primary higher level interface to the [QueueManager](#). From here you can push messages and set the callbacks to use on received messages.

- void [push_message](#) (const protobuf::ModemDataTransmission &new_message)
Push a message using a QueueKey as a key.
- void [flush_queue](#) (const protobuf::QueueFlush &flush)
Flush (delete all messages in) a queue.

Modem Slots

These methods are the interface to the [QueueManager](#) from the modem driver.

- void [handle_modem_data_request](#) (const protobuf::ModemDataRequest &msg_request, protobuf::ModemDataTransmission *msg_data)
Finds data to send to the modem.
- void [handle_modem_receive](#) (const protobuf::ModemDataTransmission &message)
Receive incoming data from the modem.
- void [handle_modem_ack](#) (const protobuf::ModemDataAck &message)
Receive acknowledgements from the modem.

Control

Call these methods when you want the [QueueManager](#) to perform time sensitive tasks (such as expiring old messages)

- void [do_work](#) ()
Calculates which messages have expired and emits the [goby::acomms::QueueManager::signal_expire](#) as necessary.

Informational Methods

- std::string [summary](#) () const
- const ManipulatorManager & [manip_manager](#) () const

Static Public Member Functions

Static helpers

- static void [add_flex_groups](#) (util::FlexOstream *tout)
Registers the group names used for the FlexOstream logger.

Public Attributes

Application Signals

- boost::signal< void(const protobuf::ModemDataAck &ack_msg)> [signal_ack](#)
Signals when acknowledgment of proper message receipt has been received. This is only sent for queues with QueueConfig::ack() == true with an explicit destination (ModemMessageBase::dest() != 0)
- boost::signal< void(const protobuf::ModemDataTransmission &msg)> [signal_receive](#)
Signals when a DCCL message is received.
- boost::signal< void(const protobuf::ModemDataTransmission &msg)> [signal_receive_ccl](#)
Signals when a CCL message is received.
- boost::signal< void(const protobuf::ModemDataExpire &expire_msg)> [signal_expire](#)
Signals when a message expires (exceeds its time-to-live or ttl) before being sent (if QueueConfig::ack() == false) or before being acknowledged (if QueueConfig::ack() == true).
- boost::signal< void(const protobuf::ModemDataRequest &request_msg, × protobuf::ModemDataTransmission *data_msg) [signal_data_on_demand](#))
Forwards the data request to the application layer. This advanced feature is used with the ON_DEMAND manipulator and allows for the application to provide data immediately before it is actually sent (for highly time sensitive data)
- boost::signal< void(protobuf::QueueSize size)> [signal_queue_size_change](#)
Signals when any queue changes size (message is popped or pushed)

Friends

- class **Queue**

15.8.1 Detailed Description

provides an API to the goby-acomms Queuing Library.

See also

[queue.proto](#) and [modem_message.proto](#) for definition of Google Protocol Buffers messages (namespace [goby::acomms::protobuf](#)).

Examples:

[acomms/chat/chat.cpp](#), and [libqueue/queue_simple/queue_simple.cpp](#).

Definition at line 54 of file [queue_manager.h](#).

15.8.2 Constructor & Destructor Documentation

15.8.2.1 goby::acomms::QueueManager::QueueManager (std::ostream * log = 0)

Default constructor.

Parameters

<i>log</i>	std::ostream object or FlexOstream to capture all humanly readable runtime and debug information (optional).
------------	--

Definition at line 34 of file [queue_manager.cpp](#).

15.8.3 Member Function Documentation

15.8.3.1 void goby::acomms::QueueManager::flush_queue (const protobuf::QueueFlush & flush)

Flush (delete all messages in) a queue.

Parameters

<i>QueueFlush</i>	flush: object containing details about queues to flush
-------------------	--

Definition at line 147 of file [queue_manager.cpp](#).

15.8.3.2 void goby::acomms::QueueManager::handle_modem_ack (const protobuf::ModemDataAck & message)

Receive acknowledgements from the modem.

If using one of the classes inheriting [ModemDriverBase](#), this method should be bound and passed to [ModemDriverBase::set_ack_cb](#).

Parameters

<i>message</i>	The ModemMessage corresponding to the acknowledgement (dest, src, frame#)
----------------	---

Definition at line 424 of file queue_manager.cpp.

15.8.3.3 void goby::acomms::QueueManager::handle_modem_data-
_request (const protobuf::ModemDataRequest & *msg_request*,
protobuf::ModemDataTransmission * *msg_data*)

Finds data to send to the modem.

Data from the highest priority queue(s) will be combined to form a message equal or less than the size requested in ModemMessage *message_in*. If using one of the classes inheriting [ModemDriverBase](#), this method should be bound and passed to ModemDriverBase::set_datarequest_cb.

Parameters

<i>message_in</i>	The ModemMessage containing the details of the request (source, destination, size, etc.)
<i>message_out</i>	The packed ModemMessage ready for sending by the modem. This will be populated by this function.

Returns

true if successful in finding data to send, false if no data is available

Definition at line 188 of file queue_manager.cpp.

15.8.3.4 void goby::acomms::QueueManager::handle_modem_receive (const
protobuf::ModemDataTransmission & *message*)

Receive incoming data from the modem.

If using one of the classes inheriting [ModemDriverBase](#), this method should be bound and passed to ModemDriverBase::set_receive_cb.

Parameters

<i>message</i>	The received ModemMessage.
----------------	----------------------------

Definition at line 482 of file queue_manager.cpp.

15.8.3.5 void goby::acomms::QueueManager::push_message (const
protobuf::ModemDataTransmission & *new_message*)

Push a message using a QueueKey as a key.

Parameters

<i>key</i>	QueueKey that references the queue to push the message to.
<i>new_message</i>	ModemMessage to push.

Definition at line 110 of file queue_manager.cpp.

15.8.3.6 std::string goby::acomms::QueueManager::summary () const

Returns

human readable summary of all loaded queues

Definition at line 164 of file queue_manager.cpp.

15.8.4 Member Data Documentation

15.8.4.1 boost::signal<void (const protobuf::ModemDataAck& ack_msg)>
goby::acomms::QueueManager::signal_ack

Signals when acknowledgment of proper message receipt has been received. This is only sent for queues with QueueConfig::ack() == true with an explicit destination (- ModemMessageBase::dest() != 0)

Parameters

<i>ack_msg</i>	a message containing details of the acknowledgment and the acknowledged transmission. (protobuf::ModemMsgAck is defined in modem_message.proto)
----------------	--

Definition at line 163 of file queue_manager.h.

15.8.4.2 boost::signal<void (const protobuf::ModemDataRequest& request_msg, protobuf::ModemDataTransmission* data_msg)
goby::acomms::QueueManager::signal_data_on_demand)

Forwards the data request to the application layer. This advanced feature is used with the ON_DEMAND manipulator and allows for the application to provide data immediately before it is actually sent (for highly time sensitive data)

Parameters

<i>request_msg</i>	the details of the requested data. (protobuf::ModemDataRequest is defined in modem_message.proto)
<i>data_msg</i>	pointer to store the supplied data. (protobuf::ModemDataTransmission is defined in modem_message.proto)

Definition at line 184 of file queue_manager.h.

15.8.4.3 boost::signal<void (const protobuf::ModemDataExpire& expire_msg)> goby::acomms::QueueManager::signal_expire

Signals when a message is expires (exceeds its time-to-live or ttl) before being sent (if QueueConfig::ack() == false) or before being acknowledged (if QueueConfig::ack() == true).

Parameters

<i>expire_msg</i>	the expired transmission. (protobuf::ModemDataExpire is defined in modem_message.proto)
-------------------	--

Definition at line 177 of file queue_manager.h.

15.8.4.4 boost::signal<void (protobuf::QueueSize size)> goby::acomms::QueueManager::signal_queue_size_change

Signals when any queue changes size (message is popped or pushed)

Parameters

<i>size</i>	message containing the queue that changed size and its new size (protobuf::QueueSize is defined in queue.proto).
-------------	---

Definition at line 189 of file queue_manager.h.

15.8.4.5 boost::signal<void (const protobuf::ModemDataTransmission& msg)> goby::acomms::QueueManager::signal_receive

Signals when a DCCL message is received.

Parameters

<i>msg</i>	the received transmission. (protobuf::ModemDataTransmission is defined in modem_message.proto)
------------	---

Definition at line 167 of file queue_manager.h.

15.8.4.6 boost::signal<void (const protobuf::ModemDataTransmission& msg)> goby::acomms::QueueManager::signal_receive_ccl

Signals when a CCL message is received.

Parameters

<i>msg</i>	the received transmission. (protobuf::ModemDataTransmission is defined in modem_message.proto)
------------	---

Definition at line 172 of file queue_manager.h.

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

- `acomms/libqueue/queue_manager.h`
- `acomms/libqueue/queue_manager.cpp`

15.9 goby::ConfigException Class Reference

indicates a problem with the runtime command line or .cfg file configuration (or --help was given)

```
#include <core/libcore/exception.h>
```

Inherits [goby::Exception](#).

Public Member Functions

- **ConfigException** (const std::string &s)
- void **set_error** (bool b)
- bool **error** ()

15.9.1 Detailed Description

indicates a problem with the runtime command line or .cfg file configuration (or --help was given)

Definition at line 35 of file `exception.h`.

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

- `core/libcore/exception.h`

15.10 goby::Exception Class Reference

simple exception class for goby applications

```
#include <core/libcore/exception.h>
```

Inherits `std::runtime_error`.

Inherited by [goby::acomms::DCCLEException](#), and [goby::ConfigException](#).

Public Member Functions

- **Exception** (const std::string &s)

15.10.1 Detailed Description

simple exception class for goby applications

Definition at line 24 of file `exception.h`.

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

- core/libcore/exception.h

15.11 goby::util::Colors Struct Reference

Represents the eight available terminal colors (and bold variants)

```
#include <util/liblogger/term_color.h>
```

Public Types

- enum [Color](#) { [nocolor](#), [red](#), [lt_red](#), [green](#), [lt_green](#), [yellow](#), [lt_yellow](#), [blue](#), [lt_blue](#), [magenta](#), [lt_magenta](#), [cyan](#), [lt_cyan](#), [white](#), [lt_white](#) }

The eight terminal colors (and bold or "light" variants)

15.11.1 Detailed Description

Represents the eight available terminal colors (and bold variants)

Definition at line 118 of file `term_color.h`.

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

- util/liblogger/term_color.h

15.12 goby::util::FlexNCurses Class Reference

Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.

```
#include <util/liblogger/flex_ncurses.h>
```

Public Member Functions

Constructors / Destructor

- **FlexNCurses** ()
- **~FlexNCurses** ()

Initialization

- void **startup** ()
- void **add_win** (const [Group](#) *title)

Use

- void [insert](#) (boost::posix_time::ptime t, const std::string &s, [Group](#) *g)
Add a string to the gui.

Utility

- `size_t panel_from_group (Group *g)`
- `void recalculate_win ()`
- `void cleanup ()`
- `void alive (bool alive)`

User Input Thread

- `void run_input ()`
run in its own thread to take input from the user

15.12.1 Detailed Description

Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.

Definition at line 38 of file flex_ncurses.h.

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

- util/liblogger/flex_ncurses.h
- util/liblogger/flex_ncurses.cpp

15.13 goby::util::FlexOstream Class Reference

Forms the basis of the Goby logger: std::ostream derived class for holding the [FlexOStreamBuf](#).

```
#include <util/liblogger/flex_ostream.h>
```

Inherits std::ostream.

Public Member Functions

- `void refresh ()`
- `void set_group (const std::string &s)`

Initialization

- `void add_group (const std::string &name, Colors::Color color=Colors::nocolor, const std::string &description="")`
- `void set_name (const std::string &s)`
Set the name of the application that the logger is serving.
- `void add_stream (const std::string &verbosity, std::ostream *os=0)`
- `void add_stream (Logger::Verbosity verbosity=Logger::verbose, std::ostream *os=0)`
Attach a stream object (e.g. std::cout, std::ofstream, ...) to the logger with desired verbosity.

Overloaded insert stream operator<<

- `std::ostream & operator<< (FlexOstream &(*pf)(FlexOstream &))`
- `std::ostream & operator<< (std::ostream &(*pf)(std::ostream &))`
- `std::ostream & operator<< (bool &val)`
- `std::ostream & operator<< (const short &val)`
- `std::ostream & operator<< (const unsigned short &val)`
- `std::ostream & operator<< (const int &val)`
- `std::ostream & operator<< (const unsigned int &val)`
- `std::ostream & operator<< (const long &val)`
- `std::ostream & operator<< (const unsigned long &val)`
- `std::ostream & operator<< (const float &val)`
- `std::ostream & operator<< (const double &val)`
- `std::ostream & operator<< (const long double &val)`
- `std::ostream & operator<< (std::streambuf *sb)`
- `std::ostream & operator<< (std::ios &(*pf)(std::ios &))`
- `std::ostream & operator<< (std::ios_base &(*pf)(std::ios_base &))`

Thread safety related

- `boost::mutex & mutex ()`
Get a reference to the Goby logger mutex for scoped locking.

Friends

- `FlexOstream & glogger (logger_lock::LockAction lock_action)`
Access the Goby logger through this function.
- `template<typename T >`
`void boost::checked_delete (T *)`
- `std::ostream & operator<< (FlexOstream &out, char c)`
- `std::ostream & operator<< (FlexOstream &out, signed char c)`
- `std::ostream & operator<< (FlexOstream &out, unsigned char c)`
- `std::ostream & operator<< (FlexOstream &out, const char *s)`
- `std::ostream & operator<< (FlexOstream &out, const signed char *s)`
- `std::ostream & operator<< (FlexOstream &out, const unsigned char *s)`

15.13.1 Detailed Description

Forms the basis of the Goby logger: `std::ostream` derived class for holding the `FlexOStreamBuf`.

Definition at line 45 of file `flex_ostream.h`.

15.13.2 Member Function Documentation

15.13.2.1 `void goby::util::FlexOstream::add_group ( const std::string & name, Colors::Color color = Colors::nocolor, const std::string & description = "" )`

Add another group to the logger. A group provides related manipulator for categorizing log messages. For thread safe use use `boost::scoped_lock` on `Logger::mutex`

Definition at line 36 of file flex_ostream.cpp.

15.13.3 Friends And Related Function Documentation

15.13.3.1 FlexOstream& glogger (logger_lock::LockAction lock_action) [friend]

Access the Goby logger through this function.

For normal (non thread safe use), do not pass any parameters: `glogger() << "some text" << std::endl;`

To group messages, pass the group(group_name) manipulator, where group_name is a previously defined group (by call to `glogger().add_group(Group)`). For example: `glogger() << group("incoming") << "received message foo" << std::endl;`

For thread safe use, use `glogger(lock)` and then insert the "unlock" manipulator when relinquishing the lock. The "unlock" manipulator **MUST** be inserted before the next call to `glogger(lock)`. **Nothing** must throw exceptions between `glogger(lock)` and `unlock`. For example: `glogger(lock) << "my thread is the best" << std::endl << unlock;`

Parameters

<i>lock_action</i>	logger_lock::lock to lock access to the logger (for thread safety) or logger_lock::none for no mutex action (typical)
--------------------	---

Returns

reference to Goby logger (std::ostream derived class [FlexOstream](#))

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

- util/liblogger/flex_ostream.h
- util/liblogger/flex_ostream.cpp

15.14 goby::util::FlexOStreamBuf Class Reference

Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by [goby::util::FlexOstream](#) for the Goby [Logger](#) (glogger)

```
#include <util/liblogger/flex_ostreambuf.h>
```

Inherits streambuf.

Public Member Functions

- int [sync](#) ()
virtual inherited from std::streambuf. Called when std::endl or std::flush is inserted into the stream
- int [overflow](#) (int c=EOF)

virtual inherited from std::streambuf. Called when something is inserted into the stream

- void [name](#) (const std::string &s)
name of the application being served
- void [add_stream](#) (Logger::Verbosity verbosity, std::ostream *os)
add a stream to the logger
- bool [is_quiet](#) ()
do all attached streams have Verbosity == quiet?
- bool [is_gui](#) ()
is there an attached stream with Verbosity == gui (ncurses GUI)
- void [group_name](#) (const std::string &s)
current group name (last insertion of group("") into the stream)
- void [set_die_flag](#) (bool b)
exit on error at the next call to [sync\(\)](#)
- void [set_debug_flag](#) (bool b)
label stream contents as "debug" until the next call to [sync\(\)](#)
- void [set_warn_flag](#) (bool b)
label stream contents as "warning" until the next call to [sync\(\)](#)
- void [add_group](#) (const std::string &name, Group g)
add a new group
- std::string [color2esc_code](#) (Colors::Color color)
TODO(tes): unnecessary?
- void [refresh](#) ()
refresh the display (does nothing if !is_gui())

15.14.1 Detailed Description

Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by [goby::util::FlexOstream](#) for the Goby [Logger](#) (glogger)

Definition at line 51 of file flex_ostreambuf.h.

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

- util/liblogger/flex_ostreambuf.h
- util/liblogger/flex_ostreambuf.cpp

15.15 goby::util::LineBasedInterface Class Reference

basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)

```
#include <util/liblinebasedcomms/interface.h>
```

Inherited by [goby::util::LineBasedClient< ASIOAsyncReadStream >](#), [goby::util::TCP-Server](#), [goby::util::LineBasedClient< boost::asio::ip::tcp::socket >](#), and [goby::util::LineBasedClient< boost::asio::serial_port >](#).

Public Types

- enum **AccessOrder** { **NEWEST_FIRST**, **OLDEST_FIRST** }

Public Member Functions

- void **start** ()
- void **close** ()
- bool **active** ()
- bool **readline** (std::string *s, AccessOrder order=OLDEST_FIRST)
returns string line (including delimiter)
- bool **readline** (protobuf::Datagram *msg, AccessOrder order=OLDEST_FIRST)
- void **write** (const std::string &s)
- void **write** (const protobuf::Datagram &msg)
- void **clear** ()
- void **set_delimiter** (const std::string &s)
- std::string **delimiter** () const

Protected Member Functions

- **LineBasedInterface** (const std::string &delimiter)
- virtual void **do_start** ()=0
- virtual void **do_write** (const protobuf::Datagram &line)=0
- virtual void **do_close** (const boost::system::error_code &error)=0
- void **set_active** (bool active)

Static Protected Attributes

- static std::string **delimiter_**
- static boost::asio::io_service **io_service_**
- static std::deque < protobuf::Datagram > **in_**
- static boost::mutex **in_mutex_**

15.15.1 Detailed Description

basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)

Definition at line 39 of file interface.h.

15.15.2 Member Function Documentation

15.15.2.1 bool **goby::util::LineBasedInterface::readline** (std::string * s, AccessOrder order = OLDEST_FIRST) [inline]

returns string line (including delimiter)

Returns

true if data was read, false if no data to read

Definition at line 55 of file interface.h.

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

- util/liblinebasedcomms/interface.h
- util/liblinebasedcomms/interface.cpp

15.16 goby::util::Logger Struct Reference

Holds static objects of the Goby [Logger](#).

```
#include <util/liblogger/flex_ostreambuf.h>
```

Public Types

- enum **Verbosity** { **quiet**, **warn**, **verbose**, **debug**, **gui** }

Static Public Attributes

- static boost::mutex **mutex**

15.16.1 Detailed Description

Holds static objects of the Goby [Logger](#).

Definition at line 43 of file flex_ostreambuf.h.

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

- util/liblogger/flex_ostreambuf.h
- util/liblogger/flex_ostreambuf.cpp

15.17 goby::util::SerialClient Class Reference

provides a basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control

```
#include <util/liblinebasedcomms/serial_client.h>
```

Inherits goby::util::LineBasedClient< ASIOAsyncReadStream >.

Public Member Functions

- [SerialClient](#) (const std::string &name="", unsigned [baud](#)=9600, const std::string &delimiter="\r\n")

- create a serial client*
- void [set_name](#) (const std::string &[name](#))
set serial port name, e.g. "/dev/ttyS0"
- void [set_baud](#) (unsigned [baud](#))
baud rate, e.g. 4800
- std::string [name](#) () const
serial port name, e.g. "/dev/ttyS0"
- unsigned [baud](#) () const
baud rate, e.g. 4800
- boost::asio::serial_port & [socket](#) ()
- std::string [local_endpoint](#) ()
our serial port, e.g. "/dev/ttyUSB1"
- std::string [remote_endpoint](#) ()
who knows where the serial port goes?! (empty string)

15.17.1 Detailed Description

provides a basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control

Definition at line 30 of file `serial_client.h`.

15.17.2 Constructor & Destructor Documentation

- 15.17.2.1 **goby::util::SerialClient::SerialClient** (const std::string & *name* = " ", unsigned *baud* = 9600, const std::string & *delimiter* = "\r\n")

create a serial client

Parameters

<i>name</i>	name of the serial connection (e.g. "/dev/ttyS0")
<i>baud</i>	baud rate of the serial connection (e.g. 9600)
<i>delimiter</i>	string used to split lines

Definition at line 25 of file `serial_client.cpp`.

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

- `util/liblinebasedcomms/serial_client.h`
- `util/liblinebasedcomms/serial_client.cpp`

15.18 goby::util::TCPClient Class Reference

provides a basic TCP client for line by line text based communications to a remote TCP server

```
#include <util/liblinebasedcomms/tcp_client.h>
```

Inherits goby::util::LineBasedClient< ASIOAsyncReadStream >.

Public Member Functions

- [TCPClient](#) (const std::string &server, unsigned port, const std::string &delimiter="\r\n")
create a [TCPClient](#)
- boost::asio::ip::tcp::socket & **socket** ()
- std::string [local_endpoint](#) ()
string representation of the local endpoint (e.g. 192.168.1.105:54230)
- std::string [remote_endpoint](#) ()
string representation of the remote endpoint, (e.g. 192.168.1.106:50000)

15.18.1 Detailed Description

provides a basic TCP client for line by line text based communications to a remote TCP server

Definition at line 29 of file tcp_client.h.

15.18.2 Constructor & Destructor Documentation

- 15.18.2.1 **goby::util::TCPClient::TCPClient** (const std::string & *server*, unsigned *port*, const std::string & *delimiter* = "\r\n")

create a [TCPClient](#)

Parameters

<i>server</i>	domain name or IP address of the remote server
<i>port</i>	port of the remote server
<i>delimiter</i>	string used to split lines

Definition at line 22 of file tcp_client.cpp.

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

- util/liblinebasedcomms/tcp_client.h
- util/liblinebasedcomms/tcp_client.cpp

15.19 goby::util::TCPServer Class Reference

provides a basic TCP server for line by line text based communications to a one or more remote TCP clients

```
#include <util/liblinebasedcomms/tcp_server.h>
```

Inherits [goby::util::LineBasedInterface](#).

Public Member Functions

- [TCPServer](#) (unsigned port, const std::string &delimiter="\r\n")
create a TCP server
- std::string [local_endpoint](#) ()
string representation of the local endpoint (e.g. 192.168.1.105:54230)

Friends

- class **TCPConnection**
- class **LineBasedConnection**< boost::asio::ip::tcp::socket >

15.19.1 Detailed Description

provides a basic TCP server for line by line text based communications to a one or more remote TCP clients

Definition at line 44 of file tcp_server.h.

15.19.2 Constructor & Destructor Documentation

15.19.2.1 goby::util::TCPServer::TCPServer (unsigned port, const std::string & delimiter = "\r\n") [inline]

create a TCP server

Parameters

<i>port</i>	port of the server (use 50000+ to avoid problems with special system ports)
<i>delimiter</i>	string used to split lines

Definition at line 51 of file tcp_server.h.

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

- util/liblinebasedcomms/tcp_server.h
- util/liblinebasedcomms/tcp_server.cpp

15.20 goby::util::TermColor Class Reference

Converts between string, escape code, and enumeration representations of the terminal colors.

```
#include <util/liblogger/term_color.h>
```

Static Public Member Functions

- static `Colors::Color from_str` (const std::string &s)
Color enumeration from string (e.g. "blue" -> blue)
- static std::string `str_from_col` (const `Colors::Color` &c)
String from color enumeration (e.g. red -> "red")
- static `Colors::Color from_esc_code` (const std::string &s)
Color enumeration from escape code (e.g. "\33[31m" -> red)
- static std::string `esc_code_from_col` (const `Colors::Color` &c)
Escape code from color enumeration (e.g. red -> "\33[31m")
- static std::string `esc_code_from_str` (const std::string &s)
Escape code from string (e.g. "red" -> "\33[31m")

15.20.1 Detailed Description

Converts between string, escape code, and enumeration representations of the terminal colors.

Definition at line 132 of file `term_color.h`.

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

- `util/liblogger/term_color.h`
- `util/liblogger/term_color.cpp`

15.21 Group Class Reference

Defines a group of messages to be sent to the Goby logger. For Verbosity == verbose streams, all entries appear interleaved, but each group is offset with a different color. For Verbosity == gui streams, all groups have a separate subwindow.

```
#include <util/liblogger/logger_manipulators.h>
```

Public Member Functions

- **Group** (const std::string &name="", const std::string &description="", `goby::util::Colors::Color` color=`goby::util::Colors::nocolor`)

Getters

- std::string `name` () const
Name of this group (used in the group manipulator)
- std::string `description` () const
Human readable description of this group.
- `goby::util::Colors::Color` `color` () const
Color to use when displaying this group (for streams that support terminal escape codes only: std::cout, std::cerr, std::clog)

- bool **enabled** () const
Is this group enabled?

Setters

- void **name** (const std::string &s)
- void **description** (const std::string &s)
- void **color** (goby::util::Colors::Color c)
- void **enabled** (bool b)

15.21.1 Detailed Description

Defines a group of messages to be sent to the Goby logger. For Verbosity == verbose streams, all entries appear interleaved, but each group is offset with a different color. For Verbosity == gui streams, all groups have a separate subwindow.

Definition at line 42 of file logger_manipulators.h.

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

- util/liblogger/logger_manipulators.h

15.22 GroupSetter Class Reference

Helper class for enabling the group(std::string) manipulator.

```
#include <util/liblogger/logger_manipulators.h>
```

Public Member Functions

- **GroupSetter** (const std::string &s)
- void **operator()** (std::ostream &os) const
- void **operator()** (goby::util::FlexOstream &os) const

15.22.1 Detailed Description

Helper class for enabling the group(std::string) manipulator.

Definition at line 89 of file logger_manipulators.h.

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

- util/liblogger/logger_manipulators.h
- util/liblogger/logger_manipulators.cpp

16 File Documentation

16.1 moos/libmoos_util/moos_protobuf_helpers.h File Reference

Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages.

```
#include "google/protobuf/io/printer.h" #include <google/protobuf/io/tokenizer.h>
#include "goby/util/liblogger/flex_ostream.h" #include "goby/util/string.h"
```

Functions

- void [serialize_for_moos](#) (std::string *out, const google::protobuf::Message &msg)
Converts the Google Protocol Buffers message 'msg' into a suitable (human readable) string 'out' for sending via MOOS.
- void [parse_for_moos](#) (const std::string &in, google::protobuf::Message *msg)
Parses the string 'in' to Google Protocol Buffers message 'msg'. All errors are written to the [goby::util::glogger\(\)](#).
- void [from_moos_comma_equals_string_field](#) (google::protobuf::Message *proto_msg, const google::protobuf::FieldDescriptor *field_desc, const std::vector< std::string > &values, int value_key=0)
- void [from_moos_comma_equals_string](#) (google::protobuf::Message *proto_msg, const std::string &in)
- std::string [to_moos_comma_equals_string_field](#) (const google::protobuf::Message &proto_msg, const google::protobuf::FieldDescriptor *field_desc, bool write_key=true)
- void [to_moos_comma_equals_string](#) (const google::protobuf::Message &proto_msg, std::string *out)

16.1.1 Detailed Description

Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages.

Definition in file [moos_protobuf_helpers.h](#).

16.1.2 Function Documentation

16.1.2.1 void [parse_for_moos](#) (const std::string & *in*, google::protobuf::Message * *msg*)
[inline]

Parses the string 'in' to Google Protocol Buffers message 'msg'. All errors are written to the [goby::util::glogger\(\)](#).

Parameters

<i>in</i>	std::string to parse
<i>msg</i>	Google Protocol buffers message to store result

Definition at line 44 of file moos_protobuf_helpers.h.

16.1.2.2 `void serialize_for_moos ( std::string * out, const google::protobuf::Message & msg ) [inline]`

Converts the Google Protocol Buffers message 'msg' into a suitable (human readable) string 'out' for sending via MOOS.

Parameters

<i>out</i>	pointer to std::string to store serialized result
<i>msg</i>	Google Protocol buffers message to serialize

Definition at line 32 of file moos_protobuf_helpers.h.

17 Example Documentation

17.1 acomms/chat/chat.cpp

chat.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name>Chat</name>
    <id>1</id>
    <size>32</size>
    <queuing>
      <ack>true</ack>
      <newest_first>false</newest_first>
    </queuing>
    <layout>
      <string>
        <name>message</name>
        <max_length>26</max_length>
      </string>
    </layout>

    <!-- only used by pAcommsHandler (publish/subscribe)-->
    <trigger>publish</trigger> <!-- pack -->
    <trigger_var>OUT_MESSAGE</trigger_var>
    <on_receipt> <!-- unpack -->
      <publish>
        <publish_var>IN_MESSAGE</publish_var>
        <message_var>message</message_var>
      </publish>
    </on_receipt>
    <!-- end used by pAcommsHandler -->

  </message>
```

```
</message_set>
```

chat.cpp

```
// copyright 2009 t. schneider tes@mit.edu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.

// usage: connect two modems and then run
// > chat /dev/tty_modem_A 1 2 log_file_A
// > chat /dev/tty_modem_B 2 1 log_file_B

// type into a window and hit enter to send a message. messages will be queued
// and sent on a fixed rotating cycle

#include <iostream>

#include "goby/acomms/dccl.h"
#include "goby/acomms/queue.h"
#include "goby/acomms/modem_driver.h"
#include "goby/acomms/amac.h"
#include "goby/acomms/bind.h"
#include "goby/util/string.h"
#include "goby/util/time.h"

#include <boost/lexical_cast.hpp>

#include "chat_curses.h"

using goby::util::as;
using goby::util::goby_time;

int startup_failure();
void received_data(const goby::acomms::protobuf::ModemDataTransmission&);
void received_ack(const goby::acomms::protobuf::ModemDataAck&);
std::string decode_received(const std::string& data);
void monitor_mac(const goby::acomms::protobuf::ModemMsgBase* mac_msg);

std::ofstream fout_;
goby::acomms::DCCLCodec dccl_(&fout_);
goby::acomms::QueueManager q_manager_(&fout_);
goby::acomms::MMDriver mm_driver_(&fout_);
goby::acomms::MACManager mac_(&fout_);
ChatCurses curses_;
int my_id_;
int buddy_id_;

int main(int argc, char* argv[])
{
    //
```

```

// 1. Deal with command line parameters
//

if(argc != 5) return startup_failure();

std::string serial_port = argv[1];
try
{
    my_id_ = boost::lexical_cast<int>(argv[2]);
    buddy_id_ = boost::lexical_cast<int>(argv[3]);
}
catch(boost::bad_lexical_cast&)
{
    std::cerr << "bad value for my_id: " << argv[2] << " or buddy_id: " <<
    argv[3] << ". these must be unsigned integers." << std::endl;
    return startup_failure();
}

std::string log_file = argv[4];
fout_.open(log_file.c_str());
if(!fout_.is_open())
{
    std::cerr << "bad value for log_file: " << log_file << std::endl;
    return startup_failure();
}

// bind the signals of these libraries
bind(mm_driver_, q_manager_, mac_);

//
// Initiate DCCL (libdccl)
//
goby::acomms::protobuf::DCCLConfig dccl_cfg;
dccl_cfg.add_message_file()->set_path(ACOMMS_EXAMPLES_DIR "/chat/chat.xml")
;

//
// Initiate queue manager (libqueue)
//
goby::acomms::protobuf::QueueManagerConfig q_manager_cfg;
q_manager_cfg.set_modem_id(my_id_);
goby::acomms::connect(&q_manager_.signal_receive, &received_data);
goby::acomms::connect(&q_manager_.signal_ack, &received_ack);
for(int i = 0, n = dccl_cfg.message_file_size(); i < n; ++i)
    q_manager_cfg.add_message_file()->CopyFrom(dccl_cfg.message_file(i));

//
// Initiate modem driver (libmodemdriver)
//
goby::acomms::protobuf::DriverConfig driver_cfg;
driver_cfg.set_modem_id(my_id_);
driver_cfg.set_serial_port(serial_port);
driver_cfg.AddExtension(MicroModemConfig::nvram_cfg, "SRC," +
    as<std::string>(my_id_));

//
// Initiate medium access control (libamac)
//
goby::acomms::protobuf::MACConfig mac_cfg;
mac_cfg.set_type(goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED);
mac_cfg.set_modem_id(my_id_);
goby::acomms::connect(&mac_.signal_initiate_transmission, &monitor_mac);

```

```

goby::acomms::protobuf::Slot my_slot;
my_slot.set_src(my_id_);
my_slot.set_dest(buddy_id_);
my_slot.set_rate(0);
my_slot.set_slot_seconds(12);
my_slot.set_type(goby::acomms::protobuf::SLOT_DATA);

goby::acomms::protobuf::Slot buddy_slot;
buddy_slot.set_src(buddy_id_);
buddy_slot.set_dest(my_id_);
buddy_slot.set_rate(0);
buddy_slot.set_slot_seconds(12);
buddy_slot.set_type(goby::acomms::protobuf::SLOT_DATA);

if(my_id_ < buddy_id_)
{
    mac_cfg.add_slot()->CopyFrom(my_slot);
    mac_cfg.add_slot()->CopyFrom(buddy_slot);
}
else
{
    mac_cfg.add_slot()->CopyFrom(buddy_slot);
    mac_cfg.add_slot()->CopyFrom(my_slot);
}

//
// Start up everything
//
try
{
    dccl_.set_cfg(dccl_cfg);
    q_manager_.set_cfg(q_manager_cfg);
    mac_.startup(mac_cfg);
    mm_driver_.startup(driver_cfg);
}
catch(std::runtime_error& e)
{
    std::cerr << "exception at startup: " << e.what() << std::endl;
    return startup_failure();
}

curses_.set_modem_id(my_id_);
curses_.startup();

//
// Loop until terminated (CTRL-C)
//
for(;;)
{
    std::string line;
    curses_.run_input(line);

    if(!line.empty())
    {
        std::map<std::string, goby::acomms::DCCLMessageVal> vals;
        vals["message"] = line;

        std::string bytes_out;

        unsigned message_id = 1;
        dccl_.encode(message_id, bytes_out, vals);
    }
}

```

```

        goby::acomms::protobuf::ModemDataTransmission message_out;
        message_out.set_data(bytes_out);
        // send this message to my buddy!
        message_out.mutable_base()->set_dest(buddy_id_);
        // set the time of this message
        message_out.mutable_base()->set_time(as<std::string>(goby_time()));

        message_out.mutable_queue_key()->set_id(message_id);
        q_manager_.push_message(message_out);
    }

    try
    {
        mm_driver_.do_work();
        mac_.do_work();
        q_manager_.do_work();
    }
    catch(std::runtime_error& e)
    {
        curses_.cleanup();
        std::cerr << "exception while running: " << e.what() << std::endl;
        return 1;
    }
}

return 0;
}

int startup_failure()
{
    std::cerr << "usage: chat /dev/tty_modem my_id buddy_id log_file" <<
        std::endl;
    return 1;
}

void monitor_mac(const goby::acomms::protobuf::ModemMsgBase* mac_msg)
{
    if(mac_msg->src() == my_id_)
        curses_.post_message("{control} starting send to my buddy");
    else if(mac_msg->src() == buddy_id_)
        curses_.post_message("{control} my buddy might be sending to me now");
}

void received_data(const goby::acomms::protobuf::ModemDataTransmission&
    message_in)
{
    curses_.post_message(message_in.base().src(), decode_received(message_in.
        data()));
}

void received_ack(const goby::acomms::protobuf::ModemDataAck& ack_message)
{
    curses_.post_message
        (ack_message.base().src(),
         std::string("{ acknowledged receiving message starting with: "
                     + decode_received(ack_message.orig_msg().data()).substr(0,
                     5)) + " }");
}

std::string decode_received(const std::string& data)
{

```

```

        std::map<std::string, goby::acomms::DCCLMessageVal> vals;
        dccl_.decode(data, vals);
        return vals["message"];
    }

```

17.2 libamac/amac_simple/amac_simple.cpp

amac_simple.cpp

```

// copyright 2009 t. schneider tes@mit.edu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.

#include "goby/acomms/amac.h"
#include "goby/acomms/connect.h"
#include <iostream>

using goby::acomms::operator<<;

void init_transmission(goby::acomms::protobuf::ModemMsgBase* base_msg);

int main(int argc, char* argv[])
{
    //
    // 1. Create a MACManager and feed it a std::ostream to log to
    //
    goby::acomms::MACManager mac(&std::cout);

    //
    // 2. Configure it for TDMA with basic peer discovery, rate 0, 10 second
    //     slots, and expire vehicles after 2 cycles of no communications. also, we are modem
    //     id 1.
    //
    goby::acomms::protobuf::MACConfig cfg;
    cfg.set_type(goby::acomms::protobuf::MAC_AUTO_DECENTRALIZED);
    cfg.set_rate(0);
    cfg.set_slot_seconds(10);
    cfg.set_expire_cycles(2);
    cfg.set_modem_id(1);

    //
    // 3. Set up the callback
    //

    // give a callback to use for actually initiating the transmission. this
    // would be bound to goby::acomms::ModemDriverBase::initiate_transmission if using
    // libmodemdriver.

```



```

goby::acomms::connect(&mac.signal_initiate_transmission, &init_transmission
);

//
// 4. Let it run for a bit alone in the world
//
mac.startup(cfg);
for(unsigned i = 1; i < 60; ++i)
{
    mac.do_work();
    sleep(1);
}

//
// 5. Discover some friends (modem ids 2 & 3)
//

goby::acomms::protobuf::ModemMsgBase msg_from_2, msg_from_3;
msg_from_2.set_src(2);
msg_from_3.set_src(3);

mac.handle_modem_all_incoming(msg_from_2);
mac.handle_modem_all_incoming(msg_from_3);

//
// 6. Run it, hearing consistently from #3, but #2 has gone silent and will
    be expired after 2 cycles
//

for(;;)
{
    mac.do_work();
    sleep(1);
    mac.handle_modem_all_incoming(msg_from_2);
}

return 0;
}

void init_transmission(goby::acomms::protobuf::ModemMsgBase* init_message)
{
    std::cout << "starting transmission with these values: " << *init_message <
        < std::endl;
}

```

17.3 libdccl/dccl_simple/dccl_simple.cpp

simple.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name>Simple</name>
    <id>1</id>
    <size>32</size>
    <layout>
      <string>
        <name>s_key</name>
        <max_length>26</max_length>
      </string>
    </layout>
  </message>
</message_set>

```

```

    </layout>
  </message>
</message_set>

```

dccl_simple.cpp

```

// copyright 2009 t. schneider tes@mit.edu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.

// encodes/decodes a string using the DCCL codec library
// assumes prior knowledge of the message format (required fields)

#include "goby/acomms/dccl.h"
#include <iostream>

using goby::acomms::operator<<;

int main()
{
    // initialize input contents to encoder. since
    // simple.xml only has a <string/> message var
    // we only need one map.
    std::map<std::string, goby::acomms::DCCLMessageVal> val_map;

    // initialize output bytes
    std::string bytes;

    std::cout << "loading xml file: xml/simple.xml" << std::endl;

    // instantiate the parser with a single xml file (simple.xml).
    // also pass the schema, relative to simple.xml, for XML validity
    // checking (syntax).
    goby::acomms::DCCLCodec dccl;
    goby::acomms::protobuf::DCCLConfig cfg;
    cfg.add_message_file()->set_path(DCCL_EXAMPLES_DIR "/dccl_simple/simple.xml");
    dccl.set_cfg(cfg);

    // read message content (in this case from the command line)
    std::string input;
    std::cout << "input a string to send: " << std::endl;
    getline(std::cin, input);

    // key is <name/> child for a given message_var
    // (i.e. child of <int/>, <string/>, etc)
    val_map["s_key"] = input;

    std::cout << "passing values to encoder:\n" << val_map;

```

```

// we are encoding for message id 1 - given in simple.xml
unsigned id = 1;
dccl.encode(id, bytes, val_map);

std::cout << "received hexadecimal string: " << goby::acomms::hex_encode(
    bytes) << std::endl;

val_map.clear();

// input contents right back to decoder
std::cout << "passed hexadecimal string to decoder: " <<
    goby::acomms::hex_encode(bytes) << std::endl;

dccl.decode(bytes, val_map);

std::cout << "received values:" << std::endl
    << val_map;

// now you can use it...
std::string output = val_map["s_key"];

return 0;
}

```

17.4 libdccl/plusnet/plusnet.cpp

naicon_command.xml

```

<?xml version="1.0" encoding="UTF8"?>
<message_set>
  <message>
    <name>SENSOR_DEPLOY</name>
    <destination_var key="DestinationPlatformId">PLUSNET_MESSAGES</destination_var>
    <trigger>publish</trigger>
    <trigger_var mandatory_content="MessageType=SENSOR_DEPLOY">
      PLUSNET_MESSAGES
    </trigger_var>
    <size>32</size>
    <header>
      <id>10</id>
      <time>
        <name>Timestamp</name>
      </time>
      <src_id>
        <name>SourcePlatformId</name>
      </src_id>
      <dest_id>
        <name>DestinationPlatformId</name>
      </dest_id>
    </header>
    <layout>
      <static>
        <name>MessageType</name>
        <value>SENSOR_DEPLOY</value>
      </static>
      <static>
        <name>SensorCommandType</name>
        <value>0</value>
      </static>
    </layout>
  </message>
</message_set>

```

```

<float>
  <name>DeployLatitude</name>
  <precision>5</precision>
  <max>90</max>
  <min>-90</min>
</float>
<float>
  <name>DeployLongitude</name>
  <precision>5</precision>
  <max>180</max>
  <min>-180</min>
</float>
<int>
  <name>DeployDepth</name>
  <max>254</max>
  <min>0</min>
</int>
<int>
  <name>DeployDuration</name>
  <max>62</max>
  <min>1</min>
</int>
<float>
  <name>AbortLatitude</name>
  <precision>4</precision>
  <max>90</max>
  <min>-90</min>
</float>
<float>
  <name>AbortLongitude</name>
  <precision>4</precision>
  <max>180</max>
  <min>-180</min>
</float>
<int>
  <name>AbortDepth</name>
  <max>510</max>
  <min>0</min>
</int>
<int>
  <name>MissionType</name>
  <max>6</max>
  <min>0</min>
</int>
<int>
  <name>OperatingRadius</name>
  <max>1022</max>
  <min>0</min>
</int>
<float algorithm="angle_0_360">
  <name>DCLFOVStartHeading</name>
  <max>360</max>
  <min>0</min>
  <precision>1</precision>
</float>
<float algorithm="angle_0_360">
  <name>DCLFOVEndHeading</name>
  <max>360</max>
  <min>0</min>
  <precision>1</precision>
</float>
<float>

```

```

        <name>DCLSearchRate</name>
        <max>254</max>
        <min>0</min>
        <precision>1</precision>
    </float>
</layout>
<on_receipt>
    <publish>
        <publish_var>NAFCON_MESSAGES</publish_var>
        <all />
    </publish>
</on_receipt>
</message>
<message>
    <name>SENSOR_PROSECUTE</name>
    <destination_var key="DestinationPlatformId">PLUSNET_MESSAGES</destination_var>
    <trigger>publish</trigger>
    <trigger_var mandatory_content="MessageType=SENSOR_PROSECUTE">
        PLUSNET_MESSAGES
    </trigger_var>
    <size>32</size>
    <header>
        <id>11</id>
        <time>
            <name>TargetTimestamp</name>
        </time>
        <src_id>
            <name>SourcePlatformId</name>
        </src_id>
        <dest_id>
            <name>DestinationPlatformId</name>
        </dest_id>
    </header>
    <layout>
        <static>
            <name>MessageType</name>
            <value>SENSOR_PROSECUTE</value>
        </static>
        <int>
            <name>SensorCommandType</name>
            <min>1</min>
            <max>4</max>
        </int>
        <int>
            <name>PlatformID</name>
            <max>30</max>
            <min>0</min>
        </int>
        <int>
            <name>TrackNumber</name>
            <max>254</max>
            <min>0</min>
        </int>
        <float>
            <name>TargetLatitude</name>
            <precision>5</precision>
            <max>90</max>
            <min>-90</min>
        </float>
        <float>
            <name>TargetLongitude</name>
            <precision>5</precision>

```

```

        <max>180</max>
        <min>-180</min>
    </float>
    <int>
        <name>TargetDepth</name>
        <max>1022</max>
        <min>0</min>
    </int>
    <float algorithm="angle_0_360">
        <name>TargetHeading</name>
        <max>360</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <float>
        <name>TargetSpeed</name>
        <max>12.6</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <int>
        <name>TargetSpectralLevel1</name>
        <max>126</max>
        <min>0</min>
    </int>
    <int>
        <name>TargetFrequency1</name>
        <max>4094</max>
        <min>0</min>
    </int>
    <int>
        <name>TargetBandwidth1</name>
        <max>20</max>
        <min>0</min>
    </int>
    <int>
        <name>TargetSpectralLevel2</name>
        <max>126</max>
        <min>0</min>
    </int>
    <int>
        <name>TargetFrequency2</name>
        <max>4094</max>
        <min>0</min>
    </int>
    <int>
        <name>TargetBandwidth2</name>
        <max>20</max>
        <min>0</min>
    </int>
    <int>
        <name>ProsecuteDuration</name>
        <max>62</max>
        <min>1</min>
    </int>
    <float>
        <name>AbortLatitude</name>
        <precision>4</precision>
        <max>90</max>
        <min>-90</min>
    </float>
</float>

```

```

        <name>AbortLongitude</name>
        <precision>4</precision>
        <max>180</max>
        <min>-180</min>
    </float>
    <int>
        <name>AbortDepth</name>
        <max>126</max>
        <min>0</min>
    </int>
</layout>
<on_receipt>
    <publish>
        <publish_var>NAFCON_MESSAGES</publish_var>
        <all />
    </publish>
</on_receipt>
</message>
</message_set>

```

nafcon_report.xml

```

<?xml version="1.0" encoding="UTF8" ?>
<message_set>
    <message>
        <name>SENSOR_STATUS</name>
        <trigger>publish</trigger> <!-- package upon publish to some moos var -->
        <trigger_moos_var mandatory_content="MessageType=SENSOR_STATUS">
            PLUSNET_MESSAGES
        </trigger_moos_var>
        <size>32</size>
        <header>
            <id>12</id>
            <time>
                <name>Timestamp</name>
            </time>
            <src_id>
                <name>SourcePlatformId</name>
            </src_id>
            <dest_id>
                <name>DestinationPlatformId</name>
            </dest_id>
        </header>
        <layout>
            <static>
                <name>MessageType</name>
                <value>SENSOR_STATUS</value>
            </static>
            <static>
                <name>SensorReportType</name>
                <value>0</value>
            </static>
            <float>
                <name>NodeLatitude</name>
                <precision>5</precision>
                <max>90</max>
                <min>-90</min>
            </float>
            <float>
                <name>NodeLongitude</name>
                <precision>5</precision>
                <max>180</max>
            </float>
        </layout>
    </message>
</message_set>

```

```

        <min>-180</min>
    </float>
    <int>
        <name>NodeDepth</name>
        <max>1022</max>
        <min>0</min>
    </int>
    <int>
        <name>NodeCEP</name>
        <max>1022</max>
        <min>0</min>
    </int>
    <float algorithm="angle_0_360">
        <name>NodeHeading</name>
        <max>360</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <float>
        <name>NodeSpeed</name>
        <max>12.6</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <int>
        <name>MissionState</name>
        <max>6</max>
        <min>0</min>
    </int>
    <int>
        <name>MissionType</name>
        <max>6</max>
        <min>0</min>
    </int>
    <int>
        <name>LastGPSTimestamp</name>
        <max>2000000000</max>
        <min>1000000000</min>
    </int>
    <int>
        <name>PowerLife</name>
        <max>1022</max>
        <min>1</min>
    </int>
    <int>
        <name>SensorHealth</name>
        <max>14</max>
        <min>0</min>
    </int>
    <int>
        <name>RecorderState</name>
        <max>1</max>
        <min>0</min>
    </int>
    <int>
        <name>RecorderLife</name>
        <max>30</max>
        <min>0</min>
    </int>
    <int>
        <name>NodeSpecificInfo0</name>
        <max>254</max>

```



```

        <min>0</min>
    </int>
    <int>
        <name>NodeSpecificInfo1</name>
        <max>254</max>
        <min>0</min>
    </int>
    <int>
        <name>NodeSpecificInfo2</name>
        <max>254</max>
        <min>0</min>
    </int>
    <int>
        <name>NodeSpecificInfo3</name>
        <max>254</max>
        <min>0</min>
    </int>
    <int>
        <name>NodeSpecificInfo4</name>
        <max>254</max>
        <min>0</min>
    </int>
    <int>
        <name>NodeSpecificInfo5</name>
        <max>254</max>
        <min>0</min>
    </int>
</layout>
<on_receipt>
    <publish>
        <moos_var>NAFCON_MESSAGES</moos_var>
        <all />
    </publish>
</on_receipt>
</message>
<message>
    <name>SENSOR_CONTACT</name>
    <trigger>publish</trigger>
    <trigger_moos_var mandatory_content="MessageType=SENSOR_CONTACT">
        PLUSNET_MESSAGES
    </trigger_moos_var>
    <size>32</size>
    <header>
        <id>13</id>
        <time>
            <name>ContactTimestamp</name>
        </time>
        <src_id>
            <name>SourcePlatformId</name>
        </src_id>
        <dest_id>
            <name>DestinationPlatformId</name>
        </dest_id>
    </header>
    <layout>
        <static>
            <name>MessageType</name>
            <value>SENSOR_CONTACT</value>
        </static>
        <static>
            <name>SensorReportType</name>
            <value>1</value>
        </static>
    </layout>

```

```

</static>
<float algorithm="angle_0_360">
  <name>SensorHeading</name>
  <max>360</max>
  <min>0</min>
  <precision>1</precision>
</float>
<float>
  <name>SensorPitch</name>
  <max>90</max>
  <min>-90</min>
  <precision>0</precision>
</float>
<float>
  <name>SensorRoll</name>
  <max>180</max>
  <min>-180</min>
  <precision>0</precision>
</float>
<float>
  <name>SensorLatitude</name>
  <precision>5</precision>
  <max>90</max>
  <min>-90</min>
</float>
<float>
  <name>SensorLongitude</name>
  <precision>5</precision>
  <max>180</max>
  <min>-180</min>
</float>
<int>
  <name>SensorDepth</name>
  <max>1022</max>
  <min>0</min>
</int>
<int>
  <name>SensorCEP</name>
  <max>1022</max>
  <min>0</min>
</int>
<float>
  <name>ContactBearing</name>
  <max>360</max>
  <min>0</min>
  <precision>1</precision>
</float>
<float>
  <name>ContactBearingUncertainty</name>
  <max>10</max>
  <min>0</min>
  <precision>1</precision>
</float>
<float>
  <name>ContactBearingRate</name>
  <max>10</max>
  <min>-10</min>
  <precision>1</precision>
</float>
<float>
  <name>ContactBearingRateUncertainty</name>
  <max>3.0</max>

```

```

        <min>0</min>
        <precision>1</precision>
    </float>
    <float>
        <name>ContactElevation</name>
        <max>90</max>
        <min>-90</min>
        <precision>1</precision>
    </float>
    <float>
        <name>ContactElevationUncertainty</name>
        <max>10</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <int>
        <name>ContactSpectralLevel1</name>
        <max>126</max>
        <min>0</min>
    </int>
    <int>
        <name>ContactFrequency1</name>
        <max>4094</max>
        <min>0</min>
    </int>
    <int>
        <name>ContactBandwidth1</name>
        <max>20</max>
        <min>0</min>
    </int>
    <int>
        <name>ContactSpectralLevel2</name>
        <max>126</max>
        <min>0</min>
    </int>
    <int>
        <name>ContactFrequency2</name>
        <max>4094</max>
        <min>0</min>
    </int>
    <int>
        <name>ContactBandwidth2</name>
        <max>20</max>
        <min>0</min>
    </int>
</layout>
<on_receipt>
    <publish>
        <moos_var>NAFCON_MESSAGES</moos_var>
        <all />
    </publish>
</on_receipt>
</message>
<message>
    <name>SENSOR_TRACK</name>
    <trigger>publish</trigger>
    <trigger_moos_var mandatory_content="MessageType=SENSOR_TRACK">
        PLUSNET_MESSAGES
    </trigger_moos_var>
    <size>32</size>

<header>

```

```

<id>14</id>
<time>
  <name>TrackTimestamp</name>
</time>
<src_id>
  <name>SourcePlatformId</name>
</src_id>
<dest_id>
  <name>DestinationPlatformId</name>
</dest_id>
</header>

<layout>
  <static>
    <name>MessageType</name>
    <value>SENSOR_TRACK</value>
  </static>
  <static>
    <name>SensorReportType</name>
    <value>2</value>
  </static>
  <int>
    <name>PlatformID</name>
    <max>30</max>
    <min>0</min>
  </int>
  <int>
    <name>TrackNumber</name>
    <max>254</max>
    <min>0</min>
  </int>
  <float>
    <name>TrackLatitude</name>
    <precision>5</precision>
    <max>90</max>
    <min>-90</min>
  </float>
  <float>
    <name>TrackLongitude</name>
    <precision>5</precision>
    <max>180</max>
    <min>-180</min>
  </float>
  <int>
    <name>TrackCEP</name>
    <max>1022</max>
    <min>0</min>
  </int>
  <int>
    <name>TrackDepth</name>
    <max>1022</max>
    <min>0</min>
  </int>
  <int>
    <name>TrackDepthUncertainty</name>
    <max>62</max>
    <min>0</min>
  </int>
  <float algorithm="angle_0_360">
    <name>TrackHeading</name>
    <max>360</max>
    <min>0</min>

```

```

        <precision>1</precision>
    </float>
    <float>
        <name>TrackHeadingUncertainty</name>
        <max>10</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <float>
        <name>TrackSpeed</name>
        <max>12.6</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <float>
        <name>TrackSpeedUncertainty</name>
        <max>3</max>
        <min>0</min>
        <precision>1</precision>
    </float>
    <int>
        <name>DepthClassification</name>
        <max>6</max>
        <min>0</min>
    </int>
    <int>
        <name>TrackClassification</name>
        <max>6</max>
        <min>0</min>
    </int>
    <int>
        <name>TrackSpectralLevel1</name>
        <max>126</max>
        <min>0</min>
    </int>
    <int>
        <name>TrackFrequency1</name>
        <max>4094</max>
        <min>0</min>
    </int>
    <int>
        <name>TrackBandwidth1</name>
        <max>20</max>
        <min>0</min>
    </int>
    <int>
        <name>TrackSpectralLevel2</name>
        <max>126</max>
        <min>0</min>
    </int>
    <int>
        <name>TrackFrequency2</name>
        <max>4094</max>
        <min>0</min>
    </int>
    <int>
        <name>TrackBandwidth2</name>
        <max>20</max>
        <min>0</min>
    </int>
</layout>
<on_receipt>

```

```

    <publish>
      <moos_var>NAFCON_MESSAGES</moos_var>
      <all />
    </publish>
  </on_receipt>
</message>
</message_set>

```

plusnet.cpp

```

// t. schneider tes@mit.edu 3.31.09
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.

// encodes/decodes the message given in the pGeneralCodec documentation
// also includes the simple.xml file to show example of DCCLCodec instantiation
// with multiple files

// this is an example showing some of the "MOOS" related features of libdccl
// that can be used (if desired) in the absence of MOOS

#include "goby/acomms/dccl.h"

#include <exception>
#include <iostream>

using namespace goby;
using goby::acomms::operator<<;

int main()
{
    std::cout << "loading nafcon xml files" << std::endl;

    acomms::DCCLCodec dccl;

    goby::acomms::protobuf::DCCLConfig cfg;
    cfg.add_message_file()->set_path(DCCL_EXAMPLES_DIR "
        /plusnet/nafcon_command.xml");
    cfg.add_message_file()->set_path(DCCL_EXAMPLES_DIR "
        /plusnet/nafcon_report.xml");

    dccl.set_cfg(cfg);

    std::cout << std::string(30, '#') << std::endl
              << "detailed message summary:" << std::endl
              << std::string(30, '#') << std::endl
              << dccl;

    std::cout << std::string(30, '#') << std::endl
              << "ENCODE / DECODE example:" << std::endl

```

```

        << std::string(30, '#') << std::endl;

// initialize input contents to encoder
std::map<std::string, aocomms::DCCLMessageVal> in_vals;

// initialize output message
aocomms::protobuf::ModemDataTransmission msg;

in_vals["PLUSNET_MESSAGES"] = "
    MessageType=SENSOR_STATUS,SensorReportType=0,
    SourcePlatformId=1,DestinationPlatformId=3,Timestamp=1191947446.91117,NodeLatitu
    de=47.7448,NodeLongitude=-122.845,NodeDepth=0.26,NodeCEP=0,NodeHeading=169.06,No
    deSpeed=0,MissionState=2,MissionType=2,LastGPSTimestamp=1191947440,PowerLife=6,S
    ensorHealth=0,RecorderState=1,RecorderLife=0,NodeSpecificInfo0=0,NodeSpecificInf
    o1=0,NodeSpecificInfo2=23,NodeSpecificInfo3=0,NodeSpecificInfo4=3,NodeSpecificInfo5=0";

std::cout << "passing values to encoder:" << std::endl
    << in_vals;

dccl.pubsub_encode("SENSOR_STATUS", &msg, in_vals);

std::cout << "received: "
    << msg
    << std::endl;

std::multimap<std::string, aocomms::DCCLMessageVal> out_vals;

std::cout << "passed to decoder: "
    << msg
    << std::endl;

dccl.pubsub_decode(msg, &out_vals);

std::cout << "received values:" << std::endl
    << out_vals;

return 0;
}

```

17.5 libdccl/test/test.cpp

test.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name>TEST</name>
    <size>128</size>
    <id>4</id>
    <repeat/>
    <layout>
      <static>
        <name>Stat</name>
        <value>hi</value>
      </static>
    </layout>
  </message>
</message_set>

```

```

    <float algorithm="sum:F:I">
      <name>SUM</name>
      <max>200</max>
      <min>-100</min>
      <precision>2</precision>

    </float>

    <float algorithm="*2">
      <name>F</name>
      <max>100</max>
      <min>-50</min>
      <precision>2</precision>
    </float>

    <bool algorithm="invert">
      <name>B</name>

    </bool>

    <enum>
      <name>E</name>
      <value>cat</value>
      <value>dog</value>
      <value>mouse</value>
      <value>emu</value>

    </enum>

    <string algorithm="prepend_fat">
      <name>S</name>
      <max_length>15</max_length>

    </string>

    <float>
      <name>my_NAN</name>
      <max>100</max>
      <min>-50</min>
      <precision>0</precision>
    </float>

    <int algorithm="+1">
      <name>I</name>
      <max>100</max>
      <min>-50</min>
      <max_delta>10</max_delta>
    </int>

    <!--      <hex>
      <name>H</name>
      <num_bytes>4</num_bytes>
      </hex> -->

  </layout>
</message>
</message_set>

```

test.cpp

```

// t. schneider tes@mit.edu 11.20.09
//

```



```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.

#include "goby/acoms/dccl.h"
#include <iostream>
#include <cassert>

using namespace goby;
using goby::acomms::operator<<;

void plus1(acoms::DCCLMessageVal& mv)
{
    long l = mv;
    ++l;
    mv = l;
}

void times2(acoms::DCCLMessageVal& mv)
{
    double d = mv;
    d *= 2;
    mv = d;
}

void prepend_fat(acoms::DCCLMessageVal& mv)
{
    std::string s = mv;
    s = "fat_" + s;
    mv = s;
}

void invert(acoms::DCCLMessageVal& mv)
{
    bool b = mv;
    b ^= 1;
    mv = b;
}

void algsum(acoms::DCCLMessageVal& mv, const
            std::vector<acomms::DCCLMessageVal>& ref_vals)
{
    double d = 0;
    // index 0 is the name ("sum"), so start at 1
    for(size_t i = 0, n = ref_vals.size(); i < n; ++i)
    {
        d += double(ref_vals[i]);
    }
    mv = d;
}
```

```

int main()
{
    std::cout << "loading xml file: " << DCCL_EXAMPLES_DIR "/test/test.xml" <<
        std::endl;

    // instantiate the parser with a single xml file
    goby::acomms::DCCLCodec dccl(&std::cerr);
    goby::acomms::protobuf::DCCLConfig cfg;
    cfg.add_message_file()->set_path(DCCL_EXAMPLES_DIR "/test/test.xml");
    // must be kept secret!
    cfg.set_crypto_passphrase("my_passphrase!");
    dccl.set_cfg(cfg);

    std::cout << dccl << std::endl;

    // load up the algorithms
    dccl.add_algorithm("prepend_fat", &prepend_fat);
    dccl.add_algorithm("+1", &plus1);
    dccl.add_algorithm("*2", &times2);
    dccl.add_algorithm("invert", &invert);
    dccl.add_adv_algorithm("sum", &algsum);

    std::map<std::string, std::vector<acomms::DCCLMessageVal> > in;

    bool b = true;
    std::vector<acomms::DCCLMessageVal> e;
    e.push_back("dog");
    e.push_back("cat");
    e.push_back("emu");

    std::string s = "raccoon";
    std::vector<acomms::DCCLMessageVal> i;
    i.push_back(30);
    i.push_back(40);
    std::vector<acomms::DCCLMessageVal> f;
    f.push_back(-12.5);
    f.push_back(1);

    std::string h = "abcd1234";
    std::vector<acomms::DCCLMessageVal> sum(2,0);

    in["B"] = std::vector<acomms::DCCLMessageVal>(1,b);
    in["E"] = e;
    in["S"] = std::vector<acomms::DCCLMessageVal>(1,s);
    in["I"] = i;
    in["F"] = f;
    in["H"] = std::vector<acomms::DCCLMessageVal>(1,h);
    in["SUM"] = sum;

    std::string bytes;
    std::cout << "sent values:" << std::endl
        << in;

    dccl.encode(4, bytes, in);

    std::cout << "hex out: " << goby::acomms::hex_encode(bytes) << std::endl;
    bytes.resize(bytes.length() + 20, '0');
    std::cout << "hex in: " << goby::acomms::hex_encode(bytes) << std::endl;
}

```

```

std::map<std::string, std::vector<acomms::DCCLMessageVal> > out;

dccl.decode(bytes, out);

std::cout << "received values:" << std::endl
          << out;

sum[0] = double(i[0]) + double(f[0]);
sum[1] = double(i[1]) + double(f[1]);
i[0] = int(i[0]) + 1;
i[1] = int(i[1]) + 1;

acomms::DCCLMessageVal tmp = b;
invert(tmp);
b = tmp;

tmp = s;
prepend_fat(tmp);
tmp.get(s);

tmp = f[0];
times2(tmp);
f[0] = tmp;

tmp = f[1];
times2(tmp);
f[1] = tmp;

assert(out["B"][0] == b);
assert(out["E"][0] == e[0]);
assert(out["E"][1] == e[1]);
assert(out["E"][2] == e[2]);
assert(out["S"][0] == s);
assert(out["F"][0] == f[0]);
assert(out["F"][1] == f[1]);
assert(out["SUM"][0] == sum[0]);
assert(out["SUM"][1] == sum[1]);
assert(out["I"][0] == i[0]);
assert(out["I"][1] == i[1]);
//assert(out["H"][0] == h);

std::cout << "all tests passed" << std::endl;
}

```

17.6 libdccl/two_message/two_message.cpp

two_message.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name>GoToCommand</name>
    <id>2</id>
    <size>32</size>
    <header>
      <dest_id>
        <name>destination</name>
      </dest_id>
    </header>
  </message>
</message_set>

```

```

    <static>
      <name>type</name>
      <value>goto</value>
    </static>
    <int>
      <name>goto_x</name>
      <max>10000</max>
      <min>0</min>
    </int>
    <int>
      <name>goto_y</name>
      <max>10000</max>
      <min>0</min>
    </int>
    <bool>
      <name>lights_on</name>
    </bool>
    <string>
      <name>new_instructions</name>
      <max_length>10</max_length>
    </string>
    <float>
      <name>goto_speed</name>
      <max>3</max>
      <min>0</min>
      <precision>2</precision>
    </float>
  </layout>
</message>
<message>
  <name>VehicleStatus</name>
  <id>3</id>
  <size>32</size>
  <layout>
    <float>
      <name>nav_x</name>
      <max>10000</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <float>
      <name>nav_y</name>
      <max>10000</max>
      <min>0</min>
      <precision>1</precision>
    </float>
    <enum>
      <name>health</name>
      <value>good</value>
      <value>low_battery</value>
      <value>abort</value>
    </enum>
  </layout>
</message>
</message_set>

```

two_message.cpp

```

// t. schneider tes@mit.edu 3.31.09
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by

```

```

// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.

// encodes/decodes the message given in the pGeneralCodec documentation
// also includes the simple.xml file to show example of DCCLCodec instantiation
// with multiple files
#include "goby/acoms/dccl.h"
#include <exception>
#include <iostream>

using namespace goby;
using goby::acomms::operator<<;

int main()
{
    std::cout << "loading xml files: xml/simple.xml, xml/two_message.xml"
               << std::endl;

    goby::acomms::DCCLCodec dccl;
    goby::acomms::protobuf::DCCLConfig cfg;
    cfg.add_message_file()->set_path(DCCL_EXAMPLES_DIR "/dccl_simple/simple.xml"
                                     "");
    cfg.add_message_file()->set_path(DCCL_EXAMPLES_DIR "
                                     /two_message/two_message.xml");
    // must be kept secret!
    cfg.set_crypto_passphrase("my_passphrase!");
    dccl.set_cfg(cfg);

    // show some useful information about all the loaded messages
    std::cout << std::string(30, '#') << std::endl
               << "number of messages loaded: " << dccl.message_count() <<
               std::endl
               << std::string(30, '#') << std::endl;

    std::cout << std::string(30, '#') << std::endl
               << "detailed message summary:" << std::endl
               << std::string(30, '#') << std::endl
               << dccl;

    std::cout << std::string(30, '#') << std::endl
               << "brief message summary:" << std::endl
               << std::string(30, '#') << std::endl
               << dccl.brief_summary();

    std::cout << std::string(30, '#') << std::endl
               << "all loaded ids:" << std::endl
               << std::string(30, '#') << std::endl
               << dccl.all_message_ids();

    std::cout << std::string(30, '#') << std::endl
               << "all loaded names:" << std::endl

```

```

        << std::string(30, '#') << std::endl
        << dccl.all_message_names();

std::cout << std::string(30, '#') << std::endl
        << "all required message var names:" << std::endl
        << std::string(30, '#') << std::endl;

std::set<unsigned> s = dccl.all_message_ids();
for (std::set<unsigned>::const_iterator it = s.begin(), n = s.end(); it !=
    n; ++it)
    std::cout << (*it) << ": " << dccl.message_var_names(*it) << std::endl;

std::cout << std::string(30, '#') << std::endl
        << "ENCODE / DECODE example:" << std::endl
        << std::string(30, '#') << std::endl;

// initialize input contents to encoder
std::map<std::string, acommms::DCCLMessageVal> vals;

// initialize output hexadecimal
std::string bytes2, bytes3;

// id = 2, name = GoToCommand
vals["destination"] = 2;
vals["goto_x"] = 423;
vals["goto_y"] = 523;
vals["lights_on"] = true;
vals["new_instructions"] = "make_toast";
vals["goto_speed"] = 2.3456;

// id = 3, name = VehicleStatus
vals["nav_x"] = 234.5;
vals["nav_y"] = 451.3;
vals["health"] = "abort";

std::cout << "passing values to encoder:" << std::endl
        << vals;

dccl.encode(2, bytes2, vals);
dccl.encode(3, bytes3, vals);

std::cout << "received hexadecimal string for message 2 (GoToCommand): "
        << goby::acomms::hex_encode(bytes2)
        << std::endl;

std::cout << "received hexadecimal string for message 3 (VehicleStatus): "
        << goby::acomms::hex_encode(bytes3)
        << std::endl;

vals.clear();

std::cout << "passed hexadecimal string for message 2 to decoder: "
        << goby::acomms::hex_encode(bytes2)
        << std::endl;

std::cout << "passed hexadecimal string for message 3 to decoder: "
        << goby::acomms::hex_encode(bytes3)
        << std::endl;

```

```

    dccl.decode(bytes2, vals);
    dccl.decode(bytes3, vals);

    std::cout << "received values:" << std::endl
               << vals;

    return 0;
}

```

17.7 libmodemdriver/driver_simple/driver_simple.cpp

```

// copyright 2009-2011 t. schneider tes@mit.edu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.
//
// Usage (WHOI Micro-Modem): run
// > driver_simple /dev/tty_of_modem_A 1
//
// wait a few seconds
//
// > driver_simple /dev/tty_of_modem_B 2
//
// be careful of collisions if you start them at the same time (this is why
// libamac exists!)
//
// Usage (example ABCModem): run
// > driver_simple /dev/tty_of_modem_A 1 ABCDriver
// > driver_simple /dev/tty_of_modem_B 2 ABCDriver
// Also see abc_modem_simulator.cpp

#include "goby/acomms/modem_driver.h"
#include "goby/acomms/connect.h"

#include <iostream>

using goby::acomms::operator<<;

void handle_data_request(const goby::acomms::protobuf::ModemDataRequest&
    request_msg,
                        goby::acomms::protobuf::ModemDataTransmission*
    data_msg);
void handle_data_receive(const goby::acomms::protobuf::ModemDataTransmission&
    data_msg);

int main(int argc, char* argv[])
{

```

```

if(argc < 3)
{
    std::cout << "usage: driver_simple /dev/tty_of_modem modem_id [type:
MMDriver (default)|ABCDriver]" << std::endl;
    return 1;
}

//
// 1. Create and initialize the driver we want
//
goby::acomms::ModemDriverBase* driver = 0;
goby::acomms::protobuf::DriverConfig cfg;

// set the serial port given on the command line
cfg.set_serial_port(argv[1]);
using google::protobuf::uint32;
// set the source id of this modem
uint32 our_id = goby::util::as<uint32>(argv[2]);
cfg.set_modem_id(our_id);

if(argc == 4)
{
    if(boost::iequals(argv[3], "ABCDriver"))
    {
        std::cout << "Starting Example driver ABCDriver" << std::endl;
        driver = new goby::acomms::ABCDriver(&std::clog);
    }
}

// default to WHOI MicroModem
if(!driver)
{
    std::cout << "Starting WHOI Micro-Modem MMDriver" << std::endl;
    driver = new goby::acomms::MMDriver(&std::clog);
    // turn data quality factor message on
    // (example of setting NVRAM configuration)
    cfg.AddExtension(MicroModemConfig::nvram_cfg, "DQF,1");
}

// for handling $CADDRQ
goby::acomms::connect(&driver->signal_receive, &handle_data_receive);
goby::acomms::connect(&driver->signal_data_request, &handle_data_request);

//
// 2. Startup the driver
//
driver->startup(cfg);

//
// 3. Initiate a transmission cycle
//

goby::acomms::protobuf::ModemMsgBase transmit_init_message;
transmit_init_message.set_src(goby::util::as<unsigned>(our_id));
transmit_init_message.set_dest(goby::acomms::BROADCAST_ID);
transmit_init_message.set_rate(0);

std::cout << transmit_init_message << std::endl;

driver->handle_initiate_transmission(&transmit_init_message);

```



```

//
// 4. Run the driver
//

// 10 hz is good
int i = 0;
while(1)
{
    ++i;
    driver->do_work();

    // send another transmission every 60 seconds
    if(!(i % 60))
        driver->handle_initiate_transmission(&transmit_init_message);

    // in here you can initiate more transmissions as you want
    usleep(100000);
}
return 0;
}

//
// 5. Handle the data request ($CADRQ)
//

void handle_data_request(const goby::acomms::protobuf::ModemDataRequest&
    request_msg,
                        goby::acomms::protobuf::ModemDataTransmission*
    data_msg)
{
    data_msg->set_data(goby::acomms::hex_decode("aa1100bbccdddef0987654321"));
    data_msg->set_ack_requested(false);
}

//
// 6. Post the received data
//

void handle_data_receive(const goby::acomms::protobuf::ModemDataTransmission&
    data_msg)
{
    std::cout << "got a message: " << data_msg << std::endl;
}

```

17.8 libqueue/queue_simple/queue_simple.cpp

simple.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<message_set>
  <message>
    <name>Simple</name>
    <id>1</id>
    <size>32</size>
    <queuing>
      <ack>false</ack>
      <value_base>1</value_base>
      <t1>1800</t1>
    </queuing>
  </message>
</message_set>

```

```

    <!-- used by libdccl alone and not needed for libqueue, but left for reference-->
    <layout>
        <string>
            <name>s_key</name>
            <max_length>24</max_length>
        </string>
    </layout>
    <!-- end used by libdccl -->
</message>
</message_set>

```

queue_simple.cpp

```

// copyright 2009 t. schneider tes@mit.edu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This software is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this software. If not, see <http://www.gnu.org/licenses/>.

// queues a single message from the DCCL library

#include "goby/acomms/queue.h"
#include "goby/acomms/connect.h"
#include <iostream>

using goby::acomms::operator<<;

void received_data(const goby::acomms::protobuf::ModemDataTransmission& msg);

int main()
{
    //
    // 1. Initialize the QueueManager
    //

    // create a QueueManager for all our queues
    // and at the same time add our message as a DCCL queue
    goby::acomms::QueueManager q_manager(&std::clog);

    // our modem id (arbitrary, but must be unique in the network)
    int our_id = 1;

    goby::acomms::protobuf::QueueManagerConfig cfg;
    cfg.set_modem_id(our_id);
    cfg.add_message_file()->set_path(QueueManagerConfig::QUEUE_EXAMPLES_DIR "
        /queue_simple/simple.xml");
    q_manager.set_cfg(cfg);

    // set up the callback to handle received DCCL messages
    goby::acomms::connect(&q_manager.signal_receive, &received_data);
}

```

```

// see what our QueueManager contains
std::cout << "1. " << q_manager << std::endl;

//
// 2. Push a message to a queue
//

// let's make a message to store in the queue
goby::acomms::protobuf::ModemDataTransmission data_msg;

unsigned dest = 0;
data_msg.mutable_base()->set_dest(dest);
// typically grab these data from DCCLCodec::encode, but here we'll just
// enter an example
// hexadecimal string
data_msg.set_data(goby::acomms::hex_decode("2000802500006162636431323334"))
;

// push to queue 1 (which is the Simple message <id/>)
data_msg.mutable_queue_key()->set_id(1);
q_manager.push_message(data_msg);

std::cout << "2. pushing message to queue 1: " << data_msg << std::endl;
std::cout << "\tdata as hex: " << goby::acomms::hex_encode(data_msg.data())
<< std::endl;

//
// 3. Create a loopback to simulate the Link Layer (libmodemdriver & modem
// firmware)
//

std::cout << "3. executing loopback (simulating sending a message to
ourselves over the modem link)" << std::endl;

// pretend the modem is requesting data of up to 32 bytes

goby::acomms::protobuf::ModemDataRequest request_msg;
request_msg.set_max_bytes(32);

data_msg.Clear();
q_manager.handle_modem_data_request(request_msg, &data_msg);

std::cout << "4. requesting data, got: " << data_msg << std::endl;
std::cout << "\tdata as hex: " << goby::acomms::hex_encode(data_msg.data())
<< std::endl;

//
// 4. Pass the received message to the QueueManager (same as outgoing
// message)
//
q_manager.handle_modem_receive(data_msg);

return 0;
}

//
// 5. Do something with the received message
//
void received_data(const goby::acomms::protobuf::ModemDataTransmission& msg)
{
    std::cout << "5. received message: " << msg << std::endl;
    std::cout << "\tdata as hex: " << goby::acomms::hex_encode(msg.data()) <<

```

```
        std::endl;  
    }
```