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",
```

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

MODULE *Protocol*

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

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

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
$Unspecified \leftarrow$ "Unspecified",
$TransferSyntaxError \leftarrow$ "TransferSyntaxError",
$AbstractSyntaxErrorReject \leftarrow$ "AbstractSyntaxErrorReject",
$AbstractSyntaxErrorIgnoreAndNotify \leftarrow$ "AbstractSyntaxErrorIgnoreAndNotify",
$MessageNotCompatibleWithReceiverState \leftarrow$ "MessageNotCompatibleWithReceiverState",
$SemanticError \leftarrow$ "SemanticError",
$AbstractSyntaxErrorFalselyConstructedMessage \leftarrow$ "AbstractSyntaxErrorFalselyConstructedMessage"

MODULE *RIC*

CONSTANTS

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

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

$\textsc{assume}\ \forall c \in \mathit{All} : c \in \textsc{string}$

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

$\mathit{RIC} \triangleq \textsc{instance}\ \mathit{RIC}\ \textsc{with}$
$\quad \mathit{Unspecified} \leftarrow$ "Unspecified",
$\quad \mathit{RANFunctionIDInvalid} \leftarrow$ "RANFunctionIDInvalid",
$\quad \mathit{ActionNotSupported} \leftarrow$ "ActionNotSupported",
$\quad \mathit{ExcessiveActions} \leftarrow$ "ExcessiveActions",

$DuplicateAction \leftarrow$ "DuplicateAction",
$DuplicateEvent \leftarrow$ "DuplicateEvent",
$FunctionResourceLimit \leftarrow$ "FunctionResourceLimit",
$RequestIDUnknown \leftarrow$ "RequestIDUnknown",
$InconsistentActionSubsequentActionSequence \leftarrow$ "InconsistentActionSubsequentActionSequence",
$ControlMessageInvalid \leftarrow$ "ControlMessageInvalid",
$CallProcessIDInvalid \leftarrow$ "CallProcessIDInvalid"

MODULE *RICService*

CONSTANTS
$Unspecified$,
$FunctionNotRequired$,
$ExcessiveFunctions$,
$RICResourceLimit$

$All \triangleq$
$\{Unspecified,$
$FunctionNotRequired,$
$ExcessiveFunctions,$
$RICResourceLimit\}$

ASSUME $\forall c \in All : c \in$ 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
$Unspecified$,
$TransportResourceUnavailable$

$All \triangleq$
$\{Unspecified,$
$TransportResourceUnavailable\}$

ASSUME $\forall c \in All : c \in$ STRING

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

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

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

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

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

```
    $RICControlResponse$,
    $RICControlFailure$
CONSTANTS
    $E2ConnectionUpdate$,
    $E2ConnectionUpdateAcknowledge$,
    $E2ConnectionUpdateFailure$
CONSTANTS
    $E2NodeConfigurationUpdate$,
    $E2NodeConfigurationUpdateAcknowledge$,
    $E2NodeConfigurationUpdateFailure$

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

  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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

LOCAL ValidE2ConnectionUpdateAcknowledge(m) ≜
    ∧   ∧ "transactionId" ∈ DOMAIN m
        ∧ m["transactionId"] ∈ Nat

LOCAL ValidE2ConnectionUpdateFailure(m) ≜
    ∧   ∧ "transactionId" ∈ DOMAIN m
        ∧ m["transactionId"] ∈ Nat
    ∧   ∧ "cause" ∈ DOMAIN m
```

```
            $\land m[\text{“cause”}] \in Cause!All$

$\textsc{local}$ $ValidE2NodeConfigurationUpdate(m) \triangleq$
    $\land$    $\land \text{“transactionId”} \in \textsc{domain}\ m$
               $\land m[\text{“transactionId”}] \in Nat$
    $\land$    $\land \text{“globalE2NodeId”} \in \textsc{domain}\ m$
               $\land m[\text{“globalE2NodeId”}] \in Nat$
    $\land$    $\text{“add”} \in \textsc{domain}\ m \Rightarrow IsFiniteSet(m[\text{“add”}])$
    $\land$    $\text{“update”} \in \textsc{domain}\ m \Rightarrow IsFiniteSet(m[\text{“update”}])$
    $\land$    $\text{“remove”} \in \textsc{domain}\ m \Rightarrow IsFiniteSet(m[\text{“remove”}])$

$\textsc{local}$ $ValidE2NodeConfigurationUpdateAcknowledge(m) \triangleq$
    $\land$    $\land \text{“transactionId”} \in \textsc{domain}\ m$
               $\land m[\text{“transactionId”}] \in Nat$
    $\land$    $\text{“add”} \in \textsc{domain}\ m \Rightarrow IsFiniteSet(m[\text{“add”}])$
    $\land$    $\text{“update”} \in \textsc{domain}\ m \Rightarrow IsFiniteSet(m[\text{“update”}])$
    $\land$    $\text{“remove”} \in \textsc{domain}\ m \Rightarrow IsFiniteSet(m[\text{“remove”}])$

$\textsc{local}$ $ValidE2NodeConfigurationUpdateFailure(m) \triangleq$
    $\land$    $\land \text{“transactionId”} \in \textsc{domain}\ m$
               $\land m[\text{“transactionId”}] \in Nat$
    $\land$    $\land \text{“cause”} \in \textsc{domain}\ m$
               $\land m[\text{“cause”}] \in Cause!All$

$\textsc{local}$ $ValidRICSubscriptionRequest(m) \triangleq$
    $\land$    $\land \text{“requestId”} \in \textsc{domain}\ m$
               $\land m[\text{“requestId”}] \in Nat$

$\textsc{local}$ $ValidRICSubscriptionResponse(m) \triangleq$
    $\land$    $\land \text{“requestId”} \in \textsc{domain}\ m$
               $\land m[\text{“requestId”}] \in Nat$

$\textsc{local}$ $ValidRICSubscriptionFailure(m) \triangleq$
    $\land$    $\land \text{“requestId”} \in \textsc{domain}\ m$
               $\land m[\text{“requestId”}] \in Nat$
    $\land$    $\land \text{“cause”} \in \textsc{domain}\ m$
               $\land m[\text{“cause”}] \in Cause!All$

$\textsc{local}$ $ValidRICSubscriptionDeleteRequest(m) \triangleq$
    $\land$    $\land \text{“requestId”} \in \textsc{domain}\ m$
               $\land m[\text{“requestId”}] \in Nat$

$\textsc{local}$ $ValidRICSubscriptionDeleteResponse(m) \triangleq$
    $\land$    $\land \text{“requestId”} \in \textsc{domain}\ m$
               $\land m[\text{“requestId”}] \in Nat$

$\textsc{local}$ $ValidRICSubscriptionDeleteFailure(m) \triangleq$
    $\land$    $\land \text{“requestId”} \in \textsc{domain}\ m$
```

```
          ∧ 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
```

```
$WithRICServiceUpdateAcknowledge(m) \triangleq$
  IF $Assert(ValidRICServiceUpdateAcknowledge(m),$ "Invalid RICServiceUpdateAcknowledge"$)$
   THEN $SetType(m, RICServiceUpdateAcknowledge)$
   ELSE $Nil$

$WithRICServiceUpdateFailure(m, c) \triangleq$
  IF $Assert(ValidRICServiceUpdateFailure(m),$ "Invalid RICServiceUpdateFailure"$)$
   THEN $SetType(m, SetFailureCause(RICServiceUpdateFailure, c))$
   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

WithRICIndication(m) ≜
   IF Assert(ValidRICIndication(m), "Invalid RICIndication")
    THEN SetType(m, RICIndication)
    ELSE  Nil

WithRICControlRequest(m) ≜
   IF Assert(ValidRICControlRequest(m), "Invalid RICControlRequest")
    THEN SetType(m, RICControlRequest)
    ELSE  Nil

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

WithRICControlFailure(m, c) ≜
   IF Assert(ValidRICControlFailure(m), "Invalid RICControlFailure")
    THEN SetType(m, SetFailureCause(RICControlFailure, c))
    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 $Messages \triangleq$ INSTANCE $Messages$ WITH
    $E2SetupRequest \leftarrow$ "E2SetupRequest",
    $E2SetupResponse \leftarrow$ "E2SetupResponse",
    $E2SetupFailure \leftarrow$ "E2SetupFailure",
    $ResetRequest \leftarrow$ "ResetRequest",
    $ResetResponse \leftarrow$ "ResetResponse",
    $RICSubscriptionRequest \leftarrow$ "RICSubscriptionRequest",
    $RICSubscriptionResponse \leftarrow$ "RICSubscriptionResponse",
    $RICSubscriptionFailure \leftarrow$ "RICSubscriptionFailure",
    $RICSubscriptionDeleteRequest \leftarrow$ "RICSubscriptionDeleteRequest",
    $RICSubscriptionDeleteResponse \leftarrow$ "RICSubscriptionDeleteResponse",
    $RICSubscriptionDeleteFailure \leftarrow$ "RICSubscriptionDeleteFailure",
    $RICIndication \leftarrow$ "RICIndication",
    $RICControlRequest \leftarrow$ "RICControlRequest",
    $RICControlResponse \leftarrow$ "RICControlResponse",
    $RICControlFailure \leftarrow$ "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.

MODULE *Send*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

$$\begin{aligned}&RICSubscriptionResponse(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithRICSubscriptionResponse(msg))\end{aligned}$$

$$\begin{aligned}&RICSubscriptionFailure(conn, msg, cause) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithRICSubscriptionFailure(msg, cause))\end{aligned}$$

$$\begin{aligned}&RICSubscriptionDeleteResponse(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithRICSubscriptionDeleteResponse(msg))\end{aligned}$$

$$\begin{aligned}&RICSubscriptionDeleteFailure(conn, msg, cause) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithRICSubscriptionDeleteFailure(msg, cause))\end{aligned}$$

$$\begin{aligned}&RICIndication(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithRICIndication(msg))\end{aligned}$$

$$\begin{aligned}&RICControlAcknowledge(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithRICControlAcknowledge(msg))\end{aligned}$$

$$\begin{aligned}&RICControlFailure(conn, msg, cause) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithRICControlFailure(msg, cause))\end{aligned}$$

$$\begin{aligned}&E2ConnectionUpdate(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithE2ConnectionUpdate(msg))\end{aligned}$$

$$\begin{aligned}&E2ConnectionUpdateAcknowledge(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))\end{aligned}$$

$$\begin{aligned}&E2NodeConfigurationUpdate(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithE2NodeConfigurationUpdate(msg))\end{aligned}$$

$$\begin{aligned}&E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq\\&\quad \land\ SCTP!Client!Reply(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))\end{aligned}$$

Instantiate the *E2AP!Client!Reply* module

$Reply \triangleq \textsc{instance}\ Reply$

MODULE *Receive*

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

$$\begin{aligned}&E2SetupResponse(conn, handler(\_)) \triangleq\\&\quad SCTP!Server!Handle(conn, \textsc{lambda}\ x, m :\\&\qquad \land\ Messages!IsE2SetupResponse(m)\\&\qquad \land\ SCTP!Client!Receive(conn)\\&\qquad \land\ handler(m))\end{aligned}$$

$$\begin{aligned}&RICServiceUpdateAcknowledge(conn, handler(\_)) \triangleq\\&\quad SCTP!Server!Handle(conn, \textsc{lambda}\ x, m :\end{aligned}$$

```
        ∧ Messages!IsRICServiceUpdateAcknowledge(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

RICServiceUpdateFailure(conn, handler(_)) ≜
    SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICServiceUpdateFailure(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

ResetRequest(conn, handler(_)) ≜
    SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsResetRequest(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

ResetResponse(conn, handler(_)) ≜
    SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsResetResponse(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

RICSusbcriptionRequest(conn, handler(_)) ≜
    SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICSubscriptionRequest(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

RICSubscriptionDeleteRequest(conn, handler(_)) ≜
    SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICSubscriptionDeleteRequest(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

RICControlRequest(conn, handler(_)) ≜
    SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICControlRequest(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

E2ConnectionUpdate(conn, handler(_)) ≜
    SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsE2ConnectionUpdate(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

E2ConnectionUpdateAcknowledge(conn, handler(_)) ≜
```

```
        SCTP!Server!Handle(conn, LAMBDA x, m :
            ∧ Messages!IsE2ConnectionUpdateAcknowledge(m)
            ∧ SCTP!Client!Receive(conn)
            ∧ handler(m))

    E2NodeConfigurationUpdate(conn, handler(_)) ≜
        SCTP!Server!Handle(conn, LAMBDA x, m :
            ∧ Messages!IsE2NodeConfigurationUpdate(m)
            ∧ SCTP!Client!Receive(conn)
            ∧ handler(m))

    E2NodeConfigurationUpdateAcknowledge(conn, handler(_)) ≜
        SCTP!Server!Handle(conn, LAMBDA x, m :
            ∧ Messages!IsE2NodeConfigurationUpdateAcknowledge(m)
            ∧ SCTP!Client!Receive(conn)
            ∧ handler(m))
```

Instantiate the *E2AP!Client!*Responses module

```
  Handle ≜ INSTANCE Receive

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

  Disconnect(c) ≜ SCTP!Client!Disconnect(c)
```

Provides operators for the *E2AP* client

```
E2Node ≜ 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.

MODULE *Send*

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

```
    E2SetupResponse(conn, msg) ≜
        ∧ SCTP!Server!Send(conn, Messages!WithE2SetupResponse(msg))

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

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

    ResetRequest(conn, msg) ≜
        ∧ SCTP!Server!Send(conn, Messages!WithResetRequest(msg))
```

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

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

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

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

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

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.

$E2SetupResponse(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithE2SetupResponse(msg))$

$RICServiceUpdateAcknowledge(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithRICServiceUpdateAcknowledge(msg))$

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

$ResetRequest(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithResetRequest(msg))$

$ResetResponse(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithResetResponse(msg))$

$E2ConnectionUpdate(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq$
    $\land\ SCTP!Server!Reply(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

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!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsE2SetupRequest(m)
        ∧ SCTP!Server!Receive(conn)
        ∧ handler(m))

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

  ResetRequest(conn, handler(_)) ≜
     SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsResetRequest(m)
        ∧ SCTP!Server!Receive(conn)
        ∧ handler(m))

  ResetResponse(conn, handler(_)) ≜
     SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsResetResponse(m)
        ∧ SCTP!Server!Receive(conn)
        ∧ handler(m))

  RICSubscriptionResponse(conn, handler(_)) ≜
     SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICSubscriptionResponse(m)
        ∧ SCTP!Server!Receive(conn)
        ∧ handler(m))

  RICSubscriptionDeleteResponse(conn, handler(_)) ≜
     SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICSubscriptionDeleteResponse(m)
        ∧ SCTP!Server!Receive(conn)
        ∧ handler(m))

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

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

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

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

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

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

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

Provides operators for the $E2AP$ server
$RIC \triangleq$ INSTANCE $RIC$

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

$Init \triangleq SCTP!Init$

$Next \triangleq SCTP!Next$
```

\* Modification History
\* Last modified *Tue Sep* 21 00:26:48 *PDT* 2021 by *jordanhalterman*
\* Created *Mon Sep* 13 10:53:17 *PDT* 2021 by *jordanhalterman*