

BOE Code generator

efficient, customized code directly from YAML specifications

"Built for Flexibility, Streamlined for Maintenance, Optimized for Performance"

Motivation

Handwritten code per version

- New version, New code
- (BOEv-1.x, code-1) (BOEv-2.x, code-2) ... (BOEv-n.x, code-n)

Handwritten code with inheritance/C++ features.

- base-code and extending it with inheritance to support BOEv-version.x
- reduced handwritten work and flexibility at the cost of latency.

Code generation

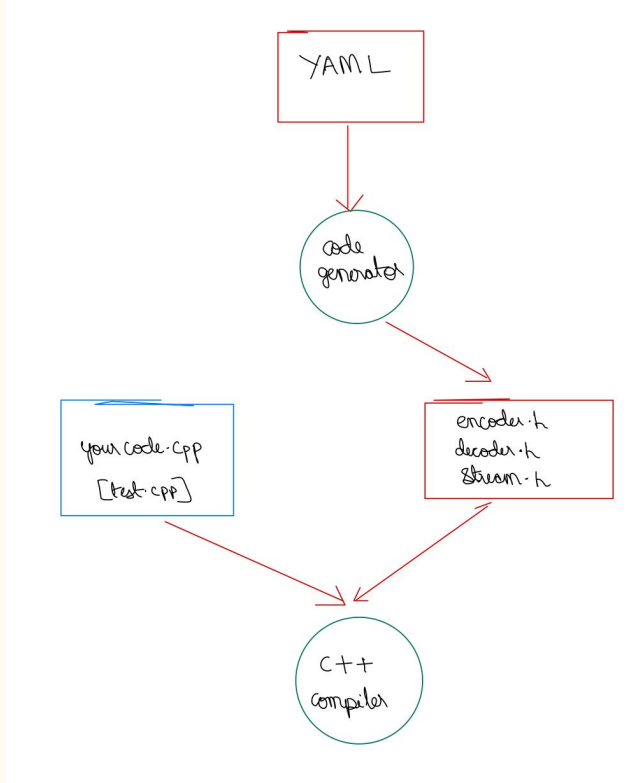
- spec driven code generation of type safe C++ code.
- spec driven run-time message validation.

Approach	Manual per Version	Inheritance-based	Code Generator Approach
Description	Separate code for each BOE version	Base code extended with inheritance for each new version	Code generator handles versions automatically
Flexibility (New protocol-version)	Zero, new code	Moderate, but extensions can be complex	High, only YAML config changes needed
Latency	High	Moderate	High
Maintenance (Error Management or New C++ feature)	Hard, version-specific code edits	Complex, maintenance for each extension	Centralized in YAML & templates
Consistency	Proportional to manual efforts	Partial	Automated

BOE Code generator

BOE Code Generator is a python program

- takes YAML spec file for BOE protocol BOEv-x.y
- generates efficient C++ code for BOEv-x.y
 - BOE_Msgs.h
 - BOE_Encoder.h
 - BOE_Decoder.h
 - BOE_Stream.h
 - Common file: BOE_Handler.h, BOE_Common.h
 - test.cpp for unit testing
- Using templating engine to keep meta-codes (templates) as modular as possible using macros, control, and other features of templating engine.



Introduction

Padala SSSS V Sri Vishnu Subhash

Experience

- B.Tech CSE'23 @IIT Palakkad
 - C++ SDE @Arista Networks
 - L2O RA @UNC, @UT Austin
 - AI R&D Engineer @Webaverse
 - Open source @HuggingFace
-

Code-generator

—

Code overview

Generated code

- BOE_Msgs.h
 - Has generated class definitions for all messages and groups along with accessors, constructors, reset member functions according to the spec file.
- BOE_Encoder.h
 - Has generated Encoder class definition, having multiple overloaded public encode functions which takes message class object and converts it into payload using multiple private encode functions.
- BOE_Decoder.h
 - Has generated Decoder class definition, having decode function which takes the input payload, populates corresponding message class object.
 - It allows user to access the decoded message class object.
 - It automatically calls handler on the decoded message object.
- BOE_Stream.h
 - Has generated Ostream operator for all the messages and groups, which pretty print the message class object.

Static code

- BOE_Handler.h
 - The handler class definition has multiple functions, which needs to be completed by the user.
 - By default it calls the custom ostream operator on the message object.
- BOE_Common.h
 - Contains helper classes and functions which could be used across different protocols.
 - Status class is defined which will be populated during the encoder and decoder if any errors happen during their tasks like buffer overflow, corrupted payload, invalid message type, etc.
- test.cpp
 - Contains unit tests

YAML overview

Modular code structure to easily identify changes required.

- tpl.BOE_Msgs.h.jinja2
 - renderStringLenEnum
 - renderOptionalEnumerators
 - renderEnumerators
 - renderMembers
 - renderGroupAccessors
 - renderAccessors
 - renderConstructorList
 - renderEnumeratorOptions
 - renderReset
 - renderGroups
 - renderMessages
- tpl.BOE_Encoder.h.jinja2
 - renderBaseEncoders
 - renderMembers
 - renderGroups
 - renderMessages
- tpl.BOE_Decoder.h.jinja2
 - renderHelperMembers
 - renderHelpers
 - renderMembers
 - renderGroups
 - renderMessages
- tpl.BOE_Stream.h.jinja2
 - renderHelpers
 - renderMembers
 - renderGroups
 - renderMessages

Repetitive and Intuitive structure across all the macros

- to handle different cases such as data types, constraints, and groups.

```
templates > tpl.BOE_Msgs.cpp.jinja2
9 > {% macro renderCamelCase(field) -%}...
11 {% endmacro -%}
12
13 > {% macro renderMemberName(field) -%}...
15 {% endmacro -%}
16
17 > {% macro renderOptionalEnumerators(data) -%}...
38 {% endmacro -%}
39
40 > {% macro renderStringLenEnum( data ) -%}...
49 {% endmacro -%}
50
51 > {% macro renderEnumerators() -%}...
71 {% endmacro -%}
72
73 > {% macro renderMembers(data) -%}...
122 {% endmacro -%}
123
124 > {% macro renderGroupAccessors(member, group) -%}...
140 {% endmacro -%}
141
142 > {% macro renderAccessors(data) -%}...
182 {% endmacro -%}
183
184 > {% macro renderConstructorList(prefix, data) -%}...
248 {% endmacro -%}
249
250 > {% macro renderReset(prefix, data) -%}...
370 {% endmacro -%}
```

```
templates > tpl.BOE_Stream.cpp.jinja2
1 > {% macro renderCamelCase(field) -%}...
3 {% endmacro -%}
4
5 > {% macro renderMemberName(field) -%}...
7 {% endmacro -%}
8
9 > {% macro renderMembers(prefix, data) -%}...
53 {% endmacro -%}
54
55 > {% macro renderGroups(data) -%}...
62 {% endmacro -%}
63
64 > {% macro renderMessages(data) -%}...
71 {% endmacro -%}
72
73 > {% macro renderHelpers() -%}...
83 {% endmacro -%}
```

```
templates > tpl.BOE_Encoder.cpp.jinja2
6 > {% macro renderCamelCase(field) -%}...
8 {% endmacro -%}
9
10 > {% macro renderMemberName(field) -%}...
12 {% endmacro -%}
13
14 > {% macro renderBaseEncoders() -%}...
98 {% endmacro -%}
99
100 > {% macro renderMembers(prefix, data) -%}...
142 {% endmacro -%}
143
144 > {% macro renderGroups(data) -%}...
157 {% endmacro -%}
158
159 > {% macro renderMessages(data) -%}...
167 {% endmacro -%}
168
```

```
templates > tpl.BOE_Decoder.cpp.jinja2
1 > {% macro renderCamelCase(field) -%}...
3 {% endmacro -%}
4
5 > {% macro renderMemberName(field) -%}...
7 {% endmacro -%}
8
9 > {% macro renderHelperMembers() -%}...
18 {% endmacro -%}
19
20 > {% macro renderMembers(prefix, data) -%}...
106 {% endmacro -%}
107
108 > {% macro renderGroups(data) -%}...
118 {% endmacro -%}
119
120 > {% macro renderMessages(data) -%}...
130 {% endmacro -%}
131
132 > {% macro renderHelpers() -%}...
189 {% endmacro -%}
190
```

Overview

```
! boe.yaml
1 > Messages: ...
47 |
48 > Groups: ...
95
96 > Optionals: ...
116
117 > Enums: ...
141
142 > Fields:|...
```

```
52
53 > RepeatingGroupsOfNewOrderCross: ...
68
69 > NumberOfUnits: ...
73
74 > UnitSequences: ...
84
85   NumberOfReturnBitfields:
86     Fields:
87       ReturnBitfield:
88
89   ReturnBitfields:
90     Fields:
91       ParamGroupLength:
92       ParamGroupType: ParamGroupTypesEnum::RETURNBITFIELDS
93       MessageType:
94       NumberOfReturnBitfields:
95       ParamGroups:
96         NumberOfReturnBitfields:
```

```
144   Fields:
145     StartOfMessage:
146       dataType: uint16_t
147     MessageLength:
148       dataType: uint16_t
149     MessageType:
150       dataType: uint8_t
151     MatchingUnit:
152       dataType: uint8_t
153     SequenceNumber:
154       dataType: uint32_t
155     SessionSubID:
156       dataType: char*
157       dataLen: 4
158     Username:
159       dataType: char*
160       dataLen: 4
161     Password:
162       dataType: char*
163       dataLen: 10
```

```
119   Enums:
120     Messages:
121       data:
122         LOGINREQUEST: 0x37
123         LOGINRESPONSE: 0x24
124         NEWORDERCROSS: 0xA1
125         dataType: uint8_t
126     ParamGroupTypes:
127       data:
128         RETURNBITFIELDS: 0x81
129         UNITSEQUENCES: 0x80
130       dataType: uint8_t
131     StartOfMessage:
132       data:
133         StartOfMessage: 0xbaba
134       dataType: uint16_t
135     MatchingUnit:
136       data:
137         MatchingUnit: 0x0
138       dataType: uint8_t
139     SequenceNumber:
140       data:
141         SequenceNumber: 0x0
142       dataType: uint32_t
```

```
98   Optionals:
99     NewOrderCross:
100       0:
101         Symbol:
102         MaturityDate:
103         StrikePrice:
104         PutonCall:
105         ExecInst:
106         AttributedQuote:
107         TargetPartyID:
108         PreventMatch:
109       1:
110         AutoMatch:
111         AutoMatchPrice:
112         LastPriority:
113         Account:
114         CMTANumber:
115         ClearingAccount:
116         RoutingFirmID:
117         ClearingOptionalData:
```

```
1   Messages:
2     LoginRequest:
3       Fields:
4         StartOfMessage: StartOfMessageEnum::STARTOFMESSAGE
5         MessageLength:
6         MessageType: MessagesEnum::LOGINREQUEST
7         MatchingUnit: MatchingUnitEnum::MATCHINGUNIT
8         SequenceNumber: SequenceNumberEnum::SEQUENCENUMBER
9         SessionSubID:
10        Username:
11        Password:
12        NumberOfParamGroups:
13          ParamGroups:
14            UnitSequences:
15            ReturnBitfields:
16
17 > NewOrderCross: ...
```

Highlights schema

Modular Design

- Intuitive to understand, Meaningful to interpret, Easy to edit
 - Similar structure at all levels.
- Detangling specification from presence
 - Same group or Field can be present in multiple messages and groups.
 - Keeping Group and Field specifications away from presence at a common place, allows us to quickly edit configuration, with minimal changes.
- Yet allowing constraints on Fields per Message, and per Group
 - Constant fields
 - Restricted fields
 - Optional fields

Where and How to specify atomic fields?

Where and How to specify common enumerators?

```
! boe.yaml
1 > Messages: ...
47
48 > Groups: ...
95
96 > Optionals: ...
116
117 > Enums: ...
141
142 > Fields: | ...
```

Highlights schema

Where to specify a message or group?

- message specification in Messages section, and group specification in Groups section
- Both message and group specification have the same schema.

How do we specify a Message and Group?

- List all fields present in Fields section, along with CONSTRAINTS.
 - Fields with Enum are considered as **constant fields**.
 - Field with List are considered as **restricted fields**.
 - Fields with Bitmap are considered as **optional fields**.
 - Fields with Param Groups section is considered as **group count fields**.
 - The ParamGroups section lists all the groups which could be present.
 - One group indicates special case **group**.
 - More than one group indicates **param groups**.
 - Group specification having Group indicates **nested group**.
- metaData has information
 - whether it has optional fields
 - which bitmap the optional fields are associated with
 - which message the group with optional field is part of
- Constraints are specified at message-level or group-level
 - A field can different constraints based on where they are present.
- According to BOE repeating groups can only have optional fields, and they have one-to-one mapping to message.

Common schema to write the presence of Param Groups, Groups, Repeating Groups.

Custom names for group-count fields and many other fields.

```
NewOrderCross:
  Fields:
    StartOfMessage: StartOfMessageEnum::STARTOFMESSAGE
    MessageLength:
    MessageType: MessagesEnum::NEWORDERCROSS
    MatchingUnit: MatchingUnitEnum::MATCHINGUNIT
    SequenceNumber:
    CrossID:
    CrossType: [{Unknown: 0}, {AIM: 1}, {QCC: 2}, {SAM: 3}, {P...
    CrossPrioritization: [{Buy: 1}, {Sell: 2}, {Unknown: 0}]
```

```
Groups:
>   NumberOfNewOrderCrossBitfields: ...
>   RepeatingGroupsOfNewOrderCross: ...
>   NumberofUnits: ...
>   UnitSequences: ...

  NumberOfReturnBitfields:
    Fields:
      ReturnBitfield:

  ReturnBitfields:
    Fields:
      ParamGroupLength:
      ParamGroupType: ParamGroupType
      MessageType:
      NumberOfReturnBitfields:
      ParamGroups:
        NumberOfReturnBitfields:
```

Highlights schema

Bitfields and Optional Fields

- How is field specified as optional field?
 - By the value passed, for example `NumberOfNewOrderCrossBitfields`
 - Same interface for message or group
- How to specify repeating group with optional fields?
 - By setting `metaData`, specifying of which message, and bitfield it depends on.
 - According to BOE repeating groups are associated with a unique message only, and repeating groups can only have optional fields.
- How to specify which bitfield and bit will represent which optional field?
 - Optionals section specifies mapping b/w Optional fields and their associated BitfieldIdx, BitIdx
- How to specify which optional fields should be passed at the end-of-message?
 - Simply append them at the end-of message configuration
 - This solves two issues
 - Not passing optional fields part of repeating group
 - Those which are disabled by the BOE
 - Merits of solution
 - Easy to edit configuration or enable new optional fields.
 - Keeps jinja2 simple, which makes it easy to manage and maintain.
- How are we handling presence of group with optional fields, and constraints on not to pass the information at the end?
 - Explained above

Divide the problem into two pieces and attack them individually to keep the Jinja2 template simple.

```
1 Messages:
2   NewOrderCross:
3     Fields:
4       CrossType: [{Unknown: 0}, {AIM: 1}, {OCC: 2}, {
5       CrossPrioritization: [{Buy: 1}, {Sell: 2}, {Unk
6       Price:
7       OrderQty:
8       NumberOfNewOrderCrossBitfields:
9       ParamGroups:
10        NumberOfNewOrderCrossBitfields:
11        GroupCnt:
12        ParamGroups:
13        RepeatingGroupsOfNewOrderCross:
14        Symbol: NumberOfNewOrderCrossBitfields
15        MaturityDate: NumberOfNewOrderCrossBitfields
16        StrikePrice: NumberOfNewOrderCrossBitfields
17        PutOrCall: NumberOfNewOrderCrossBitfields
18        ExecInst: NumberOfNewOrderCrossBitfields
19        AttributedQuote: NumberOfNewOrderCrossBitfields
20        PreventMatch: NumberOfNewOrderCrossBitfields
21        TargetPartyID: NumberOfNewOrderCrossBitfields
22        AutoMatch: NumberOfNewOrderCrossBitfields
23        AutoMatchPrice: NumberOfNewOrderCrossBitfields
24        LastPriority: NumberOfNewOrderCrossBitfields
25        RoutingFirmID: NumberOfNewOrderCrossBitfields
```

```
3 Groups:
4   NumberOfNewOrderCrossBitfields:
5     Fields:
6       NewOrderCrossBitfield:
7
8   RepeatingGroupsOfNewOrderCross:
9     Fields:
10      Side: [{Buy: 1}, {Sell: 2}, {Unknown: 0}]
11      AllocQty:
12      ClOrdID:
13      Capacity: [{Unknown: 'Z'}, {Customer: 'C'}, {Mark
14      OpenClose: [{Unknown: 'Z'}, {Open: 'O'}, {Close:
15      GiveUpFirmID:
16      Account: NumberOfNewOrderCrossBitfields
17      CMTANumber: NumberOfNewOrderCrossBitfields
18      ClearingAccount: NumberOfNewOrderCrossBitfields
19      ClearingOptionalData: NumberOfNewOrderCrossBitfie
20      metaData:
21        hasOptional: NumberOfNewOrderCrossBitfields
22        message: NewOrderCross
```

```
98 Optionals:
99   NewOrderCross:
100     0:
101       Symbol:
102       MaturityDate:
103       StrikePrice:
104       PutOrCall:
105       ExecInst:
106       AttributedQuote:
107       TargetPartyID:
108       PreventMatch:
109       1:
```

Highlights YAML config file

Constant fields can be specified per message.

Allowing flexibility across messages.

Code generator

- Optimizes then as compile time constants using constexpr.
- automatically removes the setter for constant fields
- runtime validation for constant fields according to spec

```
! boe.yaml
1  Messages:
2    LoginRequest:
3      Fields:
4        StartOfMessage: StartOfMessageEnum::STARTOFMESSAGE
5        MessageLength:
6        MessageType: MessagesEnum::LOGINREQUEST
7        MatchingUnit: MatchingUnitEnum::MATCHINGUNIT
8        SequenceNumber: SequenceNumberEnum::SEQUENCENUMBER
9        SessionSubID:
10       Username:
11       Password:
12       NumberOfParamGroups:
13         ParamGroups:
14           UnitSequences:
15           ReturnBitFields:
16
17 > NewOrderCross: ...
47
```

Highlights

Enumerators to handle constant and constrained fields.

What are different types of enumerators?

- Global and within class, optional enumerators to represent different use cases.
- Common enumerators which are shared across different messages and groups are placed global scope.

Support for Constrained Fields

- Message and Group class set Constrained Fields with unknown value when the message object is initialized without any arguments, If the unknown value is not modified the encoder is equipped detect this and mark it as error.
- Encoder and decoder ensures that payload is not corrupted, and populates status object accordingly.
- Unknown is used to ensure that payload is corrupted.
- Message object is by default initialized with Unknown for constrained types.

Code generator takes the responsibility to identifying ParamGroup or Group based on spec.

- abstracts the setter for Group Size, and automating it inside the setter for adding group element.

```
74 UnitSequences:
75   Fields:
76     ParamGroupLength: # might need to handle the name
77     ParamGroupType: ParamGroupTypeEnum:UNITSEQUENCE
78     NoUnspecifiedUnitReplay: [{True : 0x01},
79                               {False: 0x00},
80                               {Unknown : 0x02} ]
81     NoUnspecifiedUnitReplay:
```

```
! boeyaml
116
117 Enums:
118   Messages:
119     data:
120       LOGINREQUEST: 0x37
121       LOGINRESPONSE: 0x24
122       NEWORDERCROSS: 0x41
123       dataType: uint8_t
124   ParamGroupTypes:
125     data:
126       RETURNBITFIELDS: 0x81
127       UNITSEQUENCES: 0x80
128       dataType: uint8_t
129   StartOfMessage:
130     data:
131       StartOfMessage: 0xbaba
132       dataType: uint16_t
```

BOE Protocol

—

Introduction

- Unlike FIX, BOE has **No delimiters, No Field Names, Only Field Data**
- Login Request Payload
 - BABA3D00**37**0000000000303030315445535454455354494E47000000030F00800102014
ABB0100020000000008008125030041050B00812C06004107004000
- New Order Cross Payload
 - BABAB000**41**00640000004E5A315637424A5F4163636570744275790000003131204E0000000000
006400000002413003003164000000514C37535A37435F6167656E63790000000000004343444546
4700000000000000003228000000514C394B3855565F636F6E747261310000000000464F4142434
4270200005758595A323C000000514C39543559445F636F6E747261320000000000464F41424344
7B0000005758595A303051306B41000043444546
- Complex structure with fields, optional fields, nested repeating groups, param groups
- Multiple protocol versions, each with distinct differences.

Each message has 10 byte header

Layout Identifier



Field	Offset	Length	Data Type	Description
<i>StartOfMessage</i>	0	2	Binary	Must be 0xBA 0xBA.
<i>MessageLength</i>	2	2	Binary	Number of bytes for the message, including this field but not including the two bytes for the <i>StartOfMessage</i> field.
<i>MessageType</i>	4	1	Binary	Message type.
<i>MatchingUnit</i>	5	1	Binary	<p>The matching unit which created this message. Matching units in BOE correspond to matching units on Multicast PITCH.</p> <p>For session level traffic, the unit is set to 0. For messages from Member to Cboe, the unit must be 0.</p>
<i>SequenceNumber</i>	6	4	Binary	<p>The sequence number for this message. Messages from Cboe to Member are sequenced distinctly per matching unit.</p> <p>Messages from Member to Cboe are sequenced across all matching units with a single sequence stream.</p> <p>Member can optionally send a 0 sequence number on all messages from Member to Cboe. Cboe highly recommends that Members send sequence numbers on all inbound messages.</p>

Login Request

Field	Offset	Length	Data Type
<i>StartOfMessage</i>	0	2	Binary
<i>MessageLength</i>	2	2	Binary
<i>MessageType</i>	4	1	Binary
<i>MatchingUnit</i>	5	1	Binary
<i>SequenceNumber</i>	6	4	Binary
<i>SessionSubID</i>	10	4	Alphanumeric
<i>Username</i>	14	4	Alphanumeric
<i>Password</i>	18	10	Alphanumeric
<i>NumberOfParam Groups</i>	28	1	Binary
<i>ParamGroup₁</i>			
...			
<i>ParamGroup_n</i>			

Field Name	Hexadecimal	Notes
<i>StartOfMessage</i>	BA BA	Start of message bytes.
<i>MessageLength</i>	3D 00	61 bytes
<i>MessageType</i>	37	Login Request
<i>MatchingUnit</i>	00	Always 0 for inbound messages
<i>SequenceNumber</i>	00 00 00 00	Always 0 for session level messages
<i>SessionSubID</i>	30 30 30 31	0001
<i>Username</i>	54 45 53 54	TEST
<i>Password</i>	54 45 53 54 49 4E 47 00 00 00	TESTING
<i>NumberOfParam Groups</i>	03	3 parameter groups
<i>ParamGroupLength</i>	0F 00	15 bytes for this parameter group
<i>ParamGroupType</i>	80	0x80 = Unit Sequences
<i>NoUnspecified</i>	01	True (replay only specified units)
<i>UnitReplay</i>		
<i>NumberOfUnits</i>	02	Two unit/sequence pairs to follow;
<i>UnitNumber₁</i>	01	Unit 1
<i>UnitSequence₁</i>	4A BB 01 00	Last received sequence of 113,482
<i>UnitNumber₂</i>	02	Unit 2
<i>UnitSequence₂</i>	00 00 00 00	Last received sequence of 0
<i>ParamGroupLength</i>	08 00	8 bytes for this parameter group
<i>ParamGroupType</i>	81	0x81 = Return Bitfields
<i>MessageType</i>	25	0x25 = Order Acknowledgment
<i>NumberOfReturn Bitfields</i>	03	3 bitfields to follow
<i>ReturnBitfield₁</i>	00	No bitfields from byte 1
<i>ReturnBitfield₂</i>	41	Symbol, Capacity
<i>ReturnBitfield₃</i>	05	Account, ClearingAccount
<i>ParamGroupLength</i>	0B 00	11 bytes for this parameter group
<i>ParamGroupType</i>	81	0x81 = Return Bitfields
<i>MessageType</i>	2C	0x2C = Order Execution
<i>NumberOfReturn Bitfields</i>	06	6 bitfields to follow
<i>ReturnBitfield₁</i>	00	No bitfields from byte 1
<i>ReturnBitfield₂</i>	41	Symbol, Capacity
<i>ReturnBitfield₃</i>	07	Account, ClearingFirm, ClearingAccount
<i>ReturnBitfield₄</i>	00	No bitfields from byte 4
<i>ReturnBitfield₅</i>	40	BaseLiquidityIndicator

Terminology

Message is a set of fields.

Atomic field are fields which cannot be broken down further.

- StartOfMessage, GrpCnt, ReturnBitfield are atomic.
- Groups, ParamGroups, Bitfields are not atomic.

Groups

$$G(N, \{A_1, A_2, \dots A_m\}, K) = \{\{A_1, A_2, \dots A_m\}, \{A_1, A_2, \dots A_m\} \dots N \text{ times}\}$$

<i>NumberOfUnits</i>	02	Two unit/sequence pairs to follow;
<i>UnitNumber₁</i>	01	Unit 1
<i>UnitSequence₁</i>	4A BB 01 00	Last received sequence of 113,482
<i>UnitNumber₂</i>	02	Unit 2
<i>UnitSequence₂</i>	00 00 00 00	Last received sequence of 0

A set of atomic fields, repeated k times.

Nested Groups

Can groups have groups?

$G_2(N_2, \{X, G_1(N_1, \{A, B\}), Z\})$

$G_2(N_2, \{X, G_1, Z\})$

<i>NumberOfParam Groups</i>	03	3 parameter groups
<i>ParamGroupLength</i>	0F 00	15 bytes for this parameter group
<i>ParamGroupType</i>	80	0x80 = Unit Sequences
<i>NoUnspecified UnitReplay</i>	01	True (replay only specified units)
<i>NumberOfUnits</i>	02	Two unit/sequence pairs to follow;
<i>UnitNumber₁</i>	01	Unit 1
<i>UnitSequence₁</i>	4A BB 01 00	Last received sequence of 113,482
<i>UnitNumber₂</i>	02	Unit 2
<i>UnitSequence₂</i>	00 00 00 00	Last received sequence of 0
<i>ParamGroupLength</i>	08 00	8 bytes for this parameter group
<i>ParamGroupType</i>	81	0x81 = Return Bitfields
<i>MessageType</i>	25	0x25 = Order Acknowledgment
<i>NumberOfReturn Bitfields</i>	03	3 bitfields to follow
<i>ReturnBitfield₁</i>	00	No bitfields from byte 1
<i>ReturnBitfield₂</i>	41	<i>Symbol, Capacity</i>
<i>ReturnBitfield₃</i>	05	<i>Account, ClearingAccount</i>
<i>ParamGroupLength</i>	0B 00	11 bytes for this parameter group
<i>ParamGroupType</i>	81	0x81 = Return Bitfields
<i>MessageType</i>	2C	0x2C = Order Execution
<i>NumberOfReturn Bitfields</i>	06	6 bitfields to follow
<i>ReturnBitfield₁</i>	00	No bitfields from byte 1
<i>ReturnBitfield₂</i>	41	<i>Symbol, Capacity</i>
<i>ReturnBitfield₃</i>	07	<i>Account, ClearingFirm, ClearingAccount</i>
<i>ReturnBitfield₄</i>	00	No bitfields from byte 4
<i>ReturnBitfield₅</i>	40	<i>BaseLiquidityIndicator</i>
<i>ReturnBitfield₆</i>	00	No bitfields from byte 6

Param Groups

Group whose repeating units can vary.

$$G(N, \{F_1, F_2\}, \{A_1, A_2, A_3\})$$

$$G(2, \{F_1, F_2\}, \{A_1, A_2, A_3\})$$

can be

$$\{\{F_1, F_2\}, \{F_1, F_2\}\} \text{ Or }$$

$$\{\{F_1, F_2\}, \{A_1, A_2, A_3\}\} \text{ Or }$$

$$\{\{A_1, A_2, A_3\}, \{A_1, A_2, A_3\}\}$$

<i>NumberOfParam Groups</i>	03	3 parameter groups
<i>ParamGroupLength</i>	0F 00	15 bytes for this parameter group
<i>ParamGroupType</i>	80	0x80 = Unit Sequences
<i>NoUnspecified UnitReplay</i>	01	True (replay only specified units)
<i>NumberOfUnits</i>	02	Two unit/sequence pairs to follow;
<i>UnitNumber₁</i>	01	Unit 1
<i>UnitSequence₁</i>	4A BB 01 00	Last received sequence of 113,482
<i>UnitNumber₂</i>	02	Unit 2
<i>UnitSequence₂</i>	00 00 00 00	Last received sequence of 0
<i>ParamGroupLength</i>	08 00	8 bytes for this parameter group
<i>ParamGroupType</i>	81	0x81 = Return Bitfields
<i>MessageType</i>	25	0x25 = Order Acknowledgment
<i>NumberOfReturn Bitfields</i>	03	3 bitfields to follow
<i>ReturnBitfield₁</i>	00	No bitfields from byte 1
<i>ReturnBitfield₂</i>	41	<i>Symbol, Capacity</i>
<i>ReturnBitfield₃</i>	05	<i>Account, ClearingAccount</i>

Optional Fields & Bitfields

<i>ClOrdID</i>	10	20	Text	<p>Corresponds to <i>ClOrdID</i> (11) in Cboe FIX.</p> <p>ID chosen by the client. Characters in the ASCII range 33-126 are allowed, except for comma, semicolon, pipe, the 'at' symbol (@) and double quotes.</p> <p>If the <i>ClOrdID</i> matches a live order, the order will be rejected as duplicate.</p> <p>Note: Cboe only enforces uniqueness of <i>ClOrdID</i> values among currently live orders, which includes long-lived, persisting GTC/GTD orders. However, we strongly recommend that you keep your <i>ClOrdID</i> values unique.</p>
<i>Side</i>	30	1	Alphanumeric	<p>Corresponds to <i>Side</i> (54) in Cboe FIX.</p> <p>1 = Buy 2 = Sell</p>
<i>OrderQty</i>	31	4	Binary	<p>Corresponds to <i>OrderQty</i> (38) in Cboe FIX.</p> <p>Order quantity. System limit is 999,999 contracts.</p>
<i>NumberOfNewOrderBitfields</i>	35	1	Binary	Bitfield identifying which bitfields are set. Field values must be appended to the end of the message.
<i>NewOrderBitfield¹</i>	36	1	Binary	Bitfield identifying fields to follow.
....				
<i>NewOrderBitfieldⁿ</i>		1	Binary	Last bitfield.
<i>Optional fields. . .</i>				

Byte	Bit	Field	
1	1	<i>ClearingFirm</i>	●
	2	<i>ClearingAccount</i>	●
	4	<i>Price</i>	●
	8	<i>ExecInst</i>	●
	16	<i>OrdType</i>	●
	32	<i>TimeInForce</i>	●
	64	<i>MinQty</i>	●
	128	<i>MaxFloor</i>	●
2	1	<i>Symbol</i>	R
	2	<i>SymbolSfx</i>	
	4	<i>Currency</i>	
	8	<i>IdSource</i>	
	16	<i>SecurityId</i>	
	32	<i>SecurityExchange</i>	
	64	<i>Capacity</i>	R
	128	<i>RoutingInst</i>	●

Optional Fields & Bitfields

Bitfields control which optional fields are expected.

Optional fields part of groups are not supplied at end-of-message.

Others will be appended at the end.

<i>NumberOfNewOrder Bitfields</i>	04	Four bitfields to follow
<i>NewOrderBitfield1</i>	04	<i>Price</i>
<i>NewOrderBitfield2</i>	C1	<i>Symbol, Capacity, RoutingInst</i>
<i>NewOrderBitfield3</i>	01	<i>Account</i>
<i>NewOrderBitfield4</i>	17	<i>MaturityDate, StrikePrice, PutOrCall, OpenClose</i>
<i>Price</i>	70 17 00 00 00 00 00 00	<i>0.60</i>
<i>Symbol</i>	4D 53 46 54 00 00 00 00	<i>MSFT</i>
<i>Capacity</i>	43	<i>C = Customer</i>
<i>RoutingInst</i>	52 00 00 00	<i>R = Routable</i>
<i>Account</i>	44 45 46 47 00 00 00 00 00 00	<i>DEFG</i>
<i>MaturityDate</i>	EF DB 32 01	<i>2011-03-19</i>
<i>StrikePrice</i>	98 AB 02 00 00 00 00 00	<i>17.50</i>
<i>PutOrCall</i>	31	<i>1 = Call</i>
<i>OpenClose</i>	4F	<i>O = Open</i>

Optional Fields & Bitfields

<i>NumberOfReturn Bitfields</i>	02	Two bitfields to follow
<i>ReturnBitfield1</i>	00	No fields from byte 1
<i>ReturnBitfield2</i>	41	<i>Symbol, Capacity</i>
<i>GroupCnt</i>	03 00	Two repeating groups to follow
<i>CIOrdID</i>	4E 5A 31 56 37 47 4E 5F 61 67 65 6E 63 79 00 00 00 00 00 00	NZ1V7GN_agency
<i>OrderID</i>	02 C0 91 A2 94 AB 78 04	2G4GYK000002 (base 36)
<i>Capacity</i>	43	C = Customer
<i>CIOrdID</i>	4E 5A 31 56 37 4B 46 5F 63 6F 6E 74 72 61 31 00 00 00 00 00	NZ1V7KF_contra1
<i>OrderID</i>	03 C0 91 A2 94 AB 78 04	2G4GYK000003 (base 36)
<i>Capacity</i>	46	F = Firm
<i>CIOrderID</i>	4E 5A 31 56 37 4E 48 5F 63 6F 6E 74 72 61 32 00 00 00 00 00	NZ1V7NH_contra2
<i>OrderID</i>	04 C0 91 A2 94 AB 78 04	2G4GYK000004 (base 36)
<i>Capacity</i>	46	F = Firm
<i>Symbol</i>	30 30 51 30 6B 41 00 00	00Q0kA

<i>NumberOfReturn Bitfields</i>	47	1	Binary
<i>ReturnBitfield¹</i>	48	1	Binary
...			
<i>ReturnBitfieldⁿ</i>		1	Binary
<i>GroupCnt</i>		2	Binary
<i>Repeating Groups of...</i>			
<i>CIOrdId</i>		20	Text
<i>OrderId</i>		8	Binary
<i>Side (Optional)</i>		1	Alphanumeric
<i>AllocQty (Optional)</i>		4	Binary
<i>Capacity (Optional)</i>		1	Alpha
<i>OpenClose (Optional)</i>		1	Alphanumeric
<i>GiveUpFirmID (Optional)</i>		4	Alpha
<i>Account (Optional)</i>		16	Text
<i>CMTANumber (Optional)</i>		4	Binary
<i>ClearingAccount (Optional)</i>		4	Text
<i>Optional fields. . .</i>			