```
┌──────────────────────── MODULE AJupiter ────────────────────────┐
 Specification of the Jupiter protocol presented by Hagit Attiya and others.
 EXTENDS JupiterInterface
├──────────────────────────────────────────────────────────────────┤
 Messages between the Server and the Clients.
 Msg ≜ [c : Client, ack : Int, op : Op ∪ {Nop}] ∪   messages sent to the Server from a client c ∈ Client
       [ack : Int, op : Op ∪ {Nop}]   messages broadcast to Clients from the Server
├──────────────────────────────────────────────────────────────────┤
 VARIABLES
    For the client replicas:
    cbuf,        cbuf[c]: buffer (of operations) at the client c ∈ Client
    crec,        crec[c]: the number of new messages have been received by the client c ∈ Client
                          since the last time a message was sent
    For the server replica:
    sbuf,        sbuf[c]: buffer (of operations) at the Server, one per client c ∈ Client
    srec,        srec[c]: the number of new messages have been ..., one per client c ∈ Client
    For all replicas:
    state,     state[r]: state (the list content) of replica r ∈ Replica
    For communication
    cincoming,   cincoming[c]: incoming channel at the client c ∈ Client
    sincoming,   incoming channel at the Server
    For model checking:
    chins      a set of chars to insert
├──────────────────────────────────────────────────────────────────┤
 vars ≜ ⟨chins, cbuf, crec, sbuf, srec, cincoming, sincoming, state⟩
├──────────────────────────────────────────────────────────────────┤
 comm ≜ INSTANCE CSComm  WITH Msg ← Msg
├──────────────────────────────────────────────────────────────────┤
 TypeOK ≜
     ∧ cbuf ∈ [Client → Seq(Op ∪ {Nop})]
     ∧ crec ∈ [Client → Int]
     ∧ sbuf ∈ [Client → Seq(Op ∪ {Nop})]
     ∧ srec ∈ [Client → Int]
     ∧ state ∈ [Replica → List]
     ∧ comm!TypeOK
     ∧ chins ∈ SUBSET Char
├──────────────────────────────────────────────────────────────────┤
 Init ≜
     ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
     ∧ crec = [c ∈ Client ↦ 0]
     ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
     ∧ srec = [c ∈ Client ↦ 0]
     ∧ state = [r ∈ Replica ↦ InitState]
```

```
      ∧ comm!Init
      ∧ chins = Char
├──────────────────────────────────────────────────────────────────────────────┤
 Client c ∈ Client issues an operation op.
DoOp(c, op) ≜
        ∧ state′ = [state EXCEPT ![c] = Apply(op, @)]
        ∧ cbuf′ = [cbuf EXCEPT ![c] = Append(@, op)]
        ∧ crec′  = [crec EXCEPT ![c]  = 0]
        ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])

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

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

Do(c) ≜
       ∨ DoIns(c)
       ∨ DoDel(c)
 Client c ∈ 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   $\land\ srec' = [cl \in Client \mapsto$
                          IF $cl = c$
                           THEN $srec[cl] + 1$   receive one more operation from client $c \in Client$
                           ELSE $0]$   reset $srec$ for other clients than $c \in Client$
             $\land\ sbuf' = [cl \in Client \mapsto$
                          IF $cl = c$
                           THEN $xcBuf$   transformed buffer for client $c \in Client$
                           ELSE $Append(sbuf[cl], xop)]$   store transformed $xop$ into other clients' bufs
             $\land\ state' = [state$ EXCEPT $![Server] = Apply(xop, @)]$   apply the transformed operation
             $\land\ comm!SSend(c, [cl \in Client \mapsto [ack \mapsto srec[cl], op \mapsto xop]])$
     $\land$ UNCHANGED $\langle chins, cbuf, crec \rangle$
-----------------------------------------------------------------------------
$Next \triangleq$
    $\lor\ \exists\, c \in Client : Do(c) \lor Rev(c)$
    $\lor\ SRev$

Fairness: There is no requirement that the clients ever generate operations.
$Fairness \triangleq$
    $\mathrm{WF}_{vars}(SRev \lor \exists\, c \in Client : Rev(c))$

$Spec \triangleq Init \land \Box[Next]_{vars}$   $\land\ Fairness$
-----------------------------------------------------------------------------
Quiescent Consistency ($QC$)
$QC \triangleq$
    $comm!EmptyChannel \Rightarrow Cardinality(Range(state)) = 1$

THEOREM $Spec \Rightarrow \Box QC$
=============================================================================
\* Modification History
\* Last modified Tue Dec 04 19:34:10 CST 2018 by hengxin
\* Created Sat Jun 23 17:14:18 CST 2018 by hengxin
```