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
    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 $\mathit{ValidE2SetupRequest}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2SetupResponse}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2SetupFailure}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICServiceUpdate}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICServiceUpdateAcknowledge}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICServiceUpdateFailure}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidResetRequest}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidResetResponse}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICSubscriptionRequest}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICSubscriptionResponse}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICSubscriptionFailure}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICSubscriptionDeleteRequest}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICSubscriptionDeleteResponse}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICSubscriptionDeleteFailure}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICIndication}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICControlRequest}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICControlResponse}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidRICControlFailure}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2ConnectionUpdate}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2ConnectionUpdateAcknowledge}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2ConnectionUpdateFailure}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2NodeConfigurationUpdate}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2NodeConfigurationUpdateAcknowledge}(m) \triangleq$ TRUE

LOCAL $\mathit{ValidE2NodeConfigurationUpdateFailure}(m) \triangleq$ TRUE

This section defines operators for constructing $E2AP$ messages.

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

```
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) ≜
```

```
  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 $Nil$

$WithRICControlResponse(m) \triangleq$
  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 $Nil$

$WithE2ConnectionUpdate(m) \triangleq$
  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 ← "RICServiceUpdate",
  RICServiceUpdateAcknowledge ← "RICServiceUpdateAcknowledge",
RICServiceUpdateFailure ← "RICServiceUpdateFailure",
E2ConnectionUpdate ← "E2ConnectionUpdate",
E2ConnectionUpdateAcknowledge ← "E2ConnectionUpdateAcknowledge",
E2ConnectionUpdateFailure ← "E2ConnectionUpdateFailure",
E2NodeConfigurationUpdate ← "E2NodeConfigurationUpdate",
E2NodeConfigurationUpdateAcknowledge ← "E2NodeConfigurationUpdateAcknowledge",
E2NodeConfigurationUpdateFailure ← "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 ≜
    {Unspecified,
     ControlProcessingOverload,
     HardwareFailure,
     OMIntervention}

ASSUME ∀ c ∈ failureCauses : 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 ← "ControlProcessingOverload",
    HardwareFailure ← "HardwareFailure",
    OMIntervention ← "OMIntervention"
```

─────────────── MODULE *Protocol* ───────────────

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

LOCAL failureCauses ≜
    {Unspecified,
     TransferSyntaxError,
     AbstractSyntaxErrorReject,
     AbstractSyntaxErrorIgnoreAndNotify,
     MessageNotCompatibleWithReceiverState,
     SemanticError,
     AbstractSyntaxErrorFalselyConstructedMessage}

ASSUME ∀ c ∈ failureCauses : c ∈ STRING

IsUnspecified(m) ≜ m.cause = Unspecified
IsTransferSyntaxError(m) ≜ m.cause = TransferSyntaxError
IsAbstractSyntaxErrorReject(m) ≜ m.cause = AbstractSyntaxErrorReject
IsAbstractSyntaxErrorIgnoreAndNotify(m) ≜ m.cause = AbstractSyntaxErrorIgnoreAndNotify
IsMessageNotCompatibleWithReceiverState(m) ≜ m.cause = MessageNotCompatibleWithReceiverState
IsSemanticError(m) ≜ m.cause = SemanticError
IsAbstractSyntaxErrorFalselyConstructedMessage(m) ≜ m.cause = AbstractSyntaxErrorFalselyConstru
└──────────────────────────────────────────────────────────────────────┘

Protocol ≜ INSTANCE Protocol WITH
    Unspecified ← "Unspecified",
    TransferSyntaxError ← "TransferSyntaxError",
    AbstractSyntaxErrorReject ← "AbstractSyntaxErrorReject",
    AbstractSyntaxErrorIgnoreAndNotify ← "AbstractSyntaxErrorIgnoreAndNotify",
    MessageNotCompatibleWithReceiverState ← "MessageNotCompatibleWithReceiverState",
    SemanticError ← "SemanticError",
    AbstractSyntaxErrorFalselyConstructedMessage ← "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 ← "RequestIDUnknown",
  InconsistentActionSubsequentActionSequence ← "InconsistentActionSubsequentActionSequence",
  ControlMessageInvalid ← "ControlMessageInvalid",
  CallProcessIDInvalid ← "CallProcessIDInvalid"

  ──────────────── MODULE RICService ────────────────

  CONSTANTS
```

```
    Unspecified,
    FunctionNotRequired,
    ExcessiveFunctions,
    RICResourceLimit

LOCAL failureCauses ≜
    {Unspecified,
     FunctionNotRequired,
     ExcessiveFunctions,
     RICResourceLimit}

ASSUME ∀ c ∈ failureCauses : 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

LOCAL failureCauses ≜
    {Unspecified,
     TransportResourceUnavailable}

ASSUME ∀ c ∈ failureCauses : c ∈ STRING

IsUnspecified(m) ≜ m.cause = Unspecified
IsTransportResourceUnavailable(m) ≜ m.cause = TransportResourceUnavailable
```

```
Transport ≜ INSTANCE Transport WITH
    Unspecified ← "Unspecified",
    TransportResourceUnavailable ← "TransportResourceUnavailable"
```

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

The *Cause* module provides failure causes

$Cause \triangleq$ INSTANCE $Cause$

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) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithE2SetupResponse(msg))

RICServiceUpdate(conn, msg) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICServiceUpdate(msg))

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

ResetResponse(conn, msg) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithResetResponse(msg))

RICSubscriptionResponse(conn, msg) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICSubscriptionResponse(msg))

RICSubscriptionFailure(conn, msg, cause) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICSubscriptionFailure(msg, cause))

RICSubscriptionDeleteResponse(conn, msg) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICSubscriptionDeleteResponse(msg))

RICSubscriptionDeleteFailure(conn, msg, cause) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICSubscriptionDeleteFailure(msg, cause))

RICIndication(conn, msg) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICIndication(msg))

RICControlResponse(conn, msg) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICControlResponse(msg))

RICControlFailure(conn, msg, cause) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithRICControlFailure(msg, cause))

E2ConnectionUpdate(conn, msg) ≜
    ∧ SCTP!Client!Send(conn, Messages!WithE2ConnectionUpdate(msg))

E2ConnectionUpdateAcknowledge(conn, msg) ≜
```

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

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

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

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.

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

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

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

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

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

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

$RICControlResponse(conn,\ msg)\ \triangleq$
$\quad \land\ SCTP!Client!Reply(conn,\ Messages!WithRICControlResponse(msg))$

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

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

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

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

```
  E2NodeConfigurationUpdateAcknowledge(conn, msg) ≜
     ∧ SCTP!Client!Reply(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))
```

Instantiate the *E2AP*!*Client*!*Reply* module

```
Reply ≜ INSTANCE Reply
```

MODULE *Receive*

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

```
  E2SetupResponse(conn, handler(_)) ≜
     SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsE2SetupResponse(m)
        ∧ SCTP!Client!Receive(conn)
        ∧ handler(m))

  RICServiceUpdateAcknowledge(conn, handler(_)) ≜
     SCTP!Server!Handle(conn, LAMBDA x, m :
        ∧ 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))
```

$RICSubscriptionDeleteRequest(conn, handler(\_)) \triangleq$
$\quad SCTP!Server!Handle(conn, \text{LAMBDA } x, m :$
$\qquad \land Messages!IsRICSubscriptionDeleteRequest(m)$
$\qquad \land SCTP!Client!Receive(conn)$
$\qquad \land handler(m))$

$RICControlRequest(conn, handler(\_)) \triangleq$
$\quad SCTP!Server!Handle(conn, \text{LAMBDA } x, m :$
$\qquad \land Messages!IsRICControlRequest(m)$
$\qquad \land SCTP!Client!Receive(conn)$
$\qquad \land handler(m))$

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

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

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

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

Instantiate the *E2AP!Client!Responses* module

$Handle \triangleq \text{INSTANCE } Receive$

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

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

Provides operators for the *E2AP* client

$E2Node \triangleq$ 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.

MODULE $Send$

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

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

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

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

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

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

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

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

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

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

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.

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

$RICServiceUpdateAcknowledge(conn, msg) \triangleq$

```
    ∧ SCTP!Server!Reply(conn, Messages!WithRICServiceUpdateAcknowledge(msg))

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

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

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

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

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

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

E2NodeConfigurationUpdateAcknowledge(conn, msg) ≜
    ∧ 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))
```

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

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

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

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

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

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

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

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

```
$E2NodeConfigurationUpdateAcknowledge(conn, handler(\_)) \triangleq$
    $SCTP!Server!Handle(conn,$ 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 *Mon Sep* 13 19:04:07 *PDT* 2021 by *jordanhalterman*
\* *Created Mon Sep* 13 10:53:17 *PDT* 2021 by *jordanhalterman*