
MODULE *Protocols*

LOCAL INSTANCE *Naturals*

LOCAL INSTANCE *Sequences*

LOCAL INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

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.

CONSTANT *Nil*

VARIABLE *servers*, *conns*

vars \triangleq $\langle \textit{servers}, \textit{conns} \rangle$

The *E2AP* protocol is implemented on *SCTP*

LOCAL *SCTP* \triangleq INSTANCE *SCTP*

Message type constants

CONSTANTS

E2SetupRequestType,
E2SetupResponseType,
E2SetupFailureType

CONSTANTS

ResetRequestType,
ResetResponseType

CONSTANTS

RICSubscriptionRequestType,
RICSubscriptionResponseType,
RICSubscriptionFailureType

CONSTANTS

RICSubscriptionDeleteRequestType,
RICSubscriptionDeleteResponseType,
RICSubscriptionDeleteFailureType

CONSTANTS

RICControlRequestType,
RICControlResponseType,
RICControlFailureType,
RICServiceUpdateType

CONSTANTS

E2ConnectionUpdateType,
E2ConnectionUpdateAcknowledgeType,

E2ConnectionUpdateFailureType

CONSTANTS

E2NodeConfigurationUpdateType,
E2NodeConfigurationUpdateAcknowledgeType,
E2NodeConfigurationUpdateFailureType

LOCAL *messageTypes* \triangleq

{*E2SetupRequestType*,
E2SetupResponseType,
E2SetupFailureType,
ResetRequestType,
ResetResponseType,
RICSubscriptionRequestType,
RICSubscriptionResponseType,
RICSubscriptionFailureType,
RICSubscriptionDeleteRequestType,
RICSubscriptionDeleteResponseType,
RICSubscriptionDeleteFailureType,
RICControlRequestType,
RICControlResponseType,
RICControlFailureType,
RICServiceUpdateType,
E2ConnectionUpdateType,
E2ConnectionUpdateAcknowledgeType,
E2ConnectionUpdateFailureType,
E2NodeConfigurationUpdateType,
E2NodeConfigurationUpdateAcknowledgeType,
E2NodeConfigurationUpdateFailureType}

Message types should be defined as strings to simplify debugging

ASSUME $\forall m \in \text{messageTypes} : m \in \text{STRING}$

Failure cause constants

CONSTANTS

MiscFailureUnspecified,
MiscFailureControlProcessingOverload,
MiscFailureHardwareFailure,
MiscFailureOMIntervention

CONSTANTS

ProtocolFailureUnspecified,
ProtocolFailureTransferSyntaxError,
ProtocolFailureAbstractSyntaxErrorReject,
ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify,
ProtocolFailureMessageNotCompatibleWithReceiverState,
ProtocolFailureSemanticError,
ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage

CONSTANTS

RICFailureUnspecified,
RICFailureRANFunctionIDInvalid,
RICFailureActionNotSupported,
RICFailureExcessiveActions,
RICFailureDuplicateAction,
RICFailureDuplicateEvent,
RICFailureFunctionResourceLimit,
RICFailureRequestIDUnknown,
RICFailureInconsistentActionSubsequentActionSequence,
RICFailureControlMessageInvalid,
RICFailureCallProcessIDInvalid

CONSTANTS

RICServiceFailureUnspecified,
RICServiceFailureFunctionNotRequired,
RICServiceFailureExcessiveFunctions,
RICServiceFailureRICResourceLimit

CONSTANTS

TransportFailureUnspecified,
TransportFailureTransportResourceUnavailable

LOCAL *failureCauses* \triangleq

{ *MiscFailureUnspecified,*
MiscFailureControlProcessingOverload,
MiscFailureHardwareFailure,
MiscFailureOMIntervention,
ProtocolFailureUnspecified,
ProtocolFailureTransferSyntaxError,
ProtocolFailureAbstractSyntaxErrorReject,
ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify,
ProtocolFailureMessageNotCompatibleWithReceiverState,
ProtocolFailureSemanticError,
ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage,
RICFailureUnspecified,
RICFailureRANFunctionIDInvalid,
RICFailureActionNotSupported,
RICFailureExcessiveActions,
RICFailureDuplicateAction,
RICFailureDuplicateEvent,
RICFailureFunctionResourceLimit,
RICFailureRequestIDUnknown,
RICFailureInconsistentActionSubsequentActionSequence,
RICFailureControlMessageInvalid,
RICFailureCallProcessIDInvalid,
RICServiceFailureUnspecified,

$RICServiceFailureFunctionNotRequired,$
 $RICServiceFailureExcessiveFunctions,$
 $RICServiceFailureRICResourceLimit,$
 $TransportFailureUnspecified,$
 $TransportFailureTransportResourceUnavailable\}$

Failure causes should be defined as strings to simplify debugging
 ASSUME $\forall c \in failureCauses : c \in \text{STRING}$

┌────────────────── MODULE *Messages* ───────────────────┐
 The *Messages* module defines predicates for receiving, sending, and verifying all the messages
 supported by *E2AP*.
 └──────────────────┘

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

$IsE2SetupRequest(m) \triangleq m.type = E2SetupRequestType$
 $IsE2SetupResponse(m) \triangleq m.type = E2SetupResponseType$
 $IsE2SetupFailure(m) \triangleq m.type = E2SetupFailureType$
 $IsResetRequest(m) \triangleq m.type = ResetRequestType$
 $IsResetResponse(m) \triangleq m.type = ResetResponseType$
 $IsRICSubscriptionRequest(m) \triangleq m.type = RICSubscriptionRequestType$
 $IsRICSubscriptionResponse(m) \triangleq m.type = RICSubscriptionResponseType$
 $IsRICSubscriptionFailure(m) \triangleq m.type = RICSubscriptionFailureType$
 $IsRICSubscriptionDeleteRequest(m) \triangleq m.type = RICSubscriptionDeleteRequestType$
 $IsRICSubscriptionDeleteResponse(m) \triangleq m.type = RICSubscriptionDeleteResponseType$
 $IsRICSubscriptionDeleteFailure(m) \triangleq m.type = RICSubscriptionDeleteFailureType$
 $IsRICControlRequest(m) \triangleq m.type = RICControlRequestType$
 $IsRICControlResponse(m) \triangleq m.type = RICControlResponseType$
 $IsRICControlFailure(m) \triangleq m.type = RICControlFailureType$
 $IsRICServiceUpdate(m) \triangleq m.type = RICServiceUpdateType$
 $IsE2ConnectionUpdate(m) \triangleq m.type = E2ConnectionUpdateType$
 $IsE2ConnectionUpdateAcknowledge(m) \triangleq m.type = E2ConnectionUpdateAcknowledgeType$
 $IsE2ConnectionUpdateFailure(m) \triangleq m.type = E2ConnectionUpdateFailureType$

$IsE2NodeConfigurationUpdate(m) \triangleq m.type = E2NodeConfigurationUpdateType$

$IsE2NodeConfigurationUpdateAcknowledge(m) \triangleq m.type = E2NodeConfigurationUpdateAcknowledgeType$

$IsE2NodeConfigurationUpdateFailure(m) \triangleq m.type = E2NodeConfigurationUpdateFailureType$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

This section defines operators for constructing *E2AP* messages.

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

$E2SetupRequest(m) \triangleq$
 IF $Assert(ValidE2SetupRequest(m), \text{"Invalid E2SetupRequest"})$
 THEN $SetType(m, E2SetupRequestType)$
 ELSE Nil

$E2SetupResponse(m) \triangleq$
 IF $Assert(ValidE2SetupResponse(m), \text{"Invalid E2SetupResponse"})$
 THEN $SetType(m, E2SetupResponseType)$
 ELSE Nil

$E2SetupFailure(m) \triangleq$
 IF $Assert(ValidE2SetupFailure(m), \text{"Invalid E2SetupFailure"})$
 THEN $SetType(m, E2SetupFailureType)$
 ELSE Nil

$ResetRequest(m) \triangleq$
 IF $Assert(ValidResetRequest(m), \text{"Invalid ResetRequest"})$
 THEN $SetType(m, ResetRequestType)$
 ELSE Nil

$ResetResponse(m) \triangleq$
 IF $Assert(ValidResetResponse(m), \text{"Invalid ResetResponse"})$
 THEN $SetType(m, ResetResponseType)$
 ELSE Nil

$RICSubscriptionRequest(m) \triangleq$
 IF $Assert(ValidRICSubscriptionRequest(m), \text{"Invalid RICSubscriptionRequest"})$
 THEN $SetType(m, RICSubscriptionRequestType)$
 ELSE Nil

$RICSubscriptionResponse(m) \triangleq$
 IF $Assert(ValidRICSubscriptionResponse(m), \text{"Invalid RICSubscriptionResponse"})$
 THEN $SetType(m, RICSubscriptionResponseType)$
 ELSE Nil

$RICSubscriptionFailure(m) \triangleq$
 IF $Assert(ValidRICSubscriptionFailure(m), \text{"Invalid RICSubscriptionFailure"})$
 THEN $SetType(m, RICSubscriptionFailureType)$
 ELSE Nil

$RICSubscriptionDeleteRequest(m) \triangleq$
 IF $Assert(ValidRICSubscriptionDeleteRequest(m), \text{"Invalid RICSubscriptionDeleteRequest"})$
 THEN $SetType(m, RICSubscriptionDeleteRequestType)$

```

ELSE Nil

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

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

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

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

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

RICServiceUpdate(m)  $\triangleq$ 
  IF Assert(ValidRICServiceUpdate(m), "Invalid RICServiceUpdate")
  THEN SetType(m, RICServiceUpdateType)
  ELSE Nil

E2ConnectionUpdate(m)  $\triangleq$ 
  IF Assert(ValidE2ConnectionUpdate(m), "Invalid E2ConnectionUpdate")
  THEN SetType(m, E2ConnectionUpdateType)
  ELSE Nil

E2ConnectionUpdateAcknowledge(m)  $\triangleq$ 
  IF Assert(ValidE2ConnectionUpdateAcknowledge(m), "Invalid E2ConnectionUpdateAcknowledge")
  THEN SetType(m, E2ConnectionUpdateAcknowledgeType)
  ELSE Nil

E2ConnectionUpdateFailure(m)  $\triangleq$ 
  IF Assert(ValidE2ConnectionUpdateFailure(m), "Invalid E2ConnectionUpdateFailure")
  THEN SetType(m, E2ConnectionUpdateFailureType)
  ELSE Nil

E2NodeConfigurationUpdate(m)  $\triangleq$ 

```

```

IF Assert(ValidE2NodeConfigurationUpdate(m), "Invalid E2NodeConfigurationUpdate")
THEN SetType(m, E2NodeConfigurationUpdateType)
ELSE Nil

```

```

E2NodeConfigurationUpdateAcknowledge(m)  $\triangleq$ 
IF Assert(ValidE2NodeConfigurationUpdateAcknowledge(m), "Invalid E2NodeConfigurationUpdateAcknowledge")
THEN SetType(m, E2NodeConfigurationUpdateAcknowledgeType)
ELSE Nil

```

```

E2NodeConfigurationUpdateFailure(m)  $\triangleq$ 
IF Assert(ValidE2NodeConfigurationUpdateFailure(m), "Invalid E2NodeConfigurationUpdateFailure")
THEN SetType(m, E2NodeConfigurationUpdateFailureType)
ELSE Nil

```

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

```

LOCAL Messages  $\triangleq$  INSTANCE Messages

```

MODULE *Client*

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

MODULE *Send*

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

```

E2SetupRequest(c, m)  $\triangleq$ 
   $\wedge$  SCTP!Client!Send(c, Messages!E2SetupResponse(m))

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

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

```

Instantiate the *E2AP!Client!Send* module

```

Send  $\triangleq$  INSTANCE Send

```

MODULE *Receive*

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

```

E2SetupResponse(c, h(-, -))  $\triangleq$ 
  SCTP!Server!Handle(c, LAMBDA x, m :
     $\wedge$  Messages!IsE2SetupResponse(m)

```


$$\begin{aligned}
& \wedge SCTP!Client!Receive(c) \\
& \wedge h(c, m)) \\
\\
ResetRequest(c, h(-, -)) & \triangleq \\
& SCTP!Server!Handle(c, LAMBDA x, m : \\
& \quad \wedge Messages!IsResetRequest(m) \\
& \quad \wedge SCTP!Client!Receive(c) \\
& \quad \wedge h(c, m)) \\
\\
ResetResponse(c, h(-, -)) & \triangleq \\
& SCTP!Server!Handle(c, LAMBDA x, m : \\
& \quad \wedge Messages!IsResetResponse(m) \\
& \quad \wedge SCTP!Client!Receive(c) \\
& \quad \wedge h(c, m))
\end{aligned}$$

Instantiate the *E2AP!Client!Receive* module

$$Receive \triangleq \text{INSTANCE } Receive$$

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

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

Provides operators for the *E2AP* client

$$Client \triangleq \text{INSTANCE } Client$$

MODULE *Server*

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

MODULE *Send*

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

$$\begin{aligned}
E2SetupResponse(c, m) & \triangleq \\
& \wedge SCTP!Server!Reply(c, Messages!E2SetupResponse(m)) \\
\\
ResetRequest(c, m) & \triangleq \\
& \wedge SCTP!Server!Send(c, Messages!ResetRequest(m)) \\
\\
ResetResponse(c, m) & \triangleq \\
& \wedge SCTP!Server!Reply(c, Messages!ResetResponse(m))
\end{aligned}$$

Instantiate the *E2AP!Server!Send* module

$$Send \triangleq \text{INSTANCE } Send$$

MODULE *Receive*

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

$$\begin{aligned} E2SetupRequest(c, h(-, -)) &\triangleq \\ SCTP!Server!Handle(c, \text{LAMBDA } x, m : & \\ \quad \wedge Messages!IsE2SetupRequest(m) & \\ \quad \wedge SCTP!Server!Receive(c) & \\ \quad \wedge h(c, m)) & \end{aligned}$$

$$\begin{aligned} ResetRequest(c, h(-, -)) &\triangleq \\ SCTP!Server!Handle(c, \text{LAMBDA } x, m : & \\ \quad \wedge Messages!IsResetRequest(m) & \\ \quad \wedge SCTP!Server!Receive(c) & \\ \quad \wedge h(c, m)) & \end{aligned}$$

$$\begin{aligned} ResetResponse(c, h(-, -)) &\triangleq \\ SCTP!Server!Handle(c, \text{LAMBDA } x, m : & \\ \quad \wedge Messages!IsResetResponse(m) & \\ \quad \wedge SCTP!Server!Receive(c) & \\ \quad \wedge h(c, m)) & \end{aligned}$$

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

Starts a new *E2AP* server
 $Serve(s) \triangleq SCTP!Server!Start(s)$

Stops the given *E2AP* server
 $Stop(s) \triangleq SCTP!Server!Stop(s)$

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

The set of all running *E2AP* servers
 $Servers \triangleq SCTP!Servers$

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

$Init \triangleq SCTP!Init$

$Next \triangleq SCTP!Next$

VARIABLES $e2apServers$, $e2apConns$

$E2AP \triangleq$ INSTANCE $E2AP$ WITH

$servers \leftarrow e2apServers$,
 $conns \leftarrow e2apConns$,
 $Nil \leftarrow [type \mapsto ""]$,
 $E2SetupRequestType \leftarrow \text{"E2SetupRequest"}$,
 $E2SetupResponseType \leftarrow \text{"E2SetupResponse"}$,
 $E2SetupFailureType \leftarrow \text{"E2SetupFailure"}$,
 $ResetRequestType \leftarrow \text{"ResetRequest"}$,
 $ResetResponseType \leftarrow \text{"ResetResponse"}$,
 $RICSubscriptionRequestType \leftarrow \text{"RICSubscriptionRequest"}$,
 $RICSubscriptionResponseType \leftarrow \text{"RICSubscriptionResponse"}$,
 $RICSubscriptionFailureType \leftarrow \text{"RICSubscriptionFailure"}$,
 $RICSubscriptionDeleteRequestType \leftarrow \text{"RICSubscriptionDeleteRequest"}$,
 $RICSubscriptionDeleteResponseType \leftarrow \text{"RICSubscriptionDeleteResponse"}$,
 $RICSubscriptionDeleteFailureType \leftarrow \text{"RICSubscriptionDeleteFailure"}$,
 $RICControlRequestType \leftarrow \text{"RICControlRequest"}$,
 $RICControlResponseType \leftarrow \text{"RICControlResponse"}$,
 $RICControlFailureType \leftarrow \text{"RICControlFailure"}$,
 $RICServiceUpdateType \leftarrow \text{"RICServiceUpdate"}$,
 $E2ConnectionUpdateType \leftarrow \text{"E2ConnectionUpdate"}$,
 $E2ConnectionUpdateAcknowledgeType \leftarrow \text{"E2ConnectionUpdateAcknowledge"}$,
 $E2ConnectionUpdateFailureType \leftarrow \text{"E2ConnectionUpdateFailure"}$,
 $E2NodeConfigurationUpdateType \leftarrow \text{"E2NodeConfigurationUpdate"}$,
 $E2NodeConfigurationUpdateAcknowledgeType \leftarrow \text{"E2NodeConfigurationUpdateAcknowledge"}$,
 $E2NodeConfigurationUpdateFailureType \leftarrow \text{"E2NodeConfigurationUpdateFailure"}$,
 $MiscFailureUnspecified \leftarrow \text{"MiscFailureUnspecified"}$,
 $MiscFailureControlProcessingOverload \leftarrow \text{"MiscFailureControlProcessingOverload"}$,
 $MiscFailureHardwareFailure \leftarrow \text{"MiscFailureHardwareFailure"}$,
 $MiscFailureOMIntervention \leftarrow \text{"MiscFailureOMIntervention"}$,
 $ProtocolFailureUnspecified \leftarrow \text{"ProtocolFailureUnspecified"}$,
 $ProtocolFailureTransferSyntaxError \leftarrow \text{"ProtocolFailureTransferSyntaxError"}$,
 $ProtocolFailureAbstractSyntaxErrorReject \leftarrow \text{"ProtocolFailureAbstractSyntaxErrorReject"}$,
 $ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify \leftarrow \text{"ProtocolFailureAbstractSyntaxErrorIgnoreAndNotify"}$,
 $ProtocolFailureMessageNotCompatibleWithReceiverState \leftarrow \text{"ProtocolFailureMessageNotCompatibleWithReceiverState"}$,
 $ProtocolFailureSemanticError \leftarrow \text{"ProtocolFailureSemanticError"}$,
 $ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage \leftarrow \text{"ProtocolFailureAbstractSyntaxErrorFalselyConstructedMessage"}$,
 $RICFailureUnspecified \leftarrow \text{"RICFailureUnspecified"}$,
 $RICFailureRANFunctionIDInvalid \leftarrow \text{"RICFailureRANFunctionIDInvalid"}$,
 $RICFailureActionNotSupported \leftarrow \text{"RICFailureActionNotSupported"}$,
 $RICFailureExcessiveActions \leftarrow \text{"RICFailureExcessiveActions"}$,
 $RICFailureDuplicateAction \leftarrow \text{"RICFailureDuplicateAction"}$,
 $RICFailureDuplicateEvent \leftarrow \text{"RICFailureDuplicateEvent"}$,
 $RICFailureFunctionResourceLimit \leftarrow \text{"RICFailureFunctionResourceLimit"}$,

$RICFailureRequestIDUnknown \leftarrow \text{"RICFailureRequestIDUnknown"},$
 $RICFailureInconsistentActionSubsequentActionSequence \leftarrow \text{"RICFailureInconsistentActionSubsequentActionSequence"},$
 $RICFailureControlMessageInvalid \leftarrow \text{"RICFailureControlMessageInvalid"},$
 $RICFailureCallProcessIDInvalid \leftarrow \text{"RICFailureCallProcessIDInvalid"},$
 $RICServiceFailureUnspecified \leftarrow \text{"RICServiceFailureUnspecified"},$
 $RICServiceFailureFunctionNotRequired \leftarrow \text{"RICServiceFailureFunctionNotRequired"},$
 $RICServiceFailureExcessiveFunctions \leftarrow \text{"RICServiceFailureExcessiveFunctions"},$
 $RICServiceFailureRICResourceLimit \leftarrow \text{"RICServiceFailureRICResourceLimit"},$
 $TransportFailureUnspecified \leftarrow \text{"TransportFailureUnspecified"},$
 $TransportFailureTransportResourceUnavailable \leftarrow \text{"TransportFailureTransportResourceUnavailable"}$

MODULE *E2TService*

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

CONSTANT *Nil*

VARIABLE *servers, conns*

$\text{vars} \triangleq \langle \text{servers}, \text{conns} \rangle$

The *E2T API* is implemented as a *gRPC* service

LOCAL *GRPC* \triangleq INSTANCE *GRPC*

Message type constants

CONSTANT

SubscribeRequestType,
SubscribeResponseType

CONSTANTS

UnsubscribeRequestType,
UnsubscribeResponseType

CONSTANTS

ControlRequestType,
ControlResponseType

LOCAL *messageTypes* \triangleq

{*SubscribeRequestType,*
SubscribeResponseType,
UnsubscribeRequestType,
UnsubscribeResponseType,
ControlRequestType,
ControlResponseType}

Message types should be defined as strings to simplify debugging

ASSUME $\forall m \in \text{messageTypes} : m \in \text{STRING}$

MODULE *Messages*

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

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

$$\begin{aligned} \text{IsSubscribeRequest}(m) &\triangleq m.type = \text{SubscribeRequestType} \\ \text{IsSubscribeResponse}(m) &\triangleq m.type = \text{SubscribeResponseType} \\ \text{IsUnsubscribeRequest}(m) &\triangleq m.type = \text{UnsubscribeRequestType} \\ \text{IsUnsubscribeResponse}(m) &\triangleq m.type = \text{UnsubscribeResponseType} \\ \text{IsControlRequest}(m) &\triangleq m.type = \text{ControlRequestType} \\ \text{IsControlResponse}(m) &\triangleq m.type = \text{ControlResponseType} \end{aligned}$$

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

$$\begin{aligned} \text{LOCAL ValidSubscribeRequest}(m) &\triangleq \text{TRUE} \\ \text{LOCAL ValidSubscribeResponse}(m) &\triangleq \text{TRUE} \\ \text{LOCAL ValidUnsubscribeRequest}(m) &\triangleq \text{TRUE} \\ \text{LOCAL ValidUnsubscribeResponse}(m) &\triangleq \text{TRUE} \\ \text{LOCAL ValidControlRequest}(m) &\triangleq \text{TRUE} \\ \text{LOCAL ValidControlResponse}(m) &\triangleq \text{TRUE} \end{aligned}$$

This section defines operators for constructing *E2T* messages.

$$\begin{aligned} \text{LOCAL SetType}(m, t) &\triangleq [m \text{ EXCEPT } !.type = t] \\ \text{SubscribeRequest}(m) &\triangleq \\ &\quad \text{IF Assert(ValidSubscribeRequest}(m), \text{"Invalid SubscribeRequest"})} \\ &\quad \text{THEN SetType}(m, \text{SubscribeRequestType}) \\ &\quad \text{ELSE Nil} \\ \text{SubscribeResponse}(m) &\triangleq \\ &\quad \text{IF Assert(ValidSubscribeResponse}(m), \text{"Invalid SubscribeResponse"})} \\ &\quad \text{THEN SetType}(m, \text{SubscribeResponseType}) \\ &\quad \text{ELSE Nil} \\ \text{UnsubscribeRequest}(m) &\triangleq \\ &\quad \text{IF Assert(ValidUnsubscribeRequest}(m), \text{"Invalid UnsubscribeRequest"})} \end{aligned}$$

```

    THEN SetType(m, UnsubscribeRequestType)
    ELSE Nil

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

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

ControlResponse(m)  $\triangleq$ 
  IF Assert(ValidControlResponse(m), "Invalid ControlResponse")
  THEN SetType(m, ControlResponseType)
  ELSE Nil

```

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

```
LOCAL Messages  $\triangleq$  INSTANCE Messages
```

```

    ─────────────────── MODULE Client ───────────────────

```

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

```

    ─────────────────── MODULE Send ───────────────────

```

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

```

    SubscribeRequest(c, m)  $\triangleq$ 
       $\wedge$  GRPC! Client! Send(c, Messages! SubscribeRequest(m))

    UnsubscribeRequest(c, m)  $\triangleq$ 
       $\wedge$  GRPC! Client! Send(c, Messages! UnsubscribeRequest(m))

    ControlRequest(c, m)  $\triangleq$ 
       $\wedge$  GRPC! Client! Send(c, Messages! ControlRequest(m))

```

Instantiate the *E2T*! *Client*! *Send* module

```
Send  $\triangleq$  INSTANCE Send
```

```

    ─────────────────── MODULE Receive ───────────────────

```

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

$$\begin{aligned} \text{SubscribeResponse}(c, h(-, -)) &\triangleq \\ &\text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\ &\quad \wedge \text{Messages!IsSubscribeResponse}(m) \\ &\quad \wedge \text{GRPC!Client!Receive}(c) \\ &\quad \wedge h(c, m)) \end{aligned}$$

$$\begin{aligned} \text{UnsubscribeResponse}(c, h(-, -)) &\triangleq \\ &\text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\ &\quad \wedge \text{Messages!IsUnsubscribeResponse}(m) \\ &\quad \wedge \text{GRPC!Client!Receive}(c) \\ &\quad \wedge h(c, m)) \end{aligned}$$

$$\begin{aligned} \text{ControlResponse}(c, h(-, -)) &\triangleq \\ &\text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\ &\quad \wedge \text{Messages!IsControlResponse}(m) \\ &\quad \wedge \text{GRPC!Client!Receive}(c) \\ &\quad \wedge h(c, m)) \end{aligned}$$

Instantiate the *E2T!Client!Receive* module
 $\text{Receive} \triangleq \text{INSTANCE } \text{Receive}$

$$\text{Connect}(s, d) \triangleq \text{GRPC!Client!Connect}(s, d)$$

$$\text{Disconnect}(c) \triangleq \text{GRPC!Client!Disconnect}(c)$$

Provides operators for the *E2T* client
 $\text{Client} \triangleq \text{INSTANCE } \text{Client}$

MODULE *Server*

The *Server* module provides operators for managing and operating on *E2T* servers and specifies the message types supported for the server.

MODULE *Send*

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

$$\begin{aligned} \text{SubscribeResponse}(c, m) &\triangleq \\ &\quad \wedge \text{GRPC!Server!Reply}(c, \text{Messages!SubscribeResponse}(m)) \end{aligned}$$

$$\begin{aligned} \text{UnsubscribeResponse}(c, m) &\triangleq \\ &\quad \wedge \text{GRPC!Server!Reply}(c, \text{Messages!UnsubscribeResponse}(m)) \end{aligned}$$

$$\begin{aligned} \text{ControlResponse}(c, m) &\triangleq \\ &\quad \wedge \text{GRPC!Server!Reply}(c, \text{Messages!ControlResponse}(m)) \end{aligned}$$

Instantiate the $E2T!Server!Send$ module
 $Send \triangleq \text{INSTANCE } Send$

MODULE *Receive*

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

$SubscribeRequest(c, h(-, -)) \triangleq$
 $GRPC!Server!Handle(c, \text{LAMBDA } x, m :$
 $\quad \wedge Messages!IsSubscribeRequest(m)$
 $\quad \wedge GRPC!Server!Receive(c)$
 $\quad \wedge h(c, m))$

$UnsubscribeRequest(c, h(-, -)) \triangleq$
 $GRPC!Server!Handle(c, \text{LAMBDA } x, m :$
 $\quad \wedge Messages!IsUnsubscribeRequest(m)$
 $\quad \wedge GRPC!Server!Receive(c)$
 $\quad \wedge h(c, m))$

$ControlRequest(c, h(-, -)) \triangleq$
 $GRPC!Server!Handle(c, \text{LAMBDA } x, m :$
 $\quad \wedge Messages!IsControlRequest(m)$
 $\quad \wedge GRPC!Server!Receive(c)$
 $\quad \wedge h(c, m))$

Instantiate the $E2T!Server!Receive$ module
 $Receive \triangleq \text{INSTANCE } Receive$

Starts a new $E2T$ server
 $Serve(s) \triangleq GRPC!Server!Start(s)$

Stops the given $E2T$ server
 $Stop(s) \triangleq GRPC!Server!Stop(s)$

Provides operators for the $E2T$ server
 $Server \triangleq \text{INSTANCE } Server$

The set of all running $E2T$ servers
 $Servers \triangleq GRPC!Servers$

The set of all open $E2T$ connections
 $Connections \triangleq GRPC!Connections$

$Init \triangleq GRPC!Init$

$Next \triangleq GRPC!Next$

VARIABLES $e2tServers, e2tConns$

$E2T \triangleq$ INSTANCE $E2TService$ WITH
 $servers \leftarrow e2tServers,$
 $conns \leftarrow e2tConns,$
 $Nil \leftarrow [type \mapsto ""]],$
 $SubscribeRequestType \leftarrow \text{"SubscribeRequest"},$
 $SubscribeResponseType \leftarrow \text{"SubscribeResponse"},$
 $UnsubscribeRequestType \leftarrow \text{"UnsubscribeRequest"},$
 $UnsubscribeResponseType \leftarrow \text{"UnsubscribeResponse"},$
 $ControlRequestType \leftarrow \text{"ControlRequest"},$
 $ControlResponseType \leftarrow \text{"ControlResponse"}$

MODULE $TopoService$

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

CONSTANT Nil

VARIABLES $servers, conns$

$vars \triangleq \langle servers, conns \rangle$

The *Topo API* is implemented as a *gRPC* service
 LOCAL $GRPC \triangleq$ INSTANCE $GRPC$

Message type constants

CONSTANT

$CreateRequestType,$
 $CreateResponseType$

CONSTANTS

$UpdateRequestType,$
 $UpdateResponseType$

CONSTANTS

$DeleteRequestType,$
 $DeleteResponseType$

CONSTANT

$GetRequestType,$
 $GetResponseType$

CONSTANT

$ListRequestType,$
 $ListResponseType$

CONSTANT

WatchRequestType,
WatchResponseType

LOCAL *messageTypes* \triangleq
 { *CreateRequestType*,
 CreateResponseType,
 UpdateRequestType,
 UpdateResponseType,
 DeleteRequestType,
 DeleteResponseType,
 GetRequestType,
 GetResponseType,
 ListRequestType,
 ListResponseType,
 WatchRequestType,
 WatchResponseType }

Message types should be defined as strings to simplify debugging

ASSUME $\forall m \in \text{messageTypes} : m \in \text{STRING}$

MODULE *Messages*

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

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

IsCreateRequest(*m*) $\triangleq m.type = \text{CreateRequestType}$
IsCreateResponse(*m*) $\triangleq m.type = \text{CreateResponseType}$
IsUpdateRequest(*m*) $\triangleq m.type = \text{UpdateRequestType}$
IsUpdateResponse(*m*) $\triangleq m.type = \text{UpdateResponseType}$
IsDeleteRequest(*m*) $\triangleq m.type = \text{DeleteRequestType}$
IsDeleteResponse(*m*) $\triangleq m.type = \text{DeleteResponseType}$
IsGetRequest(*m*) $\triangleq m.type = \text{GetRequestType}$
IsGetResponse(*m*) $\triangleq m.type = \text{GetResponseType}$
IsListRequest(*m*) $\triangleq m.type = \text{ListRequestType}$
IsListResponse(*m*) $\triangleq m.type = \text{ListResponseType}$
IsWatchRequest(*m*) $\triangleq m.type = \text{WatchRequestType}$
IsWatchResponse(*m*) $\triangleq m.type = \text{WatchResponseType}$

This section defines predicates for validating *ONOS Topo* 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 $\text{ValidCreateRequest}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidCreateResponse}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidUpdateRequest}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidUpdateResponse}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidDeleteRequest}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidDeleteResponse}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidGetRequest}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidGetResponse}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidListRequest}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidListResponse}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidWatchRequest}(m) \triangleq \text{TRUE}$
 LOCAL $\text{ValidWatchResponse}(m) \triangleq \text{TRUE}$

This section defines operators for constructing *ONOS Topo* messages.

LOCAL $\text{SetType}(m, t) \triangleq [m \text{ EXCEPT } !.type = t]$
 $\text{CreateRequest}(m) \triangleq$
 IF $\text{Assert}(\text{ValidCreateRequest}(m), \text{"Invalid CreateRequest"})$
 THEN $\text{SetType}(m, \text{CreateRequestType})$
 ELSE Nil
 $\text{CreateResponse}(m) \triangleq$
 IF $\text{Assert}(\text{ValidCreateResponse}(m), \text{"Invalid CreateResponse"})$
 THEN $\text{SetType}(m, \text{CreateResponseType})$
 ELSE Nil
 $\text{UpdateRequest}(m) \triangleq$
 IF $\text{Assert}(\text{ValidUpdateRequest}(m), \text{"Invalid UpdateRequest"})$
 THEN $\text{SetType}(m, \text{UpdateRequestType})$
 ELSE Nil
 $\text{UpdateResponse}(m) \triangleq$
 IF $\text{Assert}(\text{ValidUpdateResponse}(m), \text{"Invalid UpdateResponse"})$

```

      THEN SetType(m, UpdateResponseType)
      ELSE Nil

DeleteRequest(m)  $\triangleq$ 
  IF Assert(ValidDeleteRequest(m), "Invalid DeleteRequest")
  THEN SetType(m, DeleteRequestType)
  ELSE Nil

DeleteResponse(m)  $\triangleq$ 
  IF Assert(ValidDeleteResponse(m), "Invalid DeleteResponse")
  THEN SetType(m, DeleteResponseType)
  ELSE Nil

GetRequest(m)  $\triangleq$ 
  IF Assert(ValidGetRequest(m), "Invalid GetRequest")
  THEN SetType(m, GetRequestType)
  ELSE Nil

GetResponse(m)  $\triangleq$ 
  IF Assert(ValidGetResponse(m), "Invalid GetResponse")
  THEN SetType(m, GetResponseType)
  ELSE Nil

ListRequest(m)  $\triangleq$ 
  IF Assert(ValidListRequest(m), "Invalid ListRequest")
  THEN SetType(m, ListRequestType)
  ELSE Nil

ListResponse(m)  $\triangleq$ 
  IF Assert(ValidListResponse(m), "Invalid ListResponse")
  THEN SetType(m, ListResponseType)
  ELSE Nil

WatchRequest(m)  $\triangleq$ 
  IF Assert(ValidWatchRequest(m), "Invalid WatchRequest")
  THEN SetType(m, WatchRequestType)
  ELSE Nil

WatchResponse(m)  $\triangleq$ 
  IF Assert(ValidWatchResponse(m), "Invalid WatchResponse")
  THEN SetType(m, WatchResponseType)
  ELSE Nil

```

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

LOCAL *Messages* \triangleq INSTANCE *Messages*

MODULE *Client*

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

$$\begin{aligned}
 & \text{CreateRequest}(c, m) \triangleq \\
 & \quad \wedge \text{GRPC!Client!Send}(c, \text{Messages!CreateRequest}(m)) \\
 & \text{UpdateRequest}(c, m) \triangleq \\
 & \quad \wedge \text{GRPC!Client!Send}(c, \text{Messages!UpdateRequest}(m)) \\
 & \text{DeleteRequest}(c, m) \triangleq \\
 & \quad \wedge \text{GRPC!Client!Send}(c, \text{Messages!DeleteRequest}(m)) \\
 & \text{GetRequest}(c, m) \triangleq \\
 & \quad \wedge \text{GRPC!Client!Send}(c, \text{Messages!GetRequest}(m)) \\
 & \text{ListRequest}(c, m) \triangleq \\
 & \quad \wedge \text{GRPC!Client!Send}(c, \text{Messages!ListRequest}(m)) \\
 & \text{WatchRequest}(c, m) \triangleq \\
 & \quad \wedge \text{GRPC!Client!Send}(c, \text{Messages!WatchRequest}(m))
 \end{aligned}$$

Instantiate the *Topo!Client!Send* module

Send \triangleq INSTANCE *Send*

MODULE *Receive*

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

$$\begin{aligned}
 & \text{CreateResponse}(c, h(-, -)) \triangleq \\
 & \quad \text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\
 & \quad \quad \wedge \text{Messages!IsCreateResponse}(m) \\
 & \quad \quad \wedge \text{GRPC!Client!Receive}(c) \\
 & \quad \quad \wedge h(c, m)) \\
 & \text{UpdateResponse}(c, h(-, -)) \triangleq \\
 & \quad \text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\
 & \quad \quad \wedge \text{Messages!IsUpdateResponse}(m) \\
 & \quad \quad \wedge \text{GRPC!Client!Receive}(c) \\
 & \quad \quad \wedge h(c, m)) \\
 & \text{DeleteResponse}(c, h(-, -)) \triangleq \\
 & \quad \text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m :
 \end{aligned}$$

$$\begin{aligned}
& \wedge \text{Messages!IsDeleteResponse}(m) \\
& \wedge \text{GRPC!Client!Receive}(c) \\
& \wedge h(c, m)) \\
\text{GetResponse}(c, h(-, -)) & \triangleq \\
& \text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\
& \quad \wedge \text{Messages!IsGetResponse}(m) \\
& \quad \wedge \text{GRPC!Client!Receive}(c) \\
& \quad \wedge h(c, m)) \\
\text{ListResponse}(c, h(-, -)) & \triangleq \\
& \text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\
& \quad \wedge \text{Messages!IsListResponse}(m) \\
& \quad \wedge \text{GRPC!Client!Receive}(c) \\
& \quad \wedge h(c, m)) \\
\text{WatchResponse}(c, h(-, -)) & \triangleq \\
& \text{GRPC!Client!Handle}(c, \text{LAMBDA } x, m : \\
& \quad \wedge \text{Messages!IsWatchResponse}(m) \\
& \quad \wedge \text{GRPC!Client!Receive}(c) \\
& \quad \wedge h(c, m))
\end{aligned}$$

Instantiate the *Topo!Client!Receive* module

$$\text{Receive} \triangleq \text{INSTANCE } \text{Receive}$$

$$\text{Connect}(s, d) \triangleq \text{GRPC!Client!Connect}(s, d)$$

$$\text{Disconnect}(c) \triangleq \text{GRPC!Client!Disconnect}(c)$$

Provides operators for the *Topo* client

$$\text{Client} \triangleq \text{INSTANCE } \text{Client}$$

MODULE *Server*

The *Server* module provides operators for managing and operating on *Topo* 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 *Topo* server.

$$\begin{aligned}
\text{CreateResponse}(c, m) & \triangleq \\
& \wedge \text{GRPC!Server!Reply}(c, \text{Messages!CreateResponse}(m))
\end{aligned}$$

$$\begin{aligned}
\text{UpdateResponse}(c, m) & \triangleq \\
& \wedge \text{GRPC!Server!Reply}(c, \text{Messages!UpdateResponse}(m))
\end{aligned}$$

$$\begin{aligned}
DeleteResponse(c, m) &\triangleq \\
&\wedge GRPC!Server!Reply(c, Messages!DeleteResponse(m)) \\
GetResponse(c, m) &\triangleq \\
&\wedge GRPC!Server!Reply(c, Messages!GetResponse(m)) \\
ListResponse(c, m) &\triangleq \\
&\wedge GRPC!Server!Reply(c, Messages!ListResponse(m)) \\
WatchResponse(c, m) &\triangleq \\
&\wedge GRPC!Server!Reply(c, Messages!WatchResponse(m))
\end{aligned}$$

Instantiate the *Topo!Server!Send* module
 $Send \triangleq \text{INSTANCE } Send$

MODULE *Receive*

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

$$\begin{aligned}
CreateRequest(c, h(-, -)) &\triangleq \\
&GRPC!Server!Handle(c, \text{LAMBDA } x, m : \\
&\quad \wedge Messages!IsCreateRequest(m) \\
&\quad \wedge GRPC!Server!Receive(c) \\
&\quad \wedge h(c, m)) \\
UpdateRequest(c, h(-, -)) &\triangleq \\
&GRPC!Server!Handle(c, \text{LAMBDA } x, m : \\
&\quad \wedge Messages!IsUpdateRequest(m) \\
&\quad \wedge GRPC!Server!Receive(c) \\
&\quad \wedge h(c, m)) \\
DeleteRequest(c, h(-, -)) &\triangleq \\
&GRPC!Server!Handle(c, \text{LAMBDA } x, m : \\
&\quad \wedge Messages!IsDeleteRequest(m) \\
&\quad \wedge GRPC!Server!Receive(c) \\
&\quad \wedge h(c, m)) \\
GetRequest(c, h(-, -)) &\triangleq \\
&GRPC!Server!Handle(c, \text{LAMBDA } x, m : \\
&\quad \wedge Messages!IsGetRequest(m) \\
&\quad \wedge GRPC!Server!Receive(c) \\
&\quad \wedge h(c, m)) \\
ListRequest(c, h(-, -)) &\triangleq \\
&GRPC!Server!Handle(c, \text{LAMBDA } x, m : \\
&\quad \wedge Messages!IsListRequest(m)
\end{aligned}$$

$$\wedge \text{GRPC!Server!Receive}(c) \\ \wedge h(c, m))$$

$$\text{WatchRequest}(c, h(-, -)) \triangleq \\ \text{GRPC!Server!Handle}(c, \text{LAMBDA } x, m : \\ \wedge \text{Messages!IsWatchRequest}(m) \\ \wedge \text{GRPC!Server!Receive}(c) \\ \wedge h(c, m))$$

Instantiate the *Topo!Server!Receive* module
 $\text{Receive} \triangleq \text{INSTANCE } \text{Receive}$

Starts a new *Topo* server
 $\text{Serve}(s) \triangleq \text{GRPC!Server!Start}(s)$

Stops the given *Topo* server
 $\text{Stop}(s) \triangleq \text{GRPC!Server!Stop}(s)$

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

The set of all running *Topo* servers
 $\text{Servers} \triangleq \text{GRPC!Servers}$

The set of all open *Topo* connections
 $\text{Connections} \triangleq \text{GRPC!Connections}$

$$\text{Init} \triangleq \text{GRPC!Init}$$

$$\text{Next} \triangleq \text{GRPC!Next}$$

VARIABLE *topoServers, topoConns*

$$\text{Topo} \triangleq \text{INSTANCE } \text{TopoService} \text{ WITH} \\ \text{servers} \leftarrow \text{topoServers}, \\ \text{conns} \leftarrow \text{topoConns}, \\ \text{Nil} \leftarrow [\text{type} \mapsto ""], \\ \text{CreateRequestType} \leftarrow \text{"CreateRequest"}, \\ \text{CreateResponseType} \leftarrow \text{"CreateResponse"}, \\ \text{UpdateRequestType} \leftarrow \text{"UpdateRequest"}, \\ \text{UpdateResponseType} \leftarrow \text{"UpdateResponse"}, \\ \text{DeleteRequestType} \leftarrow \text{"DeleteRequest"}, \\ \text{DeleteResponseType} \leftarrow \text{"DeleteResponse"},$$


```

GetRequestType ← "GetRequest",
GetResponseType ← "GetResponse",
ListRequestType ← "ListRequest",
ListResponseType ← "ListResponse",
WatchRequestType ← "WatchRequest",
WatchResponseType ← "WatchResponse"

```

```

\ * Modification History
\ * Last modified Fri Aug 13 17:42:40 PDT 2021 by jordanhalterman
\ * Created Fri Aug 13 15:34:11 PDT 2021 by jordanhalterman

```