```
1  ---------- MODULE Skeen ----------
   The specification of Skeen's protocol for atomic multicast; see Section III of the DSN2019 paper
   "White-Box Atomic Multicast" by Alexey Gotsman, Anatole Lefort, and Gregory Chockler.
7  EXTENDS Naturals, Sequences, FiniteSets, TLC
8  ----------
9  Injective(f) ≜ ∀ a, b ∈ DOMAIN f : (a ≠ b) ⇒ (f[a] ≠ f[b])

11 Max(a, b) ≜ IF a > b THEN a ELSE b
12 ----------
13 CONSTANTS
14     Msg,      the set of messages, ranged over by m
15     Proc,     the set of processes, ranged over by p
16     Dest      Dest[m] ⊆ Proc: the set of destination processes of m ∈ Msg

18 ASSUME
19     ∧ Dest ∈ [Msg → SUBSET Proc]

21 Priority ≜ CHOOSE f ∈ [Proc → 1 .. Cardinality(Proc)] : Injective(f)
22 ----------
23 VARIABLES
24     clock,      clock[p]: the clock at process p ∈ Proc
25     phase,      phase[p][m]: the phase of the message m ∈ Msg at process p ∈ Proc
26     localTS,    localTS[p][m]: the local ts of the message m ∈ Msg at process p ∈ Proc
27     globalTS,   globalTS[p][m]: the global ts of the message m ∈ Msg at process p ∈ Proc
28     delivered,  delivered[p][m]: has m ∈ Msg been delivered at process p ∈ Proc
29     incoming,   incoming[p] ⊆ Message (defined below): the incoming channel of process p ∈ Proc
30     sent        sent ⊆ Msg: the set of messages that have been multicast; only for TLC

32 pvars ≜ ⟨clock, phase, localTS, globalTS, delivered⟩
33 vars ≜ ⟨clock, phase, localTS, globalTS, delivered, incoming, sent⟩
34 ----------
35 MaxCounter ≜ Cardinality(Msg) ∗ Cardinality(Proc)

37 TS ≜ [c : 0 .. MaxCounter, p : Proc]  c for counter

39 GT(u, v) ≜  Is u > v?
40     ∨ u.c > v.c
41     ∨ ∧ u.c = v.c
42       ∧ Priority[u.p] > Priority[v.p]

44 MaxV(vs) ≜ CHOOSE u ∈ vs : ∀ v ∈ vs : u ≠ v ⇒ GT(u, v)
45 ----------
46 Message ≜ [type : {"MULTICAST"}, m : Msg]
47     ∪ [type : {"PROPOSE"}, m : Msg, p : Proc, lts : TS]

49 Send(msg) ≜  Send msg ∈ Message to its destination processes
50     incoming′ = [p ∈ Proc ↦
51         IF p ∈ Dest[msg.m] THEN incoming[p] ∪ {msg}
```

```
52                                  ELSE  incoming[p]]

    Send smsg ∈ Message to its destination processes and remove rmsg ∈ Message from
    incoming[sender]

    Precondition: sender ∈ Dest[msg.m]
59  SendAndRemove(smsg, sender, rmsg) ≜
60      incoming' = [p ∈ Proc ↦
61          IF p = sender THEN (incoming[sender] ∪ {smsg}) \ {rmsg}
62                        ELSE IF p ∈ Dest[smsg.m] THEN incoming[p] ∪ {smsg}
63                                                 ELSE incoming[p]]
64  ----------------------------------------------------------------------------
65  TypeOK ≜
66      ∧  clock      ∈ [Proc → 0 .. MaxCounter]
67      ∧  phase      ∈ [Proc → [Msg → {"START", "PROPOSED", "COMMITTED"}]]
68      ∧  localTS    ∈ [Proc → [Msg → TS]]
69      ∧  globalTS   ∈ [Proc → [Msg → TS]]
70      ∧  delivered  ∈ [Proc → [Msg → BOOLEAN ]]
71      ∧  incoming   ∈ [Proc → SUBSET Message]
72      ∧  sent       ⊆ Msg
73  ----------------------------------------------------------------------------
74  Init ≜
75      ∧ clock      = [p  ∈ Proc ↦ 0]
76      ∧ phase      = [p  ∈ Proc ↦ [m ∈ Msg ↦ "START"]]
77      ∧ localTS    = [p  ∈ Proc ↦ [m ∈ Msg ↦ [c ↦ 0, p ↦ p]]]
78      ∧ globalTS   = [p  ∈ Proc ↦ [m ∈ Msg ↦ [c ↦ 0, p ↦ p]]]
79      ∧ delivered  = [p  ∈ Proc ↦ [m ∈ Msg ↦ FALSE]]
80      ∧ incoming   = [p ∈ Proc ↦ {}]
81      ∧ sent       = {}
82  ----------------------------------------------------------------------------
83  Multicast(m) ≜  Multicast m ∈ Msg
84      ∧ m ∈ Msg \ sent
85      ∧ sent' = sent ∪ {m}
86      ∧ Send([type ↦ "MULTICAST", m ↦ m])
87      ∧ UNCHANGED pvars

89  Propose(p) ≜  When p ∈ Proc receives a MULTICAST for some m ∈ Msg
90      ∃ msg ∈ incoming[p] :
91          ∧ msg.type = "MULTICAST"
92          ∧ LET m ≜ msg.m
93            IN  ∧ Assert(p ∈ Dest[m], "p should be one of the destination process of m")
94                ∧ clock' = [clock EXCEPT ![p] = @ + 1]
95                ∧ localTS' = [localTS EXCEPT ![p][m] = [c ↦ clock'[p], p ↦ p]]
96                ∧ phase' = [phase EXCEPT ![p][m] = "PROPOSED"]
97                ∧ SendAndRemove([type ↦ "PROPOSE", m ↦ m, p ↦ p,
98                                 lts  ↦ localTS'[p][m]], p, msg)
```

```
                ∧ UNCHANGED ⟨globalTS, delivered, sent⟩

Deliver(p) ≜  When p ∈ Proc receives all PROPOSE for some m ∈ Msg
   ∃ m ∈ Msg :
      LET msgofm ≜ {msg ∈ incoming[p] : msg.type = "PROPOSE" ∧ msg.m = m}
          destofm ≜ {msg.p : msg ∈ msgofm}
          ltsofm ≜ {msg.lts : msg ∈ msgofm}
      IN   ∧ destofm = Dest[m]
           ∧ globalTS' = [globalTS EXCEPT ![p][m] = MaxV(ltsofm)]
           ∧ clock' = [clock EXCEPT ![p] = Max(clock[p], globalTS'[p][m].c)]
           ∧ phase' = [phase EXCEPT ![p][m] = "COMMITTED"]
           ∧ LET readym ≜ {rm ∈ Msg :
                              ∧ phase'[p][rm] = "COMMITTED"
                              ∧ delivered[p][rm] = FALSE
                              ∧ ∀ pm ∈ Msg :
                                  phase'[p][pm] = "PROPOSED"
                                      ⇒ GT(localTS[p][pm], globalTS'[p][rm])}
             IN   delivered' = [delivered EXCEPT ![p] = [pm ∈ Msg ↦
                                  IF pm ∈ readym THEN TRUE ELSE @[pm]]]
                  TODO: deliver in globalTS[p][m] order (using deliver sequence)
           ∧ UNCHANGED ⟨localTS, sent, incoming⟩
------------------------------------------------------------------------
Next ≜
   ∨ ∃ m ∈ Msg : Multicast(m)
   ∨ ∃ p ∈ Proc :
      ∨ Propose(p)
      ∨ Deliver(p)

Spec ≜ Init ∧ □[Next]_vars
------------------------------------------------------------------------
Invariant: Global timestamps are unique for each m ∈ Msg; see Section III.
UniqueGTS ≜
   ∀ p ∈ Proc, m1, m2 ∈ Msg :
      (m1 ≠ m2 ∧ phase[p][m1] = "COMMITTED" ∧ phase[p][m2] = "COMMITTED")
          ⇒ globalTS[p][m1] ≠ globalTS[p][m2]

Invariant: Each m ∈ Msg is assigned a single global timestamp.
SameGTS ≜
   ∀ p1, p2 ∈ Proc, m ∈ Msg :
      (phase[p1][m] = "COMMITTED" ∧ phase[p2][m] = "COMMITTED")
          ⇒ globalTS[p1][m] = globalTS[p2][m]

TODO: Invariant: atomic multicast
------------------------------------------------------------------------
THEOREM TypeTheorem ≜ Spec ⇒ □TypeOK
```

151 THEOREM $UniqueGTSTheorem \triangleq Spec \Rightarrow \Box UniqueGTS$

153 THEOREM $SameGTSTheorem \triangleq Spec \Rightarrow \Box SameGTS$

154

* Modification History
* Last modified Sat *Jul* 24 07:18:44 *CST* 2021 by *hengxin*
* Last modified *Thu Jul* 08 20:18:10 *CST* 2021 by *eric*
* Created Sat *Jun* 26 16:04:39 *CST* 2021 by *eric*