```
──────────── MODULE ClientSequencer ────────────
EXTENDS Naturals, FiniteSets, Sequences, Bags, TLC

 Message types
CONSTANTS Response, Event

 A bag of messages
VARIABLE messages

 The total number of messages
VARIABLE messageCount

 A sequence of all messaging variables
messageVars ≜ ⟨messages, messageCount⟩

 Server variables
VARIABLES serverSequence, serverIndex, previousIndex

 A sequence of all server variables
serverVars ≜ ⟨serverSequence, serverIndex, previousIndex⟩

 Sequencer variables
VARIABLES responseSequence, responseIndex, eventIndex

 The sequence of all ordered responses
VARIABLE responses

 Variables used for queueing out of order responses and events
VARIABLE pendingResponses, pendingEvents

 A sequence of all client variables
clientVars ≜ ⟨responseSequence, responseIndex, eventIndex, responses, pendingResponses, pendingEvents⟩

 A sequence of all variables used in the spec
vars ≜ ⟨messageVars, serverVars, clientVars⟩

├──────────────────────────────────────────────┤

 The type invariant verifies that the ordering of responses and events in the 'responses'
 variable is sequential
TypeInvariant ≜
    ∧ ∀ r  ∈ DOMAIN responses :
      IF r > 1 THEN
          LET
              current ≜ responses[r]
              previous ≜ responses[r − 1]
          IN
              ∨ ∧ current.type = Event
```

```
            $\land previous.type = Response$
            $\land current.eventIndex \geq previous.index$
          $\lor \land current.type = Response$
            $\land previous.type = Event$
            $\land current.index > previous.eventIndex$
          $\lor \land current.type = Response$
            $\land previous.type = Response$
            $\land current.index > previous.index$
    ELSE
        TRUE
```

Helper for adding a message to a bag of messages

```
$WithMessage(m, msgs) \triangleq$
    IF $m \in$ DOMAIN $msgs$ THEN
        $[msgs$ EXCEPT $![m] = msgs[m] + 1]$
    ELSE
        $msgs$ @@ $(m :> 1)$
```

Helper for removing a message from a bag of messages

```
$WithoutMessage(m, msgs) \triangleq$
    IF $m \in$ DOMAIN $msgs$ THEN
        $[msgs$ EXCEPT $![m] = msgs[m] - 1]$
    ELSE
        $msgs$
```

Helper to send a message

```
$Send(m) \triangleq$
    $\land messages' = WithMessage(m, messages)$
    $\land messageCount' = messageCount + 1$
```

Helper to discard a message

```
$Discard(m) \triangleq$
    $\land messages' = WithoutMessage(m, messages)$
    $\land messageCount' = messageCount + 1$
```

Duplicates a message

```
$Duplicate(m) \triangleq$
    $\land Send(m)$
    $\land$ UNCHANGED $\langle clientVars, serverVars \rangle$
```

Drops a message

```
$Drop(m) \triangleq$
    $\land Discard(m)$
    $\land$ UNCHANGED $\langle clientVars, serverVars \rangle$
```

Sends a response to the client

The response will have a sequence number and index greater than all prior responses

```
$SendResponse \triangleq$
    $\land\ Send([type \mapsto Response,\ index \mapsto serverIndex + 1,\ eventIndex \mapsto previousIndex,\ sequence \mapsto serverSequ$
    $\land\ serverSequence' = serverSequence + 1$
    $\land\ serverIndex' = serverIndex + 1$
    $\land\ \text{UNCHANGED}\ \langle clientVars,\ previousIndex \rangle$
```

Sends an event to the client

The sent event will have an index greater than all prior responses

```
$SendEvent \triangleq$
    $\land\ previousIndex \neq serverIndex$
    $\land\ Send([type \mapsto Event,\ eventIndex \mapsto serverIndex,\ previousIndex \mapsto previousIndex])$
    $\land\ previousIndex' = serverIndex$
    $\land\ \text{UNCHANGED}\ \langle clientVars,\ serverSequence,\ serverIndex \rangle$
```

Accepts a response that has been received in order, adding it to the sequence of ordered responses

```
$AcceptResponse(m) \triangleq$
    $\land\ responses' = responses \circ \langle m \rangle$
    $\land\ responseSequence' = responseSequence + 1$
    $\land\ responseIndex' = m.index$
    $\land\ \text{UNCHANGED}\ \langle pendingResponses \rangle$
```

Queues a response for handling in order

```
$QueueResponse(m) \triangleq$
    $\land\ pendingResponses' = pendingResponses \cup \{m\}$
    $\land\ \text{UNCHANGED}\ \langle responses,\ responseSequence,\ responseIndex \rangle$
```

Handles a response from the cluster

```
$HandleResponse(m) \triangleq$
    $\land\ \lor\ \land\ m.sequence = responseSequence + 1$
            $\land\ m.eventIndex = eventIndex$
            $\land\ AcceptResponse(m)$
        $\lor\ QueueResponse(m)$
    $\land\ \text{UNCHANGED}\ \langle messageVars,\ serverVars,\ eventIndex,\ pendingEvents \rangle$
```

Accepts an event that has been received in order, adding it to the sequence of ordered responses

```
$AcceptEvent(m) \triangleq$
    $\land\ responses' = responses \circ \langle m \rangle$
    $\land\ eventIndex' = m.eventIndex$
    $\land\ \text{UNCHANGED}\ \langle pendingEvents \rangle$
```

Queues an event for handling in order

```
$QueueEvent(m) \triangleq$
      $\land\ pendingEvents' = pendingEvents \cup \{m\}$
      $\land$ UNCHANGED $\langle responses, eventIndex \rangle$
```

Handles an event from the cluster

```
$HandleEvent(m) \triangleq$
      $\land\ \lor\ \land\ m.previousIndex = eventIndex$
              $\land\ m.eventIndex = responseIndex$
              $\land\ AcceptEvent(m)$
          $\lor\ QueueEvent(m)$
      $\land$ UNCHANGED $\langle messageVars, serverVars, responseSequence, responseIndex, pendingResponses \rangle$
```

Receives a message from the cluster

```
$Receive(m) \triangleq$
   $\lor\ \land\ m.type = Response$
      $\land\ HandleResponse(m)$
   $\lor\ \land\ m.type = Event$
      $\land\ HandleEvent(m)$
```

Initial message variables

```
$InitMessageVars \triangleq$
      $\land\ messages = [m \in \{\} \mapsto 0]$
      $\land\ messageCount = 0$
```

Initial server variables

```
$InitServerVars \triangleq$
      $\land\ serverSequence = 1$
      $\land\ serverIndex = 1$
      $\land\ previousIndex = 0$
```

Initial client variables

```
$InitClientVars \triangleq$
      $\land\ responses = \langle\rangle$
      $\land\ pendingResponses = \{\}$
      $\land\ pendingEvents = \{\}$
      $\land\ responseSequence = 0$
      $\land\ responseIndex = 0$
      $\land\ eventIndex = 0$
```

Initial state

```
$Init \triangleq$
      $\land\ InitMessageVars$
      $\land\ InitClientVars$
      $\land\ InitServerVars$
```

Next state predicate

```
$Next \triangleq$
```

$\lor\ \mathit{SendResponse}$
$\lor\ \mathit{SendEvent}$
$\lor\ \exists\, m \in \textsc{domain}\ \mathit{messages} : \mathit{Receive}(m)$
$\lor\ \exists\, m \in \textsc{domain}\ \mathit{messages} : \mathit{Duplicate}(m)$
$\lor\ \exists\, m \in \textsc{domain}\ \mathit{messages} : \mathit{Drop}(m)$
$\lor\ \exists\, m \in \mathit{pendingResponses} : \mathit{HandleResponse}(m)$
$\lor\ \exists\, m \in \mathit{pendingEvents} : \mathit{HandleEvent}(m)$

The specification must start with an initial state and transition according to the next state predicate

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

* Modification History
* Last modified *Wed Jan* 24 21:18:32 *PST* 2018 by *jordanhalterman*
* Created *Wed Jan* 24 10:02:25 *PST* 2018 by *jordanhalterman*