# Valid states



# State transitions



1. **uninitializedStateTransition**

stateBefore

f(e,args)

stateAfter

assert(stateBefore==UNINITIALIZED => (stateAfter == UNINITIALIZED || stateAfter==PENDING))

assert(stateBefore==UNINITIALIZED && stateAfter==PENDING => f.selector == scheduleMeeting(uint,uint,uint))

1. **pendingStateTransitions**

stateBefore

f(e,args)

stateAfter

assert(stateBefore==PENDING=> (stateAfter == PENDING||stateAfter == STARTED|| stateAfter==CANCELLED))

assert(stateBefore==PENDING && stateAfter==STARTED=> f.selector == startMeeting(uint))

assert(stateBefore==PENDING && stateAfter==CANCELLED=> f.selector == cancelMeeting(uint))

1. **startedStateTransition**

stateBefore

f(e,args)

stateAfter

assert(stateBefore==STARTED=> (stateAfter == STARTED|| stateAfter==ENDED))

assert(stateBefore==STARTED && stateAfter==ENDED => f.selector == endMeeting(uint))

# Variable transitions

1. numofParticipants can only increase
2. once meeting is initialized, organizer can never be zero

# High-Level properties

1. endTime is always greater than startTime

# Unit tests

1. startMeeting: If stateBefore is PENDING then stateAfter should be STARTED
2. scheduleMeeting: if stateBefore is UNINITIALIZED then stateAfter should be PENDING
3. cancelMeeting: if stateBefore is PENDING then stateAfter should be CANCELLED
4. endMeeting: if stateBefore is STARTED then stateAfter should be ENDED