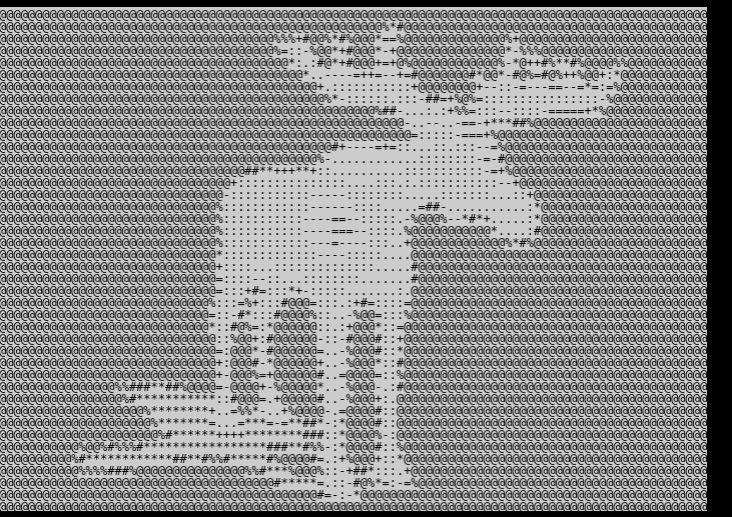
DCCL4 in MOOS

Advances in Data Marshalling for Slow (Marine) Links



Toby Schneider

GobySoft, LLC

Falmouth, MA, USA

gobysoft.org
aquatic software

MOOS-DAWG 2024, Cambridge, MA

DCCL Recap

The Dynamic Compact Control Language contains:

- an interface description language (IDL) with static (compile-time) units of measure support
- a flexible data marshalling (source encoding/decoding) C++ library (which allows user extension).

Open source (LGPL) implementation

(http://libdccl.org, https://github.com/GobySoft/dccl/).

pAcommsHandler in Goby3 (https://goby.software) provides MOOS interface

Interface Description Language (IDL)

```
message CommandMessage
{


 required int32 destination = 1

 optional string description = 2

 enum SonarPower { NOMINAL = 10; LOW = 5; OFF = 0; }
 optional SonarPower sonar_power = 10;
 required double speed = 11


 repeated int32 waypoint_depth = 12


}
```

Base Protobuf message example

DCCL acts as an "invisible" extension to Google Protocol Buffers (Protobuf).

- Code unaware of DCCL can still use the same messages (e.g. internal vehicle data)
- Extensions provide additional information (e.g. numeric bounding).

Protobuf is widely used and provides syntax checking and multi-language support (C++, Python, Java, C, ...)

IDL: Field bounds

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

message CommandMessage
{
 option (dccl.msg) = { id:  125  max_bytes:  32  codec_version: 3 }

 required int32 destination = 1
                [(dccl.field)  = { max: 31 min: 0 in_head: true }];
 optional string description = 2
                [(dccl.field).omit = true];
 enum SonarPower { NOMINAL = 10; LOW = 5; OFF = 0; }
 optional SonarPower sonar_power = 10;
 required double speed = 11
                [(dccl.field) = { units { base_dimensions: "LT^-1" }
                                  max: 2.0  min: -0.5 precision: 1 }];
 repeated int32 waypoint_depth = 12
                [(dccl.field) = { units { base_dimensions: "L" }
                                  max: 40  min: 0 max_repeat: 4 }];
}
```

DCCL message example

Numerics are bounded by

- *max*: largest value
- *min*: smallest value
- *precision*: *decimal* digits of precision preserved *or* *resolution* (non-powers-of-10)

Message level information

- id: identifies message (CommandMessage == 125)
- max_bytes: enforced upper bound for MTU targetting

DCCL4 Features

Major new features in DCCL4:

- Support for Protobuf "oneof" construct: *allows smaller max_bytes for some messages.*
- Dynamic conditions: Runtime selection of field inclusion and bounds: *allows smaller messages in some cases.*
- Message hashing: DCCL messages must match at sender and receiver: *hashing allows for static or dynamic checking of this requirement.*

Full backwards compatibility with DCCL3 (via codec_version setting).

Demo repository

All the demos in this talk can be run on your own:

- https://github.com/GobySoft/moos-dawg-24
  (or)
- docker run -it gobysoft/moos-dawg-2024
  # cd /opt/moos-dawg-24

Demos use mix of shell (“dccl” tool), C++, and Python to show various ways you can interact with DCCL.

Demo 1: oneof (~= union)

```
message Command {
  option (.dccl.msg) = { codec_version: 4 ... };
   message ThermoclineParams { ... }
   message FrontParams { ... }
   message PlaneParams { ... }
   enum Action {
    MEASURE_THERMOCLINE = 1;
    SEARCH_FOR_OCEAN_FRONT = 2;
    SEARCH_FOR_MISSING_AIRPLANE = 3;
  }
  required .moos.dawg.Command.Action action = 1;
  oneof parameters {
    .moos.dawg.Command.ThermoclineParams thermocline = 2;   or
    .moos.dawg.Command.FrontParams front = 3;               or
    .moos.dawg.Command.PlaneParams airplane = 4;            or unset
  }
}
```

Demo 1: oneof: Why?

Reduces maximum message size

- Easier to target small MTU values (e.g. 32B acomms message)

```
== 125: moos.dawg.CommandWithoutOneOf {0x02c62506a7b294a2} ==
Actual maximum size of message: 6 bytes / 48 bits
```

```
======== 124: moos.dawg.Command {0x3dabf069f95d1b7b} ========
Actual maximum size of message: 4 bytes / 32 bits
```

Note that this doesn't typically change actual encoded size:

- Original: **action: MEASURE_THERMOCLINE thermocline { max_search_depth: 100 }**
- Encoded (hex)
  - without oneof: fad402 (**3 bytes**)
  - oneof: f8a105 (**3 bytes**)

Demo 2: Dynamic Conditions

Dynamic conditions:

- Lua scripts embedded in DCCL definition that allow for changing encoding based on other message fields.
- required_if, omit_if, only_if: Dynamic presence or absence of field.
- max, min: Dynamic bounds

A few important notes:

- These do not change the max_bytes of the message (no way at compile time to know effect of dynamics).
- dynamic min/max are subset of static min/max.

Demo 2: Dynamic Conditions

```
message Status {
  option (.dccl.msg) = { codec_version: 4 ...  };
  enum VehicleType {
    USV = 1;   AUV_WITH_CTD = 2;    AUV_WITH_SONAR = 3;
    DEEP_AUV = 4;  }
  required .moos.dawg.Status.VehicleType type = 1;
  message Location {
    required int32 x = 1 [ ... ];
    required int32 y = 2 [ ... ];
    optional int32 depth = 3 [(.dccl.field) = {
      min: 0, max: 5000, resolution: 0.1
      dynamic_conditions {
        omit_if: "root.type == 'USV'"
        max: "if root.type == 'DEEP_AUV' then return 5000
else return 100 end"
      }
    }];
 required .moos.dawg.Status.Location location = 2; ...
```

Demo 2: Dynamic Conditions

```
message Status {
    ...
 optional double temperature = 3 [(.dccl.field) = {
    min: 0, max: 40, resolution: 0.01
    dynamic_conditions {
      only_if: "this.type == 'AUV_WITH_CTD' or this.type ==
'DEEP_AUV'"
    }
  }];
  optional .moos.dawg.Status.AirplaneDetection airplane_de-
tection = 4 [(.dccl.field) = {
    dynamic_conditions {
      only_if: "this.type == 'AUV_WITH_SONAR'"
    }
  }];
}
```

D2: Dynamic Conditions: Why?

Why?

- Reduced message size in some cases

```
• Original: type: AUV_WITH_CTD location { x: 500 y: 700 depth: 50 } temperature: 12.3
• Encoded (hex)
  • without dynamic conditions: fe11a4985333e09900 (9 bytes)
  • dynamic conditions: fc11a49853f53913 (8 bytes)

• Original: type: AUV_WITH_SONAR location { x: 500 y: 700 depth: 10 } airplane_detection { detected_
• Encoded (hex)
  • without dynamic conditions: fe12a498530b0000a2804201 (12 bytes)
  • dynamic conditions: fc12a4985365440185 02 (10 bytes)
```

- Improved feedback for UIs using commands (e.g. goby_liaison will hide fields based on dynamic conditions)

Demo 3: Hash

What if we update our Command?

- All senders and receivers must have same .proto message (or it will be decoded as incorrect data).

```
--- command.proto   2024-08-01 16:58:45.071139498 -0400
+++ updated_command.proto   2024-08-01 16:58:49.106890224 -0400
@@ -18,6 +18,7 @@
         MEASURE_THERMOCLINE = 1;
         SEARCH_FOR_OCEAN_FRONT = 2;
         SEARCH_FOR_MISSING_AIRPLANE = 3;
+        FIND_LIFE_ON_MARS = 4;
     }

     message ThermoclineParams
```

Demo 3: Hash

What if we update our Command?

- We can check hash via the command line (useful for adding to CMake to ensure we don't accidentally cause backwards compatibility issue):

```
• Original Hash: 0x3dabf069f95d1b7b
• Updated Hash: 0xa4cb90cf16c8027e
```

- and/or we can add to the message itself for runtime check:

```
--- updated_command.proto   2024-08-01 16:58:49.106890224 -0400
+++ updated_command_with_hash.proto 2024-08-01 16:57:54.902237354 -0400
@@ -4,7 +4,7 @@

 package moos.dawg;

-message Command
+message UpdatedCommandWithHash
 {
     option (dccl.msg) = {
         id: 124
@@ -68,4 +68,7 @@
         FrontParams front = 3;     // for action == SEARCH_FOR_OCEAN_FRONT
         PlaneParams airplane = 4;  // for action == SEARCH_FOR_MISSING_AIRPLAI
     }
+
+    required uint32 hash = 5
+        [(dccl.field).codec = "dccl.hash", (dccl.field).max = 0xFFFF];
 }
```

Demo 3: Hash

- Runtime check: takes message space but ensures (within probability based on hash size) we don't have a mismatch:

```
• Original: action: MEASURE_THERMOCLINE thermocline { max_search_depth: 100 } hash: 0
• Encoded (hex)
  • f8a1057613 (5 bytes)
• Decoded:
  • Using original message: action: MEASURE_THERMOCLINE thermocline { max_search_depth:
    100 } hash: 19928
  • Using updated message: Exception: Message f8a1057613 failed to decode. Reason:
    Message: moos.dawg.UpdatedCommandWithHash: Hash value mismatch. Expected: 21567,
    received: 19928. Ensure both sender and receiver are using identical DCCL
    definitions
```

Demo 4: MOOS

Simple scenario to put this all in context for MOOS users:

- Two platforms connected by a slow link (acoustic modem, satcomms, etc.):
  - topside (mm1.moos)
  - vehicle (mm2.moos)
- Topside (pCommand) sends a Command message to the vehicle
- Vehicle (pStatus) responds with a Status message
- pAcommsHandler (from Goby3) loads DCCL messages and handles link-layer (UDP in this case) and queuing.

Acknowledgments

- Many thanks to Davide Fenucci (National Oceanography Centre, UK) for contributing the *oneof* functionality.
- JaiaRobotics (Bristol, RI) for funding development of the *hash* functionality.
- JPAnalytics, LLC (Falmouth, MA) for funding development of the *dynamic conditions* functionality.
- Thanks to all the contributors of DCCL for feedback, bug fixes, etc.: https://github.com/GobySoft/dccl/graphs/contributors
- Please consider getting involved!