MODULE *E2AP*

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

LOCAL INSTANCE *Naturals*

LOCAL INSTANCE *Sequences*

LOCAL INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

CONSTANT *Nil*

VARIABLE *conns*

The *E2AP* protocol is implemented on *SCTP*

LOCAL $SCTP \triangleq$ INSTANCE *SCTP*

$vars \triangleq \langle conns \rangle$

MODULE *Messages*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by *E2AP*.

Message type constants

CONSTANTS
E2SetupRequest,
E2SetupResponse,
E2SetupFailure

CONSTANTS
ResetRequest,
ResetResponse

CONSTANTS
RICSubscriptionRequest,
RICSubscriptionResponse,
RICSubscriptionFailure

CONSTANTS
RICSubscriptionDeleteRequest,
RICSubscriptionDeleteResponse,
RICSubscriptionDeleteFailure

CONSTANTS
RICControlRequest,
RICControlResponse,
RICControlFailure,
RICServiceUpdate

CONSTANTS

$E2ConnectionUpdate$,
$E2ConnectionUpdateAcknowledge$,
$E2ConnectionUpdateFailure$

CONSTANTS

$E2NodeConfigurationUpdate$,
$E2NodeConfigurationUpdateAcknowledge$,
$E2NodeConfigurationUpdateFailure$

LOCAL $messageTypes \triangleq$

$\{E2SetupRequest$,
$E2SetupResponse$,
$E2SetupFailure$,
$ResetRequest$,
$ResetResponse$,
$RICSubscriptionRequest$,
$RICSubscriptionResponse$,
$RICSubscriptionFailure$,
$RICSubscriptionDeleteRequest$,
$RICSubscriptionDeleteResponse$,
$RICSubscriptionDeleteFailure$,
$RICControlRequest$,
$RICControlResponse$,
$RICControlFailure$,
$RICServiceUpdate$,
$E2ConnectionUpdate$,
$E2ConnectionUpdateAcknowledge$,
$E2ConnectionUpdateFailure$,
$E2NodeConfigurationUpdate$,
$E2NodeConfigurationUpdateAcknowledge$,
$E2NodeConfigurationUpdateFailure\}$

Message types should be defined as strings to simplify debugging

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

This section defines predicates for identifying $E2AP$ message types on the network.

$IsE2SetupRequest(m) \triangleq m.type = E2SetupRequest$

$IsE2SetupResponse(m) \triangleq m.type = E2SetupResponse$

$IsE2SetupFailure(m) \triangleq m.type = E2SetupFailure$

$IsResetRequest(m) \triangleq m.type = ResetRequest$

$IsResetResponse(m) \triangleq m.type = ResetResponse$

$IsRICSubscriptionRequest(m) \triangleq m.type = RICSubscriptionRequest$

$IsRICSubscriptionResponse(m) \triangleq m.type = RICSubscriptionResponse$

$IsRICSubscriptionFailure(m) \triangleq m.type = RICSubscriptionFailure$

$IsRICSubscriptionDeleteRequest(m) \triangleq m.type = RICSubscriptionDeleteRequest$

$IsRICSubscriptionDeleteResponse(m) \triangleq m.type = RICSubscriptionDeleteResponse$

$IsRICSubscriptionDeleteFailure(m) \triangleq m.type = RICSubscriptionDeleteFailure$

$IsRICControlRequest(m) \triangleq m.type = RICControlRequest$

$IsRICControlResponse(m) \triangleq m.type = RICControlResponse$

$IsRICControlFailure(m) \triangleq m.type = RICControlFailure$

$IsRICServiceUpdate(m) \triangleq m.type = RICServiceUpdate$

$IsE2ConnectionUpdate(m) \triangleq m.type = E2ConnectionUpdate$

$IsE2ConnectionUpdateAcknowledge(m) \triangleq m.type = E2ConnectionUpdateAcknowledge$

$IsE2ConnectionUpdateFailure(m) \triangleq m.type = E2ConnectionUpdateFailure$

$IsE2NodeConfigurationUpdate(m) \triangleq m.type = E2NodeConfigurationUpdate$

$IsE2NodeConfigurationUpdateAcknowledge(m) \triangleq m.type = E2NodeConfigurationUpdateAcknowledge$

$IsE2NodeConfigurationUpdateFailure(m) \triangleq m.type = E2NodeConfigurationUpdateFailure$

This section defines predicates for validating $E2AP$ message contents. The predicates provide precise documentation on the $E2AP$ message format and are used within the spec to verify that steps adhere to the $E2AP$ protocol specification.

LOCAL $ValidE2SetupRequest(m) \triangleq$ TRUE

LOCAL $ValidE2SetupResponse(m) \triangleq$ TRUE

LOCAL $ValidE2SetupFailure(m) \triangleq$ TRUE

LOCAL $ValidResetRequest(m) \triangleq$ TRUE

LOCAL $ValidResetResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionRequest(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionFailure(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteRequest(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteResponse(m) \triangleq$ TRUE

LOCAL $ValidRICSubscriptionDeleteFailure(m) \triangleq$ TRUE

LOCAL $ValidRICControlRequest(m) \triangleq$ TRUE

LOCAL $ValidRICControlResponse(m) \triangleq$ TRUE

LOCAL $ValidRICControlFailure(m) \triangleq$ TRUE

LOCAL $ValidRICServiceUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdateAcknowledge(m) \triangleq$ TRUE

LOCAL $ValidE2ConnectionUpdateFailure(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdate(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdateAcknowledge(m) \triangleq$ TRUE

LOCAL $ValidE2NodeConfigurationUpdateFailure(m) \triangleq$ TRUE

This section defines operators for constructing $E2AP$ messages.

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

LOCAL $SetFailureCause(m,\ c) \triangleq [m$ EXCEPT $!.cause = c]$

$WithE2SetupRequest(m) \triangleq$
  IF $Assert(ValidE2SetupRequest(m),$ "Invalid E2SetupRequest")
  THEN $SetType(m,\ E2SetupRequest)$
  ELSE Nil

$WithE2SetupResponse(m) \triangleq$
  IF $Assert(ValidE2SetupResponse(m),$ "Invalid E2SetupResponse")
  THEN $SetType(m,\ E2SetupResponse)$
  ELSE Nil

$WithE2SetupFailure(m,\ c) \triangleq$
  IF $Assert(ValidE2SetupFailure(m),$ "Invalid E2SetupFailure")
  THEN $SetType(m,\ SetFailureCause(E2SetupFailure,\ c))$
  ELSE Nil

$WithResetRequest(m) \triangleq$
  IF $Assert(ValidResetRequest(m),$ "Invalid ResetRequest")
  THEN $SetType(m,\ ResetRequest)$
  ELSE Nil

$WithResetResponse(m) \triangleq$
IF $Assert(ValidResetResponse(m),$ "Invalid ResetResponse")
THEN $SetType(m, ResetResponse)$
ELSE Nil

$WithRICSubscriptionRequest(m) \triangleq$
IF $Assert(ValidRICSubscriptionRequest(m),$ "Invalid RICSubscriptionRequest")
THEN $SetType(m, RICSubscriptionRequest)$
ELSE Nil

$WithRICSubscriptionResponse(m) \triangleq$
IF $Assert(ValidRICSubscriptionResponse(m),$ "Invalid RICSubscriptionResponse")
THEN $SetType(m, RICSubscriptionResponse)$
ELSE Nil

$WithRICSubscriptionFailure(m, c) \triangleq$
IF $Assert(ValidRICSubscriptionFailure(m),$ "Invalid RICSubscriptionFailure")
THEN $SetType(m, SetFailureCause(RICSubscriptionFailure, c))$
ELSE Nil

$WithRICSubscriptionDeleteRequest(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteRequest(m),$ "Invalid RICSubscriptionDeleteRequest")
THEN $SetType(m, RICSubscriptionDeleteRequest)$
ELSE Nil

$WithRICSubscriptionDeleteResponse(m) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteResponse(m),$ "Invalid RICSubscriptionDeleteResponse")
THEN $SetType(m, RICSubscriptionDeleteResponse)$
ELSE Nil

$WithRICSubscriptionDeleteFailure(m, c) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteFailure(m),$ "Invalid RICSubscriptionDeleteFailure")
THEN $SetType(m, SetFailureCause(RICSubscriptionDeleteFailure, c))$
ELSE Nil

$WithRICControlRequest(m) \triangleq$
IF $Assert(ValidRICControlRequest(m),$ "Invalid RICControlRequest")
THEN $SetType(m, RICControlRequest)$
ELSE Nil

$WithRICControlResponse(m) \triangleq$
IF $Assert(ValidRICControlResponse(m),$ "Invalid RICControlResponse")
THEN $SetType(m, RICControlResponse)$
ELSE Nil

$WithRICControlFailure(m, c) \triangleq$
IF $Assert(ValidRICControlFailure(m),$ "Invalid RICControlFailure")
THEN $SetType(m, SetFailureCause(RICControlFailure, c))$

```
    ELSE Nil

  WithRICServiceUpdate(m) ≜
    IF Assert(ValidRICServiceUpdate(m), "Invalid RICServiceUpdate")
     THEN SetType(m, RICServiceUpdate)
     ELSE Nil

  WithE2ConnectionUpdate(m) ≜
    IF Assert(ValidE2ConnectionUpdate(m), "Invalid E2ConnectionUpdate")
     THEN SetType(m, E2ConnectionUpdate)
     ELSE Nil

  WithE2ConnectionUpdateAcknowledge(m) ≜
    IF Assert(ValidE2ConnectionUpdateAcknowledge(m), "Invalid E2ConnectionUpdateAcknowledge")
     THEN SetType(m, E2ConnectionUpdateAcknowledge)
     ELSE Nil

  WithE2ConnectionUpdateFailure(m, c) ≜
    IF Assert(ValidE2ConnectionUpdateFailure(m), "Invalid E2ConnectionUpdateFailure")
     THEN SetType(m, SetFailureCause(E2ConnectionUpdateFailure, c))
     ELSE Nil

  WithE2NodeConfigurationUpdate(m) ≜
    IF Assert(ValidE2NodeConfigurationUpdate(m), "Invalid E2NodeConfigurationUpdate")
     THEN SetType(m, E2NodeConfigurationUpdate)
     ELSE Nil

  WithE2NodeConfigurationUpdateAcknowledge(m) ≜
    IF Assert(ValidE2NodeConfigurationUpdateAcknowledge(m), "Invalid E2NodeConfigurationUpdateAcknow
     THEN SetType(m, E2NodeConfigurationUpdateAcknowledge)
     ELSE Nil

  WithE2NodeConfigurationUpdateFailure(m, c) ≜
    IF Assert(ValidE2NodeConfigurationUpdateFailure(m), "Invalid E2NodeConfigurationUpdateFailure")
     THEN SetType(m, SetFailureCause(E2NodeConfigurationUpdateFailure, c))
     ELSE Nil
```

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

```
LOCAL Messages ≜ INSTANCE Messages WITH
  E2SetupRequest ← "E2SetupRequest",
  E2SetupResponse ← "E2SetupResponse",
  E2SetupFailure ← "E2SetupFailure",
  ResetRequest ← "ResetRequest",
  ResetResponse ← "ResetResponse",
  RICSubscriptionRequest ← "RICSubscriptionRequest",
```

$RICSubscriptionResponse \leftarrow$ "RICSubscriptionResponse",
$RICSubscriptionFailure \leftarrow$ "RICSubscriptionFailure",
$RICSubscriptionDeleteRequest \leftarrow$ "RICSubscriptionDeleteRequest",
$RICSubscriptionDeleteResponse \leftarrow$ "RICSubscriptionDeleteResponse",
$RICSubscriptionDeleteFailure \leftarrow$ "RICSubscriptionDeleteFailure",
$RICControlRequest \leftarrow$ "RICControlRequest",
$RICControlResponse \leftarrow$ "RICControlResponse",
$RICControlFailure \leftarrow$ "RICControlFailure",
$RICServiceUpdate \leftarrow$ "RICServiceUpdate",
$E2ConnectionUpdate \leftarrow$ "E2ConnectionUpdate",
$E2ConnectionUpdateAcknowledge \leftarrow$ "E2ConnectionUpdateAcknowledge",
$E2ConnectionUpdateFailure \leftarrow$ "E2ConnectionUpdateFailure",
$E2NodeConfigurationUpdate \leftarrow$ "E2NodeConfigurationUpdate",
$E2NodeConfigurationUpdateAcknowledge \leftarrow$ "E2NodeConfigurationUpdateAcknowledge",
$E2NodeConfigurationUpdateFailure \leftarrow$ "E2NodeConfigurationUpdateFailure"

MODULE *Cause*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by $E2AP$.

MODULE *Misc*

CONSTANTS
Unspecified,
ControlProcessingOverload,
HardwareFailure,
OMIntervention

LOCAL $failureCauses \triangleq$
{*Unspecified*,
ControlProcessingOverload,
HardwareFailure,
OMIntervention}

ASSUME $\forall\, c \in failureCauses : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsControlProcessingOverload(m) \triangleq m.cause = ControlProcessingOverload$
$IsHardwareFailure(m) \triangleq m.cause = HardwareFailure$
$IsOMIntervention(m) \triangleq m.cause = OMIntervention$

$Misc \triangleq$ INSTANCE *Misc* WITH
Unspecified $\leftarrow$ "Unspecified",
ControlProcessingOverload $\leftarrow$ "ControlProcessingOverload",
HardwareFailure $\leftarrow$ "HardwareFailure",
OMIntervention $\leftarrow$ "OMIntervention"

MODULE *Protocol*

CONSTANTS
Unspecified,
TransferSyntaxError,
AbstractSyntaxErrorReject,
AbstractSyntaxErrorIgnoreAndNotify,
MessageNotCompatibleWithReceiverState,
SemanticError,
AbstractSyntaxErrorFalselyConstructedMessage

LOCAL $failureCauses \triangleq$
$\{$*Unspecified*,
TransferSyntaxError,
AbstractSyntaxErrorReject,
AbstractSyntaxErrorIgnoreAndNotify,
MessageNotCompatibleWithReceiverState,
SemanticError,
AbstractSyntaxErrorFalselyConstructedMessage$\}$

ASSUME $\forall\, c \in failureCauses : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsTransferSyntaxError(m) \triangleq m.cause = TransferSyntaxError$
$IsAbstractSyntaxErrorReject(m) \triangleq m.cause = AbstractSyntaxErrorReject$
$IsAbstractSyntaxErrorIgnoreAndNotify(m) \triangleq m.cause = AbstractSyntaxErrorIgnoreAndNotify$
$IsMessageNotCompatibleWithReceiverState(m) \triangleq m.cause = MessageNotCompatibleWithReceiverState$
$IsSemanticError(m) \triangleq m.cause = SemanticError$
$IsAbstractSyntaxErrorFalselyConstructedMessage(m) \triangleq m.cause = AbstractSyntaxErrorFalselyConstru$

Protocol $\triangleq$ INSTANCE *Protocol* WITH
Unspecified $\leftarrow$ "Unspecified",
TransferSyntaxError $\leftarrow$ "TransferSyntaxError",
AbstractSyntaxErrorReject $\leftarrow$ "AbstractSyntaxErrorReject",
AbstractSyntaxErrorIgnoreAndNotify $\leftarrow$ "AbstractSyntaxErrorIgnoreAndNotify",
MessageNotCompatibleWithReceiverState $\leftarrow$ "MessageNotCompatibleWithReceiverState",
SemanticError $\leftarrow$ "SemanticError",
AbstractSyntaxErrorFalselyConstructedMessage $\leftarrow$ "AbstractSyntaxErrorFalselyConstructedMessage"

MODULE *RIC*

CONSTANTS
Unspecified,
RANFunctionIDInvalid,
ActionNotSupported,

```
    ExcessiveActions,
    DuplicateAction,
    DuplicateEvent,
    FunctionResourceLimit,
    RequestIDUnknown,
    InconsistentActionSubsequentActionSequence,
    ControlMessageInvalid,
    CallProcessIDInvalid

  LOCAL failureCauses ≜
    {Unspecified,
     RANFunctionIDInvalid,
     ActionNotSupported,
     ExcessiveActions,
     DuplicateAction,
     DuplicateEvent,
     FunctionResourceLimit,
     RequestIDUnknown,
     InconsistentActionSubsequentActionSequence,
     ControlMessageInvalid,
     CallProcessIDInvalid}

  ASSUME ∀ c ∈ failureCauses : c ∈ STRING

  IsUnspecified(m) ≜ m.cause = Unspecified
  IsRANFunctionIDInvalid(m) ≜ m.cause = RANFunctionIDInvalid
  IsActionNotSupported(m) ≜ m.cause = ActionNotSupported
  IsExcessiveActions(m) ≜ m.cause = ExcessiveActions
  IsDuplicateAction(m) ≜ m.cause = DuplicateAction
  IsDuplicateEvent(m) ≜ m.cause = DuplicateEvent
  IsFunctionResourceLimit(m) ≜ m.cause = FunctionResourceLimit
  IsRequestIDUnknown(m) ≜ m.cause = RequestIDUnknown
  IsInconsistentActionSubsequentActionSequence(m) ≜ m.cause = InconsistentActionSubsequentActionSe
  IsControlMessageInvalid(m) ≜ m.cause = ControlMessageInvalid
  IsCallProcessIDInvalid(m) ≜ m.cause = CallProcessIDInvalid

RIC ≜ INSTANCE RIC WITH
  Unspecified ← "Unspecified",
  RANFunctionIDInvalid ← "RANFunctionIDInvalid",
  ActionNotSupported ← "ActionNotSupported",
  ExcessiveActions ← "ExcessiveActions",
  DuplicateAction ← "DuplicateAction",
  DuplicateEvent ← "DuplicateEvent",
  FunctionResourceLimit ← "FunctionResourceLimit",
```

$RequestIDUnknown \leftarrow$ "RequestIDUnknown",
$InconsistentActionSubsequentActionSequence \leftarrow$ "InconsistentActionSubsequentActionSequence",
$ControlMessageInvalid \leftarrow$ "ControlMessageInvalid",
$CallProcessIDInvalid \leftarrow$ "CallProcessIDInvalid"

MODULE *RICService*

CONSTANTS
Unspecified,
FunctionNotRequired,
ExcessiveFunctions,
RICResourceLimit

LOCAL $failureCauses \triangleq$
$\{Unspecified,$
$FunctionNotRequired,$
$ExcessiveFunctions,$
$RICResourceLimit\}$

ASSUME $\forall\, c \in failureCauses : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsFunctionNotRequired(m) \triangleq m.cause = FunctionNotRequired$
$IsExcessiveFunctions(m) \triangleq m.cause = ExcessiveFunctions$
$IsRICResourceLimit(m) \triangleq m.cause = RICResourceLimit$

RICService $\triangleq$ INSTANCE *RICService* WITH
Unspecified $\leftarrow$ "Unspecified",
FunctionNotRequired $\leftarrow$ "FunctionNotRequired",
ExcessiveFunctions $\leftarrow$ "ExcessiveFunctions",
RICResourceLimit $\leftarrow$ "RICResourceLimit"

MODULE *Transport*

CONSTANTS
Unspecified,
TransportResourceUnavailable

LOCAL $failureCauses \triangleq$
$\{Unspecified,$
$TransportResourceUnavailable\}$

ASSUME $\forall\, c \in failureCauses : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsTransportResourceUnavailable(m) \triangleq m.cause = TransportResourceUnavailable$

$Transport \triangleq$ INSTANCE $Transport$ WITH
$Unspecified \leftarrow$ "Unspecified",
$TransportResourceUnavailable \leftarrow$ "TransportResourceUnavailable"

This section defines predicates for identifying $E2AP$ message types on the network.

The *Cause* module provides failure causes
$Cause \triangleq$ INSTANCE $Cause$

MODULE *Client*

The *Client* module provides operators for managing and operating on $E2AP$ 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 $E2AP$ client.

$E2SetupRequest(c,\ m) \triangleq$
$\land\ SCTP!Client!Send(c,\ Messages!WithE2SetupResponse(m))$

$ResetRequest(c,\ m) \triangleq$
$\land\ SCTP!Client!Send(c,\ Messages!WithResetRequest(m))$

$ResetResponse(c,\ m) \triangleq$
$\land\ SCTP!Client!Reply(c,\ Messages!WithResetResponse(m))$

Instantiate the $E2AP!Client!Send$ module
$Send \triangleq$ INSTANCE $Requests$

MODULE *Responses*

This module provides predicates for the types of messages that can be received by an $E2AP$ client.

$E2SetupResponse(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsE2SetupResponse(m)$
$\land\ SCTP!Client!Receive(c)$
$\land\ h(c,\ m))$

$ResetRequest(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsResetRequest(m)$
$\land\ SCTP!Client!Receive(c)$
$\land\ h(c,\ m))$

$ResetResponse(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsResetResponse(m)$
$\land\ SCTP!Client!Receive(c)$
$\land\ h(c,\ m))$

Instantiate the $E2AP!Client!Receive$ module
$Receive \triangleq$ INSTANCE $Responses$

$Connect(s,\ d) \triangleq SCTP!Client!Connect(s,\ d)$

$Disconnect(c) \triangleq SCTP!Client!Disconnect(c)$

Provides operators for the $E2AP$ client
$Client \triangleq$ INSTANCE $Client$

MODULE $Server$

The $Server$ module provides operators for managing and operating on $E2AP$ 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 $E2AP$ server.

$E2SetupResponse(c,\ m) \triangleq$
$\land\ SCTP!Server!Reply(c,\ Messages!WithE2SetupResponse(m))$

$ResetRequest(c,\ m) \triangleq$
$\land\ SCTP!Server!Send(c,\ Messages!WithResetRequest(m))$

$ResetResponse(c,\ m) \triangleq$
$\land\ SCTP!Server!Reply(c,\ Messages!WithResetResponse(m))$

Instantiate the $E2AP!Server!Send$ module
$Send \triangleq$ INSTANCE $Responses$

MODULE $Requests$

This module provides predicates for the types of messages that can be received by an $E2AP$ server.

$E2SetupRequest(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,$ LAMBDA $x,\ m :$
$\land\ Messages!IsE2SetupRequest(m)$
$\land\ SCTP!Server!Receive(c)$

$\land\ h(c,\ m))$

$ResetRequest(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,\ \text{LAMBDA}\ x,\ m :$
$\land\ Messages!IsResetRequest(m)$
$\land\ SCTP!Server!Receive(c)$
$\land\ h(c,\ m))$

$ResetResponse(c,\ h(\_,\ \_)) \triangleq$
$SCTP!Server!Handle(c,\ \text{LAMBDA}\ x,\ m :$
$\land\ Messages!IsResetResponse(m)$
$\land\ SCTP!Server!Receive(c)$
$\land\ h(c,\ m))$

Instantiate the $E2AP!Server!Receive$ module
$Receive \triangleq \text{INSTANCE}\ Requests$

Provides operators for the $E2AP$ server
$Server \triangleq \text{INSTANCE}\ Server$

The set of all open $E2AP$ connections
$Connections \triangleq SCTP!Connections$

$Init \triangleq SCTP!Init$

$Next \triangleq SCTP!Next$

---

* Modification History
* Last modified *Mon Sep* 13 16:15:39 *PDT* 2021 by *jordanhalterman*
* *Created Mon Sep* 13 10:53:17 *PDT* 2021 by *jordanhalterman*