Goby v2

Generated by Doxygen 1.8.6

Tue Nov 29 2016 23:10:37
## Contents

1 **Goby Underwater Autonomy Project** 2
   1.1 Resources .................................................. 2
   1.2 Developer manual ........................................ 2
   1.3 Publications ............................................... 2
   1.4 Download and Install Goby ................................. 2
   1.5 Building Examples ......................................... 2
   1.6 Authors ...................................................... 3

2 **goby-acomms: An overview of Acoustic Communications Library** 3
   2.1 Quick Start .................................................. 3
   2.2 Overview .................................................... 4
      2.2.1 Analogy to established networking systems .......... 4
      2.2.2 Acoustic Communications are slow .................... 4
      2.2.3 Efficiency to make messages small is good .......... 4
      2.2.4 Total throughput unrealistic: prioritize data ......... 5
      2.2.5 Despite all this, simplicity is good ................. 5
      2.2.6 Component model ....................................... 5
   2.3 dccl: Encoding and decoding ............................... 6
   2.4 queue: Priority based message queuing .................... 6
   2.5 modemdriver: Modem driver ............................... 7
   2.6 amac: Medium Access Control (MAC) ....................... 7
   2.7 Software concepts used in goby-acomms .................. 7
      2.7.1 Signal / Slot model for asynchronous events ......... 7
      2.7.2 Google Protocol Buffers ............................ 8
   2.8 UML models .................................................. 9

3 **goby-acomms: DCCL (Dynamic Compact Control Language)** 10
   3.1 Designing a message ....................................... 10
   3.2 DCCL Protobuf Options .................................. 13
   3.3 Interacting with the DCCLCodec ........................... 14
   3.4 Encryption ................................................ 15
   3.5 Example messages .......................................... 15
      3.5.1 Minimal functional DCCL message .................... 15
      3.5.2 Two Message example ................................ 16
      3.5.3 DCCL Test2 showing an embedded message encoded by a custom (non-default) codec ... 19
      3.5.4 DCCL Test3 ............................................ 19
      3.5.5 DCCL Test4 ............................................ 20
      3.5.6 DCCL Test5 ............................................ 20
      3.5.7 DCCL Test6 ............................................ 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.8</td>
<td>DCCL Test7</td>
<td>21</td>
</tr>
<tr>
<td>3.5.9</td>
<td>DCCL Test8</td>
<td>22</td>
</tr>
<tr>
<td>3.5.10</td>
<td>DCCL Test9</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>goby-accomms: queue (Message Priority Queuing)</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Understanding dynamic priority queuing</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Queuing Protobuf Options</td>
<td>24</td>
</tr>
<tr>
<td>4.3</td>
<td>Interacting with the QueueManager</td>
<td>27</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Instantiate and configure</td>
<td>27</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Signals and (application layer) slots</td>
<td>27</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Operation</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Example messages</td>
<td>28</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Minimal functional DCCL / Queue message</td>
<td>28</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Test1</td>
<td>28</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Test2, Test3, Test4</td>
<td>29</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Test5</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>goby-accomms: amac (Medium Access Control)</td>
<td>30</td>
</tr>
<tr>
<td>5.1</td>
<td>Supported MAC schemes</td>
<td>30</td>
</tr>
<tr>
<td>5.2</td>
<td>Interacting with the goby::acomms::MACManager</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>goby-accomms: modemdriver (Driver to interact with modem firmware)</td>
<td>32</td>
</tr>
<tr>
<td>6.1</td>
<td>Abstract class: ModemDriverBase</td>
<td>32</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Interacting with the goby::acomms::ModemDriverBase</td>
<td>32</td>
</tr>
<tr>
<td>6.2</td>
<td>Protobuf Message goby::acomms::protobuf::ModemTransmission</td>
<td>33</td>
</tr>
<tr>
<td>6.3</td>
<td>Writing a new driver</td>
<td>33</td>
</tr>
<tr>
<td>6.4</td>
<td>WHOI Micro-Modem Driver: MMDriver</td>
<td>37</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Supported Functionality</td>
<td>37</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Micro-Modem NMEA to Goby ModemTransmission mapping</td>
<td>37</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Sequence diagrams for various Micro-Modem features using Goby</td>
<td>45</td>
</tr>
<tr>
<td>7</td>
<td>goby-util: Overview of Utility Libraries</td>
<td>52</td>
</tr>
<tr>
<td>7.1</td>
<td>Overview</td>
<td>52</td>
</tr>
<tr>
<td>7.2</td>
<td>Logging</td>
<td>52</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Configurable extension of std::ostream - liblogger</td>
<td>52</td>
</tr>
<tr>
<td>7.3</td>
<td>TCP and Serial port communications - liblinebasedcomms</td>
<td>53</td>
</tr>
<tr>
<td>8</td>
<td>goby-moos: An overview of the Goby/MOOS interoperability library</td>
<td>55</td>
</tr>
<tr>
<td>8.1</td>
<td>iFrontSeat</td>
<td>55</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Writing a new driver for iFrontSeat</td>
<td>55</td>
</tr>
<tr>
<td>9</td>
<td>Module Index</td>
<td>61</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>9.1</td>
<td>Modules</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>Namespace Index</td>
<td>61</td>
</tr>
<tr>
<td>10.1</td>
<td>Namespace List</td>
<td>61</td>
</tr>
<tr>
<td>11</td>
<td>Hierarchical Index</td>
<td>61</td>
</tr>
<tr>
<td>11.1</td>
<td>Class Hierarchy</td>
<td>61</td>
</tr>
<tr>
<td>12</td>
<td>Class Index</td>
<td>62</td>
</tr>
<tr>
<td>12.1</td>
<td>Class List</td>
<td>62</td>
</tr>
<tr>
<td>13</td>
<td>File Index</td>
<td>64</td>
</tr>
<tr>
<td>13.1</td>
<td>File List</td>
<td>64</td>
</tr>
<tr>
<td>14</td>
<td>Module Documentation</td>
<td>78</td>
</tr>
<tr>
<td>14.1</td>
<td>API classes for the Dynamic Compact Control Language (includes writing custom encoders)</td>
<td>78</td>
</tr>
<tr>
<td>14.2</td>
<td>API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver)</td>
<td>78</td>
</tr>
<tr>
<td>14.2.1</td>
<td>Detailed Description</td>
<td>78</td>
</tr>
<tr>
<td>15</td>
<td>Namespace Documentation</td>
<td>78</td>
</tr>
<tr>
<td>15.1</td>
<td>goby Namespace Reference</td>
<td>78</td>
</tr>
<tr>
<td>15.1.1</td>
<td>Detailed Description</td>
<td>80</td>
</tr>
<tr>
<td>15.1.2</td>
<td>Function Documentation</td>
<td>80</td>
</tr>
<tr>
<td>15.2</td>
<td>goby::acomms Namespace Reference</td>
<td>80</td>
</tr>
<tr>
<td>15.2.1</td>
<td>Detailed Description</td>
<td>83</td>
</tr>
<tr>
<td>15.3</td>
<td>goby::common Namespace Reference</td>
<td>83</td>
</tr>
<tr>
<td>15.3.1</td>
<td>Detailed Description</td>
<td>85</td>
</tr>
<tr>
<td>15.4</td>
<td>goby::common::tcolor Namespace Reference</td>
<td>86</td>
</tr>
<tr>
<td>15.4.1</td>
<td>Detailed Description</td>
<td>86</td>
</tr>
<tr>
<td>15.4.2</td>
<td>Function Documentation</td>
<td>86</td>
</tr>
<tr>
<td>15.5</td>
<td>goby::pb Namespace Reference</td>
<td>87</td>
</tr>
<tr>
<td>15.5.1</td>
<td>Detailed Description</td>
<td>87</td>
</tr>
<tr>
<td>15.6</td>
<td>goby::transitional Namespace Reference</td>
<td>87</td>
</tr>
<tr>
<td>15.6.1</td>
<td>Detailed Description</td>
<td>89</td>
</tr>
<tr>
<td>15.6.2</td>
<td>Typedef Documentation</td>
<td>89</td>
</tr>
<tr>
<td>15.6.3</td>
<td>Enumeration Type Documentation</td>
<td>89</td>
</tr>
<tr>
<td>15.6.4</td>
<td>Function Documentation</td>
<td>90</td>
</tr>
<tr>
<td>15.6.5</td>
<td>Variable Documentation</td>
<td>92</td>
</tr>
<tr>
<td>16</td>
<td>Class Documentation</td>
<td>92</td>
</tr>
<tr>
<td>16.1</td>
<td>boost::asio::time_traits&lt; goby::common::GobyTime &gt; Struct Template Reference</td>
<td>92</td>
</tr>
<tr>
<td>16.1.1</td>
<td>Detailed Description</td>
<td>93</td>
</tr>
<tr>
<td>Section</td>
<td>Class Reference</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>16.2</td>
<td>ChatCurses</td>
<td></td>
</tr>
<tr>
<td>16.3</td>
<td>goby::acomms::ABCDriver</td>
<td></td>
</tr>
<tr>
<td>16.4</td>
<td>goby::acomms::MACManager</td>
<td></td>
</tr>
<tr>
<td>16.5</td>
<td>goby::acomms::MMDriver</td>
<td></td>
</tr>
<tr>
<td>16.6</td>
<td>goby::acomms::ModemDriverBase</td>
<td></td>
</tr>
<tr>
<td>16.7</td>
<td>goby::acomms::QueueException</td>
<td></td>
</tr>
<tr>
<td>16.8</td>
<td>goby::acomms::QueueManager</td>
<td></td>
</tr>
<tr>
<td>16.9</td>
<td>goby::common::Colors</td>
<td></td>
</tr>
<tr>
<td>16.10</td>
<td>goby::common::ConfigException</td>
<td></td>
</tr>
<tr>
<td>16.11</td>
<td>goby::common::FlexNCurses</td>
<td></td>
</tr>
<tr>
<td>16.12</td>
<td>goby::common::FlexOstream</td>
<td></td>
</tr>
<tr>
<td>16.13</td>
<td>goby::common::TermColor</td>
<td></td>
</tr>
<tr>
<td>16.14</td>
<td>goby::Exception</td>
<td></td>
</tr>
<tr>
<td>16.15</td>
<td>goby::moos::BluefinCommsDriver</td>
<td></td>
</tr>
<tr>
<td>16.16</td>
<td>goby::moos::UFldDriver</td>
<td></td>
</tr>
</tbody>
</table>
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. The Goby libraries are licensed under the GNU Lesser General Public License v3 [http://www.gnu.org/licenses/lgpl.html](http://www.gnu.org/licenses/lgpl.html) and the applications are licensed under the GNU General Public License v3 [http://www.gnu.org/licenses/gpl.html](http://www.gnu.org/licenses/gpl.html).

1.1 Resources

- Home page, code, bug tracking, and answers: [https://launchpad.net/goby](https://launchpad.net/goby).
- User Manual: [pdf](#).
- Developers’ Manual: [html](#) [pdf](#).

1.2 Developer manual

- **goby-acomms**: An overview of Acoustic Communications Library - tackle the extremely rate limited acoustic networking problem. This library was designed with four modules that 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/](http://www.boost.org/) for many utility tasks to fill in areas where the C++ Standard Library is insufficient or unelegant.
- **goby-moos**: An overview of the Goby/MOOS interoperability library - classes, applications (e.g. pAcomms-Handler and iFrontSeat), and functions for interoperating between Goby and the MOOS middleware.

1.3 Publications

- T. Schneider and H. Schmidt, Goby-Acomms version 2: extensible marshalling, queuing, and link layer interfacing for acoustic telemetry. 9th IFAC Conference on Manoeuvring and Control of Marine Craft ’12 / Arenzano, Italy.

1.4 Download and Install Goby


1.5 Building Examples

Please visit [http://gobysoft.com/wiki/Examples](http://gobysoft.com/wiki/Examples) 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: An overview of Acoustic Communications Library

Table of Contents for goby-acomms: An overview of Acoustic Communications Library.

- 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
- dccl: Encoding and decoding (Detailed documentation)
- queue: Priority based message queuing (Detailed documentation)
- modemdriver: Modem driver (Detailed documentation)
- amac: 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: http://gobysoft.com/wiki/- InstallingGoby.

2. Identify which components you need:
   - Encoding and decoding from C++ types to bit-packed messages: dccl.
   - Queueing of DCCL messages with priority based message selection: queue.
   - A driver for interacting with the acoustic modem firmware: modemdriver.
   - Time division multiple access (TDMA) medium access control (MAC): amac.

3. Look at the "simple" code examples that accompany each component (dccl_simple.cpp, queue_simple.cpp, driver_simple.cpp, amac_simple.cpp). Then look at the example that uses all the components 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-C-X.200-199407-I/en.

<table>
<thead>
<tr>
<th>OSI Layer</th>
<th>Goby-Acomms library component</th>
<th>API class(es)</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Not yet part of Goby</td>
<td></td>
<td>MOOS Application: pAcommsHandler</td>
</tr>
<tr>
<td>Presentation</td>
<td>dccl: Encoding and decoding</td>
<td>goby::acomms::DCCL-Codec</td>
<td>dccl_simple.cpp two_message.cpp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>chat.cpp</td>
</tr>
<tr>
<td>Session</td>
<td>Not used, sessions are established passively.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td>queue: Priority based message queuing</td>
<td>goby::acomms::Queue-Manager</td>
<td>queue_simple.cpp chat.cpp</td>
</tr>
<tr>
<td>Network</td>
<td>Does not yet exist. All transmissions are considered single hop, currently. Addressing routing over multiple hops is an open and pressing research problem.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Link</td>
<td>modemdriver: Modem driver</td>
<td>classes derived from goby::acomms::Modem-DriverBase; e.g. goby::acomms::MM-Driver</td>
<td>driver_simple.cpp chat.cpp</td>
</tr>
<tr>
<td></td>
<td>amac: Medium Access Control (MAC)</td>
<td>goby::acomms::MAC-Manager</td>
<td>amac_simple.cpp chat.cpp</td>
</tr>
<tr>
<td>Physical</td>
<td>Not part of Goby</td>
<td></td>
<td>Modem Firmware, e.g. WHOI Micro-Modem Firmware (NMEA 0183 on RS-232) (see Interface Guide)</td>
</tr>
</tbody>
</table>

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, acomms) 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-acomms 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-acomms design as a priority based queuing 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) but 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 trade-off 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-acomms 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 library showing the interface classes:
For a more detailed model, see the UML models section.

2.3 dccl: Encoding and decoding

The Dynamic Compact Control Language (DCCL) provides a structure for defining messages to be sent through an acoustic modem. The messages are configured in Google Protocol Buffers 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.whoi.edu/ccl/).

Unlike the encoder / decoder provided with Google Protocol Buffers, each field (which could be a primitive type like double, int32, string or an user-defined embedded message like CTDMessage) of a DCCL message can be encoded using a DCCL built-in or user-defined encoder. This allows the codecs to be matched to the data’s physical origins and thus make the most of the limited throughput available by making very small encoded messages.

Detailed documentation for goby-acomms: DCCL (Dynamic Compact Control Language).

2.4 queue: Priority based message queuing

The goby-acomms queuing (queue) component interacts with both the application level process and the modem driver process that talks directly to the modem.

On the application side, queue provides the ability for the application level process to push DCCL 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, queue 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 as many messages from the user into a single frame from the modem as possible using the DCCLCodec’s repeated encoding functionality. Note, however, that queue 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: queue (Message Priority Queueing).

2.5 modemdriver: Modem driver

The goby-acomms Modem driver component (modemdriver) of the Goby-Acomms library provides an interface from the rest of goby-acomms to the acoustic modem firmware. While currently the only driver publicly available is for the WHOI Micro-Modem (and for an example toy modem "ABCDriver"), this component is written in such a way that drivers for any acoustic modem that interfaces over a serial or TCP connection and can provide (or provide 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: modemdriver (Driver to interact with modem firmware).

2.6 amac: Medium Access Control (MAC)

The goby-acomms MAC component (amac) 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 amac for the frequency division multiple access (FDMA) scenario.

The Goby AMAC 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.).

- 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: amac (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 component'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 queue, I need to connect to the signal QueueManager::signal_receive. Thus, I need to define a function or class method with the same signature:

```cpp
void receive_data(const google::protobuf::Message& msg);
```
At startup, I then connect the signal to the slot:

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

If instead, I was using a member function such as

```c++
class MyApplication
{
  public:
    void receive_data(const google::protobuf::Message& msg);
};
```

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

```c++
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](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](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:

```proto
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](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](http://code.google.com/apis/protocolbuffers/docs/reference/cpp-generated.html) for precise details):

```cpp
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();
};
```
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. acomms_modem_message.proto).

2.8 UML models

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

![UML diagram showing message sending sequence]

Figure 2: 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 modemdriver-firmware interaction will depend on the acoustic modem used.

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

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

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

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
Figure 3: UML model that illustrates the set of commands needed to start up goby-acomms and keep it running.

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

Table of contents for libdcl:

- Designing a message
- Interacting with the DCCLCodec
- DCCL Protobuf Options
- Encryption
- Example messages

Return to goby-acomms: An overview of Acoustic Communications Library.

3.1 Designing a message

DCCL uses the Google Protocol Buffers (Protobuf) language to define messages. DCCL specific components are defined as extensions to the Protobuf language message and field options. You should familiarize yourself with basic Protobuf using before reading the rest of this document: see Google Protocol Buffers and http://code.google.com/apis/protocolbuffers/docs/overview.html.
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 a Protobuf message with a single string field (let's call it "telegram") to hold our command:

```proto
define message Simple {
  required string telegram = 1;
}
```

The " = 1" indicates that this is the first field on the wire in our DCCL message. All fields must have a unique index, but otherwise these index values are not particularly important. "required" means a valid "Simple" message always contains something for "telegram" (could be an empty string).

To turn this Protobuf message into a DCCL message, we need to add a few options. All the options are defined in acoms_option_extensions.proto so we include that:

```proto
import "goby/common/protobuf/option_extensions.proto";
define message Simple {
  required string telegram = 1;
}
```

At a minimum we must give a unique ID for our DCCL message and a maximum number of bytes we allow the message to be before throwing an exception when it is loaded. This allows us to ensure that we are not creating messages larger than we can send with the physical hardware. We want to have the ability to use the lowest rate WHOI Micro-Modem message size, so we pick max_bytes to be 32. We are testing so we'll use an id of 124. See [http://gobysoft.org/wiki/DcclIdTable](http://gobysoft.org/wiki/DcclIdTable) for a list of the assigned DCCL IDs.

After these additions we have:

```proto
import "goby/common/protobuf/option_extensions.proto";
define message Simple {
  option (dccl.msg).id = 124;
  option (dccl.msg).max_bytes = 32;
  required string telegram = 1;
}
```

Finally, we need to pick an encoder/decoder (codec) for each field in our message. DCCL comes with defaults for all the Protobuf types. So if we don’t specifically list a codec for a given field, the default is used. The default "string" codec is goby::acoms::DCCLDefaultStringCodec and is variable length. It uses one byte to list the length of the string and then up to 255 bytes to hold the contents. To ensure we stay within our bounds for the entire message ((goby.msg).dccl.max_bytes = 32), we have to give a maximum allowed length for a string when using the DCCLDefaultStringCodec ((goby.field).dccl.max_length).

```proto
import "dccl/protobuf/option_extensions.proto";
define message Simple {
  // see [http://gobysoft.org/wiki/DcclIdTable](http://gobysoft.org/wiki/DcclIdTable)
  option (dccl.msg).id = 124;
  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;
  required string telegram = 1 [(dccl.field).max_length=30];
}
```

See dccl_simple.cpp for an example of how to use this message.

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

---

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
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:

<table>
<thead>
<tr>
<th>message variable name</th>
<th>description</th>
<th>type</th>
<th>bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>id number of the vehicle we are commanding</td>
<td>int32</td>
<td>[0, 31]</td>
</tr>
<tr>
<td>goto_x</td>
<td>meters east to transit from datum</td>
<td>int32</td>
<td>[0, 10000]</td>
</tr>
<tr>
<td>goto_y</td>
<td>meters north to transit from datum</td>
<td>int32</td>
<td>[0, 10000]</td>
</tr>
<tr>
<td>lights_on</td>
<td>turn on the lights?</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>new_instructions</td>
<td>string instructions</td>
<td>string</td>
<td>no longer than 10 characters</td>
</tr>
<tr>
<td>goto_speed</td>
<td>transit speed (m/s)</td>
<td>float</td>
<td>[0.00, 3.00]</td>
</tr>
</tbody>
</table>

Taking all this into account, we form the first message (named GoToCommand) in the file two_message.proto (see Two Message example):

We choose a dccl.id of 125 to avoid conflicting with the message from Scenario 1 (simple.proto) and a dccl-max_bytes of 32 bytes to again allow sending in the WHOI Micro-Modem rate 0 packet.

Now, for the second message in two_message.proto. 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:

<table>
<thead>
<tr>
<th>message variable name</th>
<th>description</th>
<th>type</th>
<th>bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>nav_x</td>
<td>current vehicle position (meters east of the datum)</td>
<td>integer</td>
<td>[0, 10000]</td>
</tr>
<tr>
<td>nav_y</td>
<td>current vehicle position (meters north of the datum)</td>
<td>integer</td>
<td>[0, 10000]</td>
</tr>
<tr>
<td>health</td>
<td>vehicle state</td>
<td>enumeration</td>
<td>HEALTH_GOOD, HEALTH_GOOD, HEALTH_LOW_BATTERY, or HEALTH_ABORT</td>
</tr>
</tbody>
</table>

The resulting message, can be seen under Two Message example. An example of how to use this message is given under two_message.cpp.

You can run analyze_dccl to view more information on your messages:

```
> analyze_dccl /path/to/two_message.proto
```

When I ran the above command I got:

```
read in: two_message.proto
*** Begin DCCCodec ***
2 messages loaded.
 * Begin GoToCommand *
 Actual maximum size of message: 18 bytes / 144 bits [dccl.id head: 8, user head: 0, body: 131, padding: 5]
 Allowed maximum size of message: 32 bytes / 256 bits
 ** Begin Header **
 ** End Header **
 ** Begin Body **
GoToCommand
 required int32 destination = 1;
 :: size = 5 bit(s)
 required int32 goto_x = 3;
 :: size = 14 bit(s)
 required int32 goto_y = 4;
 :: size = 14 bit(s)
 required bool lights_on = 5;
 :: size = 1 bit(s)
 required string new_instructions = 6;
```
3.2 DCCL Protobuf Options

This section gives an overview of the DCCL message and field options available for use with DCCL and the default field codecs. The full list is available in option_extensions.proto (as messages DCCLFieldOptions and DCCLMessageOptions).

**DCCL message options:**

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>uint32</td>
<td>required</td>
<td>A unique ID for each DCCL message</td>
</tr>
<tr>
<td>max_bytes</td>
<td>uint32</td>
<td>required</td>
<td>Maximum allowed size in bytes for the encoded message</td>
</tr>
<tr>
<td>codec</td>
<td>string</td>
<td>&quot;_default&quot;</td>
<td>Name of the codec to use for encoding the base message</td>
</tr>
</tbody>
</table>

**DCCL field options:**

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>default</th>
<th>required for codecs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>codec</td>
<td>string</td>
<td>&quot;_default&quot;</td>
<td>optional</td>
<td>Name of the codec to use for encoding this field</td>
</tr>
<tr>
<td>omit</td>
<td>bool</td>
<td>false</td>
<td>optional</td>
<td>Omit this field from all DCCL encoding</td>
</tr>
</tbody>
</table>

Besides validity checking, the most useful feature of analyze_dccl 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.
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_head</td>
<td>bool</td>
<td>false</td>
<td>optional&lt;br&gt;Set true for fields in the header (will <strong>not</strong> be encrypted, rather will be used to create the encryption IV).</td>
</tr>
<tr>
<td>precision</td>
<td>int32</td>
<td>0</td>
<td>goby::acomms::DCCLDefaultNumericFieldCodec (double, float)&lt;br&gt;Number of decimal digits of precision to keep (can be negative).</td>
</tr>
<tr>
<td>min</td>
<td>double</td>
<td>0</td>
<td>goby::acomms::DCCLDefaultNumericFieldCodec (double, float, int32, uint32, int64, uint64, fixed32, fixed64, sfixed32, sfixed64)&lt;br&gt;Minimum value that can be encoded in this field.</td>
</tr>
<tr>
<td>max</td>
<td>double</td>
<td>0</td>
<td>goby::acomms::DCCLDefaultNumericFieldCodec (double, float, int32, uint32, int64, uint64, fixed32, fixed64, sfixed32, sfixed64)&lt;br&gt;Maximum value that can be encoded in this field.</td>
</tr>
<tr>
<td>static_value</td>
<td>string</td>
<td>&quot;&quot;</td>
<td>goby::acomms::DCCLStaticCodec (any type)&lt;br&gt;The static value for use on decoding this placeholder field.</td>
</tr>
<tr>
<td>max_length</td>
<td>uint32</td>
<td>0</td>
<td>goby::acomms::DCCLDefaultStringCodec, goby::acomms::DCCLDefaultBytesCodec (string)&lt;br&gt;The maximum length of the string that can be stored in this field.</td>
</tr>
<tr>
<td>max_repeat</td>
<td>uint32</td>
<td>0</td>
<td>any repeated field&lt;br&gt;The maximum length of the repeated array (or vector).</td>
</tr>
</tbody>
</table>

### 3.3 Interacting with the DCCLCodec

Using the `goby::acomms::DCCLCodec` is a fairly straightforward endeavor (this example uses `dccl_simple.cpp`). First you need to get a pointer to the DCCLCodec singleton:

```cpp
goby::acomms::DCCLCodec* codec = goby::acomms::DCCLCodec::get();
```

Validate all messages with the DCCLCodec to ensure all bounding constraints are met:

```cpp
try
  dccl->validate<Simple>();
] catch(DCCLException& e)
  |
  std::cerr << "Oh no! " << e << std::endl;
  exit(1);
```

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
3.4 Encryption

Then, to encode a message, create a Protobuf message, set its fields and pass it to goby::acomms::DCCLCodec::encode():

```cpp
Simple message;
message.set_telegram("hello");
std::string bytes;
dccl->encode(&bytes, message);
```

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.

To decode a message (stored in `bytes` as a byte string), simply pass `bytes` as a reference along with pointers to the Protobuf message to store the results.

```cpp
message.Clear();
dccl->decode(bytes, &message);
```

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 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. You will want to make sure the header (designate fields for the header with `(goby.field).dccl.in_head = true`) is a nonce by including a constantly changing value such as time.

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

3.5 Example messages

This section provides a listing of DCCL example Protobuf messages used in the code examples and unit tests.

3.5.1 Minimal functional DCCL message

```proto
simple.proto

import "dccl/protobuf/option_extensions.proto";

message Simple {
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;

  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=30];
}
```

See Also

dcll_simple.cpp
3.5.2 Two Message example

two_message.proto

import "dccl/protobuf/option_extensions.proto";
message GoToCommand {
  option (dccl.msg).id = 125;
  option (dccl.msg).max_bytes = 32;
  required int32 destination = 1 [(dccl.field).max=31,
    (dccl.field).min=0,
    (dccl.field).precision=0];
  optional string type = 2 [(dccl.field).static_value="goto",
    (dccl.field).codec="_static"];
  required int32 goto_x = 3 [(dccl.field).max=10000,
    (dccl.field).min=0,
    (dccl.field).precision=0];
  required int32 goto_y = 4 [(dccl.field).max=10000,
    (dccl.field).min=0,
    (dccl.field).precision=0];
  required bool lights_on = 5;
  required string new_instructions = 6 [(dccl.field).max_length=10];
  required double goto_speed = 7 [(dccl.field).max=3,
    (dccl.field).min=0,
    (dccl.field).precision=2];
}
message VehicleStatus {
  option (dccl.msg).id = 126;
  option (dccl.msg).max_bytes = 32;
  required double nav_x = 1 [(dccl.field).max=10000,
    (dccl.field).min=0,
    (dccl.field).precision=1];
  required double nav_y = 2 [(dccl.field).max=10000,
    (dccl.field).min=0,
    (dccl.field).precision=1];
  required HealthEnum health = 3;
  enum HealthEnum {
    HEALTH_GOOD = 0;
    HEALTH_LOW_BATTERY = 1;
    HEALTH_ABORT = 2;
  }
}

See Also
two_message.cpp

Test1 showing all Protobuf types (using default codecs):
dccl1/test.proto

import "dccl/protobuf/option_extensions.proto";

enum Enum1 {
  ENUM_A = 1;
  ENUM_B = 2;
  ENUM_C = 3;
}
message EmbeddedMsg1 {
  optional double val1 = 1 [(dccl.field).min=0,
    (dccl.field).max=126,
    (dccl.field).precision=3];
  optional EmbeddedMsg2 msg = 2;
}
message EmbeddedMsg2
{
  optional double val = 1 [(dccl.field).min=0,
    (dccl.field).max=126,
    (dccl.field).precision=2];
  optional string sval = 2 [(dccl.field).max_length=10];
  optional Enum1 enum_default = 3;
}

message TestMsg
{
  option (dccl.msg).id = 2;
  option (dccl.msg).max_bytes = 512;

  // test default enc/dec
  optional double double_default_optional = 1 [(dccl.field).min=-100,
    (dccl.field).max=126,
    (dccl.field).precision=2,
    (dccl.field).in_head=true];
  optional float float_default_optional = 2 [(dccl.field).min=-20,
    (dccl.field).max=150,
    (dccl.field).precision=3];
  optional int32 int32_default_optional = 3 [(dccl.field).min=-20,
    (dccl.field).max=3000];
  optional int64 int64_default_optional = 4 [(dccl.field).min=-710,
    (dccl.field).max=3000];
  optional uint32 uint32_default_optional = 5 [(dccl.field).min=0,
    (dccl.field).max=3000];
  optional uint64 uint64_default_optional = 6 [(dccl.field).min=5,
    (dccl.field).max=3000];
  optional sint32 sint32_default_optional = 7 [(dccl.field).min=-60,
    (dccl.field).max=3000];
  optional sint64 sint64_default_optional = 8 [(dccl.field).min=-70,
    (dccl.field).max=3000];
  optional fixed32 fixed32_default_optional = 9 [(dccl.field).min=0,
    (dccl.field).max=400];
  optional fixed64 fixed64_default_optional = 10 [(dccl.field).min=0,
    (dccl.field).max=3000];
  optional sfixed32 sfixed32_default_optional = 11 [(dccl.field).min=0,
    (dccl.field).max=3000];
  optional sfixed64 sfixed64_default_optional = 12 [(dccl.field).min=0,
    (dccl.field).max=3000];
  optional bool bool_default_optional = 13;
  optional string string_default_optional = 14 [(dccl.field).max_length=8];
  optional bytes bytes_default_optional = 15 [(dccl.field).max_length=9];
  optional Enum1 enum_default_optional = 16;
  optional EmbeddedMsg1 msg_default_optional = 17;

  required double double_default_required = 21 [(dccl.field).min=-100,
    (dccl.field).max=126,
    (dccl.field).precision=2,
    (dccl.field).in_head=true];
  required float float_default_required = 22 [(dccl.field).min=-20,
    (dccl.field).max=150,
    (dccl.field).precision=3];
  required int32 int32_default_required = 23 [(dccl.field).min=-20,
    (dccl.field).max=3000];
  required int64 int64_default_required = 24 [(dccl.field).min=-710,
    (dccl.field).max=3000];
  required uint32 uint32_default_required = 25 [(dccl.field).min=0,
    (dccl.field).max=3000];
  required uint64 uint64_default_required = 26 [(dccl.field).min=5,
    (dccl.field).max=3000];
  required sint32 sint32_default_required = 27 [(dccl.field).min=-60,
    (dccl.field).max=3000];
  required sint64 sint64_default_required = 28 [(dccl.field).min=-70,
    (dccl.field).max=3000];
  required fixed32 fixed32_default_required = 29 [(dccl.field).min=0,
    (dccl.field).max=400];
  required fixed64 fixed64_default_required = 30 [(dccl.field).min=0,
    (dccl.field).max=3000];
  required sfixed32 sfixed32_default_required = 31 [(dccl.field).min=0,
    (dccl.field).max=3000];
  required sfixed64 sfixed64_default_required = 32 [(dccl.field).min=0,
    (dccl.field).max=3000];
}

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
See Also
dccl1/test.cpp
3.5 Example messages

3.5.3 DCCL Test2 showing an embedded message encoded by a custom (non-default) codec

dccl2/test.proto

```protobuf
import "dccl/protobuf/option_extensions.proto";

message CustomMsg
{
  option (dccl.msg).id = 3;
  option (dccl.msg).max_bytes = 256;
  option (dccl.msg).codec = "custom_codec";

  optional uint32 a = 1;
  optional bool b = 2;
}

message CustomMsg2
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 256;

  optional CustomMsg msg = 1;
  repeated int32 c = 3 [ (dccl.field).max=100,
                      (dccl.field).min=0,
                      (dccl.field).max_repeat=4,
                      (dccl.field).codec="int32_test_codec" ];
}
```

See Also
dccl2/test.cpp

3.5.4 DCCL Test3
dccl3/test.proto

```protobuf
import "goby/common/protobuf/option_extensions.proto";
import "dccl/protobuf/option_extensions.proto";
import "goby/test/acoms/dccl3/header.proto";

message GobyMessage
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [ (dccl.field).max_length=10 ];
  required Header header = 2;
}
```

protobuf/header.proto

```protobuf
import "goby/common/protobuf/option_extensions.proto";
import "dccl/protobuf/option_extensions.proto";

// required fields will be filled in for you by ApplicationBase
// if you choose not to do so yourself
message Header
{
  //
  // time
  //

  // result of goby::util::as<std::string>(goby_time())
  // e.g. "2002-01-20 23:59:59.000"
  required string time = 10 [ (dccl.field).codec="_time",
```
{dccl.field}.in_head=true;

//
// source

required string source_platform = 11 [{dccl.field}.codec="_platform<->modem_id",
{dccl.field}.in_head=true];

optional string source_app = 12 [{dccl.field}.omit=true];

//
// destination

enum PublishDestination { PUBLISH_SELF = 1; PUBLISH_OTHER = 2; PUBLISH_ALL = 3; }
optional PublishDestination dest_type = 13 [default = PUBLISH_SELF, {dccl.field}.in_head=true];

optional string dest_platform = 14 [{dccl.field}.codec="_platform<->modem_id",
{dccl.field}.in_head=true]; // required if dest_type == other

See Also
dccl3/test.cpp

3.5.5 DCCL Test4
dccl4/test.proto

import "dccl/protobuf/option_extensions.proto";
import "goby/test/acoms/dccl3/header.proto";

message GobyMessage1
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  optional int32 int32_val = 1 [{dccl.field}.min=0, {dccl.field}.max=20];
}

message GobyMessage2
{
  option (dccl.msg).id = 5;
  option (dccl.msg).max_bytes = 32;

  optional bool bool_val = 1;
}

message GobyMessage3
{
  option (dccl.msg).id = 6;
  option (dccl.msg).max_bytes = 32;

  optional string string_val = 1 [{dccl.field}.max_length=10];
}

See Also
dccl4/test.cpp

3.5.6 DCCL Test5
dccl5/test.proto

See Also
dccl5/test.cpp
3.5 Example messages

3.5.7 DCCL Test6

dccl6/test.proto

import "dccl/protobuf/option_extensions.proto";

message ShortIDMsg
{
  option (dccl.msg).id = 2;
  option (dccl.msg).max_bytes = 1;
}

message ShortIDMsgWithData
{
  option (dccl.msg).id = 3;
  option (dccl.msg).max_bytes = 10;

  optional int32 in_head = 1 [(dccl.field).in_head=true, (dccl.field).min=0, (dccl.field).max=100];
  optional int32 in_body = 2 [(dccl.field).in_head=true, (dccl.field).min=0, (dccl.field).max=100];
}

message LongIDMsg
{
  option (dccl.msg).id = 10000;  option (dccl.msg).max_bytes = 2;
}

message TooLongIDMsg
{
  option (dccl.msg).id = 32768;
  option (dccl.msg).max_bytes = 32;
}

message LongIDEdgeMsg
{
  option (dccl.msg).id = 128;
  option (dccl.msg).max_bytes = 2;
}

message ShortIDEdgeMsg
{
  option (dccl.msg).id = 127;
  option (dccl.msg).max_bytes = 1;
}

See Also

dccl6/test.cpp

3.5.8 DCCL Test7

dccl7/test.proto

import "dccl/protobuf/option_extensions.proto";

message BytesMsg
{
  option (dccl.msg).id = 10;
  option (dccl.msg).max_bytes = 32;

  required bytes req_bytes = 1 [(dccl.field).max_length=8];
  optional bytes opt_bytes = 2 [(dccl.field).max_length=8];
}

See Also

dccl7/test.cpp

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
3.5.9 DCCL Test8

dccl8/test.proto

import "dccl/protobuf/option_extensions.proto";
import "goby/test/acoms/dcc13/header.proto";

message GobyMessage1
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;
  optional int32 int32_val = 1 [ (dccl.field).min=0, (dccl.field).max=20 ];
}

message GobyMessage2
{
  option (dccl.msg).id = 5;
  option (dccl.msg).max_bytes = 32;
  optional bool bool_val = 1;
}

message GobyMessage3
{
  option (dccl.msg).id = 6;
  option (dccl.msg).max_bytes = 32;
  optional string string_val = 1 [ (dccl.field).max_length=10 ];
}

See Also

dccl8/test.cpp

3.5.10 DCCL Test9

dccl9/test.proto

import "dccl/protobuf/option_extensions.proto";

message MiniUser
{
  option (dccl.msg).id = 1000001;
  option (dccl.msg).max_bytes = 2;
  required uint32 user = 1 [ (dccl.field).min=0,
    (dccl.field).max=0x03FF,
    (dccl.field).in_head=true ];
}

message MiniOWTT
{
  option (dccl.msg).id = 1000002;
  option (dccl.msg).max_bytes = 2;
  required uint32 clock_mode = 1 [ (dccl.field).min=0,
    (dccl.field).max=3,
    (dccl.field).in_head=true ];
  required uint32 tod = 2 [ (dccl.field).min=0,
    (dccl.field).max=0x0F,
    (dccl.field).in_head=true ];
  required uint32 user = 3 [ (dccl.field).min=0,
    (dccl.field).max=0x0F,
    (dccl.field).in_head=true ];
}

message MiniAbort
4 goby-acomms: queue (Message Priority Queuing)

Table of Contents for queue:

- Understanding dynamic priority queuing
- Queuing Protobuf Options
- Interacting with the QueueManager
  - Instantiate and configure
  - Signals and (application layer) slots
  - Operation

Return to goby-acomms: An overview of Acoustic Communications Library.

4.1 Understanding dynamic priority queuing

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

\[
P(t) = V_{\text{base}} \frac{(t - t_{\text{last}})}{ttl}
\]

where \( t_{\text{last}} \) is the time of the last send from this queue.

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

The following graph illustrates the priority growth over time of three queues with different \( ttl \) and \( V_{\text{base}} \). A message is sent every 100 seconds and the queue that is chosen is marked on the graph.
Figure 4: Graph of the growth of queueing priorities for queue 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 Protobuf Options

This section gives an overview of the queue configuration options available. The full list is available in queue.proto (as messages goby::acomms::protobuf::QueuedMessageEntry).

Queue message options:

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack</td>
<td>bool</td>
<td>true</td>
<td>Whether an acoustic acknowledgment should be requested for messages sent from this queue.</td>
</tr>
<tr>
<td>blackout_time</td>
<td>uint32</td>
<td>0</td>
<td>Minimum number of seconds allowed between sending messages from this queue.</td>
</tr>
<tr>
<td>max_queue</td>
<td>uint32</td>
<td>0</td>
<td>Allowed size of the queue before overflow. If newest_first is true, the oldest elements are removed upon overflow, else the newest elements are (the queue blocks). 0 is a special value signifying infinity (no maximum).</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>newest_first</td>
<td>bool</td>
<td>true</td>
<td>(true=FILO, false=FIFO) whether to send newest messages in the queue first (FILO) or not (FIFO).</td>
</tr>
<tr>
<td>ttl</td>
<td>int32</td>
<td>1800</td>
<td>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 is effectively the same as ttl = \infty)</td>
</tr>
<tr>
<td>value_base</td>
<td>double</td>
<td>1</td>
<td>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 option. Essentially, a message with low ttl will become effective quickly again after a sent message (the priority line grows faster). See Understanding dynamic priority queueing for further discussion.</td>
</tr>
</tbody>
</table>
(Advanced) enable on-demand encoding where rather than queueing data, the data request is forwarded up to the application level via the signal `goby::acomms::QueueManager::signal_data_on_demand`.

If `encode_on_demand` == true, this sets the number of seconds before data encoded on demand are considering stale and thus must be demanded again with the signal `goby::acomms::QueueManager::signal_data_on_demand`. Setting this to 0 is unadvisable as it will cause many calls to `goby::acomms::QueueManager::signal_data_on_demand` and thus waste CPU cycles needlessly encoding.

Queue Role options: Queue needs to know how to address a message (the source ID and destination ID) as well as the time the message was generated. This information either read from the fields of the of the DCCL message (setting: FIELD_VALUE) or is statically configured (setting: STATIC). In the latter case, the configuration value "static_value" is set and used for every DCCL message of this type that gets queued by this QueueManager.

In the former case (the default), you can tag a given field of a DCCL message to a particular "role." This takes the place of a fixed transport layer header that protocols such as UDP use. The fields used in a given role can be anywhere within the message. The field is identified by its name (in the configuration value "field"). Submessage fields can be used by separating the field names by periods ("." until the child is a primitive type (e.g. uint32).

<table>
<thead>
<tr>
<th>RoleType</th>
<th>allowed field types</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE_ID</td>
<td>All integer types (uint32, int32, uint64, int64, ...)</td>
<td>The value in this field is used to represent the sending address (similar to an IP address) of the message.</td>
</tr>
<tr>
<td>DESTINATION_ID</td>
<td>All integer types (uint32, int32, uint64, int64, ...)</td>
<td>The value in this field is used to represent the destination address (similar to an IP address) of the message. 0 is reserved to indicate broadcast.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>uint64 or double</td>
<td>The value in this field is used as the timestamp of the message. If the type is double, it must be seconds (and fractional seconds) since the UNIX epoch (1970-01-01 midnight UTC). If it is a uint64, it must be microseconds since the UNIX epoch. This field used for expiring messages that exceed their ttl and thus must, in general, be set and correct.</td>
</tr>
</tbody>
</table>
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 set a unique identification number for this platform (the "modem ID") through the `goby::acomms::protobuf::QueueManagerConfig`.

You can configure queues by added repeated fields to the QueueManagerConfig's message_entry field, or by calling `goby::acomms::QueueManager::add_queue()` directly.

When using `goby::acomms::QueueManager` you will not likely need to use the `goby::acomms::DCCLCodec` directly much at all. All messages are pushed to the queues unencoded and are encoded automatically by `goby::acomms::QueueManager::signal_receive`.

For example, this code configures the QueueManager with a single queue (DCCL type GobyMessage)

```cpp
    goby::acomms::protobuf::QueueManagerConfig cfg;
    cfg.set_modem_id(our_id);
    goby::acomms::protobuf::QueuedMessageEntry* q_entry = cfg.add_message_entry();
    q_entry->set_protobuf_name("GobyMessage");
    q_entry->set_newest_first(true);
    goby::acomms::protobuf::QueuedMessageEntry::Role* dest_role = q_entry->add_role();
    dest_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::DESTINATION_ID);
    dest_role->set_field("header.dest_platform");
    goby::acomms::protobuf::QueuedMessageEntry::Role* time_role = q_entry->add_role();
    time_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::TIMESTAMP);
    time_role->set_field("header.time");
    goby::acomms::protobuf::QueuedMessageEntry::Role* src_role = q_entry->add_role();
    src_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::SOURCE_ID);
    src_role->set_field("header.source_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 (and decoded) DCCL data: `goby::acomms::QueueManager::signal_receive`
  - 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 encode_on_demand option to true for that particular Protobuf message (and if desired change the on_demand_skew_seconds). You must also connect a slot that will be executed each time data is requested to the signal `goby::acomms::QueueManager::signal_data_on_demand`. 

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
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 its DCCL (Protobuf) name.

At the driver layer, messages are requested using `goby::acomms::QueueManager::handle_modem_data_request` and incoming messages (including acknowledgments) are published using `goby::acomms::QueueManager::handle_modem_receive`. 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()` if using the Goby modemdriver).

See `queue_simple.cpp` for a basic complete example.

4.4 Example messages

This section provides a listing of queue example Protobuf messages used in the code examples and unit tests.

4.4.1 Minimal functional DCCL / Queue message

`simple.proto`

```protobuf
import "dccl/protobuf/option_extensions.proto";

message Simple {
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;

  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=30];
}
```

4.4.2 Test1

`queue1/test.proto`

```protobuf
import "dccl/protobuf/option_extensions.proto";

message TestMsg {
  option (dccl.msg).id = 2;
  option (dccl.msg).max_bytes = 32;

  // test default enc/dec

  optional float float_default_optional = 2 [(dccl.field).min=-20, (dccl.field).max=150, (dccl.field).precision=3];
}
```

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
4.4 Example messages

See Also

queue1/test.cpp

4.4.3 Test2, Test3, Test4
dccl3/test.proto

import "goby/common/protobuf/option_extensions.proto";
import "dccl/protobuf/option_extensions.proto";
import "goby/test/acoms/dccl3/header.proto";

message GobyMessage
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=10];
  required Header header = 2;
}

protobuf/header.proto

import "goby/common/protobuf/option_extensions.proto";
import "dccl/protobuf/option_extensions.proto";

// required fields will be filled in for you by ApplicationBase
// if you choose not to do so yourself
message Header
{
  //
  // time
  //
  // result of goby::util::as<std::string>(goby_time())
  // e.g. "2002-01-20 23:59:59.000"
  required string time = 10 [(dccl.field).codec="_time",
                             (dccl.field).in_head=true];

  //
  // source
  //
  required string source_platform = 11 [(dccl.field).codec="_platform<->modem_id",
                                        (dccl.field).in_head=true];
  optional string source_app = 12 [(dccl.field).omit=true];

  //
  // destination
  //
  enum PublishDestination { PUBLISH_SELF = 1; PUBLISH_OTHER = 2; PUBLISH_ALL = 3; }
  optional PublishDestination dest_type = 13 [default = PUBLISH_SELF, (dccl.field).in_head=true];
  optional string dest_platform = 14 [(dccl.field).codec="_platform<->modem_id",
                                      (dccl.field).in_head=true]; // required if dest_type == other
}

See Also

queue2/test.cpp
queue3/test.cpp
queue4/test.cpp

4.4.4 Test5

queue5/test.proto
import "dccl/protobuf/option_extensions.proto";

message GobyMessage
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  // one byte
  required int32 telegram = 1 [{(dccl.field).min=0,
                              (dccl.field).max=255};
}

See Also

queue5/test.cpp

5 goby-acomms: amac (Medium Access Control)

Table of Contents for amac:

- Supported MAC schemes
- Interacting with the goby::acomms::MACManager

Return to goby-acomms: An overview of Acoustic Communications Library.

5.1 Supported MAC schemes

The Medium Access Control schemes provided by amac 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. MACManager is implemented as a timer and a std::list of goby::acomms::protobuf::ModemTransmission objects. This allows you to use amac to create a TDMA cycle for any type of transmission (data, ping, LBL, etc.) that your modem supports.

The two variations on this scheme provided by amac are:

1. Decentralized: Each vehicle initiates its own transmission at the start of its slot. (goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED): Slots are set at launch and can be updated using the std::list insert, push, pop, erase, etc. 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. Most of the time you will want to use this mode.

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. This mode requires third-party initiation of transmissions to function.

5.2 Interacting with the goby::acomms::MACManager

To use the goby::acomms::MACManager, you need to instantiate it:

goby::acomms::MACManager mac;

Then you need to provide a callback (or "slot", not to be confused with a TDMA 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 modemdriver, 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 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):

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

You can also provide a set of slots in the protobuf::MACConfig to initialize the MACManager with. Otherwise, you can add them later using the std::list calls.

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

- **goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED**: All vehicles must 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. In this example, I used the std::list push_back instead of adding the slots to the protobuf::MACConfig (see under MAC_POLLED below). Either way, you get the same result, but you can modify the std::list after startup:

```cpp
goby::acomms::protobuf::ModemTransmission 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.push_back(slot); // 1->-1@0 wait 10

slot.set_src(3);
mac.push_back(slot); // 3->-1@0 wait 10

slot.set_rate(5);
mac.push_back(slot); // 3->-1@5 wait 10

slot.set_src(4);
slot.set_rate(0);
mac.push_back(slot); // 4->-1@0 wait 10

mac.update(); // important - call update() after any modifying changes to the MACManager underlying std::list!
```

- **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 topside (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 queue 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:

```cpp
// poll ourselves (for commands, perhaps)
goby::acomms::protobuf::ModemTransmission slot;
slot.set_src(1);
slot.set_dest(goby::acomms::QUERY_DESTINATION_ID);
slot.set_rate(0);
slot.set_type(goby::acomms::protobuf::ModemTransmission::DATA);
slot.set_extension(goby::acomms::protobuf::slot_seconds, 10);
mac_cfg.add_slot(slot); // 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_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`.

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).

You can modify the MAC scheme while MACManager is running. Simply use the `std::list` insert, push, pop, erase methods to changes slots (`goby::acomms::protobuf::ModemTransmission` objects). After any changes that invalidate `std::list` iterators (insert, push, pop, erase), you **must** call `goby::acomms::MACManager::update()` before the next call to `goby::acomms::MACManager::do_work()`.

See `amac_simple.cpp` for a basic complete example.

## 6 goby-acomms: modemdriver (Driver to interact with modem firmware)

Table of contents for `modemdriver`:

- Abstract class: `ModemDriverBase`
- `protobuf` Message `goby::acomms::protobuf::ModemTransmission`
- Writing a new driver
- WHOI Micro-Modem Driver: `MMDriver`

Return to goby-acomms: An overview of Acoustic Communications Library.

### 6.1 Abstract class: `ModemDriverBase`

`goby::acomms::ModemDriverBase` defines the core functionality for an acoustic modem. It provides

- **A serial or serial-like (over TCP) reader/writer.** This is an instantiation of an appropriate derivative of the `goby::util::LineBasedInterface` class which reads the physical interface (serial or TCP) 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::DriverConfig`. 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. 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.

- **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`). The `handle_initiate_transmission` slot is typically bound to `goby::acomms::MACManager::signal_initiate_transmission`.

#### 6.1.1 Interacting with the `goby::acomms::ModemDriverBase`

To use the `goby::acomms::ModemDriverBase`, you need to create one of its implementations such as WHOI Micro-Modem Driver: `MMDriver`.

```cpp
goby::acomms::ModemDriverBase* driver = new goby::acomms::MMDriver;
```
6.2 Protobuf Message goby::acomms::protobuf::ModemTransmission

You will also need to configure the driver. At the very least this involves a serial port, baud, and modem ID (integer MAC address for the modem).

```cpp
goby::acomms::protobuf::DriverConfig cfg;
cfg.set_serial_port("/dev/ttyS0");
cfg.set_modem_id(3);
```

Most modems will have specific other configuration that is required. For example the WHOI Micro-Modem NVRAM is set using three character strings followed by a number. This modem-specific configuration is stored as Protobuf extensions to `goby::acomms::protobuf::DriverConfig`, such as `micromodem::protobuf::Config`. If we were using the WHOI Micro-Modem and wanted to add an NVRAM configuration value we could write

```cpp
cfg.AddExtension(micromodem::protobuf::Config::nvram_cfg, "DQF,1");
```

We need to connect any signals we are interested in. At a minimum this is `goby::acomms::ModemDriverBase::signal_receive`:

```cpp
goby::acomms::connect(&driver->signal_receive, &handle_data_receive);
```

where `handle_data_receive` has the signature:

```cpp
void handle_data_receive(const goby::acomms::protobuf::ModemTransmission& data_msg);
```

Next, we start up the driver with our configuration:

```cpp
driver->startup(cfg);
```

We need to call `goby::acomms::ModemDriverBase::do_work()` on some reasonable frequency (greater than 5 Hz; 10 Hz is probably good). Whenever we need to transmit something, we can either directly call `goby::acomms::ModemDriverBase::handle_initiate_transmission` or connect `goby::acomms::MACManager` to do so for us on some TDMA cycle.

### 6.2 Protobuf Message goby::acomms::protobuf::ModemTransmission

The `goby::acomms::protobuf::ModemTransmission` message is used for all outgoing (sending) and incoming (receiving) messages. The message itself only contains the subset of modem functionality that every modem is expected to support (point-to-point transmission of datagrams).

All other functionality is provided by extensions to `ModemTransmission` such as those in `mm_driver.proto` for the WHOI Micro-Modem. These extensions provide access to additional features of the WHOI Micro-Modem (such as LBL ranging, two-way pings, and comprehensive receive statistics).

By making use of the Protobuf extensions in this way, Goby can both support unique features of a given modem while at that same time remaining general and agnostic to which modem is used when the features are shared (primarily data transfer).

### 6.3 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`. You should check that a driver for your modem does not yet exist before attempting to create your own.

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 accommodate 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/acomms/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();
        void startup(const protobuf::DriverConfig& cfg);
        void shutdown();
        void do_work();
        void handle_initiate_transmission(const protobuf::ModemTransmission& m);
      private:
        protobuf::DriverConfig driver_cfg_; // configuration given to you at launch
        // 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.

```
6.3 Writing a new driver

At startup() you get your configuration from the application (e.g. pAcommsHandler)

```cpp
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);
    glog.is(DEBUG1) && glog << group("modem_out") << "ABCDriver configuration good. Starting modem...", << std::endl;
    ModemDriverBase::modem_start(driver_cfg_);
    // set your local modem id (MAC address)
    {
        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:

```cpp
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 may contain data (in the ModemTransmission::frame field). If not, 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:

```cpp
void goby::acomms::ABCDriver::handle_initiate_transmission(const protobuf::ModemTransmission& orig_msg)
{
    // copy so we can modify
    protobuf::ModemTransmission msg = orig_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).
    glog.is(DEBUG1) && glog << group("modem_out") << "We were asked to transmit from " << msg.src() << " to " << msg.dest() << " at bitrate code " << msg.rate() << std::endl;
    // let's say ABC modem uses 500 byte packet
    msg.set_max_frame_bytes(500);
    // no data given to us, let's ask for some
    if(msg.frame_size() == 0)
        ModemDriverBase::signal_data_request(&msg);
    glog.is(DEBUG1) && glog << group("modem_out") << "Sending these data now:", << msg.frame(0) << 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:" << msg.dest() << ",FROM:" << msg.src() << ",HEX:" << hex_encode(msg.frame(0)) << ",BITRATE:" << BITRATE[msg.rate()] << ",ACK:TRUE" << ",\r\n";
    signal_and_write(raw.str());
} // 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.

```cpp
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);
      // let others know about the raw feed
      protobuf::ModemRaw raw;
      raw.set_raw(in);
      ModemDriverBase::signal_raw_incoming(raw);
      protobuf::ModemTransmission msg;
      msg.set_src(goby::util::as<int32>(parsed["FROM"]));
      msg.set_dest(goby::util::as<int32>(parsed["TO"]));
      msg.set_time(goby::common::goby_time<uint64>());
      glog.is(DEBUG1) && glog << group("modem_in") << in << std::endl;
      if(parsed["KEY"] == "RECV")
      {
        msg.set_type(protobuf::ModemTransmission::DATA);
        msg.add_frame(hex_decode(parsed["HEX"]));
        glog.is(DEBUG1) && glog << group("modem_in") << "received: " << msg << std::endl;
      }
      else if(parsed["KEY"] == "ACKN")
      {
        msg.set_type(protobuf::ModemTransmission::ACK);
      }
      ModemDriverBase::signal_receive(msg);
    }
    catch(std::exception& e)
    {
      glog.is(WARN) && glog << "Bad line: " << in << std::endl;
      glog.is(WARN) && glog << "Exception: " << e.what() << std::endl;
    }
  }
} // do_work
```

- Add your driver header to goby/src/acomms/modem_driver.h
- Modify 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/modemdriver/abc_driver.h and acomms/modemdriver/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 \texttt{goby::acomms::MMDriver} for an example real implementation.
- When a message is sent to \texttt{goby::acomms::BROADCAST\_ID} (0), it should be broadcast if the modem supports such functionality. Otherwise, the driver should throw an \texttt{goby::acomms::ModemDriverException} indicating that it does not support broadcast allowing the user to reconfigure their MAC / addressing scheme.

6.4 WHOI Micro-Modem Driver: MMDriver

6.4.1 Supported Functionality

The \texttt{goby::acomms::MMDriver} extends the \texttt{goby::acomms::ModemDriverBase} for the WHOI Micro-Modem acoustic modem. It is tested to work with revision 0.94.0.00 of the Micro-Modem 1 and revision 2.0.16421 of the Micro-Modem 2 firmware, but is known to work with older firmware (at least 0.92.0.85). It is likely to work properly with newer firmware, and any problems while using newer Micro-Modem firmware should be filed as a bug in Goby. The following features of the WHOI Micro-Modem are implemented, which comprise the majority of the Micro-Modem functionality:

- FSK (rate 0) data transmission
- PSK (rates 1,2,3,4,5) data transmission
- Narrowband transponder LBL ping
- REMUS transponder LBL ping
- User mini-packet 13 bit data transmission
- Two way ping
- Flexible Data Protocol (Micro-Modem 2 only)

See the UML diagrams for a graphical diagram of using Goby for each of these features.

6.4.2 Micro-Modem NMEA to Goby ModemTransmission mapping

Mapping between \texttt{modem\_message.proto} and \texttt{mm\_driver.proto} messages and NMEA fields (see \url{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

<table>
<thead>
<tr>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>$CACYC</strong></td>
</tr>
<tr>
<td>If we did not send $CCCYC, buffer data for $CADRQ by augmenting the provided ModemTransmission and calling signal_data_request:</td>
</tr>
<tr>
<td><code>goby::acomms::protobuf::ModemTransmission.time()</code> = <code>goby::common::goby_time&lt; uint64_t &gt;()</code></td>
</tr>
<tr>
<td><code>goby::acomms::protobuf::ModemTransmission.src()</code> = ADR1</td>
</tr>
<tr>
<td><code>goby::acomms::protobuf::ModemTransmission.dest()</code> = ADR2</td>
</tr>
<tr>
<td><code>goby::acomms::protobuf::ModemTransmission.rate()</code> = Packet Type</td>
</tr>
<tr>
<td><code>goby::acomms::protobuf::ModemTransmission.max_frame_bytes()</code> = 32 for Packet Type == 0, 64 for Packet Type == 2, 256 for Packet Type == 3 or 5</td>
</tr>
<tr>
<td><code>goby::acomms::protobuf::ModemTransmission.max_num_frames()</code> = 1 for Packet Type == 0, 3 for Packet Type == 2, 2 for Packet Type == 3 or 8 for Packet Type == 5</td>
</tr>
</tbody>
</table>

| **$CARXD** |
| only for the first $CARXD for a given packet (should match with the rest though): |
| `goby::acomms::protobuf::ModemTransmission.time()` = `goby::common::goby_time< uint64_t >()` |
| `goby::acomms::protobuf::ModemTransmission.type()` = `goby::acomms::protobuf::ModemTransmission::DATA` |
| `goby::acomms::protobuf::ModemTransmission.src()` = SRC |
| `goby::acomms::protobuf::ModemTransmission.dest()` = DEST |
| `goby::acomms::protobuf::ModemTransmission.ack_requested()` = ACK |
| for each $CARXD: |
| `goby::acomms::protobuf::ModemTransmission.frame(F#-1)` = `goby::util::hex_decode(HH...HH)` |
### $SCAMSG

Used only to detect BAD_CRC frames ($SCAMSG,BAD_CRC...). (in extension micromodem::protobuf::frame_with_bad_crc) micromodem::protobuf::frame_with_bad_crc(n) = Frame with BAD CRC (assumed next frame after last good frame). n is an integer 0,1,2,... indicating the nth reported BAD_CRC frame for this packet. (not the frame number)

### $CAACK

```
goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>()
goby::acomms::protobuf::ModemTransmission.src() = SRC
goby::acomms::protobuf::ModemTransmission.dest() = DEST
goby::acomms::protobuf::ModemTransmission.acked_frame(0) = Frame#-1 (Goby starts counting at frame 0, WHOI starts with frame 1) (second CAACK for a given packet) goby::acomms::protobuf::ModemTransmission.acked_frame(1) = Frame#-1 (third CAACK for a given packet) goby::acomms::protobuf::ModemTransmission.acked_frame(2) = Frame#-1...
```

### $CAMUA

```
goby::acomms::protobuf::ModemTransmission.type() = goby::acomms::protobuf::ModemTransmission::DRIVER_SPECIFIC extension micromodem::protobuf::type = micromodem::protobuf::MICROMODEM_MINI_DATA goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>()
goby::acomms::protobuf::ModemTransmission.src() = SRC
goby::acomms::protobuf::ModemTransmission.dest() = DEST
goby::acomms::protobuf::ModemTransmission.acked_frame(0) = goby::util::hex_decode(HHHH)
```
goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>()
goby::acomms::protobuf::ModemTransmission.dest() = SRC (SRC and DEST flipped to be SRC and DEST of $CCMPC)
goby::acomms::protobuf::ModemTransmission.src() = DEST
goby::acomms::protobuf::ModemTransmission.type() = goby::acomms::protobuf::DRIVER_SPECIFIC
extension micromodem::protobuf::type = micromodem::protobuf::MICROMODEM_TWO_WAY_PING
(in extension micromodem::protobuf::ranging_reply)
micromodem::protobuf::RangingReply.one_way_travel_time(0) = Travel Time

$CAMPA

goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>()
goby::acomms::protobuf::ModemTransmission.src() = SRC
goby::acomms::protobuf::ModemTransmission.dest() = DEST
goby::acomms::protobuf::ModemTransmission.type() = goby::acomms::protobuf::DRIVER_SPECIFIC
extension micromodem::protobuf::type = micromodem::protobuf::MICROMODEM_TWO_WAY_PING

$SNTTA

goby::acomms::protobuf::ModemTransmission.time() = hhmmss.ss (converted to microseconds since 1970-01-01 00:00:00 UTC)
goby::acomms::protobuf::ModemTransmission.time_source() = goby::acomms::protobuf::MODEM_TIME
goby::acomms::protobuf::ModemTransmission.type() = goby::acomms::protobuf::DRIVER_SPECIFIC
extension micromodem::protobuf::type = micromodem::protobuf::MICROMODEM_REMUSLBL_RANGING or
micromodem::protobuf::MICROMODEM_NARROWBLBL_RANGING (depending on which LBL type was last initiated)
goby::acomms::protobuf::ModemTransmission.src() = modem ID
(in extension micromodem::protobuf::ranging_reply)
micromodem::protobuf::RangingReply.one_way_travel_time(0) = TA
micromodem::protobuf::RangingReply.one_way_travel_time(1) = TB
micromodem::protobuf::RangingReply.one_way_travel_time(2) = TC
micromodem::protobuf::RangingReply.one_way_travel_time(3) = TD
| $\text{CAXST} $ | maps onto extension `micromodem::protobuf::transmit_stat` of type `micromodem::protobuf::TransmitStatistics`. The two $\text{CAXST}$ messages (CYC and data) for a rate 0 FH-FSK transmission are grouped and reported at once. |
| $\text{CACST} $ | maps onto extension `micromodem::protobuf::receive_stat` of type `micromodem::protobuf::ReceiveStatistics`. The two $\text{CACST}$ messages for a rate 0 FH-FSK transmission are grouped and reported at once. Note that this message contains the one way time of flight for synchronous ranging (used instead of $\text{CATOA}$). Also sets (which will overwrite `goby_time()` set previously):

```cpp
    goby::acomms::protobuf::ModemTransmission.time() = TOA time (converted to microseconds since 1970-01-01 00:00:00 UTC)
    goby::acomms::protobuf::ModemTransmission.time_source() = goby::acomms::protobuf::MODEM_TIME
```

| $\text{CAREV} $ | Not translated into any of the `modem_message.proto` messages. Monitored to detect excessive clock skew (between Micro-Modem clock and system clock) or reboot (INIT) |
| $\text{CAERR} $ | Not translated into any of the `modem_message.proto` messages. Reported to `goby::glog`. |
| $\text{CACFG} $ | NVRAM setting stored internally. |
| $\text{CACLK} $ | Checked against system clock and if skew is unacceptable another $\text{CCCLK}$ will be sent. |
| $\text{CADRQ} $ | Data request is anticipated from the $\text{CCCYC}$ or $\text{CACYC}$ and buffered. Thus it is not translated into any of the Protobuf messages. |

### Code Snippet

```cpp
$\text{CARDP} $ =
```

```cpp
goby::acomms::protobuf::ModemTransmission.type() =
goby::acomms::protobuf::ModemTransmission::DRI-
VER_SPECIFIC
extension micromodem::protobuf::type =
micromodem::protobuf::MICROMODEM_FLEXIBLE_DATA
goby::acomms::protobuf::ModemTransmission.src() = src
goby::acomms::protobuf::ModemTransmission.dest() = dest
goby::acomms::protobuf::ModemTransmission.rate() = rate
goby::acomms::protobuf::ModemTransmission.-
frame(0) = goby::util::hex_decode(df1+df2+df3...dfN)
where "+" means concatenate, unless any frame fails
the CRC check, in which case this field is set to the
empty string,
micromodem::protobuf::frame_with_bad_crc(0) = 0
indicated that Goby frame 0 is bad, if any sub-frame in
the FDP has a bad CRC
```

### Control Computer to Modem ($\text{CC}$):
$CCTXD$

\[
\begin{align*}
SRC &= \text{goby::acomms::protobuf::ModemTransmission.src()} \\
DEST &= \text{goby::acomms::protobuf::ModemTransmission.dest()} \\
A &= \text{goby::acomms::protobuf::ModemTransmission.ackRequested()} \\
HH...HH &= \text{goby::acomms::hex_encode(goby::acomms::protobuf::ModemTransmission.frame(n))}, \text{ which } n \text{ is an integer } 0,1,2,... \text{ corresponding to the Goby frame that this } $CCTXD \text{ belongs to.}
\end{align*}
\]

$CCCYC$

Augment the ModemTransmission:

\[
\begin{align*}
\text{goby::acomms::protobuf::ModemTransmission.max_frame_bytes()} &= 32 \text{ for Packet Type } = 0, \text{ 64 for Packet Type } = 2, \text{ 256 for Packet Type } = 3 \text{ or } 5 \\
\text{goby::acomms::protobuf::ModemTransmission.max_num_frames()} &= 1 \text{ for Packet Type } = 0, 3 \text{ for Packet Type } = 2, 2 \text{ for Packet Type } = 3 \text{ or } 8 \text{ for Packet Type } = 5
\end{align*}
\]

If ADR1 == modem ID and frame_size() < max_frame_size(), buffer data for later $CADRQ$ by passing the ModemTransmission to signal_data_request

CMD = 0 (deprecated field)

ADR1 = goby::acomms::protobuf::ModemTransmission.src()
ADR2 = goby::acomms::protobuf::ModemTransmission.dest()
Packet Type = goby::acomms::protobuf::ModemTransmission.rate()
ACK = if ADR1 == modem ID then goby::acomms::protobuf::ModemTransmission.ackRequested() else 1
Nframes = goby::acomms::protobuf::ModemTransmission.max_num_frames()
$CCCLK
Not translated from any of the `modem_message.proto` messages. (taken from the system time using the boost::date_time library)

$CCCFG
Not translated from any of the `modem_message.proto` messages. (taken from values passed to the extension `micromodem::protobuf::Config::nvram_cfg` of `goby::acomms::protobuf::DriverConfig`). If the extension `micromodem::protobuf::Config::reset_nvram` is set to true, $CCCFG,ALL,0 will be sent before any other $CCCFG values.)

$CCCFQ
Not translated from any of the `modem_message.proto` messages. $CCCFQ,ALL sent at startup.

$CCMPC
micromodem::protobuf::MICROMODEM_TWO_WAY_PING == extension micromodem::protobuf::type
SRC =
goby::acomms::protobuf::ModemTransmission.src()
DEST =
goby::acomms::protobuf::ModemTransmission.dest()

$CCPDT
micromodem::protobuf::MICROMODEM_R-EMUS_LBL_RANGING == extension
micromodem::protobuf::type
micromodem::protobuf::REMUSLBLParams type
used to determine the parameters of the LBL ping. The object provided with configuration
(micromodem::protobuf::Config::remus_lbl) is merged
with the object provided with the ModemTransmission
(micromodem::protobuf::remus_lbl) with the latter
taking priority on fields that a set in both objects:
GRP = 1
CHANNEL = modem ID % 4 + 1 (use four consecutive
modem IDs if you need multiple vehicles pinging)
SF = 0
STO = 0
Timeout = micromodem::protobuf::REMUSBLLabels::lbl_max_range() m +2/ 1500 m/s + 1000
ms/s + micromodem::protobuf::REMUSBLLabels::turnaround_msi()
goby::acomms::protobuf::ModemRangingRequest::enable_beacons() is a set of four bit flags where the
least significant bit is AF enable, most significant bit is
DF enable. Thus b1111 == 0x0F enables all beacons
AF = micromodem::protobuf::REMUSBLLabels::enable_beacons() >> 0 & 1
BF = micromodem::protobuf::REMUSBLLabels::enable_beacons() >> 1 & 1
CF = micromodem::protobuf::REMUSBLLabels::enable_beacons() >> 2 & 1
DF = micromodem::protobuf::REMUSBLLabels::enable_beacons() >> 3 & 1
micromodem::protobuf::MICROMODEM_NARROWBANDLBL_RANGING == extension
micromodem::protobuf::type
micromodem::protobuf::NarrowBandLBLParams type used to determine the parameters of the LBL ping.
The object provided with configuration
(micromodem::protobuf::Config::narrowband_lbl) is merged with the object provided with the
ModemTransmission
(micromodem::protobuf::narrowband_lbl) with the latter taking priority on fields that a set in both objects:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ftx</td>
<td>micromodem::protobuf::NarrowBandLBLParams::transmit_freq()</td>
</tr>
<tr>
<td>Ttx</td>
<td>micromodem::protobuf::NarrowBandLBLParams::transmit_ping_ms()</td>
</tr>
<tr>
<td>Trx</td>
<td>micromodem::protobuf::NarrowBandLBLParams::receive_ping_ms()</td>
</tr>
</tbody>
</table>
| Timeout     | micromodem::protobuf::NarrowBandLBLParams::lbl_max_range() m * 2/ 1500 m/s * 1000 ms/s +
micromodem::protobuf::NarrowBandLBLParams::turnaround_ms() |
| FA          | micromodem::protobuf::NarrowBandLBLParams::receive_freq(0) or 0 if receive_freq_size() < 1     |
| FB          | micromodem::protobuf::NarrowBandLBLParams::receive_freq(1) or 0 if receive_freq_size() < 2     |
| FC          | micromodem::protobuf::NarrowBandLBLParams::receive_freq(2) or 0 if receive_freq_size() < 3     |
| FD          | micromodem::protobuf::NarrowBandLBLParams::receive_freq(3) or 0 if receive_freq_size() < 4     |
| Tflag       | micromodem::protobuf::NarrowBandLBLParams::transmit_flag()                                     |
| $\text{CCMUC}$ | SRC =
goby::acomms::protobuf::ModemTransmission.src()
DEST =
goby::acomms::protobuf::ModemTransmission.dest()
HHHH = goby::acomms::hex_encode(goby::acomms::protobuf::ModemTransmission.frame(0)) & 0x1F |
| $\text{CCTDP}$ | dest =
goby::acomms::protobuf::ModemTransmission.dest()
rate =
goby::acomms::protobuf::ModemTransmission.rate()
ack = 0 (not yet supported by the Micro-Modem 2)
reserved = 0
hexdata = goby::acomms::hex_encode(goby::acomms::protobuf::ModemTransmission.frame(0)) |

### 6.4.3 Sequence diagrams for various Micro-Modem features using Goby

FSK (rate 0) data transmission
Figure 5: FSK (rate 0) data transmission

PSK (rate 2 shown, others are similar) data transmission
Figure 6: PSK (rate 2 shown, others are similar) data transmission

Narrowband transponder LBL ping

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
Figure 7: Narrowband transponder LBL ping

REMUS transponder LBL ping
User mini-packet 13 bit data transmission

Figure 8: REMUS transponder LBL ping
Figure 9: User mini-packet 13 bit data transmission
Figure 10: Two way ping

Flexible Data Protocol (Micro-Modem 2)
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.

7.2  Logging

Because Goby is designed first and foremost as an engineering testbed and scientific research architecture, comprehensive logging is a crucial component of the software architecture. The \link acomms_api goby-acomms\endlink API classes all have a constructor which can take a pointer to std::ostream:

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

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

![Figure 11: Flexible Data Protocol](image-url)
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:

```cpp
// 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:
Figure 12: Example of the goby::util::glogger() output at different verbosity settings to the terminal window

Graphical user interface logger mode:

Figure 13: Example of the goby::util::glogger() 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

libutil_linebasedcomms provides a common interface (goby::util::LineBasedInterface) for line-based (defined as blocks of text) data sent over a TCP or serial connection. liblinebasedcomms uses 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 goby-moos: An overview of the Goby/MOOS interoperability library

Table of Contents for goby-moos: An overview of the Goby/MOOS interoperability library.

- iFrontSeat
  - Writing a new driver for iFrontSeat
    - Overview
    - State charts
    - Example "ABC" driver

8.1 iFrontSeat

iFrontSeat is a MOOS application used to interface a Goby/MOOS community (the "backseat") running pHelmIP with a given manufacturer's vehicle (the "frontseat"). The usage of iFrontSeat and the existing driver suite is explained in the Goby user manual (see Resources).

8.1.1 Writing a new driver for iFrontSeat

8.1.1.1 Overview

iFrontSeat is intended to interface to a wide range of vehicles using any interface (e.g. proprietary extensions of NMEA-0183). The purpose of the driver is to implement the Goby FrontSeatInterfaceBase in the language of the particular frontseat vehicle system. Minimally, these are the requirements of the frontseat:

- it can provide a navigation solution for the vehicle (minimally latitude, longitude, depth, and speed), and typically also the geo-referenced pose of the vehicle (heading, pitch, yaw).
- it can accept a desired heading, speed, and depth (at around 1 Hz) for the vehicle and carry out these commands as quickly as reasonably possible given the vehicle's dynamics.

Additionally, the frontseat may provide or consume:

- arbitrary sensor data (e.g. CTD samples, acoustic modem datagrams)
- additional special commands (e.g. buoyancy adjustment, activate/deactivate sensors, low power mode) that the backseat can command of the frontseat.
8.1.1.2 State charts

The state of iFrontSeat (as shown in the following diagram) is determined by a combination of the state of the frontseat and the state of pHelmIPvP. Only the state of the frontseat must be determined by each new driver, as the state of pHelmIPvP is determined by code shared by all the drivers.

![State charts of the iFrontSeat interface and connected ends (pHelmIPvP and frontseat)](image)

The state of the frontseat consists of two parallel state charts (command and data):

- **Command state**
  - FRONTSEAT_IDLE (required): The frontseat computer is alive and well, but is not running any mission (the vehicle is a standby mode).
  - FRONTSEAT_ACCEPTING_COMMANDS (required): The frontseat is accepting the backseat commands.
  - FRONTSEAT_NOT_CONNECTED (optional): No communication with the frontseat computer has been established (or a connection has been lost). If there is no way to tell whether the frontseat is alive at any given time, this state may not be implemented.
  - FRONTSEAT_IN_CONTROL (optional): The frontseat is running a mission and driving the vehicle but not accepting commands from the backseat. If the frontseat never runs missions without backseat control, this state may not be implemented.

- **Data state (not diagrammed above)**
  - frontseat_providing_data == true: The frontseat is providing all required data. What is required is determined by the specific driver, but at a minimum is the navigation solution.
  - frontseat_providing_data == false: The frontseat is not providing all required data.

The state transitions for the iFrontSeat interface states are (using the names as defined in the enumerations in moos/protobuf/frontseat.proto)
8.1 iFrontSeat

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>INTERFACE_STANDBY</td>
<td>Configuration Read</td>
</tr>
<tr>
<td>INTERFACE_STANDBY</td>
<td>INTERFACE_LISTEN</td>
<td>frontseat_providing_data == true</td>
</tr>
<tr>
<td>INTERFACE_LISTEN</td>
<td>INTERFACE_COMMAND</td>
<td>FRONTSEAT_ACCEPTING_COMMANDS &amp;&amp; HELM_DRIVE</td>
</tr>
<tr>
<td>INTERFACE_COMMAND</td>
<td>INTERFACE_LISTEN</td>
<td>(FRONTSEAT_IN_CONTROL</td>
</tr>
<tr>
<td>INTERFACE_COMMAND</td>
<td>INTERFACE_HELM_ERROR</td>
<td>HELM_NOT_RUNNING</td>
</tr>
<tr>
<td>INTERFACE_LISTEN</td>
<td></td>
<td>INTERFACE_COMMAND</td>
</tr>
<tr>
<td>INTERFACE_LISTEN</td>
<td></td>
<td>INTERFACE_COMMAND</td>
</tr>
<tr>
<td>INTERFACE_STANDBY</td>
<td>INTERFACE_FS_ERROR</td>
<td>FRONTSEAT_NOT_CONNECTED (after timeout)</td>
</tr>
<tr>
<td>INTERFACE_HELM_ERROR</td>
<td>INTERFACE_STANDBY</td>
<td>HELM_DRIVE</td>
</tr>
<tr>
<td>INTERFACE_FRONTSEAT_ERROR</td>
<td>INTERFACE_STANDBY</td>
<td>(if(ERROR_FRONTSEAT_NOT_CONNECTED) !FRONTSEAT_NOT_CONNECTED</td>
</tr>
</tbody>
</table>

8.1.1.3 Example "ABC" driver

We will show you how to a write a new driver by example. To do so, we have created a simple frontseat simulator ("abc_frontseat_simulator") that is intended to represent the real vehicle frontseat control system. The full source code for this example is given at:

- examples/moos/abc_frontseat_driver/abc_frontseat_driver.h
- examples/moos/abc_frontseat_driver/abc_frontseat_driver.cpp
- examples/moos/abc_frontseat_driver/abc_frontseat_driver_config.proto

A complete production driver is provided by BluefinFrontSeat for the Bluefin Robotics AUVs that conform to the Bluefin Standard Payload Interface version 1.8 and newer.

The transport for the ABC frontseat is TCP: the simulator (frontseat) listens on a given port and the driver connects to that machine and port. The wire protocol is a simple ascii line-based protocol where lines are terminated by carriage-return and newline (\r\n or "\r\n"). Each message has a name (key), followed by a number of comma-delimited, colon-separated fields:
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Direction (relative to frontseat)</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>Simulator initialization message</td>
<td>Receive</td>
<td>START,LAT:{latitude decimal degrees},LON:{longitude decimal degrees},DURATION:{simulation duration seconds}</td>
<td>START,LAT:42.1234,LON:-72,DURATION:600</td>
</tr>
<tr>
<td>CTRL</td>
<td>Frontseat state message</td>
<td>Transmit</td>
<td>CTRL,STATE:{PAYLOAD (if backseat control) or IDLE}</td>
<td>CTRL,STATE:PAYLOAD</td>
</tr>
<tr>
<td>NAV</td>
<td>Navigation message generated from very primitive dynamics model (depth &amp; heading changes are instantaneous)</td>
<td>Transmit</td>
<td>NAV,LAT:{latitude decimal degrees},LON:{longitude decimal degrees},DEPTH:{depth in meters},HEADING:{heading in degrees},SPEED:{speed in m/s}</td>
<td>NAV,LAT:42.1234,LON:-72.5435,DEPTH:200,HEADING:223,SPEED:1.4</td>
</tr>
<tr>
<td>CMD</td>
<td>Desired course command from backseat</td>
<td>Receive</td>
<td>CMD,HEADING:{desired heading in degrees},SPEED:{desired speed in m/s},DEPTH:{desired depth in m}</td>
<td>CMD,HEADING:260,SPEED:1.5,DEPTH:100</td>
</tr>
<tr>
<td>CMD</td>
<td>Reponse to last CMD</td>
<td>Transmit</td>
<td>CMD,RESULT:{OK or ERROR}</td>
<td>CMD,RESULT:OK</td>
</tr>
</tbody>
</table>

Your driver will be (at a minimum) a C linkage function "frontseat_driver_load" and a subclass of FrontSeatInterfaceBase. It should be compiled into a shared library (.so on Linux).

The C function is used by iFrontSeat to load your driver:

```c
extern "C" {
  FrontSeatInterfaceBase* frontseat_driver_load(iFrontSeatConfig* cfg) {
    return new AbcFrontSeat(*cfg);
  }
}
```

First you should decide what configuration your driver will accept. Your configuration object is an extension to the Google Protobuf message "iFrontSeatConfig". For the ABC frontseat driver, we use the abc_frontseat_driver.proto file to define the configuration:

```protobuf
import "goby/moos/protobuf/frontseat_config.proto";
import "goby/common/protobuf/option_extensions.proto";

message ABCFrontSeatConfig {
  required string tcp_address = 1;
  optional uint32 tcp_port = 2 [default = 54321];
  message StartParams {
    required double lat = 1;
    required double lon = 2;
  }
}
```

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
In this case, we need to know what IP address and TCP port the abc_frontseat_simulator is listening on, and the starting position of the simulator.

Next, you should fill out the virtual methods of FrontSeatInterfaceBase:

- The method "frontseat_state" reports the driver’s belief of the frontseat command state (see State charts).

```cpp
goby::moos::protobuf::FrontSeatState AbcFrontSeat::frontseat_state() const
{
    return frontseat_state_; // frontseat_state
}
```

In this case, we set the value of frontseat_status_ based on the received "CTRL" messages:

```cpp
if(parsed[*KEY*] == "CTRL")
{
    if(parsed[*STATE*] == "PAYLOAD")
        frontseat_state_ = gpb::FRONTSEAT_ACCEPTING_COMMANDS;
    else if{parsed[*STATE*] == "AUV")
        frontseat_state_ = gpb::FRONTSEAT_IN_CONTROL;
    else
        frontseat_state_ = gpb::FRONTSEAT_IDLE;
}
```

- The method "frontseat_providing_data" reports the frontseat's data state (see State charts). It must return true if the frontseat is providing data to the driver reasonably often (where reasonable is defined by the driver). Here we set the class member variable "frontseat_providing_data_" to true each time we get a "NAV" message, and then false if we have had no "NAV" messages in the last 10 seconds.

```cpp
bool AbcFrontSeat::frontseat_providing_data() const
{
    return frontseat_providing_data_; // frontseat_providing_data
}
```

- The method "send_command_to_frontseat" is called whenever iFrontSeat needs to send a command to the frontseat. This command typically contains a desired heading, speed, and depth, but could alternatively contain a special command defined via an extension to the goby::moos::protobuf::CommandRequest message.

```cpp
void AbcFrontSeat::send_command_to_frontseat(const gpb::CommandRequest& command)
{
    if(command.has_desired_course())
    {
        std::stringstream cmd_ss;
        const goby::moos::protobuf::DesiredCourse& desired_course = command.desired_course();
        cmd_ss << "CMD,"
            "HEADING:" << desired_course.heading() << "," 
            "SPEED:" << desired_course.speed() << "," 
            "DEPTH:" << desired_course.depth();
        write(cmd_ss.str());
        last_request_ = command;
    }
    else
    {
        glog.is(VERBOSE) && glog << "Unhandled command: " << command.ShortDebugString() << std::endl;
    }
} // send_command_to_frontseat
```

- The method "send_data_to_frontseat" is called whenever iFrontSeat needs to send data to the frontseat. These data could include sensor readings from instruments that are directly connected to the backseat, such as a CTD or acoustic modem. Our bare-bones example frontseat doesn’t require any data from the backseat, so we just leave an empty implementation here.
void AbcFrontSeat::send_data_to_frontseat(const gpb::FrontSeatInterfaceData& data)
  {
    // ABC driver doesn’t have any data to sent to the frontseat
    // send_data_to_frontseat
  }

– The method "send_raw_to_frontseat" is called whenever an external application wants to directly control
the frontseat. This can be left blank (or post a warning to the glog) if there is no need (or desire) to allow
for direct control of the frontseat from external applications.

– The method "loop" is called regularly (at the AppTick of iFrontSeat) and is where you can read data from
the frontseat and do other regular work.

void AbcFrontSeat::loop()
  {
    check_connection_state();
    try_receive();
    // if we haven’t gotten data for a while, set this boolean so that the
    // FrontSeatInterfaceBase class knows
    if(goby_time<double>() > last_frontseat_data_time_ + allowed_skew)
      frontseat_providing_data_ = false;
  }

Now, the final task is to call the appropriate signals in FrontSeatInterfaceBase upon receipt of data and
responses to commands. The signals are called just like normal functions with the corresponding signatures.
These signals (except signal_raw_to_frontseat) are typically called in response to data received in the loop() method.

– signal_data_from_frontseat: Call when a navigation solution is received from the frontseat. This may
have to be merged from several messages, which is why goby::moos::protobuf::NodeStatus has the *
_time_lag fields. These fields can be used to indicate the offset of certain fields from the timestamp on
the message. You can use the FrontSeatInterfaceBase::compute_missing to compute the local fix (X,
Y, Z) from the global fix (latitude, longitude, depth) or vice-versa.

– signal_command_response: Call when the frontseat acknowledges a command, if the command re-
quest includes response_requested == true. Include the success or failure of the command, and an
error code (with description) if applicable.

– signal_raw_from_frontseat: Call when a raw message (e.g. "CMD,RESULT:OK") is received from the
frontseat. This is for logging and debug purposes.

– signal_raw_to_frontseat: Call when a raw message (e.g. "CMD,HEADING:260,SPEED:1.5,DEPTH-
:100") is send to the frontseat. This is for logging and debug purposes.

For testing the ABC driver to see how it functions, you will need to run

abc_frontseat_simulator 54321

where 54321 is the port for the simulator to listen on.

Then, run iFrontSeat in a MOOS community with pHelmIvP with the following configuration:

ProcessConfig = iFrontSeat_bluefin
{
  common {
    verbosity: DEBUG1
  }
  [abc_config] { # (optional)
    tcp_address: "localhost" # (required)
    tcp_port: 54321 # (optional) (default=54321)
    start { lat: 44.0888889 lon: 9.84861111 duration: 600 }
  }
}

You can change the start position as desired.
9 Module Index

9.1 Modules

Here is a list of all modules:

- API classes for the Dynamic Compact Control Language (includes writing custom encoders). 78
- API classes for the major components of the Goby-Acomms acoustic communications library (D-CCL, Queue, AMAC, ModemDriver). 78

10 Namespace Index

10.1 Namespace List

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

- **goby**
  The global namespace for the Goby project 78

- **goby::acomms**
  Objects pertaining to acoustic communications (acomms) 80

- **goby::common**
  Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc 83

- **goby::common::tcolor**
  Contains functions for adding color to Terminal window streams 86

- **goby::pb**
  Contains objects relating to the core publish / subscribe architecture provided by Goby 87

- **goby::transitional**
  Objects pertaining to transitioning from DCCLv1 to DCCLv2 87

11 Hierarchical Index

11.1 Class Hierarchy

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

- **boost::asio::time_traits< goby::common::GobyTime >** 92
- **ChatCurses** 93
- **goby::acomms::ModemDriverBase** 100
  - **goby::acomms::ABCDriver** 94
  - **goby::acomms::MMDriver** 98
- **goby::moos::BluefinCommsDriver** 115
- **goby::moos::UFldDriver** 117
12 Class Index

12.1 Class List

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

- **boost::asio::time_traits</goby::common::GobyTime>**
  Time traits specialised for GobyTime

- **ChatCurses**
  Terminal GUI for a chat window (lower box to type and upper box to receive messages). Part of the chat.cpp example

- **goby::acomms::ABCDriver**
  API to the imaginary ABC modem (as an example how to write drivers)
goby::acomms::MACManager
API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer 96

goby::acomms::MMDriver
API to the WHOI Micro-Modem driver 98

goby::acomms::ModemDriverBase
Abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems 100

goby::acomms::QueueException
Exception class for libdcl 104

goby::acomms::QueueManager
API to the goby-acomms Queuing Library 105

goby::common::Colors
Represents the eight available terminal colors (and bold variants) 110

goby::common::ConfigException
Indicates a problem with the runtime command line or .cfg file configuration (or –help was given) 111

goby::common::FlexNCurses
Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content 112

goby::common::FlexOstream
Forms the basis of the Goby logger: std::ostream derived class for holding the FlexOStreamBuf 112

goby::common::TermColor
Converts between string, escape code, and enumeration representations of the terminal colors 114

goby::Exception
Simple exception class for goby applications 114

goby::moos::BluefinCommsDriver
Driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hardware) 115

goby::moos::UFldDriver

goby::pb::Application
Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture 118

goby::transitional::DCCLMessageVal
Defines a DCCL value 120

goby::transitional::DCCLTransitionalCodec
API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly 125

goby::util::LineBasedInterface
Basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.) 127
goby::util::SerialClient
Basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control 128

goby::util::TCPClient
Basic TCP client for line by line text based communications to a remote TCP server 129

goby::util::TCPServer
Basic TCP server for line by line text based communications to a one or more remote TCP clients 130

Group
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 132

GroupSetter
Helper class for enabling the group(std::string) manipulator 133

13 File Index
13.1 File List
Here is a list of all documented files with brief descriptions:

goby/acoms.h ??
goby/util.h ??
goby/version.h ??
goby/acoms/acoms_constants.h ??
goby/acoms/acoms_helpers.h ??
goby/acoms/amac.h ??
goby/acoms/bind.h ??
goby/acoms/connect.h ??
goby/acoms/dccl.h ??
goby/acoms/ip_codecs.h ??
goby/acoms/modem_driver.h ??
goby/acoms/queue.h ??
goby/acoms/route.h ??
goby/acoms/amac/mac_manager.h ??
goby/acoms/dccl/dccl.h ??
goby/acoms/modemdriver/abc_driver.h ??
goby/acoms/modemdriver/driver_base.h ??
goby/acoms/modemdriver/driver_exception.h ??
goby/acomms/protobuf/manipulator.proto
**
goby/acomms/protobuf/mm_driver.pb.h
**
goby/acomms/protobuf/mm_driver.proto
**
goby/acomms/protobuf/modem_driver_status.pb.h
**
goby/acomms/protobuf/modem_driver_status.proto
**
goby/acomms/protobuf/modem_message.pb.h
**
goby/acomms/protobuf/modem_message.proto
**
goby/acomms/protobuf/mosh_packet.pb.h
**
goby/acomms/protobuf/mosh_packet.proto
**
goby/acomms/protobuf/network_ack.pb.h
**
goby/acomms/protobuf/network_ack.proto
**
goby/acomms/protobuf/network_header.pb.h
**
goby/acomms/protobuf/network_header.proto
**
goby/acomms/protobuf/queue.pb.h
**
goby/acomms/protobuf/queue.proto
**
goby/acomms/protobuf/route.pb.h
**
goby/acomms/protobuf/route.proto
**
goby/acomms/protobuf/rudics_shore.pb.h
**
goby/acomms/protobuf/rudics_shore.proto
**
goby/acomms/protobuf/store_server.pb.h
**
goby/acomms/protobuf/store_server.proto
**
goby/acomms/protobuf/time_update.pb.h
**
goby/acomms/protobuf/time_update.proto
**
goby/acomms/protobuf/udp_driver.pb.h
**
goby/acomms/protobuf/udp_driver.proto
**
goby/acomms/queue/queue.h
**
goby/acomms/queue/queue_constants.h
**
goby/acomms/queue/queue_exception.h
**
goby/acomms/queue/queue_manager.h
**
goby/acomms/route/route.h
**
goby/apps/acomms/goby_bridge/bridge_config.pb.h
**
goby/apps/acomms/goby_bridge/bridge_config.proto
**
<table>
<thead>
<tr>
<th>File Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>goby/apps/acomms/goby_file_transfer/file_transfer_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_file_transfer/file_transfer_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_ip_gateway/ip_gateway_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_ip_gateway/ip_gateway_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_modemdriver/modemdriver_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_modemdriver/modemdriver_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_mosh_relay/mosh_relay_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_mosh_relay/mosh_relay_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_store_server/goby_store_server_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/acomms/goby_store_server/goby_store_server_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/moos/moos_gateway_g/moos_gateway_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/moos/moos_gateway_g/moos_gateway_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/moos/pGobyMOOSAppTemplate/pGobyMOOSAppTemplate_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/moos/pTranslator/pTranslator_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/apps/moos/pTranslator/pTranslator_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/application_base.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/configuration_reader.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/core_constants.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/core_helpers.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/exception.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/hdf5_plugin.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/liaison_container.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/logger.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/node_interface.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/pubsub_nodeWrapper.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/time.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/zeromq_application_base.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/zeromq_packet.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/zeromq_service.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/logger/flex_ncurses.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/logger/flex_ostream.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/common/logger/flex_ostreambuf.h</td>
<td>??</td>
</tr>
</tbody>
</table>
goby/common/logger/logger_manipulators.h ??
goby/common/logger/term_color.h ??
goby/common/protobuf/app_base_config.pb.h ??
goby/common/protobuf/app_base_config.proto ??
goby/common/protobuf/hdf5.pb.h ??
goby/common/protobuf/hdf5.proto ??
goby/common/protobuf/liaison_config.pb.h ??
goby/common/protobuf/liaison_config.proto ??
goby/common/protobuf/logger.pb.h ??
goby/common/protobuf/logger.proto ??
goby/common/protobuf/option_extensions.pb.h ??
goby/common/protobuf/option_extensions.proto ??
goby/common/protobuf/pubsub_node_config.pb.h ??
goby/common/protobuf/pubsub_node_config.proto ??
goby/common/protobuf/zero_mq_node_config.pb.h ??
goby/common/protobuf/zero_mq_node_config.proto ??
goby/moos/dynamic_moos_vars.h ??
goby/moos/goby_moos_app.h ??
goby/moos/liaison_acomms.h ??
goby/moos/liaison_commander.h ??
goby/moos/liaison_geodesy.h ??
goby/moos/liaison_scope.h ??
goby/moos/modem_id_convert.h ??
goby/moos/moos_bluefin_driver.h ??
goby/moos/moos_geodesy.h ??
goby/moos/moos_header.h ??
goby/moos/moos_liaison_load.h ??
goby/moos/moos_node.h ??
goby/moos/moos_protobuf_helpers.h
   Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages 133

   goby/moos/moos_serializer.h ??
goby/moos/moos_string.h ??
goby/moos/moos_translator.h ??
goby/moos/moos_ufield_sim_driver.h ??
goby/moos/frontseat/frontseat.h ??
goby/moos/frontseat/frontseat_exception.h ??
goby/moos/frontseat/bluefin/bluefin.h ??
goby/moos/frontseat/bluefin/bluefin.pb.h ??
goby/moos/frontseat/bluefin/bluefin.proto ??
goby/moos/frontseat/bluefin/bluefin_config.pb.h ??
goby/moos/frontseat/bluefin/bluefin_config.proto ??
goby/moos/protobuf/bluefin_driver.pb.h ??
goby/moos/protobuf/bluefin_driver.proto ??
goby/moos/protobuf/ctd_sample.pb.h ??
goby/moos/protobuf/ctd_sample.proto ??
goby/moos/protobuf/desired_course.pb.h ??
goby/moos/protobuf/desired_course.proto ??
goby/moos/protobuf/frontseat.pb.h ??
goby/moos/protobuf/frontseat.proto ??
goby/moos/protobuf/frontseat_config.pb.h ??
goby/moos/protobuf/frontseat_config.proto ??
goby/moos/protobuf/goby_moos_app.pb.h ??
goby/moos/protobuf/goby_moos_app.proto ??
goby/moos/protobuf/liaison_config.pb.h ??
goby/moos/protobuf/liaison_config.proto ??
goby/moos/protobuf/modem_id_lookup.pb.h ??
goby/moos/protobuf/modem_id_lookup.proto ??
goby/moos/protobuf/node_status.pb.h ??
goby/moos/protobuf/node_status.proto ??
goby/moos/protobuf/pAcommsHandler_config.pb.h ??
goby/moos/protobuf/pAcommsHandler_config.proto ??
goby/moos/protobuf/transitional.pb.h ??
goby/moos/protobuf/transitional.proto ??
goby/moos/protobuf/translator.pb.h ??
<table>
<thead>
<tr>
<th>File Path</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>goby/pb/protobuf/database_request.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/pb/protobuf/header.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/pb/protobuf/header.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/pb/protobuf/interprocess_notification.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/pb/protobuf/interprocess_notification.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/pb/protobuf/pb_modem_driver.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/pb/protobuf/pb_modem_driver.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/share/examples/acomms/chat/chat.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/share/examples/acomms/dccl1/dccl_simple/simple.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/share/examples/acomms/dccl1/two_message/two_message.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/share/examples/acomms/queue/encode_on_demand/on_demand.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/share/examples/moos/abc_frontseat_driver/abc_frontseat_driver_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/share/examples/moos/gobyexample_protobuf/simple_status.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc1/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc1/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc10/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc10/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc2/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc2/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc3/header.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc3/header.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc3/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc3/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc4/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc4/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc6/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc6/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc7/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc7/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc8/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc8/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>Path</td>
<td>Status</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>goby/test/acomms/dcc19/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/dcc19/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/mmdriver2/test_config.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/mmdriver2/test_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/queue1/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/queue1/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/queue5/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/queue5/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/queue6/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/queue6/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/route1/test.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/acomms/route1/test.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/common/hdf5/test2.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/common/hdf5/test2.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/moos/translator1/basic_node_report.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/moos/translator1/basic_node_report.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/util/dynamic_protobuf/test_a.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/util/dynamic_protobuf/test_a.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/util/dynamic_protobuf/test_b.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/test/util/dynamic_protobuf/test_b.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/as.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/base_convert.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/binary.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/dynamic_protobuf_manager.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/linebasedcomms.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/primitive_types.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/sci.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/linebasedcomms/client_base.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/linebasedcomms/connection.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/linebasedcomms/interface.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/linebasedcomms/nmea_sentence.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/linebasedcomms/serial_client.h</td>
<td>??</td>
</tr>
<tr>
<td>File Path</td>
<td>Status</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>goby/util/linebasedcomms/tcp_client.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/linebasedcomms/tcp_server.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/protobuf/linebasedcomms.pb.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/protobuf/linebasedcomms.proto</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/seawater/depth.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/seawater/pressure.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/seawater/salinity.h</td>
<td>??</td>
</tr>
<tr>
<td>goby/util/seawater/swstate.h</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/amac/amac_simple/amac_simple.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/chat/chat.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/chat/chat.proto</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/chat/chat_curses.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/chat/chat_curses.h</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/dccl/dccl_simple/dccl_simple.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/dccl/dccl_simple/simple.proto</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/dccl/two_message/two_message.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/dccl/two_message/two_message.proto</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/modemdriver/driver_simple/driver_simple.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/modemdriver/whoi_ranging/whoi_ranging.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/queue/encode_on_demand/encode_on_demand.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/queue/multimessage/multimessage.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/queue/multimessage/multimessage.proto</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/acomms/queue/multimessage/simple/multimessage.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.cpp</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.h</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.proto</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/moos/abc_frontseat_driver/abc_frontseat_driver_config.proto</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/moos/gobyexample_protobuf/simple_status.proto</td>
<td>??</td>
</tr>
<tr>
<td>share/examples/util/flexostream_simple/flexostream_simple.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/acomms/ip_codecs.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/acomms/amac/mac_manager.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/acomms/dccl/dccl.cpp</td>
<td>??</td>
</tr>
<tr>
<td>Source Path</td>
<td>Status</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>src/apps/util/serial2tcp_server/serial2tcp_server.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/application_base.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/configuration_reader.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/time.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/zeromq_service.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/logger/flex_ncurses.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/logger/flex_ostream.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/logger/flex_ostreambuf.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/logger/logger_manipulators.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/common/logger/term_color.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/goby_moos_app.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/liaison_acomms.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/liaison_commander.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/liaison_geodesy.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/liaison_scope.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/modem_id_convert.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/moos_bluefin_driver.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/moos_geodesy.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/moos_liaison_load.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/moos_node.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/moos_protobuf_helpers.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/moos_translator.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/moos_ufield_sim_driver.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/frontseat/frontseat.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/frontseat/bluefin/bluefin.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/frontseat/bluefin/bluefin_incoming.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/transitional/dccl_transitional.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/transitional/message.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/transitional/message_algorithms.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/transitional/message_publish.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/transitional/message_val.cpp</td>
<td>??</td>
</tr>
<tr>
<td>src/moos/transitional/message_var.cpp</td>
<td>??</td>
</tr>
</tbody>
</table>
src/moos/transitional/message_var_float.cpp
src/moos/transitional/message_xml_callbacks.cpp
src/moos/transitional/queue_xml_callbacks.cpp
src/pb/application.cpp
src/pb/pb_modem_driver.cpp
src/pb/protobuf_node.cpp
src/share/doc/style_example.cpp
src/share/examples/acomms/amac/amac_simple/amac_simple.cpp
src/share/examples/acomms/chat/chat.cpp
src/share/examples/acomms/chat/chat_curses.cpp
src/share/examples/acomms/dcc/dcccl_simple/dcccl_simple.cpp
src/share/examples/acomms/dcc/dcccl_two_message/dcccl_two_message.cpp
src/share/examples/acomms/modemdriver/driver_simple/driver_simple.cpp
src/share/examples/acomms/modemdriver/whoi_ranging/whoi_ranging.cpp
src/share/examples/acomms/queue/encode_on_demand/encode_on_demand.cpp
src/share/examples/acomms/queue/multimessage/multimessage.cpp
src/share/examples/acomms/queue/queue_simple/queue_simple.cpp
src/share/examples/acomms/queue/queue_simple/queue_simple.cpp
src/share/examples/acomms/queue/queue_simple/queue_simple.cpp
src/share/examples/acomms/queue/queue_simple/queue_simple.cpp
src/share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.cpp
src/share/examples/util/flexostream_simple/flexostream_simple.cpp
src/test/acomms/amac1/test.cpp
src/test/acomms/dcc1/test.cpp
src/test/acomms/dcc10/test.cpp
src/test/acomms/dcc2/test.cpp
src/test/acomms/dcc3/test.cpp
src/test/acomms/dcc4/test.cpp
src/test/acomms/dcc6/test.cpp
src/test/acomms/dcc7/test.cpp
src/test/acomms/dcc8/test.cpp
src/test/acomms/dcc9/test.cpp
src/test/acomms/driver_tester/driver_tester.cpp
src/test/acomms/ipcodecs/test.cpp
src/test/acomms/iridiunmdriver1/test.cpp
13.1 File List

src/test/acomms/mmdriver1/test.cpp
src/test/acomms/mmdriver2/test.cpp
src/test/acomms/queue1/test.cpp
src/test/acomms/queue2/test.cpp
src/test/acomms/queue3/test.cpp
src/test/acomms/queue4/test.cpp
src/test/acomms/queue5/test.cpp
src/test/acomms/queue6/test.cpp
src/test/acomms/route1/test.cpp
src/test/acomms/udpdriver1/test.cpp
src/test/acomms/udpdriver2/test.cpp
src/test/acomms/udpdriver3/test.cpp
src/test/common/hdf5/test-plugin.cpp
src/test/common/hdf5/test.cpp
src/test/common/log/test.cpp
src/test/common/zero_mq_node1/test.cpp
src/test/common/zero_mq_node2/test.cpp
src/test/common/zero_mq_node3/test.cpp
src/test/common/zero_mq_node4/test.cpp
src/test/moos/transitional1/test.cpp
src/test/moos/translator1/test.cpp
src/test/pb/pbdriver1/test.cpp
src/test/util/as/as.cpp
src/test/util/base255/base255.cpp
src/test/util/dynamic_protobuf/dynamic_protobuf.cpp
src/test/util/hex_codec/hex_codec.cpp
src/test/util/nmea/nmea.cpp
src/test/util/salinity/salinity.cpp
src/test/util/sci/sci.cpp
src/test/util/time/time.cpp
src/util/linebasedcomms/interface.cpp
src/util/linebasedcomms/nmea_sentence.cpp
14 Module Documentation

14.1 API classes for the Dynamic Compact Control Language (includes writing custom encoders).

14.2 API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver).

Classes

- `class goby::acomms::ABCDriver`: provides an API to the imaginary ABC modem (as an example how to write drivers)
- `class goby::moos::BluefinCommsDriver`: provides a driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hardware)
- `class goby::acomms::MACManager`: provides an API to the goby-acomms MAC library. `MACManager` is essentially a `std::list<protobuf::ModemTransmission>` plus a timer.
- `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.
- `class goby::transitional::DCCLTransitionalCodec`: provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.

14.2.1 Detailed Description

15 Namespace Documentation

15.1 goby Namespace Reference

The global namespace for the Goby project.

Namespaces

- `acomms`: Objects pertaining to acoustic communications (acomms)
- `common`: Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc.
• pb
  
  Contains objects relating to the core publish / subscribe architecture provided by Goby.

• transitional
  
  Objects pertaining to transitioning from DCCLv1 to DCCLv2.

Classes

• class Exception
  
  simple exception class for goby applications

Typedefs

• typedef google::protobuf::uint32 uint32
  
  an unsigned 32 bit integer

• typedef google::protobuf::int32 int32
  
  a signed 32 bit integer

• typedef google::protobuf::uint64 uint64
  
  an unsigned 64 bit integer

• typedef google::protobuf::int64 int64
  
  a signed 64 bit integer

Enumerations

• enum GobyFieldOptions_ConfigurationOptions_ConfigAction

  { GobyFieldOptions_ConfigurationOptions_ConfigAction_ALWAYS = 1, GobyFieldOptions_ConfigurationOptions_ConfigAction_NEVER = 2, GobyFieldOptions_ConfigurationOptions_ConfigAction_ADVANCED = 3 }

Functions

• template<typename App , typename Config >
  
  int run (int argc, char ∗argv[], Config ∗cfg)
  
  Run a Goby application derived from MinimalApplicationBase. blocks caller until MinimalApplicationBase::__run() returns.

• void protobuf_AddDesc_goby_commonprotobuf_2foption_5fextensions_2eproto ()

• void protobuf_AssignDesc_goby_commonprotobuf_2foption_5fextensions_2eproto ()

• void protobuf_ShutdownFile_goby_commonprotobuf_2foption_5fextensions_2eproto ()

• bool GobyFieldOptions_ConfigurationOptions_ConfigAction_IsValid (int value)

• const ::google::protobuf::EnumDescriptor ∗GobyFieldOptions_ConfigurationOptions_ConfigAction_descriptor ()

• const ::std::string &GobyFieldOptions_ConfigurationOptions_ConfigAction_Name (GobyFieldOptions_ - ConfigurationOptions_ConfigAction value)

• bool GobyFieldOptions_ConfigurationOptions_ConfigAction_Parse (const ::std::string ∗name, GobyFieldOptions_ConfigurationOptions_ConfigAction ∗value)

• std::string version_message ()
Variables

- `const` `GobyFieldOptions_ConfigurationOptions_ConfigAction_MIN = GobyFieldOptions_ConfigurationOptions_ConfigAction_ALWAYS`
- `const` `GobyFieldOptions_ConfigurationOptions_ConfigAction_MAX = GobyFieldOptions_ConfigurationOptions_ConfigAction_ADVANCED`
- `const int` `GobyFieldOptions_ConfigurationOptions_ConfigAction.ARRAYSIZ E = GobyFieldOptions_ConfigurationOptions_ConfigAction_CONFIGACTION_MAX + 1`
- `extern::google::protobuf::internal::ExtensionIdentifier< ::google::protobuf::FieldOptions, ::google::protobuf::internal::MessageTypeTraits< ::goby::GobyFieldOptions >, 11, false > field`
- `extern::google::protobuf::internal::ExtensionIdentifier< ::google::protobuf::MessageOptions, ::google::protobuf::internal::MessageTypeTraits< ::goby::GobyMessageOptions >, 11, false > msg`
- `const std::string VERSION_STRING = "2.1.4"`
- `const std::string VERSION_DATE = "2016.09.26"

Logger

- `common::FlexOstream glog`

Access the Goby logger through this object.

15.1 Detailed Description

The global namespace for the Goby project. Converts the Google Protocol Buffers message `msg` into a suitable (human readable) string `out` for sending via MOOS.

All objects related to the Goby Underwater Autonomy Project.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td>pointer to std::string to store serialized result</td>
</tr>
<tr>
<td><code>msg</code></td>
<td>Google Protocol buffers message to serialize</td>
</tr>
</tbody>
</table>

15.1.2 Function Documentation

15.1.2.1 `template<typename App , typename Config > int goby::run ( int argc, char ∗ argv[], Config ∗ cfg )`

Run a Goby application derived from MinimalApplicationBase. blocks caller until MinimalApplicationBase::run() returns.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>argc</code></td>
<td>same as int main(int argc, char ∗ argv)</td>
</tr>
<tr>
<td><code>argv</code></td>
<td>same as int main(int argc, char ∗ argv)</td>
</tr>
</tbody>
</table>

Returns

same as int main(int argc, char ∗ argv)

Definition at line 109 of file application_base.h.

15.2 goby::acomms Namespace Reference

Objects pertaining to acoustic communications (acomms)
Classes

- class **MACManager**
  
  provides an API to the goby-acomms MAC library. **MACManager** is essentially a `std::list<protobuf::ModemTransmission>` plus a timer.

- 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 **QueueException**
  
  `Exception` class for libdccl.

- class **QueueManager**
  
  provides an API to the goby-acomms Queuing Library.

Typedefs

- typedef `dccl::Exception` **DCCLEException**
- typedef `dccl::NullValueException` **DCCLNullValueException**
- typedef `dccl::DefaultIdentifierCodec` **DCCLDefaultIdentifierCodec**
- typedef `dccl::v2::DefaultBoolCodec` **DCCLDefaultBoolCodec**
- typedef `dccl::v2::DefaultStringCodec` **DCCLDefaultStringCodec**
- typedef `dccl::v2::DefaultBytesCodec` **DCCLDefaultBytesCodec**
- typedef `dccl::v2::DefaultEnumCodec` **DCCLDefaultEnumCodec**
- typedef `dccl::v2::DefaultMessageCodec` **DCCLDefaultMessageCodec**
- typedef `dccl::FieldCodecBase` **DCCLFieldCodecBase**
- typedef `dccl::FieldCodecManager` **DCCLFieldCodecManager**
- typedef `dccl::internal::FromProtoCppTypeBase` **FromProtoCppTypeBase**
- typedef `dccl::internal::TypeHelper` **DCCLTypeHelper**
- typedef `std::list<QueuedMessage>` **messages_it**
- typedef `std::multimap<unsigned, messages_it>` **waiting_for_ack_it**

Enumerations

- enum { **RATE_RUDICS** = 1, **RATE_SBD** = 0 }

Functions

- `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)
  
  binds the MAC initiate transmission callback to the driver and the driver parsed message callback to the MAC
• void bind (QueueManager &queue_manager, RouteManager &route_manager)
  creates bindings for a RouteManager to control a particular queue (QueueManager)
• void bind (ModemDriverBase &driver, QueueManager &queue_manager, MACManager &mac)
  bind all three (shortcut to calling the other three bind functions)
• void unbind (ModemDriverBase &driver, QueueManager &queue_manager)
  unbinds the driver link-layer callbacks to the QueueManager
• void unbind (MACManager &mac, ModemDriverBase &driver)
  unbinds the MAC initiate transmission callback to the driver and the driver parsed message callback to the MAC
• void unbind (QueueManager &queue_manager, RouteManager &route_manager)
  creates unbindings for a RouteManager to control a particular queue (QueueManager)
• void unbind (ModemDriverBase &driver, QueueManager &queue_manager, MACManager &mac)
  unbind all three (shortcut to calling the other three unbind functions)
• template<typename Signal, typename Slot>
  void connect (Signal ∗signal, Slot slot)
    connect a signal to a slot (e.g. function pointer)
• 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 Signal, typename Slot>
  void disconnect (Signal ∗signal, Slot slot)
    disconnect a signal to a slot (e.g. function pointer)
• template<typename Signal, typename Obj, typename A1>
  void disconnect (Signal ∗signal, Obj ∗obj, void(Obj::*mem_func)(A1))
    disconnect a signal to a member function with one argument
• template<typename Signal, typename Obj, typename A1, typename A2>
  void disconnect (Signal ∗signal, Obj ∗obj, void(Obj::*mem_func)(A1, A2))
    disconnect a signal to a member function with two arguments
• template<typename Signal, typename Obj, typename A1, typename A2, typename A3>
  void disconnect (Signal ∗signal, Obj ∗obj, void(Obj::*mem_func)(A1, A2, A3))
    disconnect a signal to a member function with three arguments
• std::ostream & operator<< (std::ostream &os, const DCCLCodec &codec)
• uint16_t net_checksum (const std::string &data)
• unsigned sbd_csum (const std::string &data)
• void serialize_rudics_packet (std::string &bytes, std::string &rudics_pkt)
• void parse_rudics_packet (std::string &bytes, std::string rudics_pkt)
• std::string uint32_to_byte_string (uint32_t i)
• uint32_t byte_string_to_uint32 (std::string s)
• std::ostream & operator<< (std::ostream &os, const Queue &oq)
• std::ostream & operator<< (std::ostream &os, const QueueManager &d)
  outputs information about all available messages (same as info_all)

Acoustic MAC Library callback function type definitions

• std::ostream & operator<< (std::ostream &os, const MACManager &mac)
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 (amac) is agnostic to the next destination. The next destination is thus set by the data provider (typically queue).
- const int QUERY_SOURCE_ID = -1
- const unsigned char DCCL_CCL_HEADER = 32
- 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)

15.2.1 Detailed Description

Objects pertaining to acoustic communications (acomms)

15.3 goby::common Namespace Reference

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

Namespaces

- tcolor
  Contains functions for adding color to Terminal window streams.

Classes

- class ConfigException
  indicates a problem with the runtime command line or .cfg file configuration (or –help was given)
- 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 Colors
  Represents the eight available terminal colors (and bold variants)
- class TermColor
  Converts between string, escape code, and enumeration representations of the terminal colors.
Enumerations

- enum MarshallingScheme {
  MARSHALLING_UNKNOWN = 0, MARSHALLING_CSTR = 1, MARSHALLING_PROTOBUF = 2, MARSHALLING_CCL = 3,
  MARSHALLING_MOOS = 4, MARSHALLING_DCCL = 5, MARSHALLING_LCM = 6, MARSHALLING_MAX = 6
}  
- enum {
  LIAISON_INTERNAL_PUBLISH_SOCKET = 1, LIAISON_INTERNAL_SUBSCRIBE_SOCKET = 2
}

Functions

- std::ostream & operator<< (std::ostream &out, const google::protobuf::Message &msg)
  
  provides stream output operator for Google Protocol Buffers Message
- void merge_app_base_cfg (AppBaseConfig *base_cfg, const boost::program_options::variables_map &var_map)
- std::ostream & operator<< (std::ostream &os, const HDF5ProtobufEntry &entry)
- const Wt::WColor goby_blue (28, 159, 203)
- const Wt::WColor goby_orange (227, 96, 52)
- std::string liaison_internal_publish_socket_name ()
- std::string liaison_internal_subscribe_socket_name ()
- 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)
- template<typename _CharT, typename _Traits, typename _Alloc>
  std::ostream & operator<< (FlexOstream &out, const std::basic_string<_CharT, _Traits, _Alloc> &s)
- 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
- uint64 ptime2unix_microsec (boost::posix_time::ptime given_time)
  
  convert from boost date_time ptime to the number of microseconds since 1/1/1970 0:00 UTC ("UNIX Time")
- boost::posix_time::ptime unix_microsec2ptime (uint64 given_time)
  
  convert to boost date_time ptime from the number of microseconds since 1/1/1970 0:00 UTC ("UNIX Time"); good to the microsecond
- boost::posix_time::ptime nmea_time2ptime (const std::string &mt)
- std::string zeromq_packet_make_header (MarshallingScheme marshalling_scheme, const std::string &identifier)
- void zeromq_packet_encode (std::string *raw, MarshallingScheme marshalling_scheme, const std::string &identifier, const std::string &body)
  
  Encodes a packet for Goby over ZeroMQ.
- void zeromq_packet_decode (const std::string &raw, MarshallingScheme *marshalling_scheme, std::string *identifier, std::string *body)
  
  Decodes a packet for Goby over ZeroMQ.
Variables

- `const int BITS_IN_UINT32 = 32`
- `const int BITS_IN_BYTE = 8`
- `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"

Time

- `boost::function0< uint64 > goby_time_function`
- `int goby_time_warp_factor = 1`
- `template<typename ReturnType> ReturnType goby_time ()`  
  Returns current UTC time as a boost::posix_time::ptime.
- `template<>
  uint64 goby_time< uint64 > ()`
  Returns current UTC time as integer microseconds since 1970-01-01 00:00:00.
- `template<>
  double goby_time< double > ()`
  Returns current UTC time as seconds and fractional seconds since 1970-01-01 00:00:00.
- `template<>
  boost::posix_time::ptime goby_time< boost::posix_time::ptime > ()`
- `template<>
  std::string goby_time< std::string > ()`
  Returns current UTC time as a human-readable string.
- `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()
- `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 ptme2time_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

15.3.1 Detailed Description

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

15.4.1 Detailed Description

Contains functions for adding color to Terminal window streams.

15.4.2 Function Documentation

15.4.2.1 std::ostream & goby::common::tcolor::add_escape_code ( std::ostream & os, const std::string & esc_code )

Append the given escape code to the stream os
15.5  goby::pb Namespace Reference

Contains objects relating to the core publish / subscribe architecture provided by Goby.

Classes

- **class Application**

  Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture.

15.5.1  Detailed Description

Contains objects relating to the core publish / subscribe architecture provided by Goby.

15.6  goby::transitional Namespace Reference

Objects pertaining to transitioning from DCCLv1 to DCCLv2.

Classes

- **class DCCLTransitionalCodec**

  provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.

- **class DCCLMessageVal**

  defines a DCCL value

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 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 }
- enum DCCLHeaderPart {
  HEAD_CCL_ID = 0, HEAD_DDCALL_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_DDCALL_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 }

Functions

- 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_protobuf_type (DCCLType type)
- std::string type_to_string (DCCLCppType type)
- std::string to_str (DCCLHeaderPart p)

- 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)

- template<typename Value>
  std::ostream & operator<<(std::ostream &out, const std::set<unsigned>&s)

- template<typename Value>
  std::ostream & operator<<(std::ostream &os, const DCCLMessageVal &mv)

- template<typename Value>
  std::ostream & operator<<(std::ostream &os, const std::vector<DCCLMessageVal>&vm)

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 (const unsigned char *c, 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 ("1001010101010101") into a hex string ("8AAA")

- std::string hex_string2binary_string (const std::string &bs)
  converts a boost::dynamic_bitset (similar to std::bitset but without compile time size requirements) into a hex string

- template<typename T>
  bool hex_string2number (const std::string &s, T &t)
  converts a hex string ("8AAA") into a dynamic_bitset

- 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
Variables

- const unsigned DCCL_NUM_HEADER_BYTES = 6
- const unsigned DCCL_NUM_HEADER_PARTS = 8
- const std::string DCCL_HEADER_NAMES []

15.6.1 Detailed Description

Objects pertaining to transitioning from DCCLv1 to DCCLv2.

15.6.2 Typedef Documentation

15.6.2.1 typedef boost::function<void (DCCLMessageVal&)> goby::transitional::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 39 of file message_algorithms.h.

15.6.2.2 typedef boost::function<void (DCCLMessageVal&, const std::vector<DCCLMessageVal>&)> goby::transitional::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 48 of file message_algorithms.h.

15.6.3 Enumeration Type Documentation

15.6.3.1 enum goby::transitional::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 57 of file dccl_constants.h.

15.6.3.2 enum goby::transitional::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 tag_static
- dccl_bool tag_bool
**dccl_int** tag_int

**dccl_float** tag_float

**dccl_enum** tag_enum

**dccl_string** tag_string

**dccl_hex** tag_hex

Definition at line 48 of file dccl_constants.h.

### 15.6.4 Function Documentation

#### 15.6.4.1 bool goby::transitional::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**

<table>
<thead>
<tr>
<th>c</th>
<th>pointer to array of char</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>reference to string to put char into as hex</td>
</tr>
<tr>
<td>n</td>
<td>length of c the first two hex chars in s are the 0 index in c</td>
</tr>
</tbody>
</table>

Definition at line 171 of file dccl_constants.h.

#### 15.6.4.2 std::string goby::transitional::hex_string2binary_string ( const std::string & bs ) [inline]

converts a boost::dynamic_bitset (similar to std::bitset but without compile time size requirements) into a hex string

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

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

Definition at line 242 of file dccl_constants.h.

#### 15.6.4.3 template<typename T> bool goby::transitional::hex_string2number ( const std::string & s, T & t )

converts a hex string ("8AAA") into a dynamic_bitset

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

**Returns**

true if conversion succeeds, false otherwise

Definition at line 273 of file dccl_constants.h.

#### 15.6.4.4 template<typename T> bool goby::transitional::number2hex_string ( std::string & s, const T & t, unsigned int width = 2 )

converts a decimal number of type T into a hex string

**Parameters**

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

**Returns**

true if successful, false otherwise

Definition at line 288 of file dccl_constants.h.
15.6.4.5 template<typename T> std::string goby::transitional::number2hex_string ( const T & t, unsigned int width = 2 )

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>decimal number to convert</td>
</tr>
<tr>
<td>width</td>
<td>desired width (in characters) of return string. Width should be twice the number of bytes</td>
</tr>
</tbody>
</table>

Returns

hex string

Definition at line 301 of file dccl_constants.h.

15.6.5 Variable Documentation

15.6.5.1 const std::string goby::transitional::DCCL_HEADER_NAMES[]

Initial value:

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

Definition at line 135 of file dccl_constants.h.

16 Class Documentation

16.1 boost::asio::time_traits< goby::common::GobyTime > Struct Template Reference

Time traits specialised for GobyTime.

```cpp
#include <goby/common/time.h>
```

Public Types

- typedef boost::posix_time::ptime time_type
  The time type.
- typedef boost::posix_time::time_duration duration_type
  The duration type.

Static Public Member Functions

- static time_type now ()
  Get the current time.
- static time_type add (const time_type &t, const duration_type &d)
  Add a duration to a time.
- static duration_type subtract (const time_type &t1, const time_type &t2)
  Subtract one time from another.
- static bool less_than (const time_type &t1, const time_type &t2)
  Test whether one time is less than another.
- static boost::posix_time::time_duration to_posix_duration (const duration_type &d)
  Convert to POSIX duration type.
16.1.1 Detailed Description

template<> struct boost::asio::time_traits< goby::common::GobyTime >

Time traits specialised for GobyTime.

Definition at line 229 of file time.h.

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

• goby/common/time.h

16.2 ChatCurses Class Reference

provides a terminal GUI for a chat window (lower box to type and upper box to receive messages). Part of the chat.cpp example.

#include "/build/goby2-gRCrru/goby2-2.1.5+DAILYDEB+640/share/examples/acomms/chat/chat_curses.h"

Public Member Functions

• void set_modem_id (unsigned id)
  
give the modem_id so we know how to label our messages

• void startup ()
  
start the display

• void run_input (std::string &line)
  
grab a character and if there's a line to return it will be returned in line

• void cleanup ()
  
end the display

• void post_message (unsigned id, const std::string &line)
  
add a message to the upper window (the chat log)

• void post_message (const std::string &line)

Constructors/Destructor

• ChatCurses ()
• ~ChatCurses ()

16.2.1 Detailed Description

provides a terminal GUI for a chat window (lower box to type and upper box to receive messages). Part of the chat.cpp example.

Examples:

acomms/chat/chat.cpp.

Definition at line 29 of file chat_curses.h.

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

• share/examples/acomms/chat/chat_curses.h
• share/examples/acomms/chat/chat_curses.cpp
16.3  

**gooby::acomms::ABCDriver Class Reference**

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

```cpp
#include <gooby/acomms/modemdriver/abc_driver.h>
```

Inheritance diagram for gooby::acomms::ABCDriver:

```
goby::acomms::ABCDriver

|<-- goby::acomms::ModemDriverBase
```

**Public Member Functions**

- void **startup**(const protobuf::DriverConfig &cfg)
  
  Starts the modem driver. Must be called before poll().

- void **shutdown**()
  
  Shuts down the modem driver.

- void **do_work**()
  
  Allows the modem driver to do its work.

- void **handle_initiate_transmission**(const protobuf::ModemTransmission &m)
  
  Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

**Additional Inherited Members**

16.3.1  **Detailed Description**

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

Examples:

```cpp
acomms/modemdriver/driver_simple/driver_simple.cpp.
```

Definition at line 39 of file abc_driver.h.

16.3.2  **Member Function Documentation**

16.3.2.1  void **gooby::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 method. Does not block.

Implements gooby::acomms::ModemDriverBase.

Definition at line 114 of file abc_driver.cpp.
16.3.2.2  void goby::acomms::ABCDriver::handle_initiate_transmission ( const protobuf::ModemTransmission & m ) [virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().
Parameters

| \[ m \] | ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request) |

Implements goby::acomms::ModemDriverBase.

Definition at line 77 of file abc_driver.cpp.

16.3.2.3 void goby::acomms::ABCDriver::startup ( const protobuf::DriverConfig & \[ \text{cfg} \] ) \[ \text{[virtual]} \]

Starts the modem driver. Must be called before poll().

Parameters

| \[ \text{cfg} \] | Startup configuration for the driver and modem. DriverConfig is defined in acomms_driver_base.proto. Derived classes can define extensions (see [http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions](http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions)) to DriverConfig to handle modem specific configuration. |

Implements goby::acomms::ModemDriverBase.

Definition at line 39 of file abc_driver.cpp.

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

- goby/acomms/modemdriver/abc_driver.h
- src/acomms/modemdriver/abc_driver.cpp

16.4 goby::acomms::MACManager Class Reference

provides an API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer.

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

Inherits std::list< T >.

Public Member Functions

Constructors/Destructor

- MACManager ()
  Default constructor.
- ~MACManager ()

Control

- void startup (const protobuf::MACConfig &\[ \text{cfg} \])
  Starts the MAC with given configuration.
- void restart ()
  Restarts the MAC with original configuration.
- void shutdown ()
  Shutdown the MAC.
- void do_work ()
  Allows the MAC timer to do its work. Does not block. If you prefer more control you can directly control the underlying boost::asio::io_service (get_io_service()) instead of using this function. This function is equivalent to get_io_service().poll();.
- void update ()
  You must call this after any change to the underlying list that would invalidate iterators or change the size (insert, push_back, erase, etc.).
- bool running ()
Modem Signals

- boost::signals2::signal<void(const protobuf::ModemTransmission &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::signals2::signal<br>
  > signal_slot_start
  
  Signals the start of all transmissions (even when we don't transmit)

- unsigned cycle_count()
- double cycle_duration()
- boost::asio::io_service & get_io_service()
- const std::string & glog_mac_group() const

16.4.1 Detailed Description

provides an API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer.

See Also

acomms_amac.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::acomms::protobuf).

Examples:


Definition at line 54 of file mac_manager.h.

16.4.2 Member Function Documentation

16.4.2.1 void goby::acomms::MACManager::startup (const protobuf::MACConfig &<br>
  
  Starts the MAC with given configuration.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfg</td>
<td>Initial configuration values (protobuf::MACConfig defined in acomms_amac.proto)</td>
</tr>
</tbody>
</table>

Examples:


Definition at line 71 of file mac_manager.cpp.

16.4.3 Member Data Documentation

16.4.3.1 boost::signals2::signal<void (const protobuf::ModemTransmission & m)>
  > goby::acomms::MACManager::signal_<br>
  > 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

| m | a message containing details of the transmission to be initiated. (protobuf::ModemMsgBase defined in acomms_modem_message.proto) |

Examples:


Definition at line 96 of file mac_manager.h.

16.4.3.2 boost::signals2::signal<\ void (const protobuf::ModemTransmission& m)> goby::acomms::MACManager::signal_slot_start

Signals the start of all transmissions (even when we don’t transmit)

Parameters

| m | a message containing details of the transmission to be initiated. (protobuf::ModemMsgBase defined in acomms_modem_message.proto) |

Definition at line 102 of file mac_manager.h.

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

- goby/acomms/amac/mac_manager.h
- src/acomms/amac/mac_manager.cpp

16.5 goby::acomms::MMDriver Class Reference

provides an API to the WHOI Micro-Modem driver

#include <goby/acomms/modem_driver.h>

Inheritance diagram for goby::acomms::MMDriver:

```
goby::acomms::ModemDriverBase

```

goby::acomms::MMDriver

Public Member Functions

- MMDriver ()
  Default constructor.
- ~MMDriver ()
  Destructor.
- void startup (const protobuf::DriverConfig &cfg)
  Starts the driver.
void update_cfg (const protobuf::DriverConfig &cfg)

Update configuration while running (not required to be implemented)

void shutdown ()

Stops the driver.

void do_work ()

See ModemDriverBase::do_work()

void handle_initiate_transmission (const protobuf::ModemTransmission &m)

See ModemDriverBase::handle_initiate_transmission()

int clk_mode ()

Current clock mode of the modem, necessary for synchronous navigation.

bool is_started () const

void set_silent (bool silent)

void write_single_cfg (const std::string &s)

Static Public Member Functions

static unsigned packet_frame_count (int rate)

static unsigned packet_size (int rate)

Additional Inherited Members

16.5.1 Detailed Description

provides an API to the WHOI Micro-Modem driver

Examples:

acomms/chat/chat.cpp, and acomms/modemdriver/driver_simple/driver_simple.cpp.

Definition at line 42 of file mm_driver.h.

16.5.2 Member Function Documentation

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

Starts the driver.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfg</td>
<td>Configuration for the Micro-Modem driver. DriverConfig is defined in acomms_driver_base.proto, and various extensions specific to the WHOI Micro-Modem are defined in acomms_mm_driver.proto.</td>
</tr>
</tbody>
</table>

Implements goby::acomms::ModemDriverBase.

Examples:

acomms/chat/chat.cpp.

Definition at line 85 of file mm_driver.cpp.

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

- gobyacomms/modemdriver/mm_driver.h
- src/acomms/modemdriver/mmm_driver.cpp
16.6 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>

Inheritance diagram for goby::acomms::ModemDriverBase:

Public Member Functions

• virtual ~ModemDriverBase ()
  Public Destructor.
• int driver_order ()

Control

• virtual void startup (const protobuf::DriverConfig &cfg)=0  
  Starts the modem driver. Must be called before poll().
• virtual void update_cfg (const protobuf::DriverConfig &cfg)
  Update configuration while running (not required to be implemented)
• 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 (const protobuf::ModemTransmission &m)=0  
  Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Public Attributes

MAC / Queue Signals

• boost::signals2::signal< void(const protobuf::ModemTransmission &message) > signal_receive  
  Called when a binary data transmission is received from the modem.
• boost::signals2::signal< void(const protobuf::ModemTransmission &message)> signal_transmit_result
  Called when a transmission is completed.
• boost::signals2::signal< void(protobuf::ModemTransmission *msg)> signal_data_request
  Called when the modem or modem driver needs data to send. The returned data should be stored in ModemTransmission::frame.
• boost::signals2::signal< void(protobuf::ModemTransmission *msg_request)> signal_modify_transmission
  Called before the modem driver begins processing a transmission. This allows a third party to modify the parameters of the transmission (such as destination or rate) on the fly.
• boost::signals2::signal< void(const protobuf::ModemRaw &msg)> signal_raw_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::signals2::signal< void(const protobuf::ModemRaw &msg)> signal_raw_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 ()
  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.
• const std::string & glog_out_group () const
• const std::string & glog_in_group () const
• util::LineBasedInterface & modem ()
  use for direct access to the modem

Static Protected Attributes

• static int count_ = 0

16.6.1 Detailed Description

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

acomms_driver_base.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::acomms::protobuf).

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

Definition at line 44 of file driver_base.h.

16.6.2 Member Function Documentation

16.6.2.1 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 method. Does not block.

Implemented in goby::acomms::MMDriver, goby::moos::BluefinCommsDriver, goby::moos::UFldDriver, and goby::acomms::ABCDriver.

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

16.6.2.2 virtual void goby::acomms::ModemDriverBase::handle_initiate_transmission ( const protobuf::ModemTransmission & m ) [pure virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Parameters

| m          | ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request) |

Implemented in goby::acomms::MMDriver, goby::moos::BluefinCommsDriver, goby::moos::UFldDriver, and goby::acomms::ABCDriver.

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

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

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

Parameters

| in | pointer to string to store line |

Returns

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

Definition at line 66 of file driver_base.cpp.

16.6.2.4 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfg</code></td>
<td>Configuration including the parameters for the physical connection. (protobuf::DriverConfig is defined in acomms_driver_base.proto).</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModemDriverException</td>
<td>Problem opening the physical connection.</td>
</tr>
</tbody>
</table>

Definition at line 81 of file driver_base.cpp.

16.6.2.5 void goby::acomms::ModemDriverBase::modem_write ( const std::string & `out` ) [protected]
write a line to the serial port.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td>reference to string to write. Must already include any end-of-line character(s).</td>
</tr>
</tbody>
</table>

Definition at line 58 of file driver_base.cpp.

16.6.2.6 virtual void goby::acomms::ModemDriverBase::startup ( const protobuf::DriverConfig & `cfg` ) [pure virtual]
Starts the modem driver. Must be called before poll().

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfg</code></td>
<td>Startup configuration for the driver and modem. DriverConfig is defined in acomms_driver_base.proto. Derived classes can define extensions (see <a href="http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions">http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions</a>) to DriverConfig to handle modem specific configuration.</td>
</tr>
</tbody>
</table>

Implemented in goby::acomms::MMDriver, goby::moos::BluefinCommsDriver, goby::moos::UFldDriver, and goby::acomms::ABCDriver.

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

16.6.3 Member Data Documentation

16.6.3.1 boost::signals2::signal<void (protobuf::ModemTransmission∗ msg) > goby::acomms::ModemDriverBase::signal_ data_request
Called when the modem or modem driver needs data to send. The returned data should be stored in ModemTransmission::frame.

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 poll. ModemTransmission is defined in acomms_modem_message.proto.

Definition at line 96 of file driver_base.h.

16.6.3.2 boost::signals2::signal<void (protobuf::ModemTransmission∗ msg_request) > goby::acomms::ModemDriverBase:: signal_modify_transmission
Called before the modem driver begins processing a transmission. This allows a third party to modify the parameters of the transmission (such as destination or rate) on the fly.

You may 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 poll. ModemTransmission is defined in acomms_modem_message.proto.

Definition at line 102 of file driver_base.h.
16.6.3.3 boost::signals2::signal<void (const protobuf::ModemRaw& msg)>
goby::acomms::ModemDriverBase::signal_raw_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 poll. ModemRaw is defined in acomms_modem_message.proto.

Definition at line 109 of file driver_base.h.

16.6.3.4 boost::signals2::signal<void (const protobuf::ModemRaw& msg)>
goby::acomms::ModemDriverBase::signal_raw_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 poll. ModemRaw is defined in acomms_modem_message.proto.

Definition at line 115 of file driver_base.h.

16.6.3.5 boost::signals2::signal<void (const protobuf::ModemTransmission& 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 poll. ModemDataTransmission is defined in acomms_modem_message.proto.

Examples:
acomms/chat/chat.cpp, and acomms/modemdriver/driver_simple/driver_simple.cpp.

Definition at line 83 of file driver_base.h.

16.6.3.6 boost::signals2::signal<void (const protobuf::ModemTransmission& message)>
goby::acomms::ModemDriverBase::signal_transmit_result

Called when a transmission is completed.

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 poll. ModemDataTransmission is defined in acomms_modem_message.proto.

Definition at line 90 of file driver_base.h.

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

- goby/acomms/modemdriver/driver_base.h
- src/acomms/modemdriver/driver_base.cpp

16.7 goby::acomms::QueueException Class Reference

Exception class for libdccl.

#include <goby/acomms/queue/queue_exception.h>
Inheritance diagram for goby::acomms::QueueException:

```plaintext
Inheritance diagram for goby::acomms::QueueException:
```

Public Member Functions

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

16.7.1 Detailed Description

**Exception** class for libdccl.

Definition at line 34 of file queue_exception.h.

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

- goby/acoms/queue/queue_exception.h

16.8 goby::acomms::QueueManager Class Reference

provides an API to the goby-acoms Queuing Library.

#include <goby/acoms/queue.h>

Public Member Functions

- **QueueManager** ()
  - 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 acoms_queue.proto)
- void **merge_cfg** (const protobuf::QueueManagerConfig &cfg)
  
  Set (and merge "repeat" fields) the current configuration. (protobuf::QueueManagerConfig defined in acoms_queue.proto)
- template<typename ProtobufMessage>
  
  void **add_queue** (const protobuf::QueuedMessageEntry &queue_cfg)
Add a DCCL queue for use with QueueManager. Note that the queue must be added before receiving messages with QueueManager.

- void add_queue (const google::protobuf::Descriptor *desc, const protobuf::QueuedMessageEntry &queue_cfg)

  Alternative method for adding Queues when using Dynamic Protobuf Messages.

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 google::protobuf::Message &new_message)
  Push a message (and add the queue if it does not exist)
- void push_message (const google::protobuf::Message &new_message, const protobuf::QueuedMessageMeta *meta)
- 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 (protobuf::ModemTransmission *msg)
  Finds data to send to the modem.
- void handle_modem_receive (const protobuf::ModemTransmission &message)
  Receive incoming data 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

- void info_all (std::ostream &os) const
  Writes a human readable summary (including DCCLCodec info) of all loaded queues.
- template<typename ProtobufMessage>
  void info (std::ostream &os) const
  Writes a human readable summary (including DCCLCodec info) of the queue for the provided DCCL type to the stream provided.
- void info (const google::protobuf::Descriptor *desc, std::ostream &os) const
  An alternative form for getting information for Queues for message types not known at compile-time ("dynamic").
- const std::string & glog_push_group ()
- const std::string & glog_pop_group ()
- const std::string & glog_priority_group ()
- const std::string & glog_out_group ()
- const std::string & glog_in_group ()
- std::string msg_string (const google::protobuf::Descriptor *desc)
- int modem_id ()
  The current modem ID (MAC address) of this node.
- protobuf::QueuedMessageMeta meta_from_msg (const google::protobuf::Message &msg)
Public Attributes

- boost::signals2::signal< void(protobuf::QueuedMessageMeta *meta, const google::protobuf::Message &data_msg, int modem_id) > signal_out_route
  
  *Used by a router to change next-hop destination (in meta)*

- boost::signals2::signal< void(const protobuf::QueuedMessageMeta &meta, const google::protobuf::Message &data_msg, int modem_id) > signal_in_route
  
  *Used by a router to intercept messages and requeue them if desired.*

Application Signals

- boost::signals2::signal< void(const google::protobuf::Message &ack_msg, const google::protobuf::Message &orig_msg) > signal_ack
  
  *Signals when acknowledgment of proper message receipt has been received. This is only sent for queues with queue.ack == true with an explicit destination (ModemMessageBase::dest() != 0)*

- boost::signals2::signal< void(const google::protobuf::Message &msg) > signal_receive
  
  *Signals when a DCCL message is received.*

- boost::signals2::signal< void(const google::protobuf::Message &orig_msg) > signal_expire
  
  *Signals when a message is expires (exceeds its time-to-live or ttl) before being sent (if queue.ack == false) or before being acknowledged (if queue.ack == true).*

- boost::signals2::signal< void(const google::protobuf::ModemTransmission &request_msg, google::protobuf::Message *data_msg) > signal_data_on_demand
  
  *Forwards the data request to the application layer. This advanced feature is used when queue.encode_on_demand == true and allows for the application to provide data immediately before it is actually sent (for highly time sensitive data)*

- boost::signals2::signal< void(protobuf::QueueSize size) > signal_queue_size_change
  
  *Signals when any queue changes size (message is popped or pushed)*

Friends

- class Queue

16.8.1 Detailed Description

provides an API to the goby-acomms Queuing Library.

See Also

acomms_queue.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::acomms::protobuf).

Examples:

acomms/chat/chat.cpp, and acomms/queue/queue_simple/queue_simple.cpp.
16.8.2 Member Function Documentation

16.8.2.1 template <typename ProtobufMessage > void goby::acomms::QueueManager::add_queue ( const protobuf::QueuedMessageEntry & queue_cfg ) [inline]

Add a DCCL queue for use with QueueManager. Note that the queue must be added before receiving messages with QueueManager.

Template Parameters

| ProtobufMessage | Any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::Message) |

Definition at line 74 of file queue_manager.cpp.

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

Flush (delete all messages in) a queue.

Parameters

| flush | QueueFlush object containing details about queue to flush |

Definition at line 164 of file queue_manager.cpp.

16.8.2.3 void goby::acomms::QueueManager::handle_modem_data_request ( protobuf::ModemTransmission * msg )

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 connected to ModemDriverBase::signal_data_request.

Parameters

| msg | The ModemTransmission containing information about the data request and is the place where the request data will be stored (in the repeated field ModemTransmission::frame). |

Returns

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

Examples:

acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 218 of file queue_manager.cpp.

16.8.2.4 void goby::acomms::QueueManager::handle_modem_receive ( const protobuf::ModemTransmission & message )

Receive incoming data from the modem.

If using one of the classes inheriting ModemDriverBase, this method should be bound and passed to ModemDriverBase::signal_receive.

Parameters

| message | The received ModemMessage. |

Examples:

acomms/queue/queue_simple/queue_simple.cpp.
16.8.5 template<typename ProtobufMessage> void goby::acomms::QueueManager::info (std::ostream * os) const

Writes a human readable summary (including DCCLCodec info) of the queue for the provided DCCL type to the stream provided.

Template Parameters

| ProtobufMessage | Any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::Message) |

Parameters

| os | Pointer to a stream to write this information |

Definition at line 147 of file queue_manager.h.

16.8.6 void goby::acomms::QueueManager::info_all (std::ostream * os) const

Writes a human readable summary (including DCCLCodec info) of all loaded queues.

Parameters

| os | Pointer to a stream to write this information |

Definition at line 181 of file queue_manager.cpp.

16.8.7 void goby::acomms::QueueManager::push_message (const google::protobuf::Message & new_message)

Push a message (and add the queue if it does not exist)

Parameters

| new_message | DCCL message to push. |

Examples:

acomms/chat/chat.cpp, and acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 137 of file queue_manager.cpp.

16.8.3 Member Data Documentation

16.8.3.1 boost::signals2::signal<void (const protobuf::ModemTransmission& ack_msg, const google::protobuf::Message& orig_msg)> goby::acomms::QueueManager::signal_ack

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

Parameters

| ack_msg | a message containing details of the acknowledgment and the acknowledged transmission. (protobuf::ModemMsgAck is defined in acomms_modem_message.proto) |

Examples:

acomms/chat/chat.cpp.

Definition at line 189 of file queue_manager.h.
Forwards the data request to the application layer. This advanced feature is used when queue.encode_on_demand == true and allows for the application to provide data immediately before it is actually sent (for highly time sensitive data).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>request_msg</em></td>
<td>the details of the requested data. (protobuf::ModemDataRequest is defined in acomms_modem_message.proto)</td>
</tr>
<tr>
<td><em>data_msg</em></td>
<td>pointer to store the supplied data. The message is of the type for this queue.</td>
</tr>
</tbody>
</table>

Definition at line 206 of file queue_manager.h.

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>expire_msg</em></td>
<td>the expired transmission. (protobuf::ModemDataExpire is defined in acomms_modem_message.proto)</td>
</tr>
</tbody>
</table>

Definition at line 199 of file queue_manager.h.

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>size</em></td>
<td>message containing the queue that changed size and its new size (protobuf::QueueSize is defined in acomms_queue.proto).</td>
</tr>
</tbody>
</table>

Definition at line 211 of file queue_manager.h.

Signals when a DCCL message is received.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>msg</em></td>
<td>the received transmission.</td>
</tr>
</tbody>
</table>

Examples:

acomms/chat/chat.cpp, and acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 194 of file queue_manager.h.

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

- goby/acomms/queue/queue_manager.h
- src/acomms/queue/queue_manager.cpp

### 16.9 goby::common::Colors Struct Reference

Represents the eight available terminal colors (and bold variants)

```
#include <goby/common/logger/term_color.h>
```
16.10  

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)*

16.9.1  Detailed Description

Represents the eight available terminal colors (and bold variants)
Definition at line 123 of file term_color.h.
The documentation for this struct was generated from the following file:

- goby/common/logger/term_color.h

16.10  

indicates a problem with the runtime command line or .cfg file configuration (or --help was given)
#include <goby/common/configuration_reader.h>
Inheritance diagram for goby::common::ConfigException:

![Inheritance diagram](#)

16.10.1  Detailed Description

indicates a problem with the runtime command line or .cfg file configuration (or --help was given)
Definition at line 54 of file configuration_reader.h.
The documentation for this class was generated from the following file:

- goby/common/configuration_reader.h
16.11  goby::common::FlexNCurses Class Reference

Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.

```cpp
#include <goby/common/logger/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*

16.11.1  Detailed Description

Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.

Definition at line 42 of file flex_ncurses.h.

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

- `goby/common/logger/flex_ncurses.h`
- `src/common/logger/flex_ncurses.cpp`

16.12  goby::common::FlexOstream Class Reference

Forms the basis of the Goby logger: std::ostream derived class for holding the FlexOstreamBuf.

```cpp
#include <goby/common/logger/flex_ostream.h>
```

Inherits std::ostream.
Public Member Functions

- void refresh ()
- void set_group (const std::string &s)
- void set_unset_verbosity ()

Initialization

- void 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.
- void set_name (const std::string &s)
  Set the name of the application that the logger is serving.
- void enable_gui ()
- bool is (goby::common::logger::Verbosity verbosity)
- 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.
- void add_stream (goby::common::protobuf::GLogConfig::Verbosity verbosity=goby::common::protobuf::GLogConfig::VERBOSE, std::ostream *os=0)
- const FlexOStreamBuf & buf ()

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::recursive_mutex & mutex ()
  Get a reference to the Goby logger mutex for scoped locking.
- void set_lock_action (logger_lock::LockAction lock_action)

Friends

- 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)
16.12.1 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.

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

- goby/common/logger/flex_ostream.h
- src/common/logger/flex_ostream.cpp

16.13 goby::common::TermColor Class Reference

Converts between string, escape code, and enumeration representations of the terminal colors.

```cpp
#include <goby/common/logger/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")

Friends

- template<typename T >
  
  void `boost::checked_delete` (T *)

16.13.1 Detailed Description

Converts between string, escape code, and enumeration representations of the terminal colors.

Definition at line 137 of file term_color.h.

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

- goby/common/logger/term_color.h
- src/common/logger/term_color.cpp

16.14 goby::Exception Class Reference

Simple exception class for goby applications

```cpp
#include <goby/common/exception.h>
```
Inheritance diagram for goby::Exception:

```
Inheritance diagram for goby::Exception:
```

Public Member Functions

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

16.14.1 Detailed Description

simple exception class for goby applications

Definition at line 33 of file exception.h.

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

- `goby/common/exception.h`

16.15 goby::moos::BluefinCommsDriver Class Reference

provides a driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hard-
ware)

```c
#include <goby/moos/moos_bluefin_driver.h>
```

Inheritance diagram for goby::moos::BluefinCommsDriver:

```
Inheritance diagram for goby::moos::BluefinCommsDriver:
```

Public Member Functions

- **BluefinCommsDriver** (goby::acomms::MACManager *mac)
• void **startup** (const goby::acomms::protobuf::DriverConfig &cfg)
  
  Starts the modem driver. Must be called before poll().

• void **shutdown**()

  Shuts down the modem driver.

• void **do_work**()

  Allows the modem driver to do its work.

• void **handle_initiate_transmission** (const goby::acomms::protobuf::ModemTransmission &m)

  Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Additional Inherited Members

16.15.1 Detailed Description

provides a driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hardware)

Definition at line 45 of file moos_bluefin_driver.h.

16.15.2 Member Function Documentation

16.15.2.1 void goby::moos::BluefinCommsDriver::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 method. Does not block.

Implements goby::acomms::ModemDriverBase.

Definition at line 186 of file moos_bluefin_driver.cpp.

16.15.2.2 void goby::moos::BluefinCommsDriver::handle_initiate_transmission (const goby::acomms::protobuf::ModemTransmission &m)

[virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Parameters

| m | ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request) |

Implements goby::acomms::ModemDriverBase.

Definition at line 98 of file moos_bluefin_driver.cpp.

16.15.2.3 void goby::moos::BluefinCommsDriver::startup (const goby::acomms::protobuf::DriverConfig & cfg)

[virtual]

Starts the modem driver. Must be called before poll().

Parameters

| cfg | Startup configuration for the driver and modem. DriverConfig is defined in acomms_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.
The documentation for this class was generated from the following files:

- `goby/moos/moos_bluefin_driver.h`
- `src/moos/moos_bluefin_driver.cpp`

**Public Member Functions**

- `void startup (const goby::acomms::protobuf::DriverConfig &cfg)`
  
  Starts the modem driver. Must be called before `poll()`.

- `void shutdown ()`
  
  Shuts down the modem driver.

- `void do_work ()`
  
  Allows the modem driver to do its work.

- `void handle_initiate_transmission (const goby::acomms::protobuf::ModemTransmission &m)`
  
  Virtual `initiate_transmission` method. Typically connected to `MACManager::signal_initiate_transmission()` using `bind()`.

**Additional Inherited Members**

16.16.1 Detailed Description


Definition at line 44 of file `moos_ufield_sim_driver.h`.

16.16.2 Member Function Documentation

16.16.2.1 `void goby::moos::UFldDriver::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 method. Does not block.

Implements goby::acomms::ModemDriverBase.

Definition at line 207 of file moos_ufield_sim_driver.cpp.

16.16.2.2 void goby::moos::UFldDriver::handle_initiate_transmission ( const goby::acomms::protobuf::ModemTransmission & m ) [virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Parameters

<table>
<thead>
<tr>
<th>m</th>
<th>ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request)</th>
</tr>
</thead>
</table>

Implements goby::acomms::ModemDriverBase.

Definition at line 89 of file moos_ufield_sim_driver.cpp.

16.16.2.3 void goby::moos::UFldDriver::startup ( const goby::acomms::protobuf::DriverConfig & cfg ) [virtual]

Starts the modem driver. Must be called before poll().

Parameters

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

Implements goby::acomms::ModemDriverBase.

Definition at line 45 of file moos_ufield_sim_driver.cpp.

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

- goby/moos/moos_ufield_sim_driver.h
- src/moos/moos_ufield_sim_driver.cpp

16.17 goby::pb::Application Class Reference

Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture.

#include <goby/pb/application.h>

Inherits goby::common::ZeroMQApplicationBase.

Inherited by goby::acomms::Bridge, goby::acomms::FileTransfer, goby::acomms::IPGateway, goby::acomms::ModemDriver, and goby::acomms::MoshRelay.

Protected Types

- typedef goby::common::Colors Colors

Protected Member Functions

- template<typename ProtoBufMessage > void subscribe (boost::function< void(const ProtoBufMessage &)> handler=boost::function< void(const ProtoBufMessage &)>(), const std::string &group="")
16.17 goby::pb::Application Class Reference

Subscribe to a message (of any type derived from google::protobuf::Message)

- template<typename ProtoBufMessage, class C>
  void subscribe (void(C::*mem_func)(const ProtoBufMessage&), C* obj, const std::string &group="")
  Subscribe for a type using a class member function as the handler.
- common::ZeroMQService & zeromq_service ()
  Fetchs the newest received message of this type.
- boost::shared_ptr<StaticProtobufPubSubNodeWrapper> pubsub_node()

Constructors / Destructor

- Application (google::protobuf::Message *cfg=0)
- virtual ~Application ()

Publish / Subscribe

- void publish (const google::protobuf::Message &msg, const std::string &group="")

16.17.1 Detailed Description

Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture.
Definition at line 45 of file application.h.

16.17.2 Constructor & Destructor Documentation

16.17.2.1 goby::pb::Application::Application (google::protobuf::Message *cfg = 0) [protected]

Parameters

| cfg | pointer to object derived from google::protobuf::Message that defines the configuration for this Application. This constructor will use the Description of cfg to read the command line parameters and configuration file (if given) and use these values to populate cfg. cfg must be a static member of the subclass or global object since member objects will be constructed after the Application constructor is called. |

Definition at line 38 of file application.cpp.

16.17.3 Member Function Documentation

16.17.3.1 template<typename ProtoBufMessage> void goby::pb::Application::subscribe (boost::function< void(const ProtoBufMessage &)> handler = boost::function<void (const ProtoBufMessage&)>(), const std::string & group = "") [inline],[protected]

Subscribe to a message (of any type derived from google::protobuf::Message)

Parameters

| handler | Function object to be called as soon as possible upon receipt of a message of this type. The signature of handler must match: void handler(const ProtoBufMessage& msg). If handler is omitted, no handler is called and only the newest message buffer is updated upon message receipt (for calls to newest<ProtoBufMessage>()). |

Definition at line 71 of file application.h.

16.17.3.2 template<typename ProtoBufMessage, class C> void goby::pb::Application::subscribe (void(C::*)(const ProtoBufMessage & mem_func, C* obj, const std::string & group = "") [inline],[protected]

Subscribe for a type using a class member function as the handler.
Parameters

<table>
<thead>
<tr>
<th>mem_func</th>
<th>Member function (method) of class C with a signature of void C::mem_func(const ProtoBuf::Message&amp; msg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>pointer to the object whose member function (mem_func) to call</td>
</tr>
</tbody>
</table>

Definition at line 87 of file application.h.

16.17.3  common::ZeroMQService& goby::pb::Application::zeromq_service ( ) [inline],[protected]

Fetches the newest received message of this type.

You must subscribe() for this type before using this method

Definition at line 111 of file application.h.

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

- goby/pb/application.h
- src/pb/application.cpp

16.18  goby::transitional::DCCLMessageVal Class Reference

defines a DCCL value

#include <goby/moos/transitional/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
  allows statements of the form
• operator bool () const
  allows statements of the form
• operator std::string () const
  allows statements of the form
• operator long () const
  allows statements of the form
• operator int () const
  allows statements of the form
• operator unsigned () const
  allows statements of the form
• operator float () const
  allows statements of the form
• 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)

16.18.1 Detailed Description

defines a DCCL value
Definition at line 36 of file message_val.h.
16.18.2 Member Function Documentation

16.18.2.1 bool goby::transitional::DCCLMessageVal::get ( std::string & s ) const

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

\texttt{s} \quad \text{std::string to store value in}

Returns

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

Definition at line 127 of file message_val.cpp.

16.18.2.2 \begin{verbatim} bool goby::transitional::DCCLMessageVal::get ( bool & b ) \end{verbatim} const

extract as bool (all reasonable casts are done)

Parameters

\begin{verbatim} b \quad \text{bool to store value in} \end{verbatim}

Returns

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

Definition at line 158 of file message_val.cpp.

16.18.2.3 \begin{verbatim} bool goby::transitional::DCCLMessageVal::get ( long & t ) \end{verbatim} const

extract as long (all reasonable casts are done)

Parameters

\begin{verbatim} t \quad \text{long to store value in} \end{verbatim}

Returns

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

Definition at line 190 of file message_val.cpp.

16.18.2.4 \begin{verbatim} bool goby::transitional::DCCLMessageVal::get ( double & d ) \end{verbatim} const

extract as double (all reasonable casts are done)

Parameters

\begin{verbatim} d \quad \text{double to store value in} \end{verbatim}

Returns

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

Definition at line 229 of file message_val.cpp.

16.18.2.5 \begin{verbatim} goby::transitional::DCCLMessageVal::operator bool ( ) \end{verbatim} const

allows statements of the form

\begin{verbatim} bool b = DCCLMessageVal("1"); \end{verbatim}

Definition at line 276 of file message_val.cpp.

\flushright

\begin{verbatim}
Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
\end{verbatim}
16.18.2.6 `goby::transitional::DCCLMessageVal::operator double ( ) const` allows statements of the form

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

Definition at line 268 of file message_val.cpp.

16.18.2.7 `goby::transitional::DCCLMessageVal::operator float ( ) const` allows statements of the form

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

Definition at line 307 of file message_val.cpp.

16.18.2.8 `goby::transitional::DCCLMessageVal::operator int ( ) const` allows statements of the form

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

Definition at line 297 of file message_val.cpp.

16.18.2.9 `goby::transitional::DCCLMessageVal::operator long ( ) const` allows statements of the form

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

Definition at line 290 of file message_val.cpp.

16.18.2.10 `goby::transitional::DCCLMessageVal::operator std::string ( ) const` allows statements of the form

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

Definition at line 283 of file message_val.cpp.

16.18.2.11 `goby::transitional::DCCLMessageVal::operator unsigned ( ) const` allows statements of the form

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

Definition at line 302 of file message_val.cpp.

16.18.2.12 `void goby::transitional::DCCLMessageVal::set ( double dval, int precision = MAX_DBL_PRECISION )` set the value with a double (overwrites previous value regardless of type)

<table>
<thead>
<tr>
<th><code>dval</code></th>
<th>values to set</th>
</tr>
</thead>
</table>

Parameters
### 16.19 goby::transitional::DCCLTransitionalCodec Class Reference

provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.

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

#### Public Member Functions

- template<typename Key >
  - const google::protobuf::Descriptor ∗ descriptor (const Key &k)
  - 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)
  - std::vector< DCCLMessage > & messages ()

#### Constructors/Destructor

- DCCLTransitionalCodec ()
  - Instantiate optionally with a ostream logger (for human readable output)
- ~DCCLTransitionalCodec ()
  - destructor

#### 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 convert_to_v2_representation (pAcommsHandlerConfig ∗cfg)

#### Detailed Description

provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.

See Also

- transitional.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::transitional::protobuf).

Definition at line 96 of file dccl_transitional.h.
16.19.2 Constructor & Destructor Documentation

16.19.2.1 `goby::transitional::DCCLTransitionalCodec::DCCLTransitionalCodec()`

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

**Parameters**

| log          | std::ostream object or FlexOstream to capture all humanly readable runtime and debug information (optional). |

Definition at line 44 of file `dccl_transitional.cpp`.

16.19.3 Member Function Documentation

16.19.3.1 `std::set<unsigned> goby::transitional::DCCLTransitionalCodec::all_message_ids()`

Returns set of all message ids loaded

16.19.3.2 `std::set<std::string> goby::transitional::DCCLTransitionalCodec::all_message_names()`

Returns set of all message names loaded

16.19.3.3 `template<typename Key> unsigned goby::transitional::DCCLTransitionalCodec::get_repeat(const Key & k)`

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

Definition at line 131 of file `dccl_transitional.h`.

16.19.3.4 `std::string goby::transitional::DCCLTransitionalCodec::id2name(unsigned id)`

Parameters

| id | message id |

Returns name of message

Definition at line 145 of file `dccl_transitional.h`.

16.19.3.5 `unsigned goby::transitional::DCCLTransitionalCodec::message_count()`

Returns number of messages loaded

Definition at line 127 of file `dccl_transitional.h`.

16.19.3.6 `template<typename Key> std::map<std::string, std::string> goby::transitional::DCCLTransitionalCodec::message_var_names(const Key & k) const`
16.20  goby::util::LineBasedInterface Class Reference

basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)

#include <goby/util/linebasedcomms/interface.h>

Inheritance diagram for goby::util::LineBasedInterface:

```
Inheritance diagram for goby::util::LineBasedInterface:
```

Public Types
- enum AccessOrder { NEWEST_FIRST, OLDEST_FIRST }

Public Member Functions
- LineBasedInterface (const std::string &delimeter)
- void start ()
- void close ()
- bool active ()
- void sleep (int sec)
- 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
- boost::asio::io_service & **io_service** ()

Protected Member Functions

- 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)
- std::string & **delimiter** ()
- std::deque<typename protobuf::Datagram>& **in** ()
- boost::mutex & **in_mutex** ()

 Protected Attributes

- std::string **delimiter_**
- boost::asio::io_service **io_service_**
- std::deque<typename protobuf::Datagram> **in_**
- boost::mutex **in_mutex_**

Friends

- template<typename ASIOAsyncReadStream>
  class LineBasedConnection

16.20.1 Detailed Description

basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)

Definition at line 45 of file interface.h.

16.20.2 Member Function Documentation

16.20.2.1 bool goby::util::LineBasedInterface::readline ( std::string * s, AccessOrder order=OLDEST_FIRST )

returns string line (including delimiter)

Returns

true if data was read, false if no data to read

Definition at line 80 of file interface.cpp.

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

- goby/util/linebasedcomms/interface.h
- src/util/linebasedcomms/interface.cpp
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

```cpp
#include <goby/util/linebasedcomms/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)

16.21.1 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 35 of file serial_client.h.

16.21.2 Constructor & Destructor Documentation

16.21.2.1 goby::util::SerialClient::SerialClient ( const std::string &name = "", unsigned baud = 9600, const std::string &delimiter = "\r\n" )

create a serial client

<table>
<thead>
<tr>
<th>name</th>
<th>name of the serial connection (e.g. &quot;/dev/ttyS0&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>baud</td>
<td>baud rate of the serial connection (e.g. 9600)</td>
</tr>
<tr>
<td>delimiter</td>
<td>string used to split lines</td>
</tr>
</tbody>
</table>

Definition at line 30 of file serial_client.cpp.

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

- goby/util/linebasedcomms/serial_client.h
- src/util/linebasedcomms/serial_client.cpp
16.22  goby::util::TCPClient Class Reference

provides a basic TCP client for line by line text based communications to a remote TCP server
#include <goby/util/linebasedcomms/tcp_client.h>
Inherits goby::util::LineBasedClient< ASIOAsyncReadStream >.

Public Member Functions

- **TCPClient**(const std::string &server, unsigned port, const std::string &delimiter="\r\n", int retry_interval=10)
  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

Friends

- class **TCPConnection**
- class LineBasedConnection< boost::asio::ip::tcp::socket >

16.22.1  Detailed Description

provides a basic TCP client for line by line text based communications to a remote TCP server
Definition at line 34 of file tcp_client.h.

16.22.2  Constructor & Destructor Documentation

16.22.2.1  goby::util::TCPClient::TCPClient**( const std::string &server, unsigned port, const std::string &delimiter = "\r\n", int retry_interval = 10 )

create a TCPClient

Parameters

<table>
<thead>
<tr>
<th>server</th>
<th>domain name or IP address of the remote server</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port of the remote server</td>
</tr>
<tr>
<td>delimiter</td>
<td>string used to split lines</td>
</tr>
</tbody>
</table>

Definition at line 28 of file tcp_client.cpp.

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

- goby/util/linebasedcomms/tcp_client.h
- src/util/linebasedcomms/tcp_client.cpp

16.23  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 <goby/util/linebasedcomms/tcp_server.h>

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
Inheritance diagram for goby::util::TCPServer:

```
goby::util::TCPServer
  └── goby::util::LineBasedInterface
```

Public Types

- typedef std::string Endpoint

Public Member Functions

- **TCPServer** (unsigned port, const std::string &delimiter="\r\n")
  - create a TCP server
- void close (const Endpoint &endpoint)
- std::string local_endpoint ()
  - string representation of the local endpoint (e.g. 192.168.1.105:54230
- const std::map< Endpoint,
  boost::shared_ptr< TCPConnection > > & connections ()

Friends

- class TCPConnection
- class LineBasedConnection< boost::asio::ip::tcp::socket >

Additional Inherited Members

16.23.1 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 49 of file tcp_server.h.

16.23.2 Constructor & Destructor Documentation

16.23.2.1 goby::util::TCPServer::TCPServer ( unsigned port, const std::string & delimiter = "\r\n" ) [inline]

create a TCP server
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>port</strong></td>
<td>port of the server (use 50000+ to avoid problems with special system ports)</td>
</tr>
<tr>
<td><strong>delimiter</strong></td>
<td>string used to split lines</td>
</tr>
</tbody>
</table>

Definition at line 56 of file tcp_server.h.

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

- goby/util/linebasedcomms/tcp_server.h
- src/util/linebasedcomms/tcp_server.cpp

16.24 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 <goby/common/logger/logger_manipulators.h>
```

Public Member Functions

- **Group** (const std::string &name="", const std::string &description="", goby::common::Colors::Color color=goby::common::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::common::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::common::Colors::Color c)
- void enabled (bool b)

16.24.1 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 69 of file logger_manipulators.h.

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

- goby/common/logger/logger_manipulators.h
16.25 GroupSetter Class Reference

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

```
#include <goby/common/logger/logger_manipulators.h>
```

Public Member Functions

- **GroupSetter** (const std::string &s)
- void **operator()** (std::ostream &os) const
- void **operator()** (goby::common::FlexOstream &os) const

16.25.1 Detailed Description

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

Definition at line 116 of file logger_manipulators.h.

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

- goby/common/logger/logger_manipulators.h
- src/common/logger/logger_manipulators.cpp

17 File Documentation

17.1 goby/moos/moos_protobuf_helpers.h File Reference

Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages.

```
#include <limits>
#include <boost/format.hpp>
#include <boost/regex.hpp>
#include <google/protobuf/io/printer.h>
#include <google/protobuf/io/tokenizer.h>
#include "goby/common/logger.h"
#include "goby/util/as.h"
#include "goby/util/binary.h"
#include "goby/util/dynamic_protobuf_manager.h"
#include "goby/moos/moos_string.h"
#include "goby/util/primitive_types.h"
#include "goby/moos/transitional/message_algorithms.h"
#include "goby/moos/transitional/message_val.h"
#include "goby/moos/protobuf/translator.pb.h"
```

Include dependency graph for moos_protobuf_helpers.h:
This graph shows which files directly or indirectly include this file:

```
Namespaces

• goby
  The global namespace for the Goby project.

Functions

• std::map<int, std::string> goby::moos::run_serialize_algorithms (const google::protobuf::Message &in, const google::protobuf::RepeatedPtrField<google::protobuf::Message> &algorithms)
• std::string goby::moos::strip_name_from_enum (const std::string &enum_value, const std::string &field_name)
• std::string goby::moos::add_name_to_enum (const std::string &enum_value, const std::string &field_name)
• bool serialize_for_moos (std::string *out, const google::protobuf::Message &msg)
• 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().
• boost::shared_ptr<google::protobuf::Message> dynamic_parse_for_moos (const std::string &in)

Variables

• const std::string goby::moos::MAGIC_PROTOBUF_HEADER = "@PB"
• goby::moos::protobuf::TranslatorEntry::ParserSerializerTechnique goby::moos::moos_technique = goby::moos::protobuf::TranslatorEntry::TECHNIQUE_PREFIXED_PROTOBUF_TEXT_FORMAT

17.1.1 Detailed Description

Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages.

Definition in file moos_protobuf_helpers.h.

17.1.2 Function Documentation

17.1.2.1 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

<table>
<thead>
<tr>
<th>in</th>
<th>std::string to parse</th>
</tr>
</thead>
</table>
```
Example Documentation

18.1 acomms/amac/amac_simple/amac_simple.cpp

```cpp
#include "goby/acomms/amac.h"
#include "goby/acomms/connect.h"
#include "goby/common/logger.h"
#include <iostream>
using goby::acomms::operator<<;

void init_transmission(const goby::acomms::protobuf::ModemTransmission& msg);

int main(int argc, char* argv[]) {
  goby::glog.set_name(argv[0]);
  goby::glog.add_stream(goby::common::logger::DEBUG1, &std::clog);
  // 1. Create a MACManager
  // goby::acomms::MACManager mac;
  // 2. Configure with a few slots
  // goby::acomms::protobuf::MACConfig cfg;
  // cfg.set_modem_id(1);
  // cfg.set_type(goby::acomms::MAC_FIXED_DECENTRALIZED);
  // goby::acomms::protobuf::ModemTransmission* slot = cfg.add_slot();
  // slot->set_src(1);
  // slot->set_rate(0);
  // slot->set_type(goby::acomms::protobuf::ModemTransmission::DATA);
  // slot->set_slot_seconds(5);
  // 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 < 150; ++i) {
    mac.do_work();
    usleep(100000);
  }
}
```

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
// 5. Add some slots [modem ids 2 & 3, and LBL ping from 1]: remember MACManager is a std::list<goby::acomms::protobuf::ModemTransmission> at heart
//
// one way we can do this
// goby::acomms::protobuf::ModemTransmission new_slot;
// new_slot.set_src(2);
// new_slot.set_rate(0);
// new_slot.set_type(goby::acomms::protobuf::ModemTransmission::DATA);
// new_slot.set_slot_seconds(5);
// mac.push_back(new_slot);
// another way
// mac.resize(mac.size()+1);
// mac.back().CopyFrom(mac.front());
// mac.back().set_src(3);
// mac.resize(mac.size()+1);
// mac.back().CopyFrom(mac.front());
// mac.back().set_type(goby::acomms::protobuf::ModemTransmission::DRIVER_SPECIFIC);
// mac.back().SetExtension(micromodem::protobuf::type,
// micromodem::protobuf::MICROMODEM_REMUS_LBL_RANGING);
// must call update after manipulating MACManager before calling do_work again.
// mac.update();
//
// 6. Run it
//
// for(;;)
// {
//     mac.do_work();
//     usleep(100000);
//     return 0;
// }

void init_transmission(const goby::acomms::protobuf::ModemTransmission& msg)
{
    std::cout << "starting transmission with these values: " << msg << std::endl;
}

18.2 acomms/chat/chat.cpp

// Copyright 2009-2016 Toby Schneider (http://gobysoft.org/index.wt/people/toby)
// GobySoft, LLC (2013-)
// Massachusetts Institute of Technology (2007-2014)
//
// This file is part of the Goby Underwater Autonomy Project Binaries
// (*The Goby Binaries*).
// The Goby Binaries are free software: you can redistribute them and/or modify
// them under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 2 of the License, or
// (at your option) any later version.
// The Goby Binaries are distributed in the hope that they 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 Goby. 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"
```cpp
#include "goby/util/as.h"
#include "goby/common/time.h"
#include "chat.pb.h"
#include <boost/lexical_cast.hpp>
#include "chat_curses.h"

using goby::util::as;
using goby::common::goby_time;

int startup_failure();
void received_data(const google::protobuf::Message&);
void received_ack(const goby::acomms::protobuf::ModemTransmission&,
                const google::protobuf::Message&);
void monitor_modem_receive(const goby::acomms::protobuf::ModemTransmission& rx_msg);
void monitor_mac(const goby::acomms::protobuf::ModemTransmission& mac_msg);

int main(int argc, char* argv[])
{
    // 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();
    }

    // Initialize logging
    goby::glog.add_stream(goby::common::logger::DEBUG1, &fout_);
    goby::glog.set_name(argv[0]);

    // bind the signals of these libraries
    bind(mm_driver_, q_manager_, mac_);

    // Initiate DCCL (libdcl)
    goby::acomms::protobuf::DCCLConfig dccl_cfg;
    dccl_->validate<ChatMessage>();

    // Initiate queue manager (libqueue)
    goby::acomms::protobuf::QueueManagerConfig q_manager_cfg;
    q_manager_cfg.set_modem_id(my_id_);
}
```
goby::acomms::protobuf::QueuedMessageEntry* q_entry = q_manager_cfg.add_message_entry();
q_entry->set_protobuf_name("ChatMessage");
#ifdef USE_FLEXIBLE_DATA_PACKET
q_entry->set_ack(false);
#endif

goby::acomms::protobuf::QueuedMessageEntry::Role* src_role = q_entry->add_role();
src_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::SOURCE_ID);
src_role->set_field("source");
goby::acomms::protobuf::QueuedMessageEntry::Role* dest_role = q_entry->add_role();
dest_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::DESTINATION_ID);
dest_role->set_field("destination");

goby::acomms::connect(&q_manager_.signal_receive, &received_data);
goby::acomms::connect(&q_manager_.signal_ack, &received_ack);

// Initiate modem driver (libmodemdriver)
//
goby::acomms::protobuf::DriverConfig driver_cfg;
driver_cfg.set_modem_id(my_id_);
driver_cfg.set_serial_port(serial_port);
#elif defined USE_FLEXIBLE_DATA_PACKET
driver_cfg.AddExtension(micromodem::protobuf::Config::nvram_cfg, "psk.packet.mod_hdr_version,1");
#endif

#if defined USE_TWO_WAY_PING

goby::acomms::connect(&mm_driver_.signal_receive, &monitor_modem_receive);
#endif

// 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::ModemTransmission my_slot;
my_slot.set_src(my_id_);
my_slot.set_dest(buddy_id_);
#endif USE_FLEXIBLE_DATA_PACKET
my_slot.set_type(goby::acomms::protobuf::ModemTransmission::DRIVER_SPECIFIC);
my_slot.set_max_frame_bytes(32);
my_slot.set_rate(1);
#endif USE_TWO_WAY_PING

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())
  {
    ChatMessage message_out;
    message_out.set_telegram(line);
    // send this message to my buddy!
    message_out.set_destination(buddy_id_);
    message_out.set_source(my_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::ModemTransmission& 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 monitor_modem_receive(const goby::acomms::protobuf::ModemTransmission& rx_msg)
{
  if(rx_msg.GetExtension(micromodem::protobuf::type) == micromodem::protobuf::MICROMODDEM_TWO_WAY_PING &&
     rx_msg.HasExtension(micromodem::protobuf::rangingReply))
    {
      const micromodem::protobuf::RangingReply& range_reply = rx_msg.GetExtension(
        micromodem::protobuf::rangingReply);
      if(range_reply.one_way_travel_time_size() > 0)
      {
        double owtt = range_reply.one_way_travel_time(0);
        curses_.post_message(range_reply.ShortDebugString());
      }
    }
}
```cpp
void received_data(const google::protobuf::Message& message_in) {
    ChatMessage typed_message_in;
    typed_message_in.CopyFrom(message_in);
    curses_.post_message(typed_message_in.source(), typed_message_in.telegram());
}

void received_ack(const goby::acomms::protobuf::ModemTransmission& ack_message,
                   const google::protobuf::Message& original_message) {
    ChatMessage typed_original_message;
    typed_original_message.CopyFrom(original_message);
    curses_.post_message(ack_message.src(),
                          std::string("{ acknowledged receiving message starting with: " +
                          typed_original_message.telegram().substr(0,5) + " }"));
}
```

### 18.3 acomms/modemdriver/driver_simple/driver_simple.cpp

```cpp
// Copyright 2009-2016 Toby Schneider (http://gobysoft.org/index.wt/people/toby)
// GobySoft, LLC (2013-)
// Massachusetts Institute of Technology (2007-2014)
//
// This file is part of the Goby Underwater Autonomy Project Binaries
// ("The Goby Binaries").
//
// The Goby Binaries are free software: you can redistribute them and/or modify
// them under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 2 of the License, or
// (at your option) any later version.
//
// The Goby Binaries are distributed in the hope that they 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 Goby. 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/util/binary.h"
#include "goby/common/logger.h"
#include "goby/acomms/connect.h"
#include <iostream>
using goby::acomms::operator<<;
void handle_data_receive(const goby::acomms::protobuf::ModemTransmission& 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<int32>(argv[2]);
cfg.set_modem_id(our_id);
goby::glog.set_name(argv[0]);
goby::glog.add_stream(goby::common::logger::DEBUG2, &std::clog);
if(argc == 4)
{
    if(boost::iequals(argv[3], "ABCDriver"))
    {
        std::cout << "Starting Example driver ABCDriver" << std::endl;
        driver = new goby::acomms::ABCDriver;
    }
}
// default to WHOI MicroModem
if(!driver)
{
    std::cout << "Starting WHOI Micro-Modem MMDriver" << std::endl;
    driver = new goby::acomms::MMDriver;
    // turn data quality factor message on
    // (example of setting NVRAM configuration)
    cfg.AddExtension(micromodem::protobuf::Config::nvram_cfg, "DQF,1");
}
goby::acomms::connect(&driver->signal_receive, &handle_data_receive);

// 2. Startup the driver
//
// driver->startup(cfg);

// 3. Initiate a transmission cycle with some data
//
goby::acomms::protobuf::ModemTransmission transmit_message;
transmit_message.set_type(goby::acomms::protobuf::ModemTransmission::DATA);
transmit_message.set_src(goby::util::as<unsigned>(our_id));
transmit_message.set_dest(goby::acomms::BROADCAST_ID);
transmit_message.set_rate(0);
transmit_message.add_frame("Hello, world!");
transmit_message.set_ack_requested(false);
std::cout << transmit_message << std::endl;

driver->handle_initiate_transmission(transmit_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 % 600))
    {
        driver->handle_initiate_transmission(transmit_message);
        // in here you can initiate more transmissions as you want
        usleep(100000);
    }
    delete driver;
    return 0;
}

// 5. Post the received data
void handle_data_receive(const goby::acomms::protobuf::ModemTransmission& data_msg)
{
    std::cout << "got a message: " << data_msg << std::endl;
}
18.4  acomms/queue/queue_simple/queue_simple.cpp

simple.proto

```protobuf
import "dccl/protobuf/option_extensions.proto";

message Simple {
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;
  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;
  required string telegram = 1 [(dccl.field).max_length=30];
}
```

queue_simple.cpp

```cpp
// Copyright 2009-2016 Toby Schneider (http://gobysoft.org/index.wt/people/toby)
// GobySoft, LLC (2013-)
// Massachusetts Institute of Technology (2007-2014)
// This file is part of the Goby Underwater Autonomy Project Binaries
// ("The Goby Binaries").
// The Goby Binaries are free software: you can redistribute them and/or modify
// them under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 2 of the License, or
// (at your option) any later version.
// The Goby Binaries are distributed in the hope that they 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 Goby. 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 "simple.pb.h"
#include "goby/util/binary.h"
#include <iostream>

using goby::acomms::operator<<;

void received_data(const google::protobuf::Message& msg);

int main() {
  // 1. Initialize the QueueManager
  //
  // 2. Push a message to a queue
  //
  // let's make a message to store in the queue
  Simple msg;
  msg.set_telegram("hello all!");
  q_manager.push_message(msg);
  std::cout << "1. pushing message to queue 1: " << msg << std::endl;
```
// see what our QueueManager contains
std::cout << "2. " << q_manager << 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

request_msg.set_max_frame_bytes(32);
request_msg.set_max_num_frames(1);
q_manager.handle_modem_data_request(&request_msg);

std::cout << "4. requesting data, got: " << request_msg << std::endl;
std::cout << "\data as hex: " << goby::util::hex_encode(request_msg.frame(0)) << std::endl;

// 4. Pass the received message to the QueueManager (same as outgoing message)
q_manager.handle_modem_receive(request_msg);
return 0;

// 5. Do something with the received message
void received_data(const google::protobuf::Message& msg)
{
    std::cout << "5. received message: " << msg << std::endl;
}
Index

API classes for the Dynamic Compact Control Language (includes writing custom encoders)., 78
API classes for the major components of the Goby--Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver)., 78

add_escape_code
    goby::common::tcolor, 86
add_queue
    goby::acomms::QueueManager, 108
AlgFunction1
    goby::transitional, 89
AlgFunction2
    goby::transitional, 89
all_message_ids
    goby::transitional::DCCLTransitionalCodec, 126
all_message_names
    goby::transitional::DCCLTransitionalCodec, 126
Application
    goby::pb::Application, 119
boost::asio::time_traits < goby::common::GobyTime >, 92
char_array2hex_string
    goby::transitional, 90
ChatCurses, 93
cpp_bool
    goby::transitional, 89
cpp_double
    goby::transitional, 89
cpp_long
    goby::transitional, 89
cpp_notype
    goby::transitional, 89
cpp_string
    goby::transitional, 89
DCCL_HEADER_NAMES
    goby::transitional, 92
DCCLCppType
    goby::transitional, 89
DCCLTransitionalCodec
    goby::transitional::DCCLTransitionalCodec, 125
DCCLType
    goby::transitional, 89
dccl_bool
    goby::transitional, 89
dccl_enum
    goby::transitional, 90
dccl_float
    goby::transitional, 90
dccl_hex
    goby::transitional, 90
dccl_int
    goby::transitional, 89
dccl_static
    goby::transitional, 89
dccl_string
    goby::transitional, 90
do_work
    goby::acomms::ABCDriver, 94
goby::acomms::ModemDriverBase, 102
goby::moos::BluefinCommsDriver, 116
goby::moos::UFldDriver, 117
flush_queue
    goby::acomms::QueueManager, 108
get
    goby::transitional::DCCLMessageVal, 122, 123
get_repeat
    goby::transitional::DCCLTransitionalCodec, 126
goby, 78
run, 80
goby/moos/moos_protobuf_helpers.h, 133
goby::transitional
cpp_bool, 89
cpp_double, 89
cpp_long, 89
cpp_notype, 89
cpp_string, 89
dccl_bool, 89
dccl_enum, 90
dccl_float, 90
dccl_hex, 90
dccl_int, 89
dccl_static, 89
dccl_string, 90
goby::Exception, 114
goby::acomms, 80
goby::acomms::ABCDriver, 94
do_work, 94
handle_initiate_transmission, 94
startup, 96
goby::acomms::MACManager, 96
signal_initiate_transmission, 97
signal_slot_start, 98
startup, 97
goby::acomms::MMDriver, 98
startup, 99
goby::acomms::ModemDriverBase, 100
do_work, 102
handle_initiate_transmission, 102
modern_read, 102
modern_start, 102
modern_write, 103
signal_data_request, 103
signal_modify_transmission, 103
signal_raw_incoming, 103
signal_raw_outgoing, 104
signal_receive, 104
signal_transmit_result, 104
INDEX 145

startup, 103

goby::acomms::QueueException, 104
goby::acomms::QueueManager, 105
  add_queue, 108
  flush_queue, 108
  handle_modem_data_request, 108
  handle_modem_receive, 108
  into, 109
  info_all, 109
  push_message, 109
  signal_ack, 109
  signal_data_on_demand, 109
  signal_expire, 110
  signal_queue_size_change, 110
  signal_receive, 110

goby::common, 83

goby::common::Colors, 110

goby::common::ConfigException, 111

goby::common::FlexNCurses, 112

goby::common::FlexOstream, 112

goby::common::TermColor, 114

goby::common::tcolor, 86
  add_escape_code, 86

goby::moos::BluefinCommsDriver, 115
  do_work, 116
  handle_initiate_transmission, 116
  startup, 116

goby::moos::UFldDriver, 117
  do_work, 117
  handle_initiate_transmission, 118
  startup, 118

goby::pb, 87

goby::pb::Application, 118
  Application, 119
  subscribe, 119
  zeromq_service, 120

goby::transitional, 87
  AlgFunction1, 89
  AlgFunction2, 89
  char_array2hex_string(), 90
  DCCL_HEADER_NAMES, 92
  DCCLCppType, 89
  DCCLType, 89
  hex_string2binary_string(), 90
  hex_string2number(), 90
  number2hex_string(), 90

goby::transitional::DCCLMessageVal, 120
  get, 122, 123
  operator bool, 123
  operator double, 123
  operator float, 124
  operator int, 124
  operator long, 124
  operator std::string, 124
  operator unsigned, 124
  set, 124

goby::transitional::DCCLTransitionalCodec, 125
  all_message_names, 126
  DCCLTransitionalCodec, 125
  get_repeat, 126
  id2name, 126
  message_count, 126
  message_var_names, 126
  name2id, 126

goby::util::LineBasedInterface, 127
  readline, 128

goby::util::SerialClient, 128
  SerialClient, 129

goby::util::TCPClient, 129

goby::util::TCPServer, 130

Group, 132
  GroupSetter, 133

handle_initiate_transmission
  goby::acomms::ABCDriver, 94
  goby::acomms::ModemDriverBase, 102
  goby::moos::BluefinCommsDriver, 116
  goby::moos::UFldDriver, 118

handle_modem_data_request
  goby::acomms::QueueManager, 108

handle_modem_receive
  goby::acomms::QueueManager, 108

hex_string2binary_string
  goby::transitional, 90

hex_string2number
  goby::transitional, 90

id2name
  goby::transitional::DCCLTransitionalCodec, 126

info
  goby::acomms::QueueManager, 109
  goby::acomms::QueueManager, 109

info_all
  goby::acomms::QueueManager, 109

message_count
  goby::transitional::DCCLTransitionalCodec, 126

message_var_names
  goby::transitional::DCCLTransitionalCodec, 126

modem_read
  goby::acomms::ModemDriverBase, 102

modem_start
  goby::acomms::ModemDriverBase, 102

modem_write
  goby::acomms::ModemDriverBase, 103

moos_protobuf_helpers.h
  parse_for_moos, 134

name2id
  goby::transitional::DCCLTransitionalCodec, 126

number2hex_string
  goby::transitional, 90

operator bool
  goby::transitional::DCCLMessageVal, 123

Generated on Tue Nov 29 2016 23:10:37 for Goby v2 by Doxygen
operator double
goby::transitional::DCCLMessageVal, 123
operator float
goby::transitional::DCCLMessageVal, 124
operator int
goby::transitional::DCCLMessageVal, 124
operator long
goby::transitional::DCCLMessageVal, 124
operator std::string
goby::transitional::DCCLMessageVal, 124
operator unsigned
goby::transitional::DCCLMessageVal, 124
parse_for_moos
moos_protobuf_helpers.h, 134
push_message
goby::acomms::QueueManager, 109
readline
goby::util::LineBasedInterface, 128
run
goby, 80
SerialClient
goby::util::SerialClient, 129
set
goby::transitional::DCCLMessageVal, 124
signal_ack
goby::acomms::QueueManager, 109
signal_data_on_demand
goby::acomms::QueueManager, 109
signal_data_request
goby::acomms::ModemDriverBase, 103
signal_expire
goby::acomms::QueueManager, 110
signal_initiate_transmission
goby::acomms::MACManager, 97
signal_modify_transmission
goby::acomms::ModemDriverBase, 103
signal_queue_size_change
goby::acomms::QueueManager, 110
signal_raw_incoming
goby::acomms::ModemDriverBase, 103
signal_raw_outgoing
goby::acomms::ModemDriverBase, 104
signal_receive
goby::acomms::ModemDriverBase, 104
goby::acomms::QueueManager, 110
signal_slot_start
goby::acomms::MACManager, 98
signal_transmit_result
goby::acomms::ModemDriverBase, 104
startup
goby::acomms::ABCDriver, 96
goby::acomms::MACManager, 97
goby::acomms::MMDriver, 99
goby::acomms::ModemDriverBase, 103
goby::moos::BluefinCommsDriver, 116
goby::moos::UFldDriver, 118
subscribe
goby::pb::Application, 119
TCPClient
goby::util::TCPClient, 130
TCPServer
goby::util::TCPServer, 131
zeromq_service
goby::pb::Application, 120