MODULE *E2T*

The *E2AP* module provides a formal specification of the *E2T* service. The spec defines the client and server interfaces for *E2T* and provides helpers for managing and operating on connections.

CONSTANT *Nil*

Message type constants

CONSTANT
    SubscribeRequestType,
    SubscribeResponseType

CONSTANTS
    UnsubscribeRequestType,
    UnsubscribeResponseType

CONSTANTS
    ControlRequestType,
    ControlResponseType

LOCAL *messageTypes* $\triangleq$
    $\{$*SubscribeRequestType*,
    SubscribeResponseType,
    UnsubscribeRequestType,
    UnsubscribeResponseType,
    ControlRequestType,
    ControlResponseType$\}$

Message types should be defined as strings to simplify debugging

ASSUME $\forall\, m \in messageTypes : m \in$ STRING

VARIABLE *conns*

LOCAL INSTANCE *API*

LOCAL INSTANCE *TLC*

vars $\triangleq$ $\langle conns \rangle$

MODULE *Messages*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by *E2T*.

This section defines predicates for identifying *E2T* message types on the network.

$IsSubscribeRequest(m) \triangleq m.type = SubscribeRequestType$

$IsSubscribeResponse(m) \triangleq m.type = SubscribeResponseType$

$IsUnsubscribeRequest(m) \triangleq m.type = UnsubscribeRequestType$

$IsUnsubscribeResponse(m) \triangleq m.type = UnsubscribeResponseType$

$IsControlRequest(m) \triangleq m.type = ControlRequestType$

$IsControlResponse(m) \triangleq m.type = ControlResponseType$

This section defines predicates for validating $E2T$ message contents. The predicates provide precise documentation on the $E2T$ message format and are used within the spec to verify that steps adhere to the $E2T$ protocol specification.

```
LOCAL $ValidSubscribeRequest(m) \triangleq$ TRUE

LOCAL $ValidSubscribeResponse(m) \triangleq$ TRUE

LOCAL $ValidUnsubscribeRequest(m) \triangleq$ TRUE

LOCAL $ValidUnsubscribeResponse(m) \triangleq$ TRUE

LOCAL $ValidControlRequest(m) \triangleq$ TRUE

LOCAL $ValidControlResponse(m) \triangleq$ TRUE
```

This section defines operators for constructing $E2T$ messages.

```
LOCAL $SetType(m, t) \triangleq [m$ EXCEPT $!.type = t]$

$SubscribeRequest(m) \triangleq$
   IF $Assert(ValidSubscribeRequest(m),$ "Invalid SubscribeRequest")
    THEN $SetType(m, SubscribeRequestType)$
    ELSE $Nil$

$SubscribeResponse(m) \triangleq$
   IF $Assert(ValidSubscribeResponse(m),$ "Invalid SubscribeResponse")
    THEN $SetType(m, SubscribeResponseType)$
    ELSE $Nil$

$UnsubscribeRequest(m) \triangleq$
   IF $Assert(ValidUnsubscribeRequest(m),$ "Invalid UnsubscribeRequest")
    THEN $SetType(m, UnsubscribeRequestType)$
    ELSE $Nil$

$UnsubscribeResponse(m) \triangleq$
   IF $Assert(ValidUnsubscribeResponse(m),$ "Invalid UnsubscribeResponse")
    THEN $SetType(m, UnsubscribeResponseType)$
    ELSE $Nil$

$ControlRequest(m) \triangleq$
   IF $Assert(ValidControlRequest(m),$ "Invalid ControlRequest")
    THEN $SetType(m, ControlRequestType)$
    ELSE $Nil$
```

```
ControlResponse(m) ≜
   IF Assert(ValidControlResponse(m), "Invalid ControlResponse")
    THEN SetType(m, ControlResponseType)
    ELSE Nil
```

The *Messages* module is instantiated locally to avoid access from outside the module.

```
LOCAL Messages ≜ INSTANCE Messages
```

──────────────── MODULE *Client* ────────────────

The *Client* module provides operators for managing and operating on $E2T$ client connections and specifies the message types supported for the client.

──────────────── MODULE *Requests* ────────────────

This module provides message type operators for the message types that can be send by the $E2T$ client.

```
SubscribeRequest(c, m) ≜
    ∧ gRPC!Client!Send(c, Messages!SubscribeRequest(m))

UnsubscribeRequest(c, m) ≜
    ∧ gRPC!Client!Send(c, Messages!UnsubscribeRequest(m))

ControlRequest(c, m) ≜
    ∧ gRPC!Client!Send(c, Messages!ControlRequest(m))
```

Instantiate the *E2T!Client!Requests* module

```
Send ≜ INSTANCE Requests
```

──────────────── MODULE *Responses* ────────────────

This module provides predicates for the types of messages that can be received by an $E2T$ client.

```
SubscribeResponse(c, h(_, _)) ≜
   gRPC!Client!Handle(c, LAMBDA x, m :
       ∧ Messages!IsSubscribeResponse(m)
       ∧ gRPC!Client!Receive(c)
       ∧ h(c, m))

UnsubscribeResponse(c, h(_, _)) ≜
   gRPC!Client!Handle(c, LAMBDA x, m :
       ∧ Messages!IsUnsubscribeResponse(m)
       ∧ gRPC!Client!Receive(c)
       ∧ h(c, m))
```

```
    ControlResponse(c, h(_, _)) ≜
        gRPC!Client!Handle(c, LAMBDA x, m :
            ∧ Messages!IsControlResponse(m)
            ∧ gRPC!Client!Receive(c)
            ∧ h(c, m))

  ────────────────────────────────────────────────────────────────

  Instantiate the E2T!Client!Responses module
  Receive ≜ INSTANCE Responses

  Connect(s, d) ≜ gRPC!Client!Connect(s, d)

  Disconnect(c) ≜ gRPC!Client!Disconnect(c)

────────────────────────────────────────────────────────────────────

Provides operators for the E2T client
Client ≜ INSTANCE Client

──────────────────────────── MODULE Server ────────────────────────────
The Server module provides operators for managing and operating on E2T servers and specifies
the message types supported for the server.
  ─────────────────────────── MODULE Responses ───────────────────────────
  This module provides message type operators for the message types that can be send by the
  E2T server.
    SubscribeResponse(c, m) ≜
        ∧ gRPC!Server!Reply(c, Messages!SubscribeResponse(m))

    UnsubscribeResponse(c, m) ≜
        ∧ gRPC!Server!Reply(c, Messages!UnsubscribeResponse(m))

    ControlResponse(c, m) ≜
        ∧ gRPC!Server!Reply(c, Messages!ControlResponse(m))

  ────────────────────────────────────────────────────────────────

  Instantiate the E2T!Server!Responses module
  Send ≜ INSTANCE Responses

  ─────────────────────────── MODULE Requests ───────────────────────────
  This module provides predicates for the types of messages that can be received by an E2T
  server.
    SubscribeRequest(c, h(_, _)) ≜
        gRPC!Server!Handle(c, LAMBDA x, m :
            ∧ Messages!IsSubscribeRequest(m)
            ∧ gRPC!Server!Receive(c)
        $\land h(c, m))$

    $UnsubscribeRequest(c, h(\_, \_)) \triangleq$
      $gRPC!Server!Handle(c,$ LAMBDA $x, m :$
        $\land Messages!IsUnsubscribeRequest(m)$
        $\land gRPC!Server!Receive(c)$
        $\land h(c, m))$

    $ControlRequest(c, h(\_, \_)) \triangleq$
      $gRPC!Server!Handle(c,$ LAMBDA $x, m :$
        $\land Messages!IsControlRequest(m)$
        $\land gRPC!Server!Receive(c)$
        $\land h(c, m))$
```

Instantiate the $E2T!Server!Requests$ module

$Receive \triangleq$ INSTANCE $Requests$

Provides operators for the $E2T$ server

$Server \triangleq$ INSTANCE $Server$

The set of all open $E2T$ connections

$Connections \triangleq gRPC!Connections$

\* Modification History
\* Last modified *Mon Sep* 13 15:16:49 *PDT* 2021 by *jordanhalterman*
\* *Created Mon Sep* 13 14:04:44 *PDT* 2021 by *jordanhalterman*