MODULE *Zab*

This is the formal specification for the *Zab* consensus algorithm,
which means *Zookeeper* Atomic *Broadcast*.

This work is driven by *Flavio P*. Junqueira,"*Zab*: High-performance broadcast for primary-backup systems"

EXTENDS *Integers*, *FiniteSets*, *Sequences*, *Naturals*, *TLC*

The set of server identifiers
CONSTANT *Server*

The set of requests that can go into history
CONSTANT *Value*

Server states
It is unnecessary to add state ELECTION, we can own it by setting *leaderOracle* to Null.
CONSTANTS *Follower*, *Leader*, *ProspectiveLeader*

Message types
CONSTANTS *CEPOCH*, *NEWEPOCH*, *ACKE*, *NEWLEADER*, *ACKLD*, *COMMITLD*, *PROPOSE*, *ACK*, C

the maximum round of epoch (initially {0, 1, 2}) currently not used
CONSTANT *Epoches*

Return the maximum value from the set S
$Maximum(S) \triangleq$ IF $S = \{\}$ THEN -1
ELSE CHOOSE $n \in S : \forall\, m \in S : n \geq m$

Return the minimum value from the set S
$Minimum(S) \triangleq$ IF $S = \{\}$ THEN -1
ELSE CHOOSE $n \in S : \forall\, m \in S : n \leq m$

$Quorums \triangleq \{Q \in \text{SUBSET } Server : Cardinality(Q) * 2 > Cardinality(Server)\}$
ASSUME $QuorumsAssumption \triangleq \land \forall\, Q \in Quorums : Q \subseteq Server$
$\land \forall\, Q1,\ Q2 \in Quorums : Q1 \cap Q2 \neq \{\}$

$None \triangleq$ CHOOSE $v : v \notin Value$

$NullPoint \triangleq$ CHOOSE $p : p \notin Server$

The server's *state*(*Follower*, *Leader*, *ProspectiveLeader*).
VARIABLE *state*

The leader's epoch or the last new epoch proposal the follower *acknowledged*(*f.p in paper*).
VARIABLE *currentEpoch*

The last new leader proposal the follower *acknowledged*(*f.a in paper*).
VARIABLE *leaderEpoch*

The identifier of the leader for followers.
VARIABLE *leaderOracle*

The history of servers as the sequence of transactions.
VARIABLE *history*

The messages repersenting requests and responses sent from one server to another.
msgs[*i*][*j*] means the input buffer of server *j* from server *i*.
VARIABLE *msgs*

The set of followers who has successfully sent *CEPOCH* to pleader in pleader.
VARIABLE *cepochRecv*

The set of followers who has successfully sent *ACK*-E to pleader in pleader.
VARIABLE *ackeRecv*

The set of followers who has successfully sent *ACK*-LD to pleader in pleader.
VARIABLE *ackldRecv*

ackIndex[*i*][*j*] means leader *i* has received how many *ACK* messages from follower *j*.
So *ackIndex*[*i*][*i*] is not used.
VARIABLE *ackIndex*

currentCounter[*i*] means the count of transactions client requests leader.
VARIABLE *currentCounter*

sendCounter[*i*] means the count of transactions leader has broadcast.
VARIABLE *sendCounter*

initialHistory[*i*] means the initial history of leader *i* in epoch *currentEpoch*[*i*].
VARIABLE *initialHistory*

commitIndex[*i*] means leader/follower *i* should commit how many proposals and sent *COMMIT* messages.
It should be more formal to add variable *applyIndex*/*deliverIndex* to represent the prefix entries of the history
that has applied to state machine, but we can tolerate that *applyIndex*(*deliverIndex here*) = *commitIndex*.
This does not violate correctness. (increases monotonically)
VARIABLE *commitIndex*

commitIndex[*i*] means leader *i* has committed how many proposals and sent *COMMIT* messages.
VARIABLE *committedIndex*

Hepler matrix for follower to stop sending *CEPOCH* to pleader in followers.
Because *CEPOCH* is the sole message which follower actively sends to pleader.
VARIABLE *cepochSent*

the maximum epoch in *CEPOCH* pleader received from followers.
VARIABLE *tempMaxEpoch*

the maximum *leaderEpoch* and most up-to-date history in *ACKE* pleader received from followers.

VARIABLE *tempMaxLastEpoch*

VARIABLE *tempInitialHistory*

the set of all broadcast messages whose tpye is proposal that any leader has sent, only used in verifying properties. So the variable will only be changed in transition *LeaderBroadcast*1.

VARIABLE *proposalMsgsLog*

$serverVars \triangleq \langle state, currentEpoch, leaderEpoch, leaderOracle, history, commitIndex\rangle$
$leaderVars \triangleq \langle cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter, sendCounter, initialHistory, co$
$tempVars \triangleq \langle tempMaxEpoch, tempMaxLastEpoch, tempInitialHistory\rangle$

$vars \triangleq \langle serverVars, msgs, leaderVars, tempVars, cepochSent, proposalMsgsLog\rangle$

$LastZxid(his) \triangleq \text{IF } Len(his) > 0 \text{ THEN } \langle his[Len(his)].epoch, his[Len(his)].counter\rangle$
$\text{ELSE } \langle -1, -1\rangle$

Add a message to $msgs -$ add a message m to $msgs[i][j]$
$Send(i, j, m) \triangleq msgs' = [msgs \text{ EXCEPT } ![i][j] = Append(msgs[i][j], m)]$

Remove a message from $msgs -$ discard head of $msgs[i][j]$
$Discard(i, j) \triangleq msgs' = \text{IF } msgs[i][j] \neq \langle\rangle \text{ THEN } [msgs \text{ EXCEPT } ![i][j] = Tail(msgs[i][j])]$
$\text{ELSE } msgs$

Leader/Pleader broadcasts a message to all other servers
$Broadcast(i, m) \triangleq msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } ii = i \wedge ij \neq i \text{ THEN } Append(msgs[ii][ij], m)$
$\text{ELSE } msgs[ii][ij]]]$

Combination of *Send* and *Discard* $-$ discard head of $msgs[j][i]$ and add m into $msgs[i][j]$
$Reply(i, j, m) \triangleq msgs' = [msgs \text{ EXCEPT } ![j][i] = Tail(msgs[j][i]),$
$![i][j] = Append(msgs[i][j], m)]$

$Reply2(i, j, m1, m2) \triangleq msgs' = [msgs \text{ EXCEPT } ![j][i] = Tail(msgs[j][i]),$
$![i][j] = Append(Append(msgs[i][j], m1), m2)]$

Define initial values for all variables

$$
\begin{aligned}
Init \triangleq\ & \wedge state && = [s \in Server \mapsto Follower] \\
& \wedge currentEpoch && = [s \in Server \mapsto 0] \\
& \wedge leaderEpoch && = [s \in Server \mapsto 0] \\
& \wedge leaderOracle && = [s \in Server \mapsto NullPoint] \\
& \wedge history && = [s \in Server \mapsto \langle\rangle] \\
& \wedge msgs && = [i \in Server \mapsto [j \in Server \mapsto \langle\rangle]] \\
& \wedge cepochRecv && = [s \in Server \mapsto \{\}] \\
& \wedge ackeRecv && = [s \in Server \mapsto \{\}] \\
& \wedge ackldRecv && = [s \in Server \mapsto \{\}] \\
& \wedge ackIndex && = [i \in Server \mapsto [j \in Server \mapsto 0]] \\
& \wedge currentCounter && = [s \in Server \mapsto 0]
\end{aligned}
$$

$\land$ $sendCounter = [s \in Server \mapsto 0]$
$\land$ $commitIndex = [s \in Server \mapsto 0]$
$\land$ $committedIndex = [s \in Server \mapsto 0]$
$\land$ $initialHistory = [s \in Server \mapsto \langle\rangle]$
$\land$ $cepochSent = [s \in Server \mapsto \text{FALSE}]$
$\land$ $tempMaxEpoch = [s \in Server \mapsto 0]$

$\land$ $tempMaxLastEpoch = [s \in Server \mapsto 0]$
$\land$ $tempInitialHistory = [s \in Server \mapsto \langle\rangle]$
$\land$ $proposalMsgsLog = \{\}$

A server becomes pleader and a quorum servers knows that.

$Election(i, Q) \triangleq$
$\land$ $i \in Q$
$\land$ $state' = [s \in Server \mapsto \text{IF } s = i \text{ THEN } ProspectiveLeader$
$\text{ELSE IF } s \in Q \text{ THEN } Follower$
$\text{ELSE } state[s]]$
$\land$ $cepochRecv' = [cepochRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land$ $ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land$ $ackldRecv' = [ackldRecv \text{ EXCEPT } ![i] = \{i\}]$
$\land$ $ackIndex' = [ii \in Server \mapsto [ij \in Server \mapsto$
$\text{IF } ii = i \text{ THEN } 0$
$\text{ELSE } ackIndex[ii][ij]]]$
$\land$ $committedIndex' = [committedIndex \text{ EXCEPT } ![i] = 0]$
$\land$ $initialHistory' = [initialHistory \text{ EXCEPT } ![i] = \langle\rangle]$
$\land$ $tempMaxEpoch' = [tempMaxEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
$\land$ $tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = currentEpoch[i]]$
$\land$ $tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = history[i]]$
$\land$ $leaderOracle' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } i$
$\text{ELSE } leaderOracle[s]]$
$\land$ $leaderEpoch' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN } currentEpoch[s]$
$\text{ELSE } leaderEpoch[s]]$
$\land$ $cepochSent' = [s \in Server \mapsto \text{IF } s \in Q \text{ THEN FALSE}$
$\text{ELSE } cepochSent[s]]$
$\land$ $msgs' = [ii \in Server \mapsto [ij \in Server \mapsto$
$\text{IF } ii \in Q \land ij \in Q \text{ THEN } \langle\rangle$
$\text{ELSE } msgs[ii][ij]]]$
$\land$ UNCHANGED $\langle currentEpoch, history, commitIndex, currentCounter, sendCounter, proposalMsgsLog$

A server halts and restarts.

$Restart(i) \triangleq$
$\land$ $state' = [state \text{ EXCEPT } ![i] = Follower]$
$\land$ $leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = NullPoint]$
$\land$ $commitIndex' = [commitIndex \text{ EXCEPT } ![i] = 0]$
$\land$ $msgs' = [ii \in Server \mapsto [ij \in Server \mapsto \text{IF } ij = i \text{ THEN } \langle\rangle$

ELSE $msgs[ii][ij]]]$

$\land$ $cepochSent'$ $= [cepochSent$ EXCEPT $![i] =$ FALSE$]$

$\land$ UNCHANGED $\langle currentEpoch, leaderEpoch, history, leaderVars, tempVars, proposalMsgsLog\rangle$

In phase $f11$, follower sends $f.p$ to pleader via $CEPOCH$.

$FollowerDiscovery1(i) \triangleq$

$\land$ $state[i] = Follower$

$\land$ $leaderOracle[i] \neq NullPoint$

$\land$ $\neg cepochSent[i]$

$\land$ LET $leader \triangleq leaderOracle[i]$

IN $Send(i, leader, [mtype \mapsto CEPOCH,$
$mepoch \mapsto currentEpoch[i]])$

$\land$ $cepochSent' = [cepochSent$ EXCEPT $![i] =$ TRUE$]$

$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, proposalMsgsLog\rangle$

In phase $l11$, pleader receives $CEPOCH$ from a quorum, and choose a new epoch e' as its own $l.p$ and sends $NEWEPOCH$ to followers.

$LeaderHandleCEPOCH(i, j) \triangleq$

$\land$ $state[i] = ProspectiveLeader$

$\land$ $msgs[j][i] \neq \langle\rangle$

$\land$ $msgs[j][i][1].mtype = CEPOCH$

$\land$ $\lor$ redundant message - just discard

$\land$ $j \in cepochRecv[i]$

$\land$ UNCHANGED $\langle tempMaxEpoch, cepochRecv\rangle$

$\lor$ new message - modify $tempMaxEpoch$ and $cepochRecv$

$\land$ $j \notin cepochRecv[i]$

$\land$ LET $newEpoch \triangleq Maximum(\{tempMaxEpoch[i], msgs[j][i][1].mepoch\})$

IN $tempMaxEpoch' = [tempMaxEpoch$ EXCEPT $![i] = newEpoch]$

$\land$ $cepochRecv' = [cepochRecv$ EXCEPT $![i] = cepochRecv[i] \cup \{j\}]$

$\land$ $Discard(j, i)$

$\land$ UNCHANGED $\langle serverVars, ackeRecv, ackldRecv, ackIndex, currentCounter, sendCounter, initialHist$
$committedIndex, cepochSent, tempMaxLastEpoch, tempInitialHistory, proposalMsgsLo$

Here I decide to change leader's epoch in $l12\&l21$, otherwise there may exist an old leader and a new leader who share the same expoch. So here I just change $leaderEpoch$, and use it in handling ACK-E.

$LeaderDiscovery1(i) \triangleq$

$\land$ $state[i] = ProspectiveLeader$

$\land$ $cepochRecv[i] \in Quorums$

$\land$ $leaderEpoch' = [leaderEpoch$ EXCEPT $![i] = tempMaxEpoch[i] + 1]$

$\land$ $cepochRecv' = [cepochRecv$ EXCEPT $![i] = \{\}]$

$\land$ $Broadcast(i, [mtype \mapsto NEWEPOCH,$
$mepoch \mapsto leaderEpoch'[i]])$

$\land$ UNCHANGED $\langle state, currentEpoch, leaderOracle, history, ackeRecv, ackldRecv, ackIndex, currentCo$
$initialHistory, commitIndex, committedIndex, cepochSent, tempVars, proposalMsgsLo$

In phase $f12$, follower receives $NEWEPOCH$. If $e' > f.p$ then sends back $ACKE$,
and $ACKE$ contains $f.a$ and hf to help pleader choose a newer history.

$$
\begin{aligned}
&FollowerDiscovery2(i,\, j) \triangleq \\
&\quad \land\, state[i] = Follower \\
&\quad \land\, msgs[j][i] \neq \langle\rangle \\
&\quad \land\, msgs[j][i][1].mtype = NEWEPOCH \\
&\quad \land\, \text{LET } msg \triangleq msgs[j][i][1] \\
&\qquad \text{IN} \quad \lor \text{ new } NEWEPOCH - \text{ accept and reply} \\
&\qquad\qquad \land\, currentEpoch[i] \leq msg.mepoch \text{ Here use } \leq \text{, because one follower may send } CEPOCH \text{ more then o} \\
&\qquad\qquad \land\, currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch] \\
&\qquad\qquad \land\, leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j] \\
&\qquad\qquad \land\, Reply(i,\, j,\, [mtype \mapsto ACKE, \\
&\qquad\qquad\qquad\qquad mepoch \mapsto msg.mepoch, \\
&\qquad\qquad\qquad\qquad mlastEpoch \mapsto leaderEpoch[i], \\
&\qquad\qquad\qquad\qquad mhf \mapsto history[i]]) \\
&\qquad\qquad \lor \text{ stale } NEWEPOCH - diacard \\
&\qquad\qquad\qquad \land\, currentEpoch[i] > msg.mepoch \\
&\qquad\qquad\qquad \land\, Discard(j,\, i) \\
&\qquad\qquad\qquad \land\, \text{UNCHANGED } \langle currentEpoch,\, leaderOracle\rangle \\
&\quad \land\, \text{UNCHANGED } \langle state,\, leaderEpoch,\, history,\, leaderVars,\, commitIndex,\, cepochSent,\, tempVars,\, proposa
\end{aligned}
$$

In phase $l12$, pleader receives $ACKE$ from a quorum,
and select the history of one most up-to-date follower to be the initial history.

$$
\begin{aligned}
&LeaderHandleACKE(i,\, j) \triangleq \\
&\quad \land\, state[i] = ProspectiveLeader \\
&\quad \land\, msgs[j][i] \neq \langle\rangle \\
&\quad \land\, msgs[j][i][1].mtype = ACKE \\
&\quad \land\, \text{LET } msg \triangleq msgs[j][i][1] \\
&\qquad\quad infoOk \triangleq \lor\, msg.mlastEpoch > tempMaxLastEpoch[i] \\
&\qquad\qquad\qquad\qquad \lor\, \land\, msg.mlastEpoch = tempMaxLastEpoch[i] \\
&\qquad\qquad\qquad\qquad\quad \land\, \lor\, LastZxid(msg.mhf)[1] > LastZxid(tempInitialHistory[i])[1] \\
&\qquad\qquad\qquad\qquad\qquad\quad \lor\, \land\, LastZxid(msg.mhf)[1] = LastZxid(tempInitialHistory[i])[1] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, LastZxid(msg.mhf)[2] \geq LastZxid(tempInitialHistory[i])[2] \\
&\qquad \text{IN} \quad \lor\, \land\, leaderEpoch[i] = msg.mepoch \\
&\qquad\qquad\quad \land\, \lor\, \land\, infoOk \\
&\qquad\qquad\qquad\qquad \land\, tempMaxLastEpoch' = [tempMaxLastEpoch \text{ EXCEPT } ![i] = msg.mlastEpoch] \\
&\qquad\qquad\qquad\qquad \land\, tempInitialHistory' = [tempInitialHistory \text{ EXCEPT } ![i] = msg.mhf] \\
&\qquad\qquad\qquad\; \lor\, \land\, \neg infoOk \\
&\qquad\qquad\qquad\qquad \land\, \text{UNCHANGED } \langle tempMaxLastEpoch,\, tempInitialHistory\rangle \\
&\qquad\qquad\quad \land\, ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \text{IF } j \notin ackeRecv[i] \text{ THEN } ackeRecv[i] \cup \{j\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE } ackeRecv[i]] \\
&\qquad\qquad\quad \lor\, \land\, leaderEpoch[i] \neq msg.mepoch \\
&\qquad\qquad\qquad \land\, \text{UNCHANGED } \langle tempMaxLastEpoch,\, tempInitialHistory,\, ackeRecv\rangle \\
&\quad \land\, Discard(j,\, i) \\
&\quad \land\, \text{UNCHANGED } \langle serverVars,\, cepochRecv,\, ackldRecv,\, ackIndex,\, currentCounter,
\end{aligned}
$$

$sendCounter$, $initialHistory$, $committedIndex$, $cepochSent$, $tempMaxEpoch$, $proposalM$

$LeaderDiscovery2Sync1(i) \triangleq$
$\wedge\ state[i] = ProspectiveLeader$
$\wedge\ ackeRecv[i] \in Quorums$
$\wedge\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = leaderEpoch[i]]$
$\wedge\ history' = [history \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\wedge\ initialHistory' = [initialHistory \text{ EXCEPT } ![i] = tempInitialHistory[i]]$
$\wedge\ ackeRecv' = [ackeRecv \text{ EXCEPT } ![i] = \{\}]$
$\wedge\ ackIndex' = [ackIndex \text{ EXCEPT } ![i] = Len(tempInitialHistory[i])]$
until now, $phase1(Discovery)$ ends
$\wedge\ Broadcast(i, [mtype \mapsto NEWLEADER,$
$mepoch \mapsto currentEpoch[i],$
$minitialHistory \mapsto history'[i]])$
$\wedge$ UNCHANGED $\langle state, leaderEpoch, leaderOracle, commitIndex, cepochRecv, ackldRecv,$
$currentCounter, sendCounter, committedIndex, cepochSent, tempVars, proposalMsgsL$

Delete the change of $commitIndex$ in $LeaderDiscovery2Sync1$. $FollowerSync1$, then we can promise that $commitIndex$ of every server increases monotonically, except that some server halts and restarts.

In phase $f21$, follower receives $NEWLEADER$. The follower updates its epoch and history, and sends back ACK-LD to pleader.
$FollowerSync1(i, j) \triangleq$
$\wedge\ state[i] = Follower$
$\wedge\ msgs[j][i] \neq \langle\rangle$
$\wedge\ msgs[j][i][1].mtype = NEWLEADER$
$\wedge$ LET $msg \triangleq msgs[j][i][1]$
IN $\vee$ new $NEWLEADER-$ accept and reply
$\wedge\ currentEpoch[i] \leq msg.mepoch$
$\wedge\ currentEpoch' = [currentEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\wedge\ leaderEpoch' = [leaderEpoch \text{ EXCEPT } ![i] = msg.mepoch]$
$\wedge\ leaderOracle' = [leaderOracle \text{ EXCEPT } ![i] = j]$
$\wedge\ history' = [history \text{ EXCEPT } ![i] = msg.minitialHistory]$
$\wedge\ Reply(i, j, [mtype \mapsto ACKLD,$
$mepoch \mapsto msg.mepoch,$
$mhistory \mapsto msg.minitialHistory])$
$\vee$ stale $NEWLEADER-$ discard
$\wedge\ currentEpoch[i] > msg.mepoch$
$\wedge\ Discard(j, i)$
$\wedge$ UNCHANGED $\langle currentEpoch, leaderEpoch, leaderOracle, history\rangle$
$\wedge$ UNCHANGED $\langle state, commitIndex, leaderVars, tempVars, cepochSent, proposalMsgsLog\rangle$

In phase $l22$, pleader receives ACK-LD from a quorum of followers, and sends $COMMIT$-LD to followers.
$LeaderHandleACKLD(i, j) \triangleq$
$\wedge\ state[i] = ProspectiveLeader$

$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACKLD$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new ACK-LD - accept
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ ackIndex' = [ackIndex$ EXCEPT $![i][j] = Len(initialHistory[i])]$
$\land\ ackldRecv' = [ackldRecv$ EXCEPT $![i] =$ IF $j \notin ackldRecv[i]$ THEN $ackldRecv[i] \cup \{j\}$
ELSE $ackldRecv[i]]$
$\lor$ stale ACK-LD - impossible
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land$ UNCHANGED $\langle ackldRecv, ackIndex\rangle$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle serverVars, cepochRecv, ackeRecv, currentCounter,$
$sendCounter, initialHistory, committedIndex, tempVars, cepochSent, proposalMsgsLo$

$LeaderSync2(i) \triangleq$
$\land\ state[i] = ProspectiveLeader$
$\land\ ackldRecv[i] \in Quorums$
$\land\ commitIndex' = [commitIndex$ EXCEPT $![i] = Len(history[i])]$
$\land\ committedIndex' = [committedIndex$ EXCEPT $![i] = Len(history[i])]$
$\land\ state' = [state$ EXCEPT $![i] = Leader]$
$\land\ currentCounter' = [currentCounter$ EXCEPT $![i] = 0]$
$\land\ sendCounter' = [sendCounter$ EXCEPT $![i] = 0]$
$\land\ ackldRecv' = [ackldRecv$ EXCEPT $![i] = \{\}]$
$\land\ Broadcast(i, [mtype \mapsto COMMITLD,$
$mepoch \mapsto currentEpoch[i]])$
$\land$ UNCHANGED $\langle currentEpoch, leaderEpoch, leaderOracle, history, cepochRecv,$
$ackeRecv, ackIndex, initialHistory, tempVars, cepochSent, proposalMsgsLog\rangle$

In phase $f22$, follower receives $COMMIT$-LD and submits all unprocessed transaction.
$FollowerSync2(i, j) \triangleq$
$\land\ state[i] = Follower$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = COMMITLD$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new $COMMIT$-LD - commit all transactions in initial history
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ commitIndex' = [commitIndex$ EXCEPT $![i] = Len(history[i])]$
$\land\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\lor$ stale $COMMIT$-LD - discard
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land$ UNCHANGED $\langle commitIndex, leaderOracle\rangle$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderOracle, history, leaderVars, tempVars, cepochSent, propos$

```
In phase l31, leader receives client request and broadcasts PROPOSE.
ClientRequest(i, v) ≜
    ∧ state[i] = Leader
    ∧ currentCounter' = [currentCounter EXCEPT ![i] = currentCounter[i] + 1]
    ∧ LET newTransaction ≜ [epoch   ↦ currentEpoch[i],
                            counter ↦ currentCounter'[i],
                            value   ↦ v]
      IN  ∧ history'   = [history   EXCEPT ![i]  = Append(history[i], newTransaction)]
          ∧ ackIndex'  = [ackIndex  EXCEPT ![i] = Len(history'[i])]
    ∧ UNCHANGED ⟨msgs, state, currentEpoch, leaderEpoch, leaderOracle, commitIndex, cepochRecv,
                 ackeRecv, ackldRecv, sendCounter, initialHistory, committedIndex, tempVars, cepoch

LeaderBroadcast1(i) ≜
    ∧ state[i] = Leader
    ∧ sendCounter[i] < currentCounter[i]
    ∧ LET toBeSentCounter ≜ sendCounter[i] + 1
          toBeSentIndex   ≜ Len(initialHistory[i]) + toBeSentCounter
          toBeSentEntry   ≜ history[i][toBeSentIndex]
      IN  ∧ Broadcast(i, [mtype     ↦ PROPOSE,
                          mepoch    ↦ currentEpoch[i],
                          mproposal ↦ toBeSentEntry])
          ∧ sendCounter' = [sendCounter EXCEPT ![i] = toBeSentCounter]
          ∧ proposalMsgsLog' = proposalMsgsLog ∪ {[msource   ↦ i,
                                                   mtype     ↦ PROPOSE,
                                                   mepoch    ↦ currentEpoch[i],
                                                   mproposal ↦ toBeSentEntry]}
    ∧ UNCHANGED ⟨serverVars, cepochRecv, ackeRecv, ackldRecv, ackIndex,
                 currentCounter, initialHistory, committedIndex, tempVars, cepochSent⟩

In phase f31, follower accepts proposal and append it to history.
FollowerBroadcast1(i, j) ≜
    ∧ state[i] = Follower
    ∧ msgs[j][i] ≠ ⟨⟩
    ∧ msgs[j][i][1].mtype = PROPOSE
    ∧ LET msg ≜ msgs[j][i][1]
      IN  ∨ It should be that msg.mproposal.counter = 1 ∨ msg.mrpoposal.counter = history[Len(history)].counter + 1
            ∧ currentEpoch[i] = msg.mepoch
            ∧ history' = [history EXCEPT ![i] = Append(history[i], msg.mproposal)]
            ∧ leaderOracle' = [leaderOracle EXCEPT ![i] = j]
            ∧ Reply(i, j, [mtype  ↦ ACK,
                           mepoch ↦ currentEpoch[i],
                           mindex ↦ Len(history'[i])])
          ∨ If happens, ≠ must be >, namely a stale leader sends it.
            ∧ currentEpoch[i] ≠ msg.mepoch
            ∧ Discard(j, i)
```

$\land$ UNCHANGED $\langle history, leaderOracle\rangle$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderEpoch, commitIndex, leaderVars, tempVars, cepochSent, p$

In phase $l32$, leader receives ack from a quorum of followers to a certain proposal, and commits the proposal.

$LeaderHandleACK(i, j) \triangleq$
$\land\ state[i] = Leader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACK$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ There should be $ackIndex[i][j] + 1 \triangleq msg.mindex$
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ ackIndex' = [ackIndex$ EXCEPT $![i][j] = Maximum(\{ackIndex[i][j], msg.mindex\})]$
$\lor$ If happens, $\neq$ must be $>$, namely a stale follower sends it.
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land$ UNCHANGED $ackIndex$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle serverVars, cepochRecv, ackeRecv, ackldRecv, currentCounter,$
$sendCounter, initialHistory, committedIndex, tempVars, cepochSent, proposalMsgsLo$

$LeaderAdvanceCommit(i) \triangleq$
$\land\ state[i] = Leader$
$\land\ commitIndex[i] < Len(history[i])$
$\land$ LET $Agree(index) \triangleq \{i\} \cup \{k \in Server : ackIndex[i][k] \geq index\}$
$agreeIndexes \triangleq \{index \in (commitIndex[i] + 1) \,..\, Len(history[i]) : Agree(index) \in Quoru$
$newCommitIndex \triangleq$ IF $agreeIndexes \neq \{\}$ THEN $Maximum(agreeIndexes)$
ELSE $commitIndex[i]$
IN $commitIndex' = [commitIndex$ EXCEPT $![i] = newCommitIndex]$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderEpoch, leaderOracle, history,$
$msgs, leaderVars, tempVars, cepochSent, proposalMsgsLog\rangle$

$LeaderBroadcast2(i) \triangleq$
$\land\ state[i] = Leader$
$\land\ committedIndex[i] < commitIndex[i]$
$\land$ LET $newCommittedIndex \triangleq committedIndex[i] + 1$
IN $\land\ Broadcast(i, [mtype \mapsto COMMIT,$
$mepoch \mapsto currentEpoch[i],$
$mindex \mapsto newCommittedIndex,$
$mcounter \mapsto history[newCommittedIndex].counter])$
$\land\ committedIndex' = [committedIndex$ EXCEPT $![i] = committedIndex[i] + 1]$
$\land$ UNCHANGED $\langle serverVars, cepochRecv, ackeRecv, ackldRecv, ackIndex, currentCounter,$
$sendCounter, initialHistory, tempVars, cepochSent, proposalMsgsLog\rangle$

In phase $f32$, follower receives $COMMIT$ and commits transaction.

$FollowerBroadcast2(i, j) \triangleq$
$\land\ state[i] = Follower$

$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = COMMIT$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor$ new *COMMIT* − commit transaction in history
$\land\ currentEpoch[i] = msg.mepoch$
$\land\ commitIndex' = [commitIndex$ EXCEPT $![i] = Maximum(\{commitIndex[i], msg.mindex\}$
$\land\ leaderOracle' = [leaderOracle$ EXCEPT $![i] = j]$
$\lor$ stale *COMMIT* − discard
$\land\ currentEpoch[i] \neq msg.mepoch$
$\land$ UNCHANGED $\langle commitIndex, leaderOracle\rangle$
$\land\ Discard(j, i)$
$\land$ UNCHANGED $\langle state, currentEpoch, leaderEpoch, history,$
$leaderVars, tempVars, cepochSent, proposalMsgsLog\rangle$

There may be two ways to make sure all followers as up-to-date as the leader.
*way*1: choose *Send* not *Broadcast* when leader is going to send *PROPOSE* and *COMMIT*.
*way*2: When one follower receives *PROPOSE* or *COMMIT* which misses some entries between its history and the newest entry, the follower send *CEPOCH* to catch pace.
Here I choose *way*2, which I need not to rewrite *PROPOSE* and *COMMIT*, but need to modify the code when follower receives *NEWLEADER* and *COMMIT*.

In phase *l*33, upon receiving *CEPOCH*, leader l proposes back *NEWEPOCH* and *NEWLEADER*.
$LeaderHandleCEPOCHinPhase3(i, j) \triangleq$
$\land\ state[i] = Leader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = CEPOCH$
$\land$ LET $msg \triangleq msgs[j][i][1]$
IN $\lor\ \land\ currentEpoch[i] \geq msg.mepoch$
$\land\ Reply2(i, j, [mtype \mapsto NEWEPOCH,$
$mepoch \mapsto currentEpoch[i]],$
$[mtype \mapsto NEWLEADER,$
$mepoch \mapsto currentEpoch[i],$
$minitialHistory \mapsto history[i]])$
$\lor\ \land\ currentEpoch[i] < msg.mepoch$
$\land$ UNCHANGED $msgs$
$\land$ UNCHANGED $\langle serverVars, leaderVars, tempVars, cepochSent, proposalMsgsLog\rangle$

In phase *l*34, upon receiving ack from f of the *NEWLEADER*, it sends a commit message to f.
Leader l also makes $Q := Q \cup \{f\}$.
$LeaderHandleACKLDinPhase3(i, j) \triangleq$
$\land\ state[i] = Leader$
$\land\ msgs[j][i] \neq \langle\rangle$
$\land\ msgs[j][i][1].mtype = ACKLD$
$\land$ LET $msg \triangleq msgs[j][i][1]$
$aimCommitIndex \triangleq Minimum(\{commitIndex[i], Len(msg.mhistory)\})$

```
        IN  ∨ ∧ currentEpoch[i] = msg.mepoch
              ∧ ackIndex' = [ackIndex EXCEPT ![i][j] = Len(msg.mhistory)]
              ∧ Reply(i, j, [mtype     ↦ COMMIT,
                             mepoch    ↦ currentEpoch[i],
                             mindex    ↦ aimCommitIndex,
                             mcounter  ↦ history[aimCommitIndex].counter])
            ∨ ∧ currentEpoch[i] ≠ msg.mepoch
              ∧ Discard(j, i)
              ∧ UNCHANGED ⟨ackIndex⟩
      ∧ UNCHANGED ⟨serverVars, cepochRecv, ackeRecv, ackldRecv, currentCounter, sendCounter,
                   initialHistory, committedIndex, tempVars, cepochSent, proposalMsgsLog⟩
```

To ensure any follower can find the correct leader, the follower should modify *leaderOracle* anytime when it receive messages from leader, because a server may restart and join the cluster Q halfway and receive the first message which is not *NEWEPOCH*. But we can delete this restriction when we ensure *Broadcast* function acts on the followers in the cluster not any servers in the whole system, then one server must has correct *leaderOracle* before it receives messages.

Let me suppose two conditions when one follower sends *CEPOCH* to leader:
0. Usually, the server becomes follower in election and sends *CEPOCH* before receiving *NEWEPOCH*.
1. The follower wants to join the cluster halfway and get the newest history.
2. The follower has received *COMMIT*, but there exists the gap between its own history and *mindex*, which means there are some transactions before *mindex* miss. Here we choose to send *CEPOCH* again, to receive the newest history from leader.

```
BecomeFollower(i) ≜
    ∧ ∃ j ∈ Server \ {i} : ∧ msgs[j][i] ≠ ⟨⟩
                           ∧ LET msg ≜ msgs[j][i][1]
                             IN  ∧ currentEpoch[i] < msg.mepoch
                                 ∧ ∨ msg.mtype = NEWEPOCH
                                   ∨ msg.mtype = NEWLEADER
                                   ∨ msg.mtype = COMMITLD
                                   ∨ msg.mtype = PROPOSE
                                   ∨ msg.mtype = COMMIT
                                 ∧ state'       = [state       EXCEPT ![i] = Follower]
                                 ∧ currentEpoch' = [currentEpoch EXCEPT ![i] = msg.mepoch]
                                 ∧ leaderOracle' = [leaderOracle EXCEPT ![i] = j]
                                 Here we should not use Discard.
    ∧ UNCHANGED ⟨leaderEpoch, history, commitIndex, msgs, leaderVars, tempVars, cepochSent, proposa
```

```
DiscardStaleMessage(i) ≜
    ∧ ∃ j ∈ Server \ {i} : ∧ msgs[j][i] ≠ ⟨⟩
                           ∧ LET msg ≜ msgs[j][i][1]
                             IN  ∨ ∧ state[i] = Follower
                                 ∧ ∨ msg.mepoch < currentEpoch[i]
                                   ∨ msg.mtype = CEPOCH
```

$\lor\ msg.mtype = ACKE$
$\lor\ msg.mtype = ACKLD$
$\lor\ msg.mtype = ACK$
$\lor\ \land\ state[i] = Leader$
$\land\ msg.mtype \neq CEPOCH$
$\land\ \lor\ msg.mepoch < currentEpoch[i]$
$\lor\ msg.mtype = ACKE$ response of *NEWEPOCH*
$\lor\ \land\ state[i] = ProspectiveLeader$
$\land\ msg.mtype \neq CEPOCH$
$\land\ \lor\ msg.mepoch < currentEpoch[i]$
$\lor\ msg.mtype = ACK$
$\land\ Discard(j,\ i)$
$\land$ UNCHANGED $\langle serverVars,\ leaderVars,\ tempVars,\ cepochSent,\ proposalMsgsLog\rangle$

Defines how the variables may transition.

$Next\ \triangleq$

$\lor \exists\, i \in Server :\ \ Restart(i)$
$\lor \exists\, i \in Server,\ Q \in Quorums : Election(i,\ Q)$
$\lor \exists\, i \in Server :\ \ FollowerDiscovery1(i)$
$\lor \exists\, i, j \in Server :\ \ LeaderHandleCEPOCH(i,\ j)$
$\lor \exists\, i \in Server :\ \ LeaderDiscovery1(i)$
$\lor \exists\, i, j \in Server :\ \ FollowerDiscovery2(i,\ j)$
$\lor \exists\, i, j \in Server :\ \ LeaderHandleACKE(i,\ j)$
$\lor \exists\, i \in Server :\ \ LeaderDiscovery2Sync1(i)$
$\lor \exists\, i, j \in Server :\ \ FollowerSync1(i,\ j)$
$\lor \exists\, i, j \in Server :\ \ LeaderHandleACKLD(i,\ j)$
$\lor \exists\, i \in Server :\ \ LeaderSync2(i)$
$\lor \exists\, i, j \in Server :\ \ FollowerSync2(i,\ j)$
$\lor \exists\, i \in Server,\ v \in Value : ClientRequest(i,\ v)$
$\lor \exists\, i \in Server :\ \ LeaderBroadcast1(i)$
$\lor \exists\, i, j \in Server :\ \ FollowerBroadcast1(i,\ j)$
$\lor \exists\, i, j \in Server :\ \ LeaderHandleACK(i,\ j)$
$\lor \exists\, i \in Server :\ \ LeaderAdvanceCommit(i)$
$\lor \exists\, i \in Server :\ \ LeaderBroadcast2(i)$
$\lor \exists\, i, j \in Server :\ \ FollowerBroadcast2(i,\ j)$
$\lor \exists\, i, j \in Server :\ \ LeaderHandleCEPOCHinPhase3(i,\ j)$
$\lor \exists\, i, j \in Server :\ \ LeaderHandleACKLDinPhase3(i,\ j)$
$\lor \exists\, i \in Server :\ \ DiscardStaleMessage(i)$
$\lor \exists\, i \in Server :\ \ BecomeFollower(i)$

$Spec\ \triangleq\ Init \land \Box[Next]_{vars}$

Define some variants, safety propoties, and liveness propoties of *Zab* consensus algorithm.

Safety properties

There is most one leader/prospective leader in a certain epoch.

$$\begin{aligned}
Leadership \triangleq\ & \forall\, i, j \in Server : \\
& \land state[i] = Leader \lor state[i] = ProspectiveLeader \\
& \land state[j] = Leader \lor state[j] = ProspectiveLeader \\
& \land currentEpoch[i] = currentEpoch[j] \\
& \Rightarrow i = j
\end{aligned}$$

Here, delivering means deliver some transaction from history to replica. We can assume $deliverIndex = commitIndex$. So we can assume the set of delivered transactions is the prefix of history with index from 1 to $commitIndex$. We can express a transaction by two-tuple $< epoch, counter >$ according to its uniqueness.

$$\begin{aligned}
equal(entry1,\ entry2) \triangleq\ & \land entry1.epoch\ \ = entry2.epoch \\
& \land entry1.counter = entry2.counter
\end{aligned}$$

$$\begin{aligned}
precede(entry1,\ entry2) \triangleq\ & \lor entry1.epoch < entry2.epoch \\
& \lor \land entry1.epoch\ \ = entry2.epoch \\
& \quad \land entry1.counter < entry2.counter
\end{aligned}$$

PrefixConsistency: The prefix that have been delivered in history in any process is the same.

$$\begin{aligned}
PrefixConsistency \triangleq\ & \forall\, i, j \in Server : \\
& \text{LET } smaller \triangleq Minimum(\{commitIndex[i],\ commitIndex[j]\}) \\
& \text{IN } \lor smaller\ \ = 0 \\
& \quad \lor \land smaller > 0 \\
& \qquad \land \forall\, index \in 1\,..\,smaller : equal(history[i][index],\ history[j][index])
\end{aligned}$$

Integrity: If some follower delivers one transaction, then some primary has broadcast it.

$$\begin{aligned}
Integrity \triangleq\ & \forall\, i \in Server : \\
& state[i] = Follower \land commitIndex[i] > 0 \\
& \Rightarrow \forall\, index \in 1\,..\,commitIndex[i] : \exists\, msg \in proposalMsgsLog : \\
& \quad equal(msg.mproposal,\ history[i][index])
\end{aligned}$$

Agreement: If some follower f delivers transaction a and some follower f' delivers transaction b, then f' delivers a or f delivers b.

$$\begin{aligned}
Agreement \triangleq\ & \forall\, i, j \in Server : \\
& \land state[i] = Follower \land commitIndex[i] > 0 \\
& \land state[j] = Follower \land commitIndex[j] > 0 \\
& \Rightarrow \\
& \forall\, index1 \in 1\,..\,commitIndex[i],\ index2 \in 1\,..\,commitIndex[j] : \\
& \quad \lor \exists\, indexj \in 1\,..\,commitIndex[j] : \\
& \qquad equal(history[j][indexj],\ history[i][index1]) \\
& \quad \lor \exists\, indexi \in 1\,..\,commitIndex[i] : \\
& \qquad equal(history[i][indexi],\ history[j][index2])
\end{aligned}$$

Total order: If some follower delivers a before b, then any process that delivers b must also deliver a and deliver a before b.

$$\begin{aligned}
TotalOrder \triangleq\ & \forall\, i, j \in Server : commitIndex[i] \geq 2 \land commitIndex[j] \geq 2 \\
& \Rightarrow \forall\, indexi1 \in 1\,..\,(commitIndex[i] - 1) : \forall\, indexi2 \in (indexi1 + 1)\,..\,commitIndex[i] :
\end{aligned}$$

$$
\begin{array}{l}
\text{LET } logOk \triangleq \exists\, index \in 1\,..\,commitIndex[j] : equal(history[i][indexi2],\ history[j][index] \\
\text{IN} \quad \lor \neg logOk \\
\qquad \lor \land logOk \\
\qquad\quad \land \text{LET } indexj2 \triangleq \text{CHOOSE } idx \in 1\,..\,commitIndex[j] : equal(history[i][indexi2], \\
\qquad\qquad \text{IN} \quad \exists\, indexj1 \in 1\,..\,(indexj2-1) : equal(history[i][indexi1],\ history[j][indexj1
\end{array}
$$

Local primary order: If a primary broadcasts a before it broadcasts b, then a follower that delivers b must also deliver a before b.

$$
\begin{array}{l}
LocalPrimaryOrder \triangleq \text{LET } mset(i,\ e) \triangleq \{msg \in proposalMsgsLog : msg.msource = i \land msg.mproposal.epoch \\
\qquad\qquad\qquad\qquad mentries(i,\ e) \triangleq \{msg.mproposal : msg \in mset(i,\ e)\} \\
\qquad\qquad\qquad \text{IN} \quad \forall\, i \in Server : \forall\, e \in 1\,..\,currentEpoch[i] : \\
\qquad\qquad\qquad\qquad \land Cardinality(mentries(i,\ e)) \geq 2 \\
\qquad\qquad\qquad\qquad \land \text{LET } tsc1 \triangleq \text{CHOOSE } p \in mentries(i,\ e) : \text{TRUE} \\
\qquad\qquad\qquad\qquad\qquad\quad tsc2 \triangleq \text{CHOOSE } p \in mentries(i,\ e) : \neg equal(p,\ tsc1) \\
\qquad\qquad\qquad\qquad\qquad\quad tscPre \triangleq \text{IF } precede(tsc1,\ tsc2) \text{ THEN } tsc1 \text{ ELSE } tsc2 \\
\qquad\qquad\qquad\qquad\qquad\quad tscNext \triangleq \text{IF } precede(tsc1,\ tsc2) \text{ THEN } tsc2 \text{ ELSE } tsc1 \\
\qquad\qquad\qquad\qquad\quad \text{IN} \quad \forall\, j \in Server : \land commitIndex[j] \geq 2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \exists\, index \qquad\quad \in 1\,..\,commitIndex[j] : equal(history[j][\\
\qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow \text{LET } index2 \triangleq \text{CHOOSE } idx \in 1\,..\,commitIndex[j] : equal(history[j][\\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{IN} \quad \land index2 > 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \exists\, index1 \in 1\,..\,(index2-1) : equal(history[j][index1],\ tscPre
\end{array}
$$

Global primary order: A follower f delivers both a with epoch e and b with epoch e', and $e < e'$, then f must deliver a before b.

$$
\begin{array}{l}
GlobalPrimaryOrder \triangleq \forall\, i \in Server : commitIndex[i] \geq 2 \\
\qquad\qquad\qquad\qquad \Rightarrow \forall\, idx1,\ idx2 \in 1\,..\,commitIndex[i] : \lor history[i][idx1].epoch \geq history[i][idx2].ep \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \lor \land history[i][idx1].epoch < history[i][idx2] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land idx1 < idx2
\end{array}
$$

Primary integrity: If primary p broadcasts a and some follower f delivers b such that b has epoch smaller than epoch of p, then p must deliver b before it broadcasts a.

$$
\begin{array}{l}
PrimaryIntegrity \triangleq \forall\, i,\ j \in Server : \land state[i] = Leader \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land state[j] = Follower \land commitIndex[j] \geq 1 \\
\qquad\qquad\qquad\qquad\qquad \Rightarrow \forall\, index \in 1\,..\,commitIndex[j] : \lor history[j][index].epoch \geq currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \lor \land history[j][index].epoch < currentEpoch[i] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \land \exists\, idx \in 1\,..\,commitIndex[i] : equal(history[i][i
\end{array}
$$

Liveness property

Suppose that :

- A quorum Q of followers are up.
- The followers in Q elect the same process l and l is up.
- Messages between a follower in Q and l are received in a timely fashion.

If l proposes a transaction a, then a is eventually committed.

$$LivenessProperty1 \triangleq \forall i, j \in Server, msg \in msgs: (state[i] = Leader) \land (msg.type = COMMIT) \leadsto (msg \in history[j]) \land (state[j] = Follower)$$

\ * Modification History
\ * Last modified Sun *Apr* 18 15:13:03 *CST* 2021 by Dell
\ * Created Sat *Dec* 05 13:32:08 *CST* 2020 by Dell