This specification is discussed in "Paxos Commit", Lecture 6 of the TLA+ Video Course.

This module specifies the *Paxos Commit* algorithm. We specify only safety properties, not liveness properties. We simplify the specification in the following ways.

- As in the specification of module *TwoPhase*, and for the same reasons, we let the variable *msgs* be the set of all messages that have ever been sent. If a message is sent to a set of recipients, only one copy of the message appears in *msgs*.
- We do not explicitly model the receipt of messages. If an operation can be performed when a process has received a certain set of messages, then the operation is represented by an action that is enabled when those messages are in the set *msgs* of sent messages. (We are specifying only safety properties, which assert what events can occur, and the operation can occur if the messages that enable it have been sent.)
- We do not model leader selection. We define actions that the current leader may perform, but do not specify who performs them.

As in the specification of Two-Phase commit in module Two-Phase, we have RMs spontaneously issue Prepared messages and we ignore Prepare messages.

EXTENDS Integers

$Maximum(S) \triangleq$

If S is a set of numbers, then this define Maximum(S) to be the maximum of those numbers, or -1 if S is empty.

```
IF S = \{\} THEN -1
ELSE CHOOSE n \in S : \forall m \in S : n \ge m
```

CONSTANT RM, The set of resource managers.

Acceptor, The set of acceptors.

Majority, The set of majorities of acceptors

Ballot The set of ballot numbers

We assume the following properties of the declared constants.

ASSUME

 $\land Ballot \subseteq Nat$

 $\wedge 0 \in Ballot$

 \land Majority \subseteq Subset Acceptor

 $\land \forall MS1, MS2 \in Majority : MS1 \cap MS2 \neq \{\}$

All we assume about the set Majority of majorities is that any two majorities have non-empty intersection.

$Messages \stackrel{\triangle}{=}$

The set of all possible messages. There are messages of type "Commit" and "Abort" to announce the decision, as well as messages for each phase of each instance of *ins* of the *Paxos* consensus algorithm. The *acc* field indicates the sender of a message from an acceptor to the leader; messages from a leader are broadcast to all acceptors.

```
[type: \{ \text{``phase1a''} \}, \ ins: RM, \ bal: Ballot \setminus \{0\}] \\ \cup \\ [type: \{ \text{``phase1b''} \}, \ ins: RM, \ mbal: Ballot, \ bal: Ballot \cup \{-1\},
```

```
val : { "prepared", "aborted", "none" }, acc : Acceptor]
  [type: \{ \text{"phase2a"} \}, ins: RM, bal: Ballot, val: \{ \text{"prepared"}, \text{"aborted"} \} ]
  [type: \{ \text{``phase2b''} \}, \ acc: Acceptor, \ ins: RM, \ bal: Ballot, 
   val : { "prepared", "aborted" }]
  [type: { "Commit", "Abort" }]
VARIABLES
  rmState,
                 rmState[r] is the state of resource manager r.
  aState,
                 aState[ins][ac] is the state of acceptor ac for instance
                 ins of the Paxos algorithm.
  msqs
                 The set of all messages ever sent.
PCTypeOK \triangleq
  The type-correctness invariant. Each acceptor maintains the values mbal, bal, and val for each
  instance of the Paxos consensus algorithm.
   \land rmState \in [RM \rightarrow \{ \text{"working"}, \text{"prepared"}, \text{"committed"}, \text{"aborted"} \}]
   \land aState \in [RM \rightarrow [Acceptor \rightarrow [mbal : Ballot,
                                               bal : Ballot \cup \{-1\},\
                                                val : \{ \text{"prepared"}, \text{"aborted"}, \text{"none"} \} ]]]
   \land msgs \subseteq Messages
PCInit \stackrel{\triangle}{=} The initial predicate.
   \land rmState = [r \in RM \mapsto "working"]
   \wedge \ aState = [r \in RM \mapsto
                      [ac \in Acceptor]
                         \mapsto [mbal \mapsto 0, bal \mapsto -1, val \mapsto "none"]]]
   \land msgs = \{\}
THE ACTIONS
Send(m) \stackrel{\triangle}{=} msgs' = msgs \cup \{m\}
  An action expression that describes the sending of message m.
RM ACTIONS
RMPrepare(r) \triangleq
  Resource manager r prepares by sending a phase 2a message for ballot number 0 with value
  "prepared".
  \land rmState[r] = "working"
   \land rmState' = [rmState \ \texttt{EXCEPT} \ ![r] = "prepared"]
   \land Send([type \mapsto "phase2a", ins \mapsto r, bal \mapsto 0, val \mapsto "prepared"])
   \land UNCHANGED aState
RMChooseToAbort(r) \triangleq
```

Resource manager r spontaneously decides to abort. It may (but need not) send a phase 2a message for ballot number 0 with value "aborted".

$RMRcvCommitMsq(r) \stackrel{\Delta}{=}$

Resource manager r is told by the leader to commit. When this action is enabled, rmState[r] must equal either "prepared" or "committed". In the latter case, the action leaves the state unchanged (it is a "stuttering step").

$RMRcvAbortMsg(r) \triangleq$

Resource manager r is told by the leader to abort. It could be in any state except "committed".

```
 \land [type \mapsto \text{``Abort''}] \in msgs \\ \land rmState' = [rmState \text{ EXCEPT } ![r] = \text{``aborted''}] \\ \land \text{UNCHANGED } \langle aState, msgs \rangle
```

LEADER ACTIONS

The following actions are performed by any process that believes itself to be the current leader. Since leader selection is not assumed to be reliable, multiple processes could simultaneously consider themselves to be the leader.

```
Phase1a(bal, r) \triangleq
```

If the leader times out without learning that a decision has been reached on resource manager r's prepare/abort decision, it can perform this action to initiate a new ballot bal. (Sending duplicate phase 1a messages is harmless.)

```
 \land Send([type \mapsto "phasela", ins \mapsto r, bal \mapsto bal]) \\ \land \texttt{UNCHANGED} \ \langle rmState, \ aState \rangle
```

$Phase2a(bal, r) \triangleq$

The action in which a leader sends a phase 2a message with ballot bal > 0 in instance r, if it has received phase 1b messages for ballot number bal from a majority of acceptors. If the leader received a phase 1b message from some acceptor that had sent a phase 2b message for this instance, then $maxbal \geq 0$ and the value val the leader sends is determined by the phase 1b messages. (If val = "prepared", then r must have prepared.) Otherwise, maxbal = -1 and the leader sends the value "aborted".

The first conjunct asserts that the action is disabled if any commit leader has already sent a phase 2a message with ballot number bal. In practice, this is implemented by having ballot numbers partitioned among potential leaders, and having a leader record in stable storage the largest ballot number for which it sent a phase 2a message.

```
LET mset \stackrel{\triangle}{=} \{m \in msgs : \land m.type = \text{"phase1b"}\}
                                             \land m.ins = r
                                             \land m.mbal = bal
                                             \land m.acc \in MS
              maxbal \stackrel{\triangle}{=} Maximum(\{m.bal : m \in mset\})
               val \stackrel{\Delta}{=} \text{if } maxbal = -1
                             THEN "aborted"
                             ELSE (CHOOSE m \in mset : m.bal = maxbal).val
                \land \forall ac \in MS : \exists m \in mset : m.acc = ac
                \land \mathit{Send}([\mathit{type} \mapsto \mathsf{"phase2a"}, \mathit{ins} \mapsto r, \mathit{bal} \mapsto \mathit{bal}, \mathit{val} \mapsto \mathit{val}])
   \land UNCHANGED \langle rmState, aState \rangle
PCDecide \triangleq
  A leader can decide that Paxos Commit has reached a result and send a message announcing
  the result if it has received the necessary phase 2b messages.
   \wedge LET Decided(r, v) \stackrel{\triangle}{=}
              True iff instance r of the Paxos consensus algorithm has chosen the value v.
              \exists b \in Ballot, MS \in Majority:
                 \forall ac \in MS : [type \mapsto "phase2b", ins \mapsto r,
                                    bal \mapsto b, \ val \mapsto v, \ acc \mapsto ac \in msgs
             \lor \land \forall r \in RM : Decided(r, "prepared")
                \land Send([type \mapsto "Commit"])
             \vee \wedge \exists r \in RM : Decided(r, "aborted")
                \land Send([type \mapsto "Abort"])
   \land UNCHANGED \langle rmState, aState \rangle
ACCEPTOR ACTIONS
Phase1b(acc) \triangleq
  \exists m \in msgs:
     \land m.type = "phase1a"
     \land aState[m.ins][acc].mbal < m.bal
     \land aState' = [aState \ EXCEPT \ ![m.ins][acc].mbal = m.bal]
     \land Send([type \mapsto "phase1b",
                 ins \mapsto m.ins,
                 mbal \mapsto m.bal,
                 bal \mapsto aState[m.ins][acc].bal,
                 val \mapsto aState[m.ins][acc].val,
                 acc \mapsto acc
     \land UNCHANGED rmState
Phase2b(acc) \triangleq
   \wedge \exists m \in msgs:
         \land m.type = \text{"phase2a"}
         \land aState[m.ins][acc].mbal \leq m.bal
         \land aState' = [aState \ EXCEPT \ ![m.ins][acc].mbal = m.bal,
```

```
![m.ins][acc].bal = m.bal, \\ ![m.ins][acc].val = m.val] \\ \wedge Send([type \mapsto \text{``phase2b''}, ins \mapsto m.ins, bal \mapsto m.bal, \\ val \mapsto m.val, acc \mapsto acc]) \\ \wedge \text{UNCHANGED } rmState
```

```
PCNext \triangleq \text{The next-state action} \\ \lor \exists \ r \in RM : \lor RMPrepare(r) \\ \lor RMChooseToAbort(r) \\ \lor RMRcvCommitMsg(r) \\ \lor RMRcvAbortMsg(r) \\ \lor \exists \ bal \in Ballot \setminus \{0\}, \ r \in RM : Phase1a(bal, \ r) \lor Phase2a(bal, \ r) \\ \lor PCDecide \\ \lor \exists \ acc \in Acceptor : Phase1b(acc) \lor Phase2b(acc)
```

The following part of the spec is not covered in Lecture 7. It will be explained in Lecture 8.

 $PCSpec \stackrel{\Delta}{=} PCInit \wedge \Box [PCNext]_{\langle rmState, \ aState, \ msgs \rangle}$

The complete spec of the Paxos Commit protocol.

THEOREM $PCSpec \Rightarrow \Box PCTypeOK$

We now assert that the $Paxos\ Commit$ protocol implements the transaction commit protocol of module TCommit. The following statement imports into the current module the definitions from module TCommit, including the definition of TCSpec.

INSTANCE TCommitOriginal

Theorem $PCSpec \Rightarrow TCSpec$