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 *Cause*

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

MODULE *Misc*

```
CONSTANTS
    Unspecified,
    ControlProcessingOverload,
    HardwareFailure,
    OMIntervention

All ≜
    {Unspecified,
     ControlProcessingOverload,
     HardwareFailure,
     OMIntervention}

ASSUME ∀ c ∈ All : c ∈ STRING

IsUnspecified(m) ≜ m.cause = Unspecified
IsControlProcessingOverload(m) ≜ m.cause = ControlProcessingOverload
IsHardwareFailure(m) ≜ m.cause = HardwareFailure
IsOMIntervention(m) ≜ m.cause = OMIntervention
```

```
Misc ≜ INSTANCE Misc WITH
    Unspecified ← "Unspecified",
```

$\quad ControlProcessingOverload \leftarrow$ "ControlProcessingOverload",
$\quad HardwareFailure \leftarrow$ "HardwareFailure",
$\quad OMIntervention \leftarrow$ "OMIntervention"

———————————— MODULE *Protocol* ————————————

CONSTANTS
$\quad Unspecified,$
$\quad TransferSyntaxError,$
$\quad AbstractSyntaxErrorReject,$
$\quad AbstractSyntaxErrorIgnoreAndNotify,$
$\quad MessageNotCompatibleWithReceiverState,$
$\quad SemanticError,$
$\quad AbstractSyntaxErrorFalselyConstructedMessage$

$All \triangleq$
$\quad \{Unspecified,$
$\quad\; TransferSyntaxError,$
$\quad\; AbstractSyntaxErrorReject,$
$\quad\; AbstractSyntaxErrorIgnoreAndNotify,$
$\quad\; MessageNotCompatibleWithReceiverState,$
$\quad\; SemanticError,$
$\quad\; AbstractSyntaxErrorFalselyConstructedMessage\}$

ASSUME $\forall c \in All : c \in$ 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
$\quad Unspecified \leftarrow$ "Unspecified",
$\quad TransferSyntaxError \leftarrow$ "TransferSyntaxError",
$\quad AbstractSyntaxErrorReject \leftarrow$ "AbstractSyntaxErrorReject",
$\quad AbstractSyntaxErrorIgnoreAndNotify \leftarrow$ "AbstractSyntaxErrorIgnoreAndNotify",
$\quad MessageNotCompatibleWithReceiverState \leftarrow$ "MessageNotCompatibleWithReceiverState",
$\quad SemanticError \leftarrow$ "SemanticError",
$\quad AbstractSyntaxErrorFalselyConstructedMessage \leftarrow$ "AbstractSyntaxErrorFalselyConstructedMessage"

———————————— MODULE *RIC* ————————————

CONSTANTS

$\mathit{Unspecified}$,
$\mathit{RANFunctionIDInvalid}$,
$\mathit{ActionNotSupported}$,
$\mathit{ExcessiveActions}$,
$\mathit{DuplicateAction}$,
$\mathit{DuplicateEvent}$,
$\mathit{FunctionResourceLimit}$,
$\mathit{RequestIDUnknown}$,
$\mathit{InconsistentActionSubsequentActionSequence}$,
$\mathit{ControlMessageInvalid}$,
$\mathit{CallProcessIDInvalid}$

$\mathit{All} \triangleq$
$\{\mathit{Unspecified}$,
$\mathit{RANFunctionIDInvalid}$,
$\mathit{ActionNotSupported}$,
$\mathit{ExcessiveActions}$,
$\mathit{DuplicateAction}$,
$\mathit{DuplicateEvent}$,
$\mathit{FunctionResourceLimit}$,
$\mathit{RequestIDUnknown}$,
$\mathit{InconsistentActionSubsequentActionSequence}$,
$\mathit{ControlMessageInvalid}$,
$\mathit{CallProcessIDInvalid}\}$

ASSUME $\forall c \in \mathit{All} : c \in$ STRING

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

$\mathit{RIC} \triangleq$ INSTANCE RIC WITH
$\mathit{Unspecified} \leftarrow$ "Unspecified",
$\mathit{RANFunctionIDInvalid} \leftarrow$ "RANFunctionIDInvalid",
$\mathit{ActionNotSupported} \leftarrow$ "ActionNotSupported",
$\mathit{ExcessiveActions} \leftarrow$ "ExcessiveActions",

```
  DuplicateAction ← "DuplicateAction",
  DuplicateEvent ← "DuplicateEvent",
  FunctionResourceLimit ← "FunctionResourceLimit",
  RequestIDUnknown ← "RequestIDUnknown",
  InconsistentActionSubsequentActionSequence ← "InconsistentActionSubsequentActionSequence",
  ControlMessageInvalid ← "ControlMessageInvalid",
  CallProcessIDInvalid ← "CallProcessIDInvalid"

┌──────────────────── MODULE RICService ────────────────────┐

  CONSTANTS
     Unspecified,
     FunctionNotRequired,
     ExcessiveFunctions,
     RICResourceLimit

  All ≜
     {Unspecified,
      FunctionNotRequired,
      ExcessiveFunctions,
      RICResourceLimit}

  ASSUME ∀ c ∈ All : c ∈ STRING

  IsUnspecified(m) ≜ m.cause = Unspecified
  IsFunctionNotRequired(m) ≜ m.cause = FunctionNotRequired
  IsExcessiveFunctions(m) ≜ m.cause = ExcessiveFunctions
  IsRICResourceLimit(m) ≜ m.cause = RICResourceLimit

└───────────────────────────────────────────────────────────┘

RICService ≜ INSTANCE RICService WITH
  Unspecified ← "Unspecified",
  FunctionNotRequired ← "FunctionNotRequired",
  ExcessiveFunctions ← "ExcessiveFunctions",
  RICResourceLimit ← "RICResourceLimit"

┌──────────────────── MODULE Transport ────────────────────┐

  CONSTANTS
     Unspecified,
     TransportResourceUnavailable

  All ≜
     {Unspecified,
      TransportResourceUnavailable}

  ASSUME ∀ c ∈ All : c ∈ STRING
```

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

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

$All \triangleq Misc!All \cup Protocol!All \cup RIC!All \cup RICService!All \cup Transport!All$

$IsCause(c) \triangleq c \in All$

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

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

MODULE *Messages*

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

Message type constants

CONSTANTS
$\quad E2SetupRequest,$
$\quad E2SetupResponse,$
$\quad E2SetupFailure$
CONSTANTS
$\quad RICServiceUpdate,$
$\quad RICServiceUpdateAcknowledge,$
$\quad RICServiceUpdateFailure$
CONSTANTS
$\quad ResetRequest,$
$\quad ResetResponse$
CONSTANTS
$\quad RICSubscriptionRequest,$
$\quad RICSubscriptionResponse,$
$\quad RICSubscriptionFailure$
CONSTANTS
$\quad RICSubscriptionDeleteRequest,$
$\quad RICSubscriptionDeleteResponse,$
$\quad RICSubscriptionDeleteFailure$
CONSTANTS
$\quad RICIndication$
CONSTANTS
$\quad RICControlRequest,$

$\quad RICControlResponse,$
$\quad RICControlFailure$

CONSTANTS
$\quad E2ConnectionUpdate,$
$\quad E2ConnectionUpdateAcknowledge,$
$\quad E2ConnectionUpdateFailure$

CONSTANTS
$\quad E2NodeConfigurationUpdate,$
$\quad E2NodeConfigurationUpdateAcknowledge,$
$\quad E2NodeConfigurationUpdateFailure$

LOCAL $messageTypes \triangleq$
$\quad \{E2SetupRequest,$
$\quad\; E2SetupResponse,$
$\quad\; E2SetupFailure,$
$\quad\; RICServiceUpdate,$
$\quad\; RICServiceUpdateAcknowledge,$
$\quad\; RICServiceUpdateFailure,$
$\quad\; ResetRequest,$
$\quad\; ResetResponse,$
$\quad\; RICSubscriptionRequest,$
$\quad\; RICSubscriptionResponse,$
$\quad\; RICSubscriptionFailure,$
$\quad\; RICSubscriptionDeleteRequest,$
$\quad\; RICSubscriptionDeleteResponse,$
$\quad\; RICSubscriptionDeleteFailure,$
$\quad\; RICControlRequest,$
$\quad\; RICControlResponse,$
$\quad\; RICControlFailure,$
$\quad\; RICServiceUpdate,$
$\quad\; E2ConnectionUpdate,$
$\quad\; E2ConnectionUpdateAcknowledge,$
$\quad\; E2ConnectionUpdateFailure,$
$\quad\; E2NodeConfigurationUpdate,$
$\quad\; E2NodeConfigurationUpdateAcknowledge,$
$\quad\; E2NodeConfigurationUpdateFailure\}$

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.

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

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

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

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

$IsRICServiceUpdateAcknowledge(m) \triangleq m.type = RICServiceUpdateAcknowledge$

$IsRICServiceUpdateFailure(m) \triangleq m.type = RICServiceUpdateFailure$

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

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

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

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

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

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

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

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

$IsRICIndication(m) \triangleq m.type = RICIndication$

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

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

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

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

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

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

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

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

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

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 $ValidE2SetupRequest(m) \triangleq$
    $\land$  $\land$ “transactionId” $\in$ DOMAIN $m$
         $\land$ $m$[“transactionId”] $\in Nat$
    $\land$  $\land$ “globalE2NodeId” $\in$ DOMAIN $m$
```

```
          ∧ m["globalE2NodeId"] ∈ Nat

LOCAL ValidE2SetupResponse(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat
    ∧     ∧ "globalRicId" ∈ DOMAIN m
          ∧ m["globalRicId"] ∈ Nat

LOCAL ValidE2SetupFailure(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat
    ∧     ∧ "cause" ∈ DOMAIN m
          ∧ m["cause"] ∈ Cause!All

LOCAL ValidRICServiceUpdate(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat

LOCAL ValidRICServiceUpdateAcknowledge(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat

LOCAL ValidRICServiceUpdateFailure(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat
    ∧     ∧ "cause" ∈ DOMAIN m
          ∧ m["cause"] ∈ Cause!All

LOCAL ValidResetRequest(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat

LOCAL ValidResetResponse(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat

LOCAL ValidE2ConnectionUpdate(m) ≜
    ∧     ∧ "transactionId" ∈ DOMAIN m
          ∧ m["transactionId"] ∈ Nat
    ∧     ∧ "add" ∈ DOMAIN m ⇒
              ∧ IsFiniteSet(m["add"])
              ∧ ∀ a ∈ m["add"] : a ∈ STRING
          ∧ "update" ∈ DOMAIN m ⇒
              ∧ IsFiniteSet(m["update"])
              ∧ ∀ a ∈ m["update"] : a ∈ STRING
          ∧ "remove" ∈ DOMAIN m ⇒
              ∧ IsFiniteSet(m["remove"])
```

```
               ∧ ∀ a ∈ m["remove"] : a ∈ STRING

LOCAL ValidE2ConnectionUpdateAcknowledge(m) ≜
   ∧    ∧ "transactionId" ∈ DOMAIN m
        ∧ m["transactionId"] ∈ Nat
   ∧    ∧ "succeeded" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["succeeded"])
           ∧ ∀ a ∈ m["succeeded"] : a ∈ STRING
        ∧ "failed" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["failed"])
           ∧ ∀ a ∈ m["failed"] : a ∈ STRING

LOCAL ValidE2ConnectionUpdateFailure(m) ≜
   ∧    ∧ "transactionId" ∈ DOMAIN m
        ∧ m["transactionId"] ∈ Nat
   ∧    ∧ "cause" ∈ DOMAIN m
        ∧ m["cause"] ∈ Cause!All

LOCAL ValidE2NodeConfigurationUpdate(m) ≜
   ∧    ∧ "transactionId" ∈ DOMAIN m
        ∧ m["transactionId"] ∈ Nat
   ∧    ∧ "globalE2NodeId" ∈ DOMAIN m
        ∧ m["globalE2NodeId"] ∈ Nat
   ∧    ∧ "add" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["add"])
        ∧ "update" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["update"])
        ∧ "remove" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["remove"])

LOCAL ValidE2NodeConfigurationUpdateAcknowledge(m) ≜
   ∧    ∧ "transactionId" ∈ DOMAIN m
        ∧ m["transactionId"] ∈ Nat
   ∧    ∧ "add" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["add"])
        ∧ "update" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["update"])
        ∧ "remove" ∈ DOMAIN m ⇒
           ∧ IsFiniteSet(m["remove"])

LOCAL ValidE2NodeConfigurationUpdateFailure(m) ≜
   ∧    ∧ "transactionId" ∈ DOMAIN m
        ∧ m["transactionId"] ∈ Nat
   ∧    ∧ "cause" ∈ DOMAIN m
        ∧ m["cause"] ∈ Cause!All

LOCAL ValidRICSubscriptionRequest(m) ≜
```

```
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat

LOCAL ValidRICSubscriptionResponse(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat

LOCAL ValidRICSubscriptionFailure(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat
    ∧  ∧ "cause" ∈ DOMAIN m
       ∧ m["cause"] ∈ Cause!All

LOCAL ValidRICSubscriptionDeleteRequest(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat

LOCAL ValidRICSubscriptionDeleteResponse(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat

LOCAL ValidRICSubscriptionDeleteFailure(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat
    ∧  ∧ "cause" ∈ DOMAIN m
       ∧ m["cause"] ∈ Cause!All

LOCAL ValidRICIndication(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat

LOCAL ValidRICControlRequest(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat

LOCAL ValidRICControlAcknowledge(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat

LOCAL ValidRICControlFailure(m) ≜
    ∧  ∧ "requestId" ∈ DOMAIN m
       ∧ m["requestId"] ∈ Nat
    ∧  ∧ "cause" ∈ DOMAIN m
       ∧ m["cause"] ∈ Cause!All
```

This section defines operators for constructing *E2AP* messages.

```
LOCAL SetType(m, t) ≜ [m EXCEPT !.type = t]

LOCAL SetFailureCause(m, c) ≜ [m EXCEPT !.cause = c]

WithE2SetupRequest(m) ≜
  IF Assert(ValidE2SetupRequest(m), "Invalid E2SetupRequest")
   THEN SetType(m, E2SetupRequest)
   ELSE Nil

WithE2SetupResponse(m) ≜
  IF Assert(ValidE2SetupResponse(m), "Invalid E2SetupResponse")
   THEN SetType(m, E2SetupResponse)
   ELSE Nil

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

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

WithRICServiceUpdateAcknowledge(m) ≜
  IF Assert(ValidRICServiceUpdateAcknowledge(m), "Invalid RICServiceUpdateAcknowledge")
   THEN SetType(m, RICServiceUpdateAcknowledge)
   ELSE Nil

WithRICServiceUpdateFailure(m, c) ≜
  IF Assert(ValidRICServiceUpdateFailure(m), "Invalid RICServiceUpdateFailure")
   THEN SetType(m, SetFailureCause(RICServiceUpdateFailure, c))
   ELSE Nil

WithResetRequest(m) ≜
  IF Assert(ValidResetRequest(m), "Invalid ResetRequest")
   THEN SetType(m, ResetRequest)
   ELSE Nil

WithResetResponse(m) ≜
  IF Assert(ValidResetResponse(m), "Invalid ResetResponse")
   THEN SetType(m, ResetResponse)
   ELSE Nil

WithRICSubscriptionRequest(m) ≜
  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 $Nil$

$WithRICIndication(m) \triangleq$
  IF $Assert(ValidRICIndication(m),$ "Invalid RICIndication"$)$
   THEN $SetType(m, RICIndication)$
   ELSE $Nil$

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

$WithRICControlAcknowledge(m) \triangleq$
  IF $Assert(ValidRICControlAcknowledge(m),$ "Invalid RICControlAcknowledge"$)$
   THEN $SetType(m, RICControlResponse)$
   ELSE $Nil$

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

$WithE2ConnectionUpdate(m) \triangleq$
  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 ← "RICSubscriptionResponse",
  RICSubscriptionFailure ← "RICSubscriptionFailure",
  RICSubscriptionDeleteRequest ← "RICSubscriptionDeleteRequest",
  RICSubscriptionDeleteResponse ← "RICSubscriptionDeleteResponse",
  RICSubscriptionDeleteFailure ← "RICSubscriptionDeleteFailure",
  RICIndication ← "RICIndication",
  RICControlRequest ← "RICControlRequest",
  RICControlResponse ← "RICControlResponse",
  RICControlFailure ← "RICControlFailure",
```

$RICServiceUpdate \leftarrow$ "RICServiceUpdate",
$RICServiceUpdateAcknowledge \leftarrow$ "RICServiceUpdateAcknowledge",
$RICServiceUpdateFailure \leftarrow$ "RICServiceUpdateFailure",
$E2ConnectionUpdate \leftarrow$ "E2ConnectionUpdate",
$E2ConnectionUpdateAcknowledge \leftarrow$ "E2ConnectionUpdateAcknowledge",
$E2ConnectionUpdateFailure \leftarrow$ "E2ConnectionUpdateFailure",
$E2NodeConfigurationUpdate \leftarrow$ "E2NodeConfigurationUpdate",
$E2NodeConfigurationUpdateAcknowledge \leftarrow$ "E2NodeConfigurationUpdateAcknowledge",
$E2NodeConfigurationUpdateFailure \leftarrow$ "E2NodeConfigurationUpdateFailure"

MODULE *E2Node*

The *Client* module provides operators for managing and operating on *E2AP* client connections and specifies the message types supported for the client.

CONSTANT *ID*

MODULE *Send*

This module provides message type operators for the message types that can be send by the *E2AP* client.

$E2SetupRequest(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithE2SetupResponse(msg))$

$RICServiceUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICServiceUpdate(msg))$

$ResetRequest(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithResetRequest(msg))$

$ResetResponse(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithResetResponse(msg))$

$RICSubscriptionResponse(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICSubscriptionResponse(msg))$

$RICSubscriptionFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICSubscriptionFailure(msg, cause))$

$RICSubscriptionDeleteResponse(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICSubscriptionDeleteResponse(msg))$

$RICSubscriptionDeleteFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICSubscriptionDeleteFailure(msg, cause))$

$RICIndication(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICIndication(msg))$

$RICControlAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICControlAcknowledge(msg))$

$RICControlFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithRICControlFailure(msg, cause))$

$E2ConnectionUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

Instantiate the $E2AP!Client!$Requests module

$Send \triangleq$ INSTANCE $Send$

MODULE *Reply*

This module provides message type operators for the message types that can be send by the $E2AP$ client.

$ResetResponse(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithResetResponse(msg))$

$RICSubscriptionResponse(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithRICSubscriptionResponse(msg))$

$RICSubscriptionFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithRICSubscriptionFailure(msg, cause))$

$RICSubscriptionDeleteResponse(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithRICSubscriptionDeleteResponse(msg))$

$RICSubscriptionDeleteFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithRICSubscriptionDeleteFailure(msg, cause))$

$RICIndication(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithRICIndication(msg))$

$RICControlAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithRICControlAcknowledge(msg))$

$RICControlFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithRICControlFailure(msg, cause))$

$E2ConnectionUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Client(ID)!Reply(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

Instantiate the $E2AP!Client!Reply$ module

$Reply \triangleq$ INSTANCE $Reply$

MODULE *Receive*

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

$E2SetupResponse(conn, handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x, m :$
$\quad\quad \land Messages!IsE2SetupResponse(m)$
$\quad\quad \land SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land handler(m))$

$RICServiceUpdateAcknowledge(conn, handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x, m :$
$\quad\quad \land Messages!IsRICServiceUpdateAcknowledge(m)$
$\quad\quad \land SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land handler(m))$

$RICServiceUpdateFailure(conn, handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x, m :$
$\quad\quad \land Messages!IsRICServiceUpdateFailure(m)$
$\quad\quad \land SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land handler(m))$

$ResetRequest(conn, handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x, m :$
$\quad\quad \land Messages!IsResetRequest(m)$
$\quad\quad \land SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land handler(m))$

$ResetResponse(conn, handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x, m :$
$\quad\quad \land Messages!IsResetResponse(m)$
$\quad\quad \land SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land handler(m))$

```
$RICSusbcriptionRequest(conn, handler(\_)) \triangleq$
    $SCTP!Client(ID)!Handle(conn, \textsc{lambda}\ x, m :$
        $\land Messages!IsRICSubscriptionRequest(m)$
        $\land SCTP!Client(ID)!Receive(conn)$
        $\land handler(m))$

$RICSubscriptionDeleteRequest(conn, handler(\_)) \triangleq$
    $SCTP!Client(ID)!Handle(conn, \textsc{lambda}\ x, m :$
        $\land Messages!IsRICSubscriptionDeleteRequest(m)$
        $\land SCTP!Client(ID)!Receive(conn)$
        $\land handler(m))$

$RICControlRequest(conn, handler(\_)) \triangleq$
    $SCTP!Client(ID)!Handle(conn, \textsc{lambda}\ x, m :$
        $\land Messages!IsRICControlRequest(m)$
        $\land SCTP!Client(ID)!Receive(conn)$
        $\land handler(m))$

$E2ConnectionUpdate(conn, handler(\_)) \triangleq$
    $SCTP!Client(ID)!Handle(conn, \textsc{lambda}\ x, m :$
        $\land Messages!IsE2ConnectionUpdate(m)$
        $\land SCTP!Client(ID)!Receive(conn)$
        $\land handler(m))$

$E2ConnectionUpdateAcknowledge(conn, handler(\_)) \triangleq$
    $SCTP!Client(ID)!Handle(conn, \textsc{lambda}\ x, m :$
        $\land Messages!IsE2ConnectionUpdateAcknowledge(m)$
        $\land SCTP!Client(ID)!Receive(conn)$
        $\land handler(m))$

$E2NodeConfigurationUpdate(conn, handler(\_)) \triangleq$
    $SCTP!Client(ID)!Handle(conn, \textsc{lambda}\ x, m :$
        $\land Messages!IsE2NodeConfigurationUpdate(m)$
        $\land SCTP!Client(ID)!Receive(conn)$
        $\land handler(m))$

$E2NodeConfigurationUpdateAcknowledge(conn, handler(\_)) \triangleq$
    $SCTP!Client(ID)!Handle(conn, \textsc{lambda}\ x, m :$
        $\land Messages!IsE2NodeConfigurationUpdateAcknowledge(m)$
        $\land SCTP!Client(ID)!Receive(conn)$
        $\land handler(m))$
```

Instantiate the $E2AP!Client!$Responses module

```
$Handle \triangleq \textsc{instance}\ Receive$

$Connect(d) \triangleq SCTP!Client(ID)!Connect(d)$
```

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

The set of all open $E2AP$ connections

$Connections \triangleq SCTP!Client(ID)!Connections$

Provides operators for the $E2AP$ client

$E2Node(ID) \triangleq$ INSTANCE $E2Node$

MODULE RIC

The $Server$ module provides operators for managing and operating on $E2AP$ servers and specifies the message types supported for the server.

CONSTANT ID

MODULE $Send$

This module provides message type operators for the message types that can be send by the $E2AP$ server.

$E2SetupResponse(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2SetupResponse(msg))$

$RICServiceUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithRICServiceUpdateAcknowledge(msg))$

$RICServiceUpdateFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithRICServiceUpdateFailure(msg, cause))$

$ResetRequest(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithResetRequest(msg))$

$ResetResponse(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithResetResponse(msg))$

$E2ConnectionUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

Instantiate the $E2AP!Server!Send$ module

$Send \triangleq$ INSTANCE $Send$

```
──────────────── MODULE Reply ────────────────
This module provides message type operators for the message types that can be send by the
E2AP server.
  E2SetupResponse(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithE2SetupResponse(msg))

  RICServiceUpdateAcknowledge(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithRICServiceUpdateAcknowledge(msg))

  RICServiceUpdateFailure(conn, msg, cause) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithRICServiceUpdateFailure(msg, cause))

  ResetRequest(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithResetRequest(msg))

  ResetResponse(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithResetResponse(msg))

  E2ConnectionUpdate(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithE2ConnectionUpdate(msg))

  E2ConnectionUpdateAcknowledge(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))

  E2NodeConfigurationUpdate(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithE2NodeConfigurationUpdate(msg))

  E2NodeConfigurationUpdateAcknowledge(conn, msg) ≜
      ∧ SCTP!Server(ID)!Reply(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))
──────────────────────────────────────────────

 Instantiate the E2AP!Server!Reply module
Reply ≜ INSTANCE Reply

──────────────── MODULE Receive ────────────────
This module provides predicates for the types of messages that can be received by an E2AP
server.
  E2SetupRequest(conn, handler(_)) ≜
      SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
          ∧ Messages!IsE2SetupRequest(m)
          ∧ SCTP!Server(ID)!Receive(conn)
          ∧ handler(m))

  RICServiceUpdate(conn, handler(_)) ≜
      SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
          ∧ Messages!IsRICServiceUpdate(m)
```

```
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$ResetRequest(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
        $\land Messages!IsResetRequest(m)$
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$ResetResponse(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
        $\land Messages!IsResetResponse(m)$
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$RICSubscriptionResponse(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
        $\land Messages!IsRICSubscriptionResponse(m)$
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$RICSubscriptionDeleteResponse(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
        $\land Messages!IsRICSubscriptionDeleteResponse(m)$
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$RICControlResponse(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
        $\land Messages!IsRICControlResponse(m)$
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$RICIndication(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
        $\land Messages!IsRICIndication(m)$
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$E2ConnectionUpdate(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
        $\land Messages!IsE2ConnectionUpdate(m)$
        $\land SCTP!Server(ID)!Receive(conn)$
        $\land handler(m))$

$E2ConnectionUpdateAcknowledge(conn, handler(\_)) \triangleq$
    $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
            $\land$ $Messages!IsE2ConnectionUpdateAcknowledge(m)$
            $\land$ $SCTP!Server(ID)!Receive(conn)$
            $\land$ $handler(m))$

    $E2NodeConfigurationUpdate(conn, handler(\_)) \triangleq$
        $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
            $\land$ $Messages!IsE2NodeConfigurationUpdate(m)$
            $\land$ $SCTP!Server(ID)!Receive(conn)$
            $\land$ $handler(m))$

    $E2NodeConfigurationUpdateAcknowledge(conn, handler(\_)) \triangleq$
        $SCTP!Server(ID)!Handle(conn,$ LAMBDA $x, m :$
            $\land$ $Messages!IsE2NodeConfigurationUpdateAcknowledge(m)$
            $\land$ $SCTP!Server(ID)!Receive(conn)$
            $\land$ $handler(m))$

  └──────────────────────────────────────────────────────┘

   Instantiate the $E2AP!Server!$Requests module
  $Handle \triangleq$ INSTANCE $Receive$

   The set of all open $E2AP$ connections
  $Connections \triangleq SCTP!Server(ID)!Connections$

└────────────────────────────────────────────────────────┘

 Provides operators for the $E2AP$ server
$RIC(ID) \triangleq$ INSTANCE $RIC$

$Init \triangleq SCTP!Init$

$Next \triangleq SCTP!Next$

└──────────────────────────────────────────────────────────┘
```

* Modification History
* Last modified *Tue Sep* 21 05:27:59 *PDT* 2021 by *jordanhalterman*
* Created *Mon Sep* 13 10:53:17 *PDT* 2021 by *jordanhalterman*