```
1  ---------- MODULE AJupiter ----------
   Specification of the Jupiter protocol presented by Hagit Attiya and others.
5  EXTENDS Integers, OT, TLC, FunctionUtils
6  ----------
7  CONSTANTS
8      Client,      the set of client replicas
9      Server,      the (unique) server replica
10     Char,        set of characters allowed
11     InitState    the initial state of each replica

13 Replica ≜ Client ∪ {Server}

15 List ≜ Seq(Char ∪ Range(InitState))    all possible lists/strings
16 MaxLen ≜ Cardinality(Char) + Len(InitState)  the max length of lists in any states;
17     We assume that all inserted elements are unique.

19 ClientNum ≜ Cardinality(Client)
20 Priority ≜ CHOOSE f ∈ [Client → 1 .. ClientNum] : Injective(f)
21 ----------
22 ASSUME
23     ∧ Range(InitState) ∩ Char = {}
24     ∧ Priority ∈ [Client → 1 .. ClientNum]
25 ----------
   The set of all operations (the positions are indexed from 1.)
29 Rd ≜ [type : {"Rd"}]
30 Del ≜ [type : {"Del"}, pos : 1 .. MaxLen]
31 Ins ≜ [type : {"Ins"}, pos : 1 .. (MaxLen + 1), ch : Char, pr : 1 .. ClientNum]  pr: priority

33 Op ≜ Ins ∪ Del   Now we don't consider Rd operations.
34 ----------
   Messages between the Server and the Clients.
38 Msg ≜ [c : Client, ack : Int, op : Op ∪ {Nop}] ∪   messages sent to the Server from a client c ∈ Client
39       [ack : Int, op : Op ∪ {Nop}]   messages broadcast to Clients from the Server
40 ----------
41 VARIABLES
       For the client replicas:
45     cbuf,    cbuf[c]: buffer (of operations) at the client c ∈ Client
46     crec,    crec[c]: the number of new messages have been received by the client c ∈ Client
47                  since the last time a message was sent
       For the server replica:
51     sbuf,    sbuf[c]: buffer (of operations) at the Server, one per client c ∈ Client
52     srec,    srec[c]: the number of new messages have been ..., one per client c ∈ Client
       For all replicas:
56     state,   state[r]: state (the list content) of replica r ∈ Replica
       For communication
```

```
60      cincoming,    cincoming[c]: incoming channel at the client c ∈ Client
61      sincoming,    incoming channel at the Server
        For model checking:
65      chins         a set of chars to insert
66  ----------------------------------------------------------------------
67  vars ≜ ⟨chins, cbuf, crec, sbuf, srec, cincoming, sincoming, state⟩
68  ----------------------------------------------------------------------
69  comm ≜ INSTANCE CSComm WITH Msg ← Msg
70  ----------------------------------------------------------------------
71  TypeOK ≜
72      ∧   cbuf ∈ [Client → Seq(Op ∪ {Nop})]
73      ∧   crec ∈ [Client → Int]
74      ∧   sbuf ∈ [Client → Seq(Op ∪ {Nop})]
75      ∧   srec ∈ [Client → Int]
76      ∧   state ∈ [Replica → List]
77      ∧   comm!TypeOK
78      ∧   chins ∈ SUBSET Char
79  ----------------------------------------------------------------------
80  Init ≜
81      ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
82      ∧ crec = [c ∈ Client ↦ 0]
83      ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
84      ∧ srec = [c ∈ Client ↦ 0]
85      ∧ state = [r ∈ Replica ↦ InitState]
86      ∧ comm!Init
87      ∧ chins = Char
88  ----------------------------------------------------------------------
    Client c ∈ Client issues an operation op.
92  DoOp(c, op) ≜
93          ∧ state′ = [state EXCEPT ![c] = Apply(op, @)]
94          ∧ cbuf′ = [cbuf EXCEPT ![c] = Append(@, op)]
95          ∧ crec′ = [crec EXCEPT ![c] = 0]
96          ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])

98  DoIns(c) ≜
99      ∃ ins ∈ {op ∈ Ins : op.pos ∈ 1 .. (Len(state[c]) + 1) ∧ op.ch ∈ chins ∧ op.pr = Priority[c]} :
100         ∧ DoOp(c, ins)
101         ∧ chins′ = chins \ {ins.ch}   We assume that all inserted elements are unique.
102         ∧ UNCHANGED ⟨sbuf, srec⟩

104 DoDel(c) ≜
105     ∃ del ∈ {op ∈ Del : op.pos ∈ 1 .. Len(state[c])} :
106         ∧ DoOp(c, del)
107         ∧ UNCHANGED ⟨chins, sbuf, srec⟩
```

```
Do(c) ≜
    ∨ DoIns(c)
    ∨ DoDel(c)
```
Client $c \in Client$ receives a message from the *Server*.
```
Rev(c) ≜
    ∧ comm!CRev(c)
    ∧ crec' = [crec EXCEPT ![c] = @ + 1]
    ∧ LET m ≜ Head(cincoming[c])
          cBuf ≜ cbuf[c]   the buffer at client c ∈ Client
          cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
          xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
          xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
      IN  ∧ cbuf' = [cbuf EXCEPT ![c] = xcBuf]
          ∧ state' = [state EXCEPT ![c] = Apply(xop, @)]   apply the transformed operation xop
    ∧ UNCHANGED ⟨chins, sbuf, srec⟩
```
The *Server* receives a message.
```
SRev ≜
    ∧ comm!SRev
    ∧ LET m ≜ Head(sincoming)   the message to handle with
          c ≜ m.c               the client c ∈ Client that sends this message
          cBuf ≜ sbuf[c]        the buffer at the Server for client c ∈ Client
          cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
          xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
          xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
      IN  ∧ srec' = [cl ∈ Client ↦
                       IF cl = c
                       THEN srec[cl] + 1   receive one more operation from client c ∈ Client
                       ELSE 0]   reset srec for other clients than c ∈ Client
          ∧ sbuf' = [cl ∈ Client ↦
                       IF cl = c
                       THEN xcBuf   transformed buffer for client c ∈ Client
                       ELSE Append(sbuf[cl], xop)]   store transformed xop into other clients' bufs
          ∧ state' = [state EXCEPT ![Server] = Apply(xop, @)]   apply the transformed operation
          ∧ comm!SSend(c, [cl ∈ Client ↦ [ack ↦ srec[cl], op ↦ xop]])
    ∧ UNCHANGED ⟨chins, cbuf, crec⟩
```

```
Next ≜
    ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
    ∨ SRev
```
Fairness: There is no requirement that the clients ever generate operations.
```
Fairness ≜
    WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))

Spec ≜ Init ∧ □[Next]_vars   ∧ Fairness
```

Quiescent Consistency (QC)

$QC \triangleq$

$comm!EmptyChannel \Rightarrow Cardinality(Range(state)) = 1$

THEOREM $Spec \Rightarrow \Box QC$

* Modification History
* Last modified *Tue Dec* 04 18:35:18 *CST* 2018 by *hengxin*
* Created Sat *Jun* 23 17:14:18 *CST* 2018 by *hengxin*