**NSI Connection Service v2.0**

Status of This Document

Group Working Draft (GWD), candidate Recommendation Proposed (R-P). Distribution is unlimited.

Copyright Notice

Copyright © Open Grid Forum (2008-2013). All Rights Reserved.

Trademark

OGSA is a registered trademark and service mark of the Open Grid Forum.

Abstract

This document describes the Connection Service v2.0 which is one of a suite of services that make up the Network Service Interface.

The NSI is a web-service based API that operates between a requester software agent and a provider software agent. The full suite of NSI services allows an application or network provider to request and manage circuit service instances. Apart from the Connection Service these include the Topology Service and the Discovery Service. The complete set of NSI services is described in GFD.173 Network Services Framework.

This Connection Service document describes the protocol, state machine, architecture and associated processes and environment in which software agents interact to deliver a Connection. A Connection is a point-to-point network circuit that can transit multiple network providers.

Contents

[1. Introduction 3](#_Toc355354811)

[1.1 NSI CS and the Network Service Framework 3](#_Toc355354812)

[1.1.1 NSI architecture 3](#_Toc355354813)

[1.1.2 Topology schema 3](#_Toc355354814)

[1.1.3 Topology Distribution Service 3](#_Toc355354815)

[1.1.4 Discovery Service 3](#_Toc355354816)

[1.2 Heterogeneous transport and Multi-Provider environments 3](#_Toc355354817)

[1.3 Representing and Managing Network resources 4](#_Toc355354818)

[1.3.1 Topology 4](#_Toc355354819)

[1.3.2 Connections 4](#_Toc355354820)

[1.3.3 Path 5](#_Toc355354821)

[1.3.4 Inter-domain vs intra-domain pathfinding 5](#_Toc355354822)

[2. Connection Service architecture 5](#_Toc355354823)

[3. NSI Messages and operations 6](#_Toc355354824)

[4. NSI State Machines 8](#_Toc355354825)

[5. NSI message usage 11](#_Toc355354826)

[5.1 Data Plane Activation 11](#_Toc355354827)

[5.2 Provision Sequence 12](#_Toc355354828)

[6. NSI Framework Header 13](#_Toc355354829)

[7. NSI Connection Services schema 16](#_Toc355354830)

[8. NSI Process Coordination and Message Transport 17](#_Toc355354831)

[8.1 Message Transport 17](#_Toc355354832)

[8.2 Message and Process Coordination 18](#_Toc355354833)

[8.3 Communications 19](#_Toc355354834)

[8.4 Per Request Information Elements 19](#_Toc355354835)

[8.5 Timeouts 20](#_Toc355354836)

[8.6 Failure Recovery 21](#_Toc355354837)

[8.7 Information 22](#_Toc355354838)

[8.8 Per Reservation Information Elements 22](#_Toc355354839)

[8.9 Reservation Versioning Information 22](#_Toc355354840)

[8.10 Data Plane Status Information 23](#_Toc355354841)

[9. Security 24](#_Toc355354842)

[9.1 Authentication and Authorization 24](#_Toc355354843)

[10. Contributors 25](#_Toc355354844)

[11. Glossary 25](#_Toc355354845)

[12. Intellectual Property Statement 26](#_Toc355354846)

[13. Disclaimer 27](#_Toc355354847)

[14. Full Copyright Notice 27](#_Toc355354848)

[15. Appendix A: XML Schema Definitions 27](#_Toc355354849)

[15.1 nsiHeader element 28](#_Toc355354850)

[15.2 Common types 29](#_Toc355354851)

[15.2.1 ServiceExceptionType 29](#_Toc355354852)

[15.2.2 NsaIdType 30](#_Toc355354853)

[15.2.3 VariablesType 30](#_Toc355354854)

[15.2.4 TypeValuePairType 30](#_Toc355354855)

[15.2.5 TypeValuePairListType 31](#_Toc355354856)

[15.2.6 UuidType 31](#_Toc355354857)

[15.2.7 DateTimeType 32](#_Toc355354858)

[15.3 NSI-CS operation specific type definitions. 32](#_Toc355354859)

[15.3.1 Reserve message elements 32](#_Toc355354860)

[15.3.2 ReserveCommit message elements 36](#_Toc355354861)

[15.3.3 reserveAbort message elements 38](#_Toc355354862)

[15.3.4 provision message elements 40](#_Toc355354863)

[15.3.5 release message elements 42](#_Toc355354864)

[15.3.6 terminate message elements 43](#_Toc355354865)

[15.3.7 querySummary message elements 45](#_Toc355354866)

[15.3.8 queryRecursive message elements 47](#_Toc355354867)

[15.3.9 querySummarySync message elements 47](#_Toc355354868)

[15.4 NSI-CS specific types 48](#_Toc355354869)

[16. Appendix B: State Machine Transition Tables 48](#_Toc355354870)

[17. Appendix C: Formal statement of coordinator 49](#_Toc355354871)

[17.1 Aggregator NSA 49](#_Toc355354872)

[17.1.1 Processing of NSI Requests 49](#_Toc355354873)

[17.1.2 Requests from State Machines 50](#_Toc355354874)

[17.2 Ultimate Provider NSA 51](#_Toc355354875)

[17.2.1 Processing of NSI Requests 51](#_Toc355354876)

[17.2.2 Requests from State Machines 51](#_Toc355354877)

[18. Appendix D: Best Practices for NSA implementation 52](#_Toc355354878)

[18.1 Message transport error handling 52](#_Toc355354879)

[18.2 ACK handling 53](#_Toc355354880)

[18.3 Guidelines on timeouts: 54](#_Toc355354881)

[18.4 Parallel processing of messages: 54](#_Toc355354882)

[18.5 NTP servers 54](#_Toc355354883)

[18.6 Transport plane failures 54](#_Toc355354884)

[19. Appendix E: Tree and chain examples 55](#_Toc355354885)

[19.1 Connection example managed by an NSA chain 55](#_Toc355354886)

[19.2 Connection example managed by an NSA tree 56](#_Toc355354887)

[20. References 57](#_Toc355354888)

# Introduction

This grid forum document defines the Connection Service (CS) protocol that enables the reservation, creation, management and removal of Connections. The Network Service Interface (NSI) is the set of protocols and parameters that are used between a software agent requesting a network service and the software agent providing that Network Service.

## NSI CS and the Network Service Framework

Network resources and capabilities are presented to the consumer through a set of Network Services, the Network Service Framework (NSF) presents a unified model for interacting with these services. Network Services include the ability to create connections, to share topologies and to perform other services needed by a federation of software agents.

### NSI architecture

The NSF describes a set of architectural elements that make up the NSI architecture [1]. Network Service Agents (NSAs) are the agents that manage service requests. An NSA can act either in the role of a requesting agent or a provider agent. The NSI is an interface between these software agents and NSI Protocol messages are exchanged over the NSI interface.

The agents and NSI protocol exist in a notional Network Service Plane. The NSF describes an environment within which network resources are treated as explicitly manageable objects. Within the framework, these network resources can be selected, allocated, interrogated, and manipulated by software agents on behalf of requesting users.

### Topology schema

The NSI topology schema is used by the NSI CS but is not defined in this document. The NSI topology schema is described in the OGF recommendation: GWD-R-P Network Service Interface Topology Representation. The document describes a normative extension to the Network Markup Language (NML) base schema version 1 which allows the description of service plane objects required for the NSI CS.

### Topology Distribution Service

The NSI topology distribution service is used by the NSI CS but is not defined in this document. NSI topology exchange is performed using the topology service defined in GWD-I ‘Network Service Interface, Topology Service Distribution Mechanisms’. The document describes a normative schema which allows the description of service plane objects required for the NSI CS. Additionally it describes a set of distribution mechanisms for the network topology descriptions.

### Discovery Service

The NSI Discovery Service is used by the NSI CS but is not defined in this document. The discovery service is a web service that allows an NSI Requester Agent (RA) to discover information about the services available in a Provider Agent (PA) and the versions of these services.

## Representing and Managing Network resources

### Heterogeneous transport and Multi-Provider environments

NSI is designed to support the creation of circuits that transit several transport network providers. This creates specific requirements for authorization and authentication which are addressed by the NSI. Traditional models of circuit services and control planes adopt a single very tightly defined data plane technology, and then hard code the service attributes into the control plane protocols. Multi-domain services need to employ many data plane technologies, to make this possible they need to recognize a wider array of service attributes. The NSI supports an abstracted notion of a Connection, and the NSI Reserve message includes a flexible schema for specifying service specific constraints. These service constraints will be evaluated against the technology available to local network service providers traversed by the service. It is up to the NSI service’s path finders to select a path that meets these constraints. In this way the NSI allows a single Service Plane protocol suite to deliver Connections that traverse heterogeneous transport technologies.

### Topology

In the NSI Topology the data plane in modeled as interconnected Networks, were the Networks are groupings of STPs. As shown in Figure 1, NSI Networks interconnect at a shared point known as an SDP. An SDP is a grouping of two adjacent STPs belonging to different Networks. A complete Connection can be built up by concatenating individual Connections at SDPs.

An overview of the NSI topology concepts is provided in can be found in OGF GFD.173: Network Service Framework v1.0 [1]. The detailed NSI topology representation can be found in GWD-R-P: Network Service Interface Topology Representation [3].

### Connections

The CS protocol supports the request and creation of Connections. In the Data Plane, an NSI Connection is a physical circuit through which information is delivered from an ingress point to an egress point. NSI CS v 2.0 only supports the point-to-point, these Connections can be unidirectional or bidirectional.

Connections can be constructed between pairs of STPs and transit Networks - see Figure 1. In NSI v2.0 it is assumed that any two STP within a particular Network are able to be connected. This implies that all Networks are non-blocking.

figure2

Figure : Inter-Network representation of a Connection

Connections within a Network are intra-domain functions, and the technology details of how two STPs are actually connected is up to the local NRM and not a concern of the NSI protocols.

### Path

Inter-Network Connections extend across multiple networks; they are constructed by concatenating connections built across the individual networks. This is done by choosing appropriate STPs such that the egress STP of one connection corresponds directly with the ingress STP of the successive connection.

A Connection request can optionally populate the Path attribute. A Path is an ordered list of STPs that describe the route that should be taken by the Connection. The STPs listed in a Path will be used as constraints by the Inter Network path-finder. The Connection will include all of the STPs in the Path in the sequence that they are listed. However a Path is not ‘strict’ in the sense that Connection is allowed to transit intermediate STPs between the STPs listed in the Path.

### Inter-domain vs intra-domain pathfinding

There are two levels of pathfinding related to the NSI architecture: the inter-domain pathfinding which is concerned with choosing a path across the global topology of Networks, and the intra-domain pathfinding concerned with the transport resources within the Network. NSI is concerned only with inter-Network pathfinding.

# Connection Service architecture

The Service plane is a notional plane that includes the Network Service Agents and the NSI interface. Each Network in the Data Plane has an associated NSA in the Service Plane. This is described in more detail in GFD.173 Network Services Framework.

Within the Service Plane NSAs have a peering relationship; a Requester Agent (RA) sends a service request to a Provider Agent (PA). The NSI protocol is made up of messages that are exchanged over this peering interface.

An NSA can take on a number of roles:

* uRA: The Ultimate Requester Agent is the originator of a service request.
* AG: The Aggregator has more than one child NSA, and has the responsibility of aggregating the responses from each child NSA.
* uPA: Ultimate Provider Agent

In addition the concept of a Network Resource Manager (NRM) is introduced. The role of the NRM is to manage the resources in the Data Plane. Typically this might be an equipment vendor’s network management system.

\\CHFILE02\Folders\guy\Desktop\nsi\Arch_diagrams.emf

Figure 1. Flexible hierarchical NSA relationship

Central to the NSI architecture is the decoupling of the Service Plane from the Data Plane. One of the differentiating features of NSI is that the Service Plane is not required to be congruent with the data plane. NSI messages do not need to transit the same NSA/Networks pairings in the same sequence that the Connection itself transits.

# NSI Messages and operations

NSI messages are classified into two types, messages that are passed from an RA to a PA and messages that are passed from a PA to an RA, these messages are summarized in Table 1 and Table 2. Messages sent from an RA to a PA are request messages; the PA is expected to send a reply message to each RA request message. Messages sent from the PA to an RA are categorized as either a reply to a request, or a notification message that is sent asynchronously.

Each Message invokes a corresponding operation in the recipient. The “Type” field in each message denotes the message type:

* If the message is of type RSM then the message is to be processed using the Reservation State Machine (RSM)
* If the message is of type PSM the message is to be processed using the Provision State Machine (PSM)
* If the message is of type LSM the message is to be processed using the Lifecycle State Machine (LSM)
* If the message is of type Query this designates a Query request and requires an associated reply messages
* If the message is of type Notification this designates asynchronous notification messages sent by a PA to an RA.

Table 1 below summarizes all of the message primitives and lists the Message Type associated with the message.

|  |  |  |
| --- | --- | --- |
| **Message**  (abbreviation) | **Type** | **Description** |
| ***Reserve***  (rsv.rq) | RSM | A Reserve request message is sent to reserve a new Connection or modify an existing Connection. To keep track of multiple modifications, a monotonically increasing *version* number attribute should be specified by the RA.  If *connectionID* attribute is not specified in the Reserve request, the RA requests the PA to Reserve network resources for a Connection between two STP’s constrained by certain service parameters. The PA should return a *connectionID* in the output of the operation. In this case, if *version* is not specified, the RA will set this by default to 0.  If a *connection\_ID* is specified, this is considered to be a request to modify an existing reservation of a Connection. In CSv2.0, modification of *startTime*, *endTime* and *bandwidth* are supported. In this case, if *version* is not specified, it is considered an illegal operation.  In this operation, PA checks the availability of requested resources. If resources are available, they are held and reserveConfirmed is returned. If resources are not available, reserveFailed is returned. The reservation data base is not updated by this operation. |
| ***reserveCommit***  (rsvcommit.rq) | RSM | A message to request a commit action on the reservation made by an earlier Reserve request. The operation will fail when one or more children uPA have timeout. |
| ***reserveAbort***  (rsvabort.rq) | RSM | A message to request the abort and release of a held reservation made by the Reserve operation. This message also enables children at the ReserveFailed state to return to the Reserved state. |
| ***provision***  (prov.rq) | PSM | A message to request that the PA transit to the Provisioned state. |
| ***release***  (release.rq) | PSM | A message to request that the PA transit to the Released state. |
| ***terminate***  (term.rq) | LSM | A message to request the PA release the Provisioned resources, and allow the PA to clean up RSM, PSM and all related data structures. |
| ***querySummary***  () | Query | A message to request a summary query |
| ***queryRecursive***  () | Query | A message to request a recursive query |
| ***querySummarySync***  () | Query | A message to request a summary query using the synchronous mode. |

Table – RA to PA messages

|  |  |  |
| --- | --- | --- |
| **Message**  (abbreviation) | **Related SM** | **Description** |
| ***reserveConfirmed***  (rsv.cf) | RSM | A message replying to a reserve request. This message confirms that the requested resources are available and the reservation is held. |
| ***reserveFailed***  (rsv.fl) | RSM | A message replying to a reserve request. This message notifies the RA that the requested resources are not available and the reservation request has been failed. |
| ***reserveCommitConfirmed***  (rsvcommit.cf) | RSM | A message replying to a commit request. This message notifies the RA that a held reservation has been successfully committed. |
| ***reserveCommitFailed***  (rsvcommit.fl) | RSM | A message replying to a commit request. This message notifies the RA that a held reservation has not been committed because a timeout happened at one or more children. |
| ***reserveAbortConfirmed***  (rsvabort.cf) | PSM | A message replying to an abort request. This message notifies the RA that a reserveAbort message has been delivered to all the children. |
| ***provisionConfirmed***  (prov.cf) | PSM | A message replying to a provision request. This message notifies the RA that a provision message has been delivered to all the children. |
| ***releaseConfirmed***  (release.cf) |  | A message replying to a release request. This message notifies the RA that a release message has been delivered to all the children. |
| ***terminateConfirmed***  (term.cf) | LSM | A message replying to a terminate request. This message notifies the RA that a terminate message has been delivered to all the children. |
| ***querySummaryConfirmed***  () | query | A message replying to a query request. This message returns the results of the summary query. |
| ***queryRecursiveConfirmed***  () | query | A message replying to a query request. This message returns the results of the recursive query. |
| ***querySummarySyncConfirmed***  () | query | A message replying to a query request. This message confirms the receipt of a synchronous summary query. |
| ***errorEvent***  () | notification | A generic message that provides a notification of an error from PA to RA |
| ***reserveTimeout***  () | notification | A message to notify that a commit timeout has occurred at a PA. |
| ***dataPlaneStatusChange***  (dataPlaneStatusChange.nt) | notification | A notification message sent up from a PA when a data plane status has changed. Possible data plane status changes are: activation, deactivation and activation version change. |
| ***messageDeliveryTimeout***  () | notification | A notification message sent upstream when the delivery of a request message has timed out either because the Message Transport Layer (MTL) has timed out due to no ack being returned for a certain period of time (defined by the MTL), or no reply has been received by the Coordinator for a certain period of time (defined by the Coordinator). |

Table – PA to RA messages

# NSI State Machines

The behavior of the NSI CS protocol is modeled in two ways: with state machines and with behavioral description of the coordinator function. In total there are three state machines, the Reservation State Machine (RSM), the Provision State Machine (PSM) and the Lifecycle State Machine (LSM). The state machines explicitly regulate sequence by which messages are processed of each for each message type: RSM, PSM and LSM.

Figure 1 below shows the Reservation State Machine. This state machine defines the sequence of operation of messages of type RSM.



Figure 1. Reservation State Machine

The NSI protocol supports advance-reservation of services. Each reservation has properties such as:

* A-point, Z-point (mandatory)
* Start-time, End-time (mandatory)
* Bandwidth, Labels (optional)

NSI Connection reservations are created using a two phase process. In the first phase (reserve) the availability of the requested resources is checked, if the resources are available, these are held. In the second phase (commit) the requester has the choice to either commit or abort the reservation that was held in the first phase.

When a reservation/modification is committed the reservation database is updated. If a requester fails to commit a held reservation after a certain period of time the provider has the option of timing out the reservation and releasing the resources from the reservation database.

Modification of a reservation is now supported in NSI v2.0. The reserve request message is used for both the initial reservation and subsequent modifications. A version number is specified in the reservation request message. The number is an integer and should be monotonically increasing with each subsequent modification. The version number is updated after a commit results in a transition to the reserved state. A query will return the currently committed reservation version number.

Currently, modification of start\_time, end\_time and bandwidth are supported.

Modification of currently active data plane may be implemented in a hitless way by the provider, however support for such hitless operation is up to the policy of the provider and is not guaranteed by the NSI protocol per se.

Figure 2 below shows the Provision State Machine. This state machine defines the sequence of operation relating to messages of type PSM.

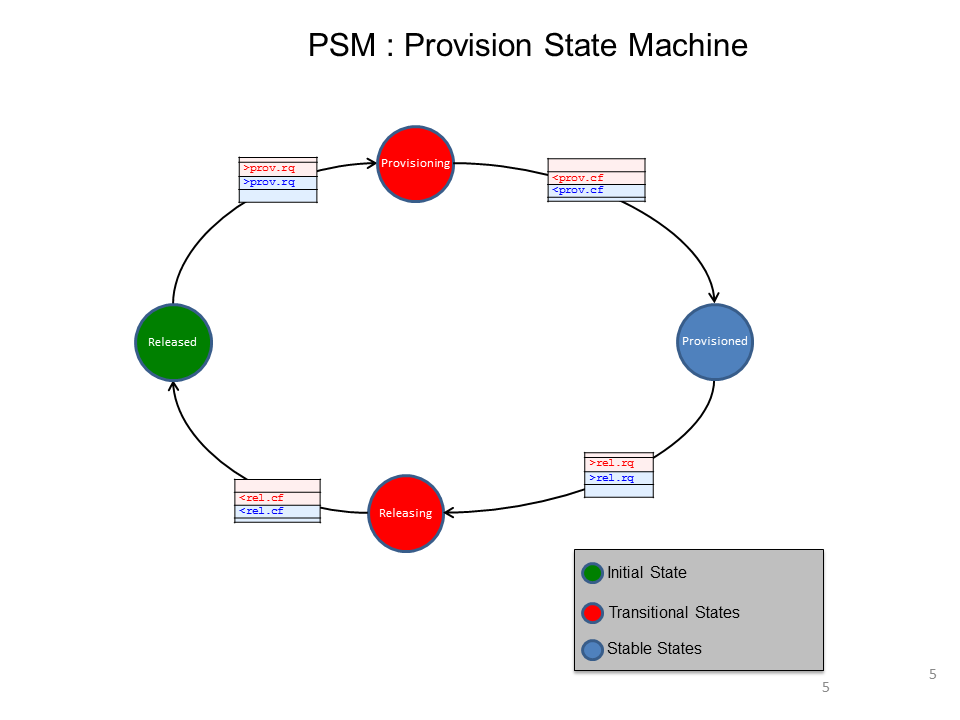


Figure 2. Provision State Machine

The Provision State Machine is a simple state machine which transits between the Provisioned and the Released state. An instance of the PSM is created when an initial reservation is committed, and at that time it remains in the Released state. The PSM transits states independent of the state of the RSM. Note that staying at the Provisioned state is necessary but not sufficient to activate the data plane. The data plane is active if the PSM is in “Provisioned” state AND startTime < current\_time < endTime.

The PSM is designed to allow a Connection to be repeatedly provisioned and released.

Figure 3 below shows the Lifecycle State Machine (LSM). This state machine defines the sequence of operation relating to messages of type LSM. The LSM allows terminate and terminateConfirm messages to be to send and received. When an errorEvent (fcd\_end) is received/sent, LSM transits to the Failed state.

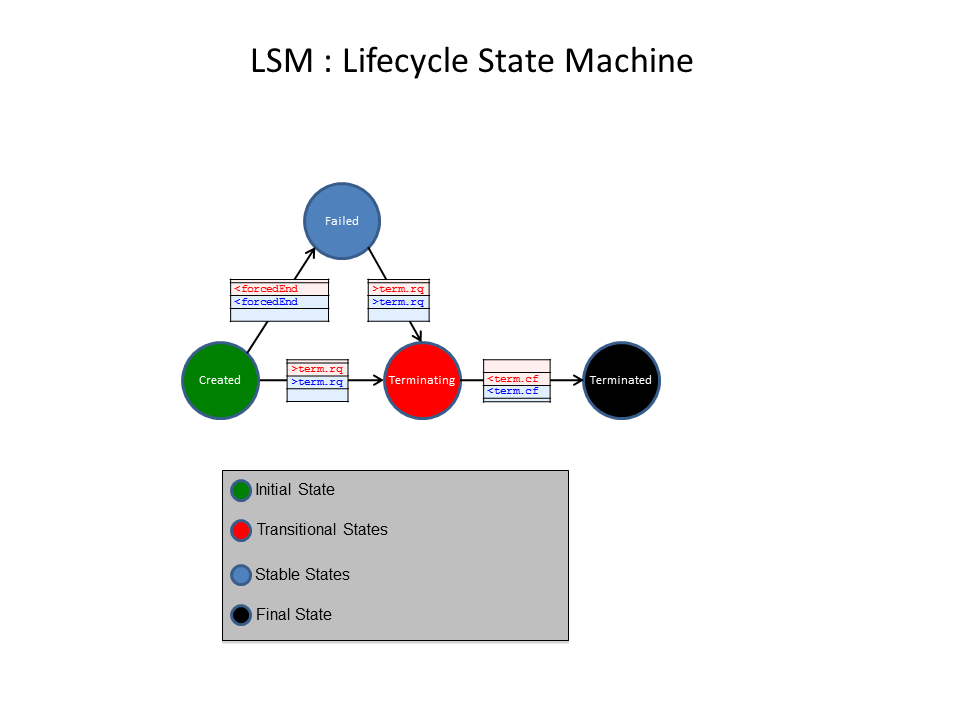


Figure 3. Lifecycle State Machine

# NSI message usage

## Data Plane Activation

Figure 4 below shows the conditions necessary for data plane activation.

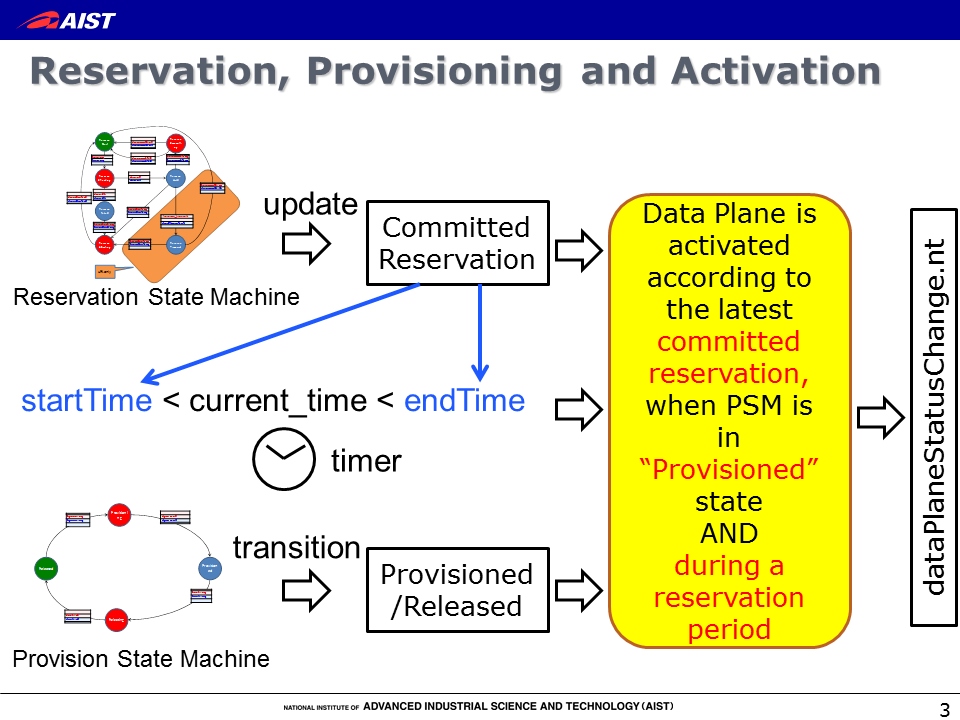


Figure 4. Data Plane activation condition

The Data Plane will become active at any of the following events (if necessary conditions are met). The activated connection will use the latest committed reservation.

* PSM transits to Provisioned state
* At the startTime is reached
* Reservation has been updated (by commit of a reservation/modify)
* Data plane is recovered from an error

The activation/deactivation of the Data Plane should be notified using the DataPlaneStateChange.nt notification message. Errors should be notified using the generic error messages as follows:

* activateFailed: Activation failed at the time when uPA attempted to activate its data plane
* deactivateFailed: Deactivation failed at the time when uPA attempted to deactivate its data plane
* dataplaneError: Data plane is deactivate unexpectedly. This error condition is recoverable.
* forcedEnd: Something unrecoverable has happened in the uPA/NRM

## Provision Sequence

Figures 5 and 6 below show two examples of how message primitives can be used to activate a Connection. As shown in Figure 5, two modes of Provisioning are possible: automatic and manual provisioning.

These provisioning methods can be realized using the semantics described in Figure 4. In the automatic provisioning mode, the provision request message is sent from the RA to the PA before the startTime, and the Data Plane is activated at the startTime, and deactivated at the endTime. If a provision request message is sent after the startTime, the data plane is activated when the provision request is received by the uPA - this sequence is referred to as manual provisioning. If the uRA wishes to activate the data plane as soon as possible, the uRA should designate a startTime which is the same as or before the time a reservation is made, and then issue a provision request message immediately after a reservation is made. This behavior can be considered as an on-demand mode of provisioning.



Figure 5. Automatic Provisioning and Manual Provisioning

A connection can be repeatedly provisioned and released by provision request messages and release request messages, as shown in Figure 6.

.



Figure 6. Release and Provisioning

# NSI Framework Header

(Move to framework doc?)

This section describes the new NSI framework header as implemented in NSI 2.0. The structural layout of the header is provided, along with definitions for each field, and a protocol compliance table outlining when fields must be included.

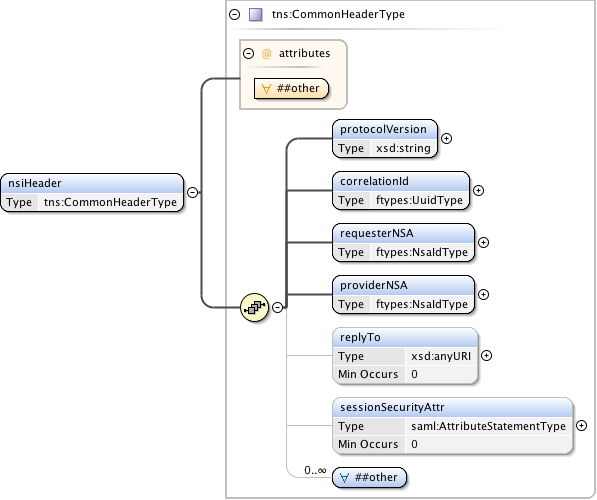


Figure – Common NSI 2.0 framework message header.

Figure 2 shows the new NSI 2.0 message header sent as part of every NSI message exchange. The original Oxford proposal was to create two separate header definitions, one for operation requests, and one for confirmed, failed, and notification messages. After further consideration a single header definition with optional elements was created to capture the semantics of both the request and response headers. The following attributes and elements are defined as part of the new NSI protocol header:

**Mandatory Elements:**

*protocolVersion*

A URI identifying the specific protocol version carried in this NSI message. The protocol version is modeled separately from the namespace of the WSDL and XML schema to capture behavioral changes that cannot be modeled in schema definition, and to avoid updating of the schema namespace.

*correlationId*

An identifier provided by the requesting NSA used to correlate to an asynchronous response from the responder. It is recommended that a Universally Unique Identifier (UUID) URN as per IETF RFC 4122 be used as a globally unique value.

*requesterNSA*

The NSA identifier for the NSA acting in the Requester Agent role for the specific NSI operation.

*providerNSA*

The NSA identifier for the NSA acting in the Provider Agent role for the specific NSI operation.

**Optional Elements:**  
  
*replyTo*

The Requester NSA's SOAP endpoint address to which asynchronous messages associated with this operation request will be delivered. This is only populated for the original operation request (i.e. reserve, provision, release, terminate, and query), and not for any additional messaging associated with the operation. If no endpoint value is provided in a operation request, then it is assumed the requester is not interested in a response and will use alternative mechanism to determine the result.

*sessionSecurityAttr*

Security attributes associated with the end user’s NSI session. This field can be used to perform authentication, authorization, and policy enforcement of end user requests. Is only provided in the operation request (i.e. reserve, provision, release, terminate, and query), and not for any additional messaging associated with the operation.

*other (any)*

Provides a flexible mechanism allowing additional elements in the protocol header for exchange between two-peered NSA. Use of this element field is beyond the current scope of this NSI specification, but may be used in the future to extend the existing protocol without requiring a schema change. Additionally, the field can be used between peered NSA to provide additional context not covered in the existing specification, however, this is left up to specific peering agreements.

**Optional Attributes:**  
*other (anyAttribute)*

Provides a flexible mechanism allowing additional attributes in the protocol header for exchange between two-peered NSA. Use of this attribute field is beyond the current scope of this NSI specification, but may be used in the future to extend the existing protocol without requiring a schema change. Additionally, the field can be used between peered NSA to provide additional context not covered in the existing specification, however, this is left up to specific peering agreements.

In addition, we identify the specific NSI CS operation being invoked using the “Soapaction:” element in the HTTP header as per section 6.1.1 of “Simple Object Access Protocol (SOAP) 1.1” found at <http://www.w3.org/TR/SOAP>. Discussion occurred in Oxford on the topic of including a specific “operation” element within the NSI header, however, this would have been a duplicate of the “Soapaction:” element, and therefore, was left out.

Table – NSI CS message use of header fields

# NSI Connection Services schema

The NSI-CS 2.0 protocol is defined using XML schema (XSD) to describe the common message header and individual Connection Services operation elements and types. Web Service Description Language (WSDL) is used to describe the interface or operation bindings, capturing the request, response, and error (fault) interactions. Finally, WSDL is once again used to provide a SOAP specific transport binding as a reference specification; however, the XML schema definitions can be utilized to encapsulate the NCI-CS protocol into other transport bindings.

The following individual namespaces are defined as part of the NSI-CS 2.0 protocol:

|  |  |
| --- | --- |
| Description | Namespace URL |
| Common types shared between NSI message header and CS operation definitions. | http://schemas.ogf.org/nsi/2013/04/framework/types |
| NSI message header definition. | http://schemas.ogf.org/nsi/2013/04/framework/headers |
| NSI CS operation specific type definitions. | http://schemas.ogf.org/nsi/2013/04/connection/types |
| NSI CS operation definitions | http://schemas.ogf.org/nsi/2013/04/connection/interface |
| Provider NSA interface SOAP binding | http://schemas.ogf.org/nsi/2013/04/connection/provider |
| Requester NSA interface SOAP binding | http://schemas.ogf.org/nsi/2013/04/connection/requester |

Table – NSI CS namespaces

Appendix E provides a detailed overview of the NSI-CS XML schema definitions.

|  |  |
| --- | --- |
| **Elements** | **Description** |
| messageDeliveryTimeout |  |
| provision | The NSI CS provision message allows a Requester NSA to transition a previously requested reservation into a provisioned state. A reservation in a provisioned state will activate associated data plane resources during the scheduled reservation time. |
| provisionConfirmed | This provisionConfirmed message is sent from a Provider NSA to Requester NSA as an indication of a successful provision request. This is in response to an original provision request from the associated Requester NSA. |
| queryRecursive |  |
| queryRecursiveConfirmed |  |
| queryRecursiveFailed |  |
| querySummary | The querySummary message provides a mechanism for a Requester NSA to query the Provider NSA for a set of connection service reservation instances between the RA-PA pair. This message can also be used as a reservation status polling mechanism.    Elements compose a filter for specifying the reservations to return in response to the querySummary request. Querying of reservations can be performed based on connectionId or globalReservationId. Filter items specified are OR'ed to build the match criteria. If no criteria are specified then all reservations associated with the requesting NSA are returned. |
| querySummaryConfirmed | This querySummaryConfirmed message is sent from the Provider NSA to Requester NSA as an indication of a successful querySummary operation. This is in response to an original querySummary request from the associated Requester NSA |
| querySummaryFailed | This querySummaryFailed message is sent from the target NSA to requesting NSA as an indication of a querySummary operation failure. This is in response to an original querySummary request from the associated Requester NSA. It is important to note that a querySummary operation that results in no matching reservations does not result in a querySummaryFailed message, but instead a querySummaryConfirmed with an empty list of reservations. |
| querySummarySync |  |
| querySummarySyncConfirmed |  |
| querySummarySyncFailed |  |
| release | The NSI CS release message allows a Requester NSA to transition a previously requested reservation into a released state. A reservation in a released state will deactivate associated data plane resources. |
| releaseConfirmed | This releaseConfirmed message is sent from a Provider NSA to Requester NSA as an indication of a successful release request. This is in response to an original release request from the associated Requester NSA |
| reserve | The NSI CS reserve message allows a Requester NSA to reserve network resources for a connection between two STP's within the network constrained by the provided service parameters. This reserve message allows a Requester NSA to check the feasibility of a connection reservation, or modification an existing connection reservation. Any resources associated with the reservation or modification operation will be allocated and held until a reserveCommit message is received for the reservation or timeout occurs (whichever arrives first). |
| reserveAbort | The NSI CS reserveAbort message allows a Requester NSA to abort a previously requested reservation or modification on a reservation. |
| reserveAbortConfirmed | This reserveAbortConfirmed message is sent from a Provider NSA to Requester NSA as an indication of a successful reserveAbort request. The reservation in question will have any pending modifications cancelled and returned to the reservation state existing before the modification. |
| reserveCommit | The NSI CS reserveCommit message allows a Requester NSA to commit a previously allocated reservation or modification on a reservation. The reserveCommit request must arrive at the Provider Agent before the reservation timeout occurs. |
| reserveCommitConfirmed | This reserveCommitConfirmed message is sent from a Provider NSA to Requester NSA as an indication of a successful reserveCommit request for a reservation previously in a Reserve Held state. |
| reserveCommitFailed | This reserveCommitFailed message is sent from a Provider NSA to Requester NSA as an indication of a reserve (or modify) commit failure. This is in response to an original reserveCommit request from the associated Requester NSA. |
| reserveConfirmed | A Provider NSA sends this positive reserveConfirmed response message to the Requester NSA that issued the original reserve request message. Receipt of this message is an indication that the requested reservation parameters were available and will be held until a reserveCommit message is received for the reservation or timeout occurs (whichever arrives first). |
| reserveFailed | A Provider NSA sends this negative reserveFailed response to the Requester NSA that issued the original reservation request message if the requested reservation criteria could not be met. This message is also sent in response to a reserve request for a modification to an existing schedule if the required modification is not possible. |
| serviceException | The service exception is raised when a fault is detected. The message includes attributes that describe an exception and include the identifier of the NSA generating the exception, the error identifier for each known fault type. The service exception supports a list of service exceptions capturing failures within the request tree. |
| terminate | The NSI CS terminate message allows a Requester NSA to transition a previously requested reservation into a terminated state. A reservation in a terminated state will release associated resources |
| terminateConfirmed | This terminateConfirmed message is sent from a Provider NSA to Requester NSA as an indication of a successful terminate request. This is in response to an original terminate request from the associated Requester NSA. |

Table – NSI CS message overview

# NSI Process Coordination and Message Transport

## Message Transport

Inherent in the NSI CSv2.0 is the flexibility to instantiate tree workflows of arbitrary complexity. This flexibility necessitates the formalization of the concept of a Message Transport Layer (MTL). The purpose of the MTL is to deliver an abstracted message delivery mechanism to the NSI layer. This formal separation of the message delivery from the message themselves it aims to simplify the operation of NSI and allow ready migration to new transport layers.

The MTL is responsible for end-to-end communications between NSAs and has two primary requirements:

* Send and receive messages; The MTL is responsible for encapsulating the datagram with all the necessary information (e.g. source/destination, port, protocol, etc) for delivery, and removing transport information when a datagram is received prior to passing it to the Coordinator.
* Verify if a message was received by the intended destination NSA; To do this, the MTL utilizes message receipt acknowledgement and timeouts to determine if a packet was or was not successfully delivered.

It should be noted that there is no inherent requirement for the MTL to be reliable or ensure delivery order because these functions can be accomplished by the higher level processes. For NSI CSv2.0, SOAP (ref ?) is used as the MTL.

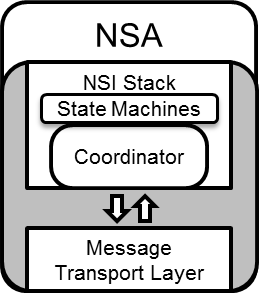


Figure 1. Coordinator and MTL in an NSA

## Message and Process Coordination

As the MTL defines only basic message transport capabilities, the NSA requires more intelligent message and process coordination to function. These capabilities are defined in a logical entity called the coordinator. Even though both the MTL and Coordinator are part of the NSA, the Coordinator is integral to the NSI Stack, whereas the MTL is functionally distinct and can be readily substituted.

Figure 2. Example workflow tree of a NSI reservation request

The Coordinator has several responsibilities, its main jobs are:

* To coordinate, track, and aggregate (if necessary) message requests, replies, and notifications
* To process or forward notifications as necessary
* To service query requests

## Communications

Reliable communications is essential to the reliable operation of the NSI. As the MTL provides only basic message transport capabilities, it is the responsibility of the Coordinator to keep track of message states and make decisions accordingly. To do this, the Coordinator maintains the following information on a per NSI request message basis:

* Who the (NSI request) message was sent to
* Was the message received (i.e. ack’ed) or not (i.e. MTL timeout)
* Which NSA has sent back an NSI reply (e.g. \*.confirm, \*.fail, \*.not\_applicable) for the initial NSI request (e.g. \*.request)

## Per Request Information Elements

For each NSI request/reply interaction, the Coordinator must maintain several pieces of information that are associated with those messages. This is particularly important for the Aggregator NSAs (AG) since it must keep track of the message status for each of its children in the request workflow. The information that must be retained includes:

* NSA IDs: A list of NSA that the messages were sent to.
* Connection ID: The name that uniquely identifies the connection request/reservation (see “ogf\_nsi\_connection\_types\_v2\_0.xsd” for more detail).
* Correlation ID: The label that identifies messages associated to a unique NSI request/reply interaction. This is used to associate NSI replies to requests, and also to identify messages for re-delivery (i.e. message retries).
* Message status: This provides the message state for each of the NSI requests sent to the various NSAs to reflect the current status, such as; MTL sent, MTL receipt acknowledged, MTL timeout, and Coordinator timeout.

In addition to the detailed information of the status for each child NSA NSI request (see “request\_segment\_list(Conn\_ID, NSA)” in Figure 4.), the Coordinator must also maintain an aggregate message status indicating if the messages were delivered successfully to all the children (see “request\_list(Conn\_ID)” Figure 4.).

## Timeouts

In order to identify communication failures, both the MTL and Coordinator have defined timeouts to detect breakdowns in certain aspects of the communication channel. The characteristics of these timeouts are outlined below:

* MTL Timeout
  + Symptoms
    - No acknowledgement of message receipt after a pre-determined time period after the message was sent.
  + Causes
    - Failure in end-to-end communication between NSAs.
* Coordinator Timeout
  + Symptoms
    - No NSI reply after a pre-determined time period after the NSI request was sent.
  + Causes
    - Failure in the MTL such that the NSI reply (from the PA) could not be delivered to the requesting NSA (the RA).
    - The NSA processing the request (e.g. PA) was unable to reply due to incapacitation.
    - The NSA processing the request (Aggr) was blocked waiting for NSI replies from downstream NSAs. (This scenario can be resolved by adjusting the Coordinator timeout value of the requester.)

As both the MTL and Coordinator timeouts are distinct and can be set exclusively, it is important to understand the interplay between the MTL and Coordinator timeouts in order to mitigate artificial “failures”.

Figure 3. Potential MH/MTL timeout sequences

In the event of an MTL or Coordinator timeout, the Coordinator generates a message delivery failure notification and sends it up the workflow tree (towards the uRA).

## Failure Recovery

In NSI CSv2.0, there is no inherent expectation that any (interim) NSAs except for the ultimate requester NSA (uRA) make a decision and take action when it receives a message delivery failure notification. Any interim (aggregator) NSA that receives the delivery failure notification MUST forward it up the workflow tree if it does not want to or cannot resolve the issue. On receiving the message delivery failure notification, the uRA has two choices:

1. Terminate the reservation; this is done by sending down a terminate request (“term.rq”) through the workflow tree.
2. Request redelivery of the original message; this is done by resending down the original message through the workflow tree.

When the original message is resent down the workflow tree, it will contain the original Correlation ID. Aggregators (AGs) receiving the duplicate request SHOULD only attempt redelivery of the message to children that it did not receive an acknowledgement for (i.e. MTL timeout) or reply to (i.e. Coordinator timeout) the original message. If the message sent with the original Correlation ID does not match the original message (e.g. different message parameters/content), the message is rejected and an error returned.

## Information

While per request information (see “Per Request Information Elements”) will only persist for the duration of the NSI request/reply interaction, the Coordinator must also store information associated with the entire reservation.

Figure 4. Information maintained by Coordinator for each Connection Reservation and NSI Request

## Per Reservation Information Elements

To support the recursive query function in NSI CS v2.0, an AG Coordinator must track the current states (i.e. RSM, PSM, LSM) of all its children as well as the condition of the data plane status. This information is persistent but updated over the lifetime of the reservation (see “connection\_segment\_list(Conn\_ID, NSA)” in Figure 4.).

* NSA IDs: A list of NSA that are part of the connection request workflow tree.
* Connection IDs: Connection IDs associated with each NSA in the workflow tree.
* Source and Destination STPs: The source and destination STP of each NSA segment composing the end-to-end circuit
* Reservation Parameters: A list of reservation parameters (e.g. start/end time, bandwidth, etc) associated with each NSA segment
* RSM States: State of children’s Reservation State Machine and current committed reservation version number
* PSM States: State of children’s Provision State Machine
* LSM States: State of Children’s Lifecycle State Machine
* Data plane states: The status of the children’s data plane (i.e. in-service/out-of-service), and the version of the reservation instantiated in the data plane if it is in-service (see “Data Plane Status Information” section for more details).

## Reservation Versioning Information

To support the modification of reservations, the notion of versioning has been introduced to identify the instance of a reservation over its lifetime. The properties and characteristics of the versioning is as follows:

* Version numbers are integer values ≥ 0 (zero)
* Version numbers are assigned by the RA when a reservation request (i.e. NSI\_rsv.rq) is made to a PA
* An integer ≥ 0 (zero) can be assigned by the RA for the initial request, however subsequent modifications to the request MUST use monotonically increasing version numbers (although they need not be sequential)
* If a version number is not specified in an NSI\_rsv.rq, it is assumed to be 0 (zero) regardless if it is the initial or subsequent requests
* An NSI\_rsv.rq with a version number ≤ the (highest) current committed reservation version number will result in a failed request and an appropriate error
* A uPA must keep track of
  + Version number of currently committed reservation
  + Version number of pending modification request (if any)
  + Version number of reservation instantiated in the data plane by the NRM
* An Aggregator must keep track of
  + Version numbers of currently committed reservations in each child segment
  + Version number of pending modification request (only one modify can be outstanding at any time)
  + Version numbers of reservations instantiated in the data plane in each child segment (see “Data Plane Status Information”)
* Version numbers of failed (e.g. timed-out) or aborted modifications are not stored, and therefore can be reused. For example:

1. Successful initial NSI\_req.rq(ver = 2) results in Reservation(v2)
2. Successful modify NSI\_req.rq(ver = 5) results in Reservation(v5)
3. Failed modify NSI\_req.rq(ver = 6) retains Reservation(v5)
4. Subsequent successful modify NSI\_req.rq(ver = 6) results in Reservation(v6)

## Data Plane Status Information

To reflect the state of the data plane, a Coordinator will maintain three flags:

* Activate (boolean): To indicate whether the data plane is in-service or out-of-service
  + uPA:
    - True => data plane is in-service
    - False => data plane is out-of-service
  + AG:
    - True => all children’s data planes are in-service
    - False => one or more children’s data plane is out-of-service
* Version (int): The version of the committed reservation instantiated in the data plane. NB: This field is only valid when ActivateFlag is true.
  + uPA: Version number of the committed reservation
  + AG: Largest version number of the committed reservation among the children
* VersionConsistent (boolean): Reflects if the “Version” numbers are consistent
  + uPA: This is always True
  + AG:
* True => all children’s “Version” numbers are the same
* False => all children’s “Version” numbers are not the same

When there is a change in the data plane status (i.e. uPA is notified by its NRM, or AG notified by one or more of its children), the Coordinator will send up the workflow tree a “DataPlaneStateChange.nt” notification with the updated Activate, Version, and VersionConsistent values.

For the AG, reporting the aggregate data plane state of its children requires some processing. The following pseudo-code describes this behavior:

if all of ChildrenDataPlaneStatus[1..n].Active are true then

{

DataPlaneStatus.Active = true

DataPlaneStatus.Version =

maximum\_of(ChildrenDataPlaneStatus[1..n].Version)

If all ChildrenDataPlaneStatus[1..n].Version are the same, and

all of ChildrenDataPlaneStatus[1..n].VersionCosistent are true then

{

DataPlaneStatus.VersionConsistent = true

}

else

{

DataPlaneStatus.VersionConsistent = false

}

}

# Security

This section describes how NSI CS protocol achieves secure communication and provides authentication data across requests. The basic setup is to use TLS between NSAs and SAML attributes to convey information regarding request authentication.

## Transport Layer Security

TLS is used to ensure secure communication between NSAs and X.509 certificates for authentication. Trust between NSAs is pairwise and must be established out-of-band. It is possible to have unidirectional trust between NSAs, i.e. reservations can only be created in one direction, as this is simply a policy special case. Transitive trust between NSAs cannot be assumed, i.e., NSAs A & B trust each other, and B & C trust each other, but this does not imply trust between A & C. However a request from A may end up using resources from C if passed through B. In the current security framework, B (if it’s policies permit) can proxy A’s request to C. From C’s point of view, it receives the request from B, and authenticates and authorizes the request using B’s credentials. This document does not describe security policies, as these will always be site specific. Note that due to the requirement for direct NSA to NSA communications (i.e. NSAs cannot forward communications via a third party NSA), message-level signing provides little value and is not used.

TLS provides message integrity, confidentiality, protects against replay attacks, and authentication via the X.509 certificates. Authorization is done at the NSAs application level. TLS version 1.0 must be supported. NSAs are free to use SSLv3 and TLS version following 1.0 if possible.

## SAML Assertions

As TLS by design only provides transport level security, an additional mechanism for conveying request authentication is required. For this, SAML assertions are used. NSAs can include SAML assertions in the CS message header, which providers can use to authorize the request. SAML attributes can describe information such as user, group, originating NSA, or even OAuth tokens. What and how to describe with SAML headers is outside the scope of this document, but will be described in a best current practices (BCP) document. The intent of such a document is to provide a baseline of what to support, but attributes can be created as needed and can be unique to NSA peerings.

# Contributors

Chin Guok, ESnet

Jeroen van der Ham, University of Amsterdam

Radek Krzywania, PSNC

Tomohiro Kudoh, AIST

John MacAuley, SURFnet

Takahiro Miyamoto, KDDI R&D Laboratories

Inder Monga, ESnet

Guy Roberts, DANTE

Jerry Sobieski, NORDUnet

Henrik Thostrup Jensen, NORDUnet

# Glossary

|  |  |
| --- | --- |
| Activate | When provisioning of a Connection has been completed the Connection is considered to be Active. A notification is sent to the RA informing them that the Connection is Active. |
| Aggregator (AG) | The Aggregator is an NSA th has more than one child NSA, and has the responsibility of aggregating the responses from each child NSA. |
| Connection | A Connection is an NSI construct that identifies the physical instance of a circuit in the data plane. A Connection has a set of properties (for instance, Connection identifier, ingress and egress STPs, capacity, or start time). Connections can be either unidirectional or bidirectional. |
| Connection Service | A Connection Service is a service that allows a Requester NSA to request and manage a Connection from a Provider NSA |
| Connection Service Protocol | A Connection Service Protocol is the protocol that describes the messages and associated attributes that are exchanged between RA and PA |
| Control and Management Planes | The Control Plane and/or Management Plane are not defined in this document, but follow common usage |
| Coordinator | The Coordinator is a set of functions that XXX |
| Data Plane | The Data Plane refers to the infrastructure that carries the physical instance of the Connection, e.g. the Ethernet switches that deliver the circuit. |
| Edge Point | A network resource that resides at the boundary of an intra-network topology, this may include for example a connector on a distribution frame, a port on an Ethernet switch, or a connector at the end of a fibre. |
| Inter-Network Topology | This is a topological description of a set of Networks and their transfer functions, and the connectivity between Networks |
| Lifecycle State Machine (LSM) | The LSM allows messages relating to terminating a Connection to be to send and received. |
| Message Transport Layer (MTL) | The MTL delivers an abstracted message delivery mechanism to the NSI layer. |
| Network | A Network is an Inter-Network topology object that describes a set of STPs with a Transfer Function between STPs |
| Network Resource Manager (NRM) | The Network Resource Manager owns a set of transport resources and has ultimate responsibility for authorizing and managing the use of these resources. Each NRM is always associated with a single NSA |
| Network Services | Network Services are the full set of services offered by an NSA. Each NSA will support one or more Network Services |
| Network Service Agent (NSA) | The Network Service Agent is a concrete piece of software that sends and receives NSI Messages. The NSA includes a set of capabilities that allow Network Services to be delivered. |
| Network Service Interface (NSI) | The NSI is the interface between Requester NSAs and Provider NSAs. The NSI defines a set of interactions or transactions between these NSAs to realize a Network Service |
| Network Services Framework (NSF) | The Network Services framework describes a NSI message based platform capable of supporting a suite of Network Services such as the Connection Service and the Topology Service |
| NSI Message | A NSI Message is a structured unit of data sent between a Requester NSA and a Provider NSA. |
| Path | A Path is an ordered list of STPs that are used a constraint in Inter-Network path finding. |
| Provision | Provisioning is the process of requesting the creation of the physical instance of a Connection in the data plane |
| Provision State Machine (PSM) | The Provision State Machine is a simple state machine which transits between the Provisioned and the Released state |
| Release | Releasing is the process of de-provisioning resources on the data-plane. When a Connection is Released on the data-plane, the Reservation is retained. |
| Requester/ Provider NSA (RA/PA) | An NSA acts in one of two possible roles relative to a particular instance of an NSI. When an NSA requests a service, it is called a Requester Agent (RA). When an NSA realizes a service, it is called a Provider Agent (PA). A particular NSA may act in different roles at different interfaces |
| Service Demarcation Point (SDP) | Service Demarcation Points (SDPs) are an NSI topology objects that identify a grouping of two Edge Points at the boundary between two Networks. |
| Service Termination Point (STP) | Service Termination Points (STPs) are an NSI topology objects that identify the Edge Points of a Network in the intra-network topology. |
| Service Plane | The Service Plane is a plane in which services are requested and managed; these services include the Network Service. The Service Plane contains a set of Network Service Agents communicating using Network Service Interfaces |
| Simple Object Access Protocol (SOAP) | SOAP is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks |
| Reservation State Machine (RSM) | The Reservation State Machine state machine defines the sequence of operation of messages for creating or modifying a reservation |
| Reserve | When a Provider Agent receives (and then confirms) a Connection Reservation request the Provider Agent then holds the resources needed by the Connection. |
| Terminate | Terminating is the process which will completely remove a Reservation and Release any associated Connections. This term has a formal definition in the CS state-machine |
| Ultimate PA (uPA) | The ultimate PA is a Provider Agent that has an associated NRM. |
| Ultimate RA (uRA) | The Ultimate RA is a Requester Agent is the originator of a service request |
| XML Schema Definition (XSD) | XSD is a schema language for XML |
| eXtensible Markup Language (XML) | XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. |

# Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

# Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

# Full Copyright Notice

Copyright (C) Open Grid Forum (2008-2012). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

# Appendix A: XML Schema Definitions

The NSI-CS 2.0 protocol is defined using XML schema (XSD) to describe the common message header and individual connection services operation elements and types. Web Service Description Language (WSDL) is used to describe the interface or operation bindings, capturing the request, response, and error (fault) interactions. Finally, WSDL is once again used to provide a SOAP specific transport binding as a reference specification; however, the XML schema definitions can be utilized to encapsulate the NCI-CS protocol into other transport bindings. This appendix provides a detailed overview of these NSI-CS XML schema definitions.

The following individual namespaces are defined as part of the NSI-CS 2.0 protocol:

|  |  |
| --- | --- |
| Description | Namespace URL |
| Common types shared between NSI message header and CS operation definitions. | http://schemas.ogf.org/nsi/2013/04/framework/types |
| NSI message header definition. | http://schemas.ogf.org/nsi/2013/04/framework/headers |
| NSI CS operation specific type definitions. | http://schemas.ogf.org/nsi/2013/04/connection/types |
| NSI CS operation definitions | http://schemas.ogf.org/nsi/2013/04/connection/interface |
| Provider NSA interface SOAP binding | http://schemas.ogf.org/nsi/2013/04/connection/provider |
| Requester NSA interface SOAP binding | http://schemas.ogf.org/nsi/2013/04/connection/requester |

This documentation is based on revision 66 of the NSI scheme as maintained in the NSI-CS code.google.com repository https://code.google.com/p/ogf-nsi-project/.

## Use of SOAP

The NSI CS protocol is specified using WSDL 1.1 and utilizes the SOAP 1.1 message encoding as identified by the namespaces:

* **soap** - "http://schemas.xmlsoap.org/soap/envelope/"
* **xsi** - "http://www.w3.org/2001/XMLSchema-instance"
* **xsd** - "http://www.w3.org/2001/XMLSchema"
* **soapenc** - "http://schemas.xmlsoap.org/soap/encoding/"
* **wsdl** - "http://schemas.xmlsoap.org/wsdl/"
* **soapbind** - "http://schemas.xmlsoap.org/wsdl/soap/"

We identify the specific NSI CS operation being invoked using the “Soapaction:” element in the HTTP header as per section 6.1.1 of “Simple Object Access Protocol (SOAP) 1.1” found at <http://www.w3.org/TR/SOAP>. This was done to allow for better compatibility between SOAP implementations even though it is not explicitly required as per WS-I Basic Profile 1.1<http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html>. The NSI header as defined with XML schema does not contain a specific “operation” element as this is included in the “Soapaction:” element and would be duplicate data.

## nsiHeader element

***Namespace definition:*** http://schemas.ogf.org/nsi/2013/04/framework/headers

The **nsiHeader** element contains attributes common to all NSI-CS operations, and therefore, is sent as part of every NSI-CS message exchange. Attributes included in the header provide protocol versioning, basic message routing for the protocol, and user security infrastructure. For the SOAP protocol binding, the **nsiHeader** element is encapsulated in the SOAP header, while the NSI specific operation is encapsulated in the SOAP body.

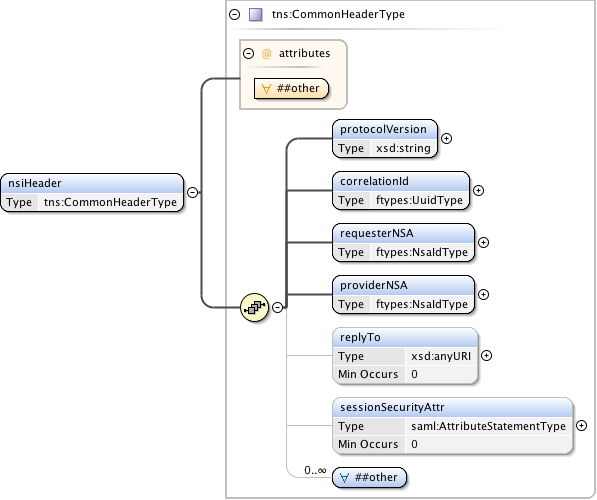


Figure – nsiHeader structure.

Parameters

The **nsiHeader** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| protocolVersion | M | A string identifying the specific protocol version carried in this NSI message. The protocol version is modeled separately from the namespace of the WSDL and XML schema to capture behavioral changes that cannot be modeled in schema definition, and to avoid updating of the schema namespace. |
| correlationId | M | An identifier provided by the requesting NSA used to correlate to an asynchronous response from the responder. It is recommended that a Universally Unique Identifier (UUID) URN as per IETF RFC 4122 be used as a globally unique value. |
| requesterNSA | M | The NSA identifier for the NSA acting in the Requester Agent role for the specific NSI operation. |
| providerNSA | M | The NSA identifier for the NSA acting in the Provider Agent role for the specific NSI operation. |
| replyTo | O | The Requester NSA's SOAP endpoint address to which asynchronous messages associated with this operation request will be delivered. This is only populated for the original operation request (reserve, provision, release, terminate, and query), and not for any additional messaging associated with the operation. If no endpoint value is provided in an operation request, then it is assumed the requester is not interested in a response and will use alternative mechanism to determine the result (i.e. polling using query). |
| sessionSecurityAttr | O | Security attributes associated with the end user's NSI session. This field can be used to perform authentication, authorization, and policy enforcement of end user requests. It is only provided in the operation request (reserve, provision, release, terminate, and query), and not for any additional messaging associated with the operation. |
| any element and anyAttribute | O | Provides a flexible mechanism allowing additional elements in the protocol header for exchange between two-peered NSA. Use of this element field is beyond the current scope of this NSI specification, but may be used in the future to extend the existing protocol without requiring a schema change. Additionally, the field can be used between peered NSA to provide additional context not covered in the existing specification, however, this is left up to specific peering agreements. |

Figure – sessionSecurityAttr type.

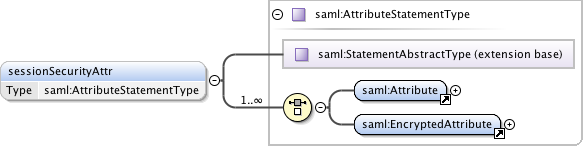
The following table describes each message and it’s use of the individual header parameters. The “Soapaction” parameter identified in the last column of the table is carried in the HTTP request attributes and not the NSI specific header.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | *Header parameters* | | | | | | | |
| M = Mandatory  O = Optional  N/A = Not Applicable | | | *protocolVersion* | *correlationId* | *requesterNSA* | *providerNSA* | *replyTo* | *sessionSecurityAttr* | *other* | *Soapaction* |
|  | | reserve | M | M | M | M | O | M | O | M |
|  | | reserveResponse | M | M | M | M | N/A | N/A | O | N/A |
|  | | reserveConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | reserveConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | reserveFailed | M | M | M | M | N/A | N/A | O | M |
|  | | reserveFailedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | |  |  |  |  |  |  |  |  |  |
|  | | reserveCommit | M | M | M | M | O | M | O | M |
|  | | reserveCommitACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | reserveCommitConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | reserveCommitConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | reserveCommitFailed | M | M | M | M | N/A | N/A | O | M |
|  | | reserveCommitFailedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | |  |  |  |  |  |  |  |  |  |
|  | | reserveAbort | M | M | M | M | O | M | O | M |
|  | | reserveAbortACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | reserveAbortConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | reserveAbortConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | |  |  |  |  |  |  |  |  |  |
|  | | provision | M | M | M | M | O | M | O | M |
|  | | provisionACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | provisionConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | provisionConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
| Messaging  Primitives | |  |  |  |  |  |  |  |  |  |
| release | M | M | M | M | O | M | O | M |
| releaseACK | M | M | M | M | N/A | N/A | O | N/A |
| releaseConfirmed | M | M | M | M | N/A | N/A | O | M |
| releaseConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
|  |  |  |  |  |  |  |  |  |
|  | | terminate | M | M | M | M | O | M | O | M |
|  | | terminateACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | terminateConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | terminateConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | |  |  |  |  |  |  |  |  |  |
|  | | querySummary | M | M | M | M | M | M | O | M |
|  | | querySummaryACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | querySummaryConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | querySummaryConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | querySummaryFailed | M | M | M | M | N/A | N/A | O | M |
|  | | querySummaryFailedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | |  |  |  |  |  |  |  |  |  |
|  | | queryRecursive | M | M | M | M | M | M | O | M |
|  | | queryRecursiveACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | queryRecursiveConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | queryRecursiveConfirmedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | queryRecursiveFailed | M | M | M | M | N/A | N/A | O | M |
|  | | queryRecursiveFailedACK | M | M | M | M | N/A | N/A | O | N/A |
|  | |  |  |  |  |  |  |  |  |  |
|  | | querySummarySync | M | M | M | M | N/A | M | O | M |
|  | | querySummarySyncConfirmed | M | M | M | M | N/A | N/A | O | M |
|  | | querySummarySyncFailed | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
|  | |  |  |  |  |  |  |  |  |  |
|  | | errorEvent | M | M | M | M | N/A | N/A | O | M |
|  | | errorEventACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | reserveTimeout | M | M | M | M | N/A | N/A | O | M |
|  | | reserveTimeoutACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | dataPlaneStateChange | M | M | M | M | N/A | N/A | O | M |
|  | | dataPlaneStateChangeACK | M | M | M | M | N/A | N/A | O | N/A |
|  | | messageDeliveryTimeout | M | M | M | M | N/A | N/A | O | M |
|  | | messageDeliveryTimeoutACK | M | M | M | M | N/A | N/A | O | N/A |

Table – NSI CS message use of header fields

### *sessionSecurityAttr type*

The *sessionSecurityAttr* element is defined using a standardized SAML *AtttributeStatementType* imported from the SAML namespace *urn:oasis:names:tc:SAML:2.0:assertion*.



## Common types

***Namespace definition:*** http://schemas.ogf.org/nsi/2013/04/framework/types

These are the common types shared between NSI message header and CS operation definitions.

### ServiceExceptionType

Common service exception used for SOAP faults and operation Failed messages.

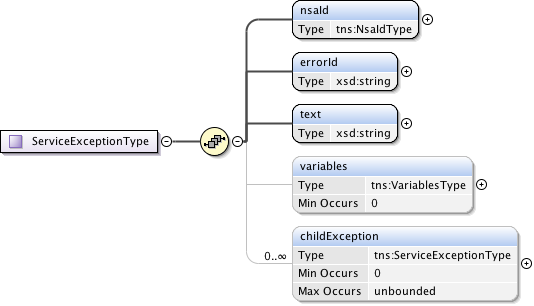


Figure – ServiceExceptionType type.

Parameters

The **ServiceExceptionType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| nsaId | M | NSA that generated the service exception. |
| errorId | M | Error identifier uniquely identifying each known fault within the protocol. |
| text | M | User-friendly message text describing the error. |
| providerNSA | M | The NSA identifier for the NSA acting in the Provider Agent role for the specific NSI operation. |
| variables | O | An optional collection of type/value pairs providing additional information relating to the error. |
| childException | O | Hierarchical list of service exceptions capturing failures within the request tree. |

### NsaIdType

NsaIdType is a specific type for a Network Services Agent (NSA) identifier that is populated with a OGF URN (reference artifact 6478 "Procedure for Registration of Subnamespace Identifiers in the URN:OGF Hierarchy") to be used for compatibility with other external systems.



Figure – NsaIdType type.

### VariablesType

A type definition providing a set of zero or more type/value variables used for modeling generic attributes.

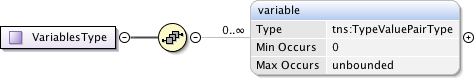


Figure – NsaIdType type.

Parameters

The **VariablesType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| variable | O | The variable containing the type/values. |

### TypeValuePairType

TypeValuePairType is a simple type and multi-value tuple. Includes simple string type and value, as well as more advanced extensions if needed. A targetNamespace attribute is included to provide additional context where needed.

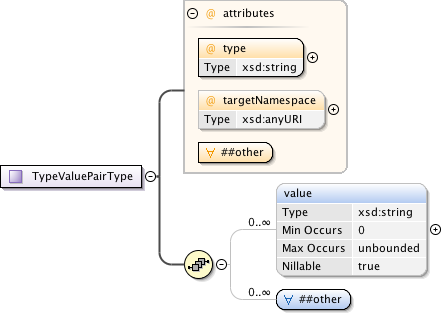


Figure – TypeValuePairType type.

Parameters

The **TypeValuePairType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| type | M | A string representing the name of the type. |
| targetNamespace | O | An optional URL to qualify the name space of the capability. |
| anyAttribute |  | Provides a flexible mechanism allowing additional attributes non-specified to be provided as needed for peer-to-peer NSA communications. Use of this attribute field is beyond the current scope of this NSI specification, but may be used in the future to extend the existing protocol without requiring a schema change. |
| value | O | A string value corresponding to type. |
| any | O | Provides a flexible mechanism allowing additional elements to be provided as an alternative, or in combination with value. Use of this element field is beyond the current scope of this NSI specification, but may be used in the future to extend the existing protocol without requiring a schema change. |

### TypeValuePairListType

A simple holder type providing a list definition for the attribute type/values structure.

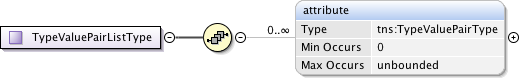


Figure – TypeValuePairListType type.

Parameters

The **TypeValuePairListType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| attribute | O | An instance of a type/value structure. |

### UuidType

Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 | ISO/IEC 9834-8:2005 and IETF RFC 4122. Values must correspond to the following pattern “urn:uuid:[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}”.



Figure – TypeValuePairListType type.

### DateTimeType

The time zone support of W3C XML Schema is quite controversial and needs some additional constraints to avoid comparison problems. These patterns can be kept relatively simple since the syntax of the datetime is already checked by the schema validator and only simple additional checks need to be added. This type definition checks that the time part ends with a "Z" or contains a sign. Values must correspond to the following pattern ".+T.+(Z|[+-].+)"



Figure – TypeValuePairListType type.

## NSI-CS operation specific type definitions.

***Namespace definition:*** <http://schemas.ogf.org/nsi/2013/04/connection/types>

### Reserve message elements

The **reserve** message is sent from a Requester NSA to a Provider NSA when a new reservation is being requested, or a modification to an existing reservation is required. The **reserveResponse** indicates that the Provider NSA has accepted the reservation request for processing and has assigned it the returned *connectionId*. A **reserveConfirmed** or **reserveFailed** message will be sent asynchronously to the Requester NSA when reserve operation has completed processing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Direction | Input | Output | Fault |
| Request | RA to PA | reserve | reserveResponse | serviceException |
| Confirmed | PA to RA | reserveConfirmed | reserveConfirmedACK | serviceException |
| Failed | PA to RA | reserveFailed | reserveFailedACK | serviceException |

#### Request: reserve

The NSI CS **reserve** message allows a Requester NSA to reserve network resources for a connection between two STP's within the network constrained by the provided service parameters. This **reserve** message allows a Requester NSA to check the feasibility of a connection reservation, or modification an existing connection reservation. Any resources associated with the reservation or modification operation will be allocated and held until a **reserveCommit** message is received for the reservation or timeout occurs (whichever arrives first).

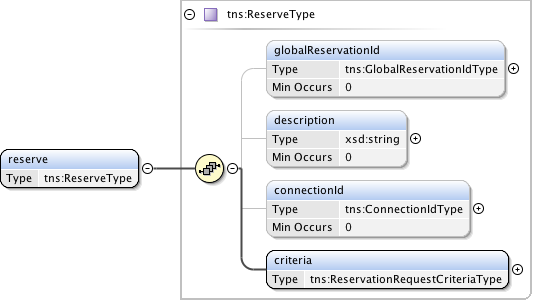


Figure – reserve request message structure.

Parameters

The **reserve** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| globalReservationId | An optional global reservation id that can be used to correlate individual related service reservations through the network. This must be populated with a Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 |ISO/IEC 9834-8:2005 and IETF RFC 4122. |
| description | An optional description for the service reservation. |
| connectionId | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. Provided in reserve request only when an existing reservation is being modified. |
| criteria | Reservation request criteria including start and end time, service attributes, and requested path for the service. |

Response

If the **reserve** operation is successful, a **reserveResponse** message is returned, otherwise a **serviceException** is returned. A Provider NSA sends this **reserveResponse** message immediately after receiving the reservation request to inform the Requester NSA of the *connectionId* allocated to their reservation request. This *connectionId* can then be used to query reservation progress.

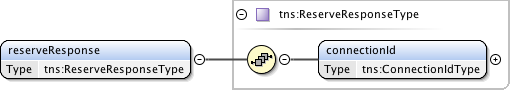


Figure – reserveResponse message structure.

The **reserveResponse** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The Provider NSA assigned *connectionId* for this reservation request. This value will be unique within the context of the Provider NSA. |

#### Confirmation: reserveConfirmed

A Provider NSA sends this positive **reserveConfirmed** response message to the Requester NSA that issued the original reserve request message. Receipt of this message is an indication that the requested reservation parameters were available and will be held until a **reserveCommit** message is received for the reservation or timeout occurs (whichever arrives first).

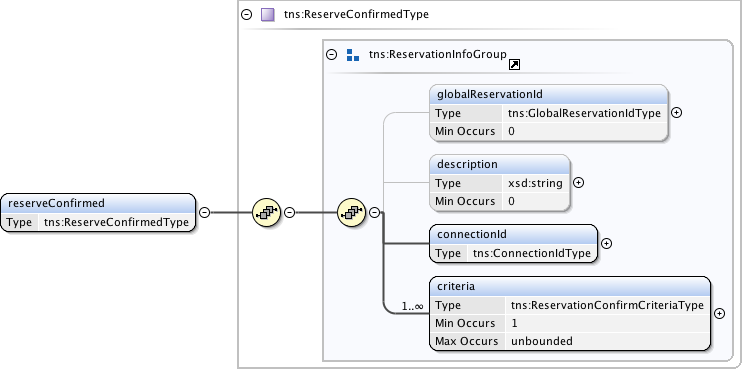


Figure – reserveConfirmed message structure.

Parameters

The **reserveConfirmed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| globalReservationId | An optional global reservation id that can be used to correlate individual related service reservations through the network. This must be populated with a Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 |ISO/IEC 9834-8:2005 and IETF RFC 4122. |
| description | An optional description for the service reservation. |
| connectionId | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. Provided in reserve request only when an existing reservation is being modified. |
| criteria | A set of versioned and confirmed reservation criteria information including start and end time, service attributes, and requested path for the service. |

Response

If the **reserveConfirmed** operation is successful, a **reserveConfirmedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **reserveConfirmedACK** message immediately after receiving the **reserveConfirmed** request to acknowledge to the Provider NSA the **reserveConfirmed** request has been accepted for processing. The **reserveConfirmedACK** message is implemented using the generic acknowledgement message.



Figure – reserveConfirmedACK message structure.

The **reserveConfirmedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Failed: reserveFailed

A Provider NSA sends this negative **reserveFailed** response to the Requester NSA that issued the original reservation request message if the requested reservation criteria could not be met. This message is also sent in response to a reserve request for a modification to an existing schedule if the required modification is not possible.

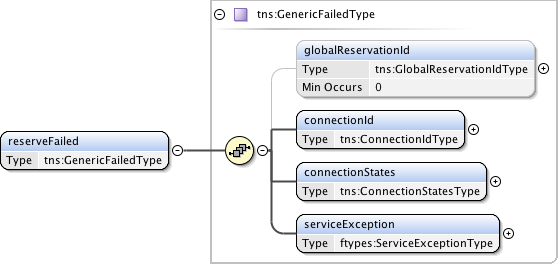


Figure – reserveFailed message structure.

Parameters

The **reserveFailed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| globalReservationId | An optional global reservation id that was originally provided in the reserve request. |
| connectionId | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |
| connectionStates | Overall connection state for the reservation. |
| serviceException | Specific error condition indicating the reason for the failure. |

Response

If the **reserveFailed** operation is successful, a **reserveFailedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **reserveFailedACK** message immediately after receiving the **reserveFailed** request to acknowledge to the Provider NSA the **reserveFailed** request has been accepted for processing. The **reserveFailedACK** message is implemented using the generic acknowledgement message.



Figure – reserveFailedACK message structure.

The **reserveFailedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

### ReserveCommit message elements

The **reserveCommit** message is sent from a Requester NSA to a Provider NSA when a reservation or modification to an existing reservation is being committed. This reservation must currently reside in the Reserve Held state for this operation to be accepted. The **reserveCommitACK** indicates that the Provider NSA has accepted the modify request for processing. A **reserveCommitConfirmed** or **reserveCommitFailed** message will be sent asynchronously to the Requester NSA when reserve or modify processing has completed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Direction | Input | Output | Fault |
| Request | RA to PA | reserveCommit | reserveCommitACK | serviceException |
| Confirmed | PA to RA | reserveCommitConfirmed | reserveCommitConfirmedACK | serviceException |
| Failed | PA to RA | reserveCommitFailed | reserveCommitFailedACK | serviceException |

#### Request: reserveCommit

The NSI CS **reserveCommit** message allows a Requester NSA to commit a previously allocated reservation or modification on a reservation. The **reserveCommit** request must arrive at the Provider Agent before the reservation timeout occurs.

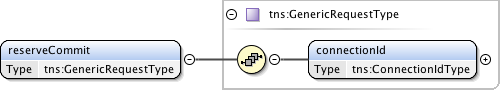


Figure – reserveCommit request message structure.

Parameters

The **reserveCommit** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The Provider NSA assigned connectionId for the reservation that is to be committed. |

Response

If the **reserveCommit** operation is successful, a **reserveCommitACK** message is returned, otherwise a **serviceException** is returned. A Provider NSA sends this **reserveCommitACK** message immediately after receiving the **reserveCommit** request to acknowledge to the Requester NSA the **reserveCommit** request has been accepted for processing. The **reserveCommitACK** message is implemented using the generic acknowledgement message.



Figure – reserveCommitACK message structure.

The **reserveCommitACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Confirmation: reserveCommitConfirmed

This **reserveCommitConfirmed** message is sent from a Provider NSA to Requester NSA as an indication of a successful **reserveCommit** request for a reservation previously in a Reserve Held state.

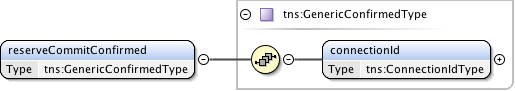


Figure – reserveCommitConfirmed message structure.

Parameters

The **reserveCommitConfirmed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The connection identifier for the reservation that was committed. |

Response

If the **reserveCommitConfirmed** operation is successful, a **reserveCommitConfirmedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **reserveCommitConfirmedACK** message immediately after receiving the **reserveCommitConfirmed** request to acknowledge to the Provider NSA the **reserveCommitConfirmed** request has been accepted for processing. The **reserveCommitConfirmedACK** message is implemented using the generic acknowledgement message.



Figure – reserveAbortConfirmedACK message structure.

The **reserveCommitConfirmedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Failed: reserveCommitFailed

This **reserveCommitFailed** message is sent from a Provider NSA to Requester NSA as an indication of a **reserve** (or modify) commit failure. This is in response to an original **reserveCommit** request from the associated Requester NSA.

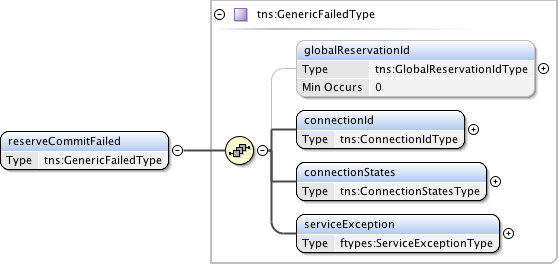


Figure – reserveCommitFailed message structure.

Parameters

The **reserveCommitFailed** message takes the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| globalReservationId | An optional global reservation id that was originally provided in the reserve request. |
| connectionId | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |
| connectionStates | Overall connection state for the reservation. |
| serviceException | Specific error condition indicating the reason for the failure. |

Response

If the **reserveCommitFailed** operation is successful, a **reserveCommitFailedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **reserveCommitFailedACK** message immediately after receiving the **reserveCommitFailed** request to acknowledge to the Provider NSA the **reserveCommitFailed** request has been accepted for processing. The **reserveCommitFailedACK** message is implemented using the generic acknowledgement message.



Figure – reserveCommitFailedACK message structure.

The **reserveCommitFailedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

### reserveAbort message elements

The **reserveAbort** message is sent from a Requester NSA to a Provider NSA when an initial reservation request, or modification to an existing reservation is to be aborted, and the reservation state machine returned to the previous version of the reservation. The **reserveAbortACK** indicates that the Provider NSA has accepted the abort request for processing. A **reserveAbortConfirmed** message will be sent asynchronously to the Requester NSA when the abort processing has completed. There is no associated Failed message for this operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Direction | Input | Output | Fault |
| Request | RA to PA | reserveAbort | reserveAbortACK | serviceException |
| Confirmed | PA to RA | reserveAbortConfirmed | reserveAbortConfirmedACK | serviceException |
| Failed | N/A | N/A | N/A | N/A |

#### Request: reserveAbort

The NSI CS **reserveAbort** message allows a Requester NSA to abort a previously requested reservation or modification on a reservation.

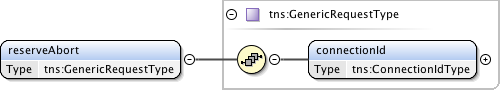


Figure – reserveAbort request message structure.

Parameters

The **reserveAbort** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The Provider NSA assigned connectionId for the reservation or modification that is to be aborted. |

Response

If the **reserveAbort** operation is successful, a **reserveAbortACK** message is returned, otherwise a **serviceException** is returned. A Provider NSA sends this **reserveAbortACK** message immediately after receiving the **reserveAbort** request to acknowledge to the Requester NSA the **reserveAbort** request has been accepted for processing. The **reserveAbortACK** message is implemented using the generic acknowledgement message.



Figure – reserveAbortACK message structure.

The **reserveAbortACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Confirmation: reserveAbortConfirmed

This **reserveAbortConfirmed** message is sent from a Provider NSA to Requester NSA as an indication of a successful **reserveAbort** request. The reservation in question will have any pending modifications cancelled and returned to the reservation state existing before the modification.

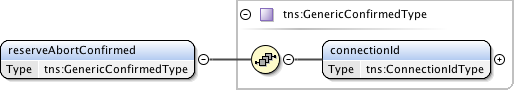


Figure – reserveAbortConfirmed message structure.

Parameters

The **reserveAbortConfirmed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The connection identifier for the reservation that was aborted. |

Response

If the **reserveAbortConfirmed** operation is successful, a **reserveAbortConfirmedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **reserveAbortConfirmedACK** message immediately after receiving the **reserveAbortConfirmed** request to acknowledge to the Provider NSA the **reserveAbortConfirmed** request has been accepted for processing. The **reserveAbortConfirmedACK** message is implemented using the generic acknowledgement message.



Figure – reserveAbortConfirmedACK message structure.

The **reserveAbortConfirmedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

### provision message elements

The **provision** message is sent from a Requester NSA to a Provider NSA when an existing reservation is to be transitioned into a provisioned state. The **provisionACK** indicates that the Provider NSA has accepted the **provision** request for processing. A **provisionConfirmed** or message will be sent asynchronously to the Requester NSA when **provision** processing has completed. There is no associated Failed message for this operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Direction | Input | Output | Fault |
| Request | RA to PA | provision | provisionACK | serviceException |
| Confirmed | PA to RA | provisionConfirmed | provisionConfirmedACK | serviceException |
| Failed | N/A | N/A | N/A | N/A |

#### Request: provision

The NSI CS **provision** message allows a Requester NSA to transition a previously requested reservation into a provisioned state. A reservation in a provisioned state will activate associated data plane resources during the scheduled reservation time.

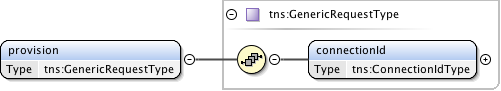


Figure – provision request message structure.

Parameters

The **provision** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The Provider NSA assigned connectionId for the reservation to be provisioned. |

Response

If the **provision** operation is successful, a **provisionACK** message is returned, otherwise a **serviceException** is returned. A Provider NSA sends this **provisionACK** message immediately after receiving the **provision** request to acknowledge to the Requester NSA the **provision** request has been accepted for processing. The **provisionACK** message is implemented using the generic acknowledgement message.



Figure – provisionACK message structure.

The **provisionACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Confirmation: provisionConfirmed

This **provisionConfirmed** message is sent from a Provider NSA to Requester NSA as an indication of a successful **provision** request. This is in response to an original **provision** request from the associated Requester NSA.

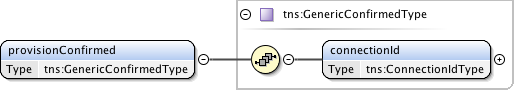


Figure – provisionConfirmed message structure.

Parameters

The **provisionConfirmed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The connection identifier for the reservation that was provisioned. |

Response

If the **provisionConfirmed** operation is successful, a **provisionConfirmedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **provisionConfirmedACK** message immediately after receiving the **provisionConfirmed** request to acknowledge to the Provider NSA the **provisionConfirmed** request has been accepted for processing. The **provisionConfirmedACK** message is implemented using the generic acknowledgement message.



Figure – provisionConfirmedACK message structure.

The **provisionConfirmedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

### release message elements

The **release** message is sent from a Requester NSA to a Provider NSA when an existing reservation is to be transitioned into a released state. The **releaseACK** indicates that the Provider NSA has accepted the release request for processing. A **releaseConfirmed** message will be sent asynchronously to the Requester NSA when release processing has completed. There is no associated Failed message for this operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Direction | Input | Output | Fault |
| Request | RA to PA | release | releaseACK | serviceException |
| Confirmed | PA to RA | releaseConfirmed | releaseConfirmedACK | serviceException |
| Failed | N/A | N/A | N/A | N/A |

#### Request: release

The NSI CS **release** message allows a Requester NSA to transition a previously requested reservation into a released state. A reservation in a released state will deactivate associated data plane resources.

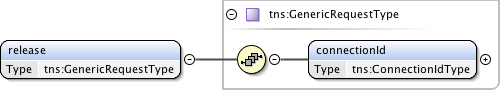


Figure – release request message structure.

Parameters

The **release** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The Provider NSA assigned connectionId for the reservation to be released. |

Response

If the **release** operation is successful, a **releaseACK** message is returned, otherwise a **serviceException** is returned. A Provider NSA sends this **releaseACK** message immediately after receiving the **release** request to acknowledge to the Requester NSA the **release** request has been accepted for processing. The **releaseACK** message is implemented using the generic acknowledgement message.



Figure – releaseACK message structure.

The **releaseACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Confirmation: releaseConfirmed

This **releaseConfirmed** message is sent from a Provider NSA to Requester NSA as an indication of a successful **release** request. This is in response to an original **release** request from the associated Requester NSA.

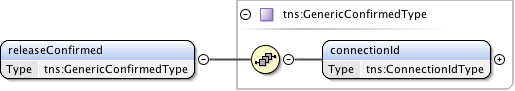


Figure – releaseConfirmed message structure.

**Parameters**

The **releaseConfirmed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The connection identifier for the reservation that was released. |

Response

If the **releaseConfirmed** operation is successful, a **releaseConfirmedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **releaseConfirmedACK** message immediately after receiving the **releaseConfirmed** request to acknowledge to the Provider NSA the **releaseConfirmed** request has been accepted for processing. The **releaseConfirmedACK** message is implemented using the generic acknowledgement message.



Figure – releaseConfirmedACK message structure.

The **releaseConfirmedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

### terminate message elements

The **terminate** message is sent from a Requester NSA to a Provider NSA when an existing reservation is to be transitioned into a terminated state and all associated resources in the network are freed. The **terminateACK** indicates that the Provider NSA has accepted the **terminate** request for processing. A **terminateConfirmed** message will be sent asynchronously to the Requester NSA when **terminate** processing has completed. There is no associated Failed message for this operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Direction | Input | Output | Fault |
| Request | RA to PA | terminate | terminateACK | serviceException |
| Confirmed | PA to RA | terminateConfirmed | terminateConfirmedACK | serviceException |
| Failed | N/A | N/A | N/A | N/A |

#### Request: terminate

The NSI CS **terminate** message allows a Requester NSA to transition a previously requested reservation into a terminated state. A reservation in a terminated state will release associated resources.

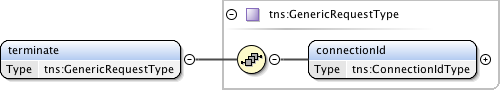


Figure – terminate request message structure.

Parameters

The **terminate** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The Provider NSA assigned connectionId for the reservation to be terminated. |

Response

If the **terminate** operation is successful, a **terminateACK** message is returned, otherwise a **serviceException** is returned. A Provider NSA sends this **terminateACK** message immediately after receiving the **terminate** request to acknowledge to the Requester NSA the **terminate** request has been accepted for processing. The **terminateACK** message is implemented using the generic acknowledgement message.



Figure – terminateACK message structure.

The **terminateACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Confirmation: terminateConfirmed

This **terminateConfirmed** message is sent from a Provider NSA to Requester NSA as an indication of a successful **terminate** request. This is in response to an original **terminate** request from the associated Requester NSA.

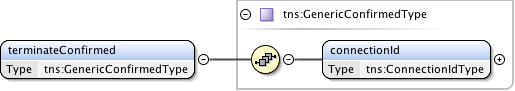


Figure – terminateConfirmed message structure.

Parameters

The **terminateConfirmed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The connection identifier for the reservation that was terminated. |

Response

If the **terminateConfirmed** operation is successful, a **terminateConfirmedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **terminateConfirmedACK** message immediately after receiving the **terminateConfirmed** request to acknowledge to the Provider NSA the **terminateConfirmed** request has been accepted for processing. The **terminateConfirmedACK** message is implemented using the generic acknowledgement message.



Figure – terminateConfirmedACK message structure.

The **terminateConfirmedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

-----------------------

### querySummary message elements

The **querySummary** message is sent from a Requester NSA to a Provider NSA to determine the status of existing reservations. The **querySummaryACK** indicates that the target NSA has accepted the **querySummary** request for processing. A **querySummaryConfirmed** or **querySummaryFailed** message will be sent asynchronously to the requesting NSA when **querySummary** processing has completed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Direction | Input | Output | Fault |
| Request | RA to PA | querySummary | querySummaryACK | serviceException |
| Confirmed | PA to RA | querySummaryConfirmed | querySummaryConfirmedACK | serviceException |
| Failed | PA to RA | querySummaryFailed | querySummaryFailedACK | serviceException |

#### Request: querySummary

The **querySummary** message provides a mechanism for a Requester NSA to query the Provider NSA for a set of connection service reservation instances between the RA-PA pair. This message can also be used as a reservation status polling mechanism.  
   
Elements compose a filter for specifying the reservations to return in response to the **querySummary** request. Querying of reservations can be performed based on *connectionId* or *globalReservationId*. Filter items specified are OR'ed to build the match criteria. If no criteria are specified then all reservations associated with the requesting NSA are returned.

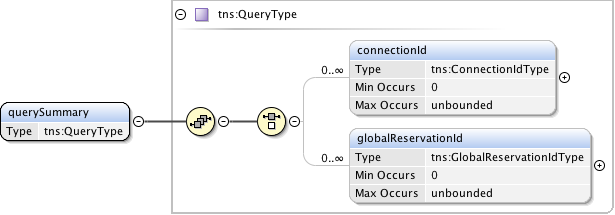


Figure – querySummary request message structure.

Parameters

The **querySummary** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| connectionId | The Provider NSA assigned connectionId for this reservation. Return reservations containing this connectionId. |
| globalReservationId | An optional global reservation id that can be used to correlate individual related service reservations through the network. Return reservations containing this globalReservationId. |

Response

If the **querySummary** operation is successful, a **querySummaryACK** message is returned, otherwise a **serviceException** is returned. A Provider NSA sends this **querySummaryACK** message immediately after receiving the **querySummary** request to acknowledge to the Requester NSA the **querySummary** request has been accepted for processing. The **querySummaryACK** message is implemented using the generic acknowledgement message.



Figure – querySummaryACK message structure.

The **querySummaryACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Confirmation: querySummaryConfirmed

This **querySummaryConfirmed** message is sent from the Provider NSA to Requester NSA as an indication of a successful **querySummary** operation. This is in response to an original **querySummary** request from the associated Requester NSA.

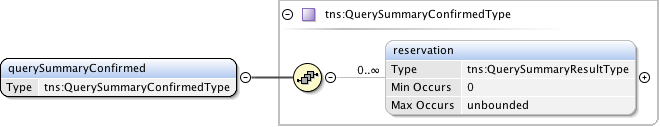


Figure – querySummaryConfirmed message structure.

Parameters

The **querySummaryConfirmed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| reservation | A set of zero or more connection reservations matching the query criteria. If there were no matches to the query then no reservation elements will be present. |

Response

If the **querySummaryConfirmed** operation is successful, a **querySummaryConfirmedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **querySummaryConfirmedACK** message immediately after receiving the **querySummaryConfirmed** request to acknowledge to the Provider NSA the **querySummaryConfirmed** request has been accepted for processing. The **querySummaryConfirmedACK** message is implemented using the generic acknowledgement message.



Figure – reserveConfirmedACK message structure.

The **querySummaryConfirmedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

#### Failed: querySummaryFailed

This **querySummaryFailed** message is sent from the target NSA to requesting NSA as an indication of a **querySummary** operation failure. This is in response to an original **querySummary** request from the associated Requester NSA. It is important to note that a **querySummary** operation that results in no matching reservations does not result in a **querySummaryFailed** message, but instead a **querySummaryConfirmed** with an empty list of reservations.

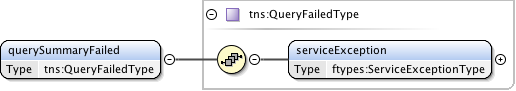


Figure – querySummaryFailed message structure.

Parameters

The **querySummaryFailed** message has the following parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| serviceException | Specific error condition indicating the reason for the failure. |

Response

If the **querySummaryFailed** operation is successful, a **querySummaryFailedACK** message is returned, otherwise a **serviceException** is returned. A Requester NSA sends this **querySummaryFailedACK** message immediately after receiving the **querySummaryFailed** request to acknowledge to the Provider NSA the **querySummaryFailed** request has been accepted for processing. The **querySummaryFailedACK** message is implemented using the generic acknowledgement message.



Figure – querySummaryFailedACK message structure.

The **querySummaryFailedACK** message has no parameters as all relevant information is carried in the NSI CS header structure.

### queryRecursive message elements

### querySummarySync message elements

## NSI-CS specific types

***Namespace definition:*** <http://schemas.ogf.org/nsi/2013/04/connection/types>

This section describes the connection services types used for the CS operation definitions.

### Element Groups

Group elements provide a convenient way to reference a common grouping of elements in-line within a complex type definition without having to specify the individual elements.

#### ReservationInfoGroup

An element group containing the common reservation elements used for complex types referencing reservation information.

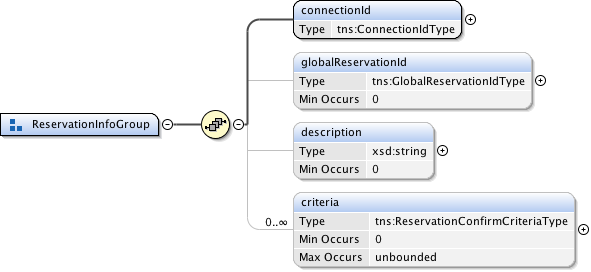


Figure – ReservationInfoGroup type.

Parameters

The **ReservationInfoGroup** type has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |
| globalReservationId | O | An optional global reservation id that can be used to correlate individual related service reservations through the network. This must be populated with a Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 | ISO/IEC 9834-8:2005 and IETF RFC 4122. |
| description | O | An optional description for the service reservation. |
| criteria | O | A set of versioned reservation criteria information. |

### Complex Types

These complex type definitions are utilized by the CS operations and are structures containing other elements and/or attributes. Types are listed in alphabetical order.

#### ChildRecursiveListType

A holder element providing an envelope that will contain the list of child NSA and associated detailed connection information. Utilized by the **QueryRecursiveResultType** to provide a nested list structure of detailed path information.

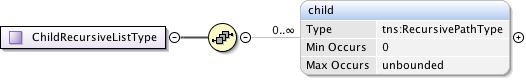


Figure – ChildRecursiveListType.

Parameters

The **ChildRecursiveListType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| child | O | Detailed path information for a child NSA. Each child element is ordered and contains a connection segment in the overall path. |

#### ChildSummaryListType

A holder element containing a list of children NSA and their associated connection information. Utilized by the **QuerySummaryResultType** to provide a nested list structure of summary path information.

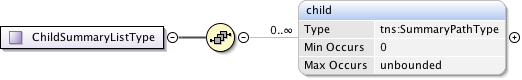


Figure – ChildSummaryListType.

Parameters

The **ChildSummaryListType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| child | O | Summary path information for a child NSA. Each child element is ordered and contains a connection segment in the overall path. |

#### ConnectionStatesType

A holder element containing the state machines associated with a connection reservation.

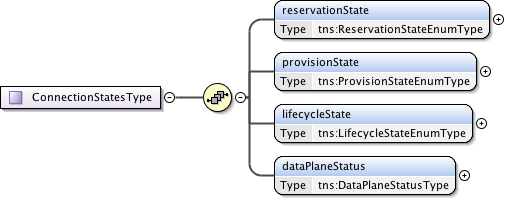


Figure – ConnectionStatesType.

Parameters

The **ConnectionStatesType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| reservationState | M | Models the current connection reservation state. |
| provisionState | M | Models the current connection provisioning state. |
| lifecycleState | M | Models the current connection lifecycle state. |
| dataPlaneStatus | M | Models the current connection data plane activation state. |

#### DataPlaneStateChangeRequestType

Type definition for the data plane state change notification message.  
   
This notification message sent up from a PA when a data plane status has changed. Possible data plane status changes are: activation, deactivation and activation version change.

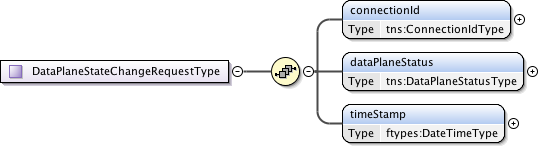


Figure – DataPlaneStateChangeRequestType.

Parameters

The **DataPlaneStateChangeRequestType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The reservation experiencing the data plane state change. |
| dataPlaneStatus | M | Current data plane activation state for the reservation identified by connectionId. |
| timeStamp | M | Time the event was generated. |

#### DataPlaneStatusType

Models the current connection activation state within the data plane.

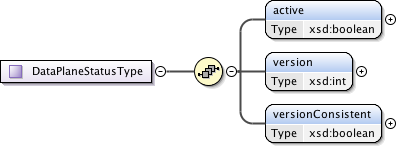


Figure – DataPlaneStatusType.

Parameters

The **DataPlaneStatusType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| active | M | True if the dataplane is active. For an aggregator, this flag is true when data plane is activated in all participating children. |
| version | M | Version of the connection reservation this entry is modeling. |
| versionConsistent | M | Always true for uPA. For an aggregator, if version numbers of all children are the same. This flag is true. This field is valid when Active is true. |

#### ErrorEventType

Type definition for an autonomous message issued from a Provider NSA to a Requester NSA when an existing reservation encounters an autonomous error condition such as being administratively terminated before the reservation's scheduled end-time.

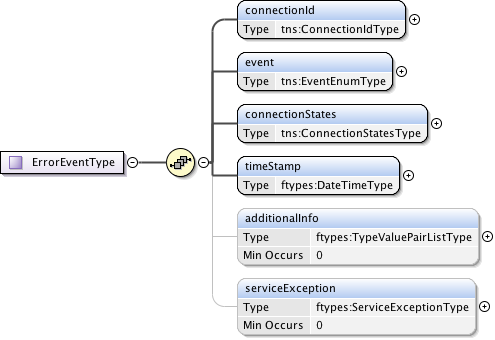


Figure – ErrorEventType.

Parameters

The **ErrorEventType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |
| event | M | The type of event that generated this notification. |
| connectionStates | M | Overall connection states for the reservation. |
| timeStamp | M | Time the event was generated. |
| additionalInfo | O | Type/value pairs that can provide additional error context as needed. |
| serviceException | O | Specific error condition - the reason for the generation of the error event. |

#### GenericAcknowledgmentType

A common acknowledgment message type definition. The correlationId has been moved to the header in CS version 3 so this is now an empty response.



Figure – GenericAcknowledgmentType.

Notes on acknowledgment:

Depending on NSA implementation and thread timing an acknowledgment to a request operation may be returned after the confirm/fail for the request has been returned to the Requesting NSA. For protocol robustness, Requesting NSA should be able to accept confirm/fail before acknowledgment.

#### GenericConfirmedType

This is a generic type definition for a "Confirmed" messages in response to a successful processing of a previous "Request" message such as provision, release, and terminate.



Figure – GenericConfirmedType.

Parameters

The **GenericConfirmedType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation request. This value will be unique within the context of the Provider NSA. |

#### GenericFailedType

A generic "Failed" message type sent as request in response to a failure to process a previous protocol "Request" message. This is used in response to all request types that can return an error.

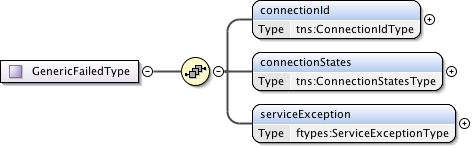


Figure – GenericFailedType.

Parameters

The **GenericFailedType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation request. This value will be unique within the context of the Provider NSA. |
| connectionStates | M | Overall connection state for the reservation. |
| serviceException | M | Specific error condition - the reason for the failure. |

#### GenericRequestType

This is a generic type definition for request messages such as provision, release, terminate, and forcedEnd that only need a connectionId as a request parameter.



Figure – GenericRequestType.

Parameters

The **GenericRequestType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation request. This value will be unique within the context of the Provider NSA. |

* + - 1. MessageDeliveryTimeoutRequestType

A notification message type definition for the Message Transport Layer (MTL) delivery timeout of a request message. In the event of an MTL timed out or Coordinator timeout, the Coordinator will generate this message delivery failure notification and send it up the workflow tree (towards the uRA).

An MTL timeout can be generated as the result of a timeout on receiving an ACK message for a corresponding send request. A Coordinator timeout can occur when no confirm or fail reply has been received to a previous request issued by the Coordinator. In both cases the local timers for these timeout conditions are locally defined.

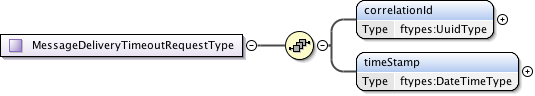


Figure – MessageDeliveryTimeoutRequestType.

Parameters

The **MessageDeliveryTimeoutRequestType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| correlationId | M | This value indicates the correlationId of the original message that the transport layer failed to send. |
| timeStamp | M | Time the event was generated. |

#### OrderedStpType

A Service Termination Point (STP) that can be ordered in a list for use in PathObject definition.

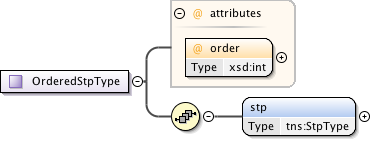


Figure – OrderedStpType.

Parameters

The **OrderedStpType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| order | M | Order attribute is provided only when the STP is part of an orderedStpList. |
| stp | M | The Service Termination Point (STP). |

#### PathType

Path of the service represented by a list of STP.

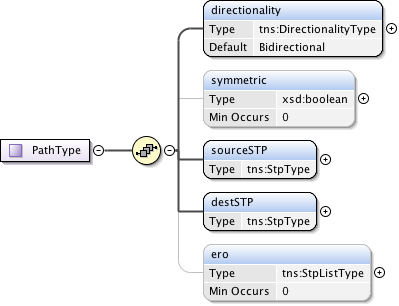


Figure – PathType.

Parameters

The **PathType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| directionality | M | The (uni or bi) directionality of the service. |
| symmetricPath | O | An indication that both directions of a bidirectional circuit must fallow the same path. Only applicable when directionality is "Bidirectional". If not specified then value is assumed to be false. |
| sourceSTP | M | Source STP of the service. |
| destSTP | M | Destination STP of the service. |
| ero | O | Hop-by-hop ordered list of STP from sourceSTP to destSTP. List does not include sourceSTP and destSTP. |

#### QueryFailedType

A query "Failed" message type sent as request in response to a failure to process a queryRequest message. This is message is returned as a result of a processing error and not for the case where a query returns an empty result set.



Figure – QueryFailedType.

Parameters

The **QueryFailedType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| ServiceException | M | Specific error condition - the reason for the failure. |

#### QueryRecursiveConfirmedType

This is the type definition for the queryRecursiveConfirmed message. An NSA sends this positive queryRecursiveRequest response to the NSA that issued the original request message. There can be zero or more results retuned in this confirmed message depending on the query parameters supplied in the request.

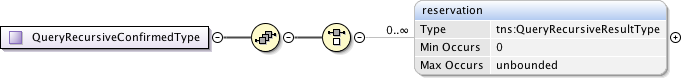


Figure – QueryRecursiveConfirmedType.

Parameters

The **QueryRecursiveConfirmedType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| reservation | O | Resulting recursive set of connection reservations matching the query criteria. If there were no matches to the query then no reservation elements will be present. |

#### QueryRecursiveResultType

This type contains the common reservation elements and detailed path data for "Recursive" query results.

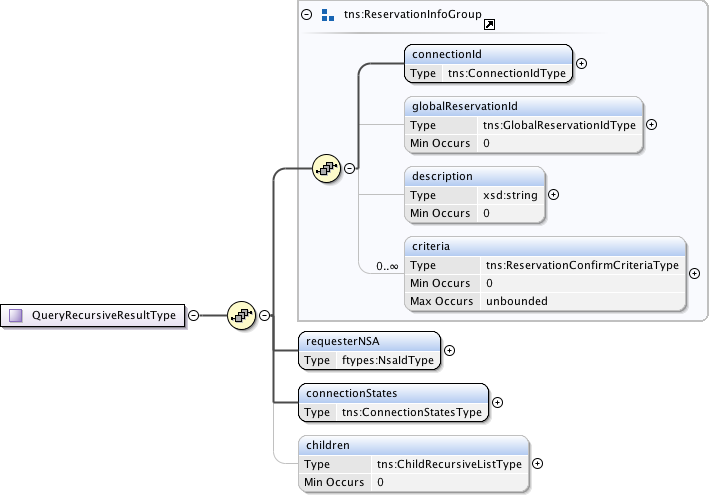


Figure – QueryRecursiveResultType.

Parameters

The **QueryRecursiveResultType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |
| globalReservationId | O | An optional global reservation id that can be used to correlate individual related service reservations through the network. This must be populated with a Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 | ISO/IEC 9834-8:2005 and IETF RFC 4122. |
| description | O | An optional description for the service reservation. |
| criteria | O | A set of versioned reservation criteria information. |
| requesterNSA | M | The requester NSA associated with the reservation. |
| connectionStates | M | The reservation's overall connection states. |
| children | O | If this connection reservation is aggregating child connections then this element contains detailed information about the child connection segment. The level of detail include is left up to the individual NSA and their authorization policies. |

#### QuerySummaryConfirmedType

This is the type definition for the querySummaryConfirmed message (both synchronous and asynchronous versions). An NSA sends this positive querySummaryRequest response to the NSA that issued the original request message. There can be zero or more results retuned in this confirmed message depending on the number of matching reservation results.

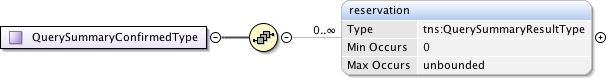


Figure – QuerySummaryConfirmedType.

Parameters

The **QuerySummaryConfirmedType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| reservation | O | Resulting recursive set of connection reservations matching the query criteria. If there were no matches to the query then no reservation elements will be present. |

#### QuerySummaryResultType

Type containing the set of reservation parameters associated with a "Summary" query result.

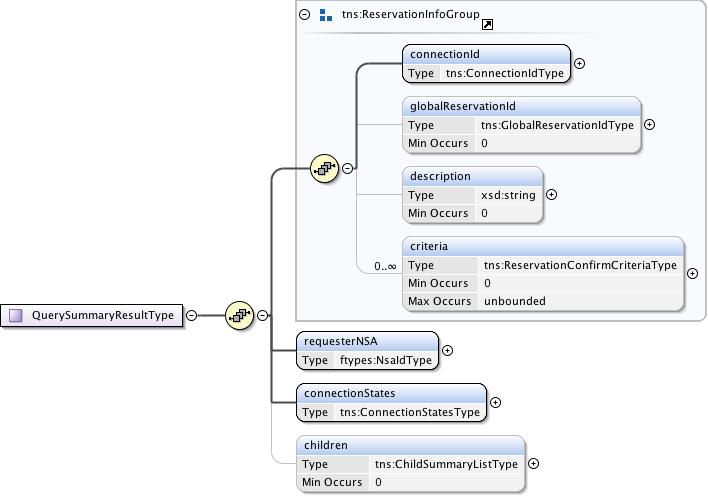


Figure – QuerySummaryResultType.

Parameters

The **QuerySummaryResultType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |
| globalReservationId | O | An optional global reservation id that can be used to correlate individual related service reservations through the network. This must be populated with a Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 | ISO/IEC 9834-8:2005 and IETF RFC 4122. |
| description | O | An optional description for the service reservation. |
| criteria | O | A set of versioned reservation criteria information. |
| requesterNSA | M | The requester NSA associated with the reservation. |
| connectionStates | M | The reservation's overall connection states. |
| children | O | If this connection reservation is aggregating child connections then this element contains detailed information about the child connection segment. The level of detail include is left up to the individual NSA and their authorization policies. |

#### QueryType

Type definition for the querySummary message providing a mechanism for either Requester or Provider NSA to query the other NSA for a set of connection service reservation instances between the RA-PA pair. This message can also be used as a status polling mechanism.

Elements compose a filter for specifying the reservations to return in response to the queryRequest. Supports the querying of reservations based on connectionId or globalReservationId. Filter items specified are OR'ed to build the match criteria. If no criteria are specified then all reservations associated with the requesting NSA are returned.

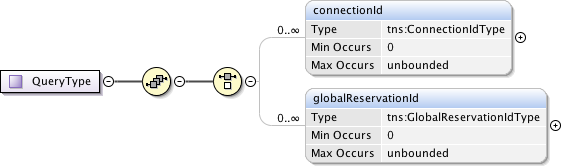


Figure – QueryType.

Parameters

The **QueryType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | O | Return reservations containing this connectionId. |
| globalReservationId | O | Return reservations containing this globalReservationId. |

#### RecursivePathType

This type is used to model a connection reservation's detailed  path information. The structure is recursive so it is possible to model both an ordered list of connection segments, as well as the hierarchical connection segments created on children NSA in either a tree and chain configuration.

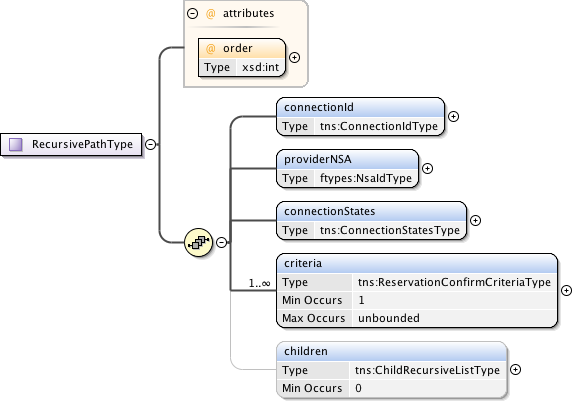


Figure – RecursivePathType.

Parameters

The **RecursivePathType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| order | M | Specification of ordered path elements. |
| connectionId | M | The connection identifier associated with the reservation and path segment. |
| providerNSA | M | The provider NSA holding the connection information associated with this instance of data. |
| connectionStates | M | This reservation's segments connection states. |
| criteria | M | A set of versioned reservation criteria information. |
| children | O | If provided this element will contain the list of connections in the context of all direct children NSA involved in the connection path. |

#### ReservationConfirmCriteriaType

A type definition for the reservation confirmation information used by PA to return reservation information to an RA. Includes the reservation version id to track version of the reservation criteria.

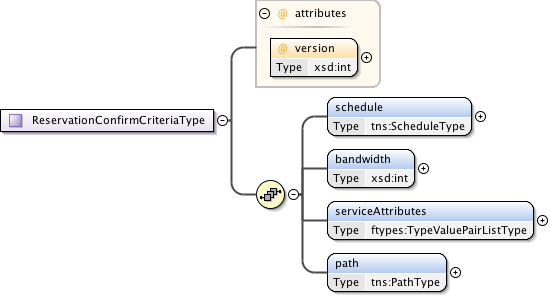


Figure – ReservationConfirmCriteriaType.

Parameters

The **ReservationConfirmCriteriaType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| version | M | Version of the reservation instance. |
| schedule | M | Time parameters specifying the life of the service. |
| bandwidth | M | Bandwidth of the service in Mb/s. |
| serviceAttributes | M | Technology specific attributes relating to the service. |
| path | M | The source and destination end points of the service. Can optionally provide additional path segments to guide path computation. |

#### ReservationRequestCriteriaType

Type definition for a reservation and modification request criteria. Only those values requiring change are specified in the modify request. The version value specified in a reservation or modify request must be a positive integer larger than the previous version number. A version value of zero is a special number indicating an allocated but not yet reserved reservation and cannot be specified by the RA.

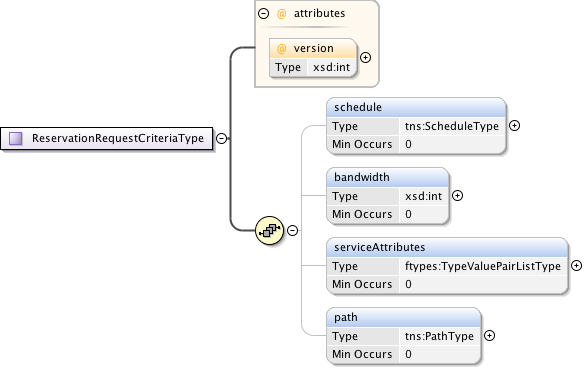


Figure – ReservationRequestCriteriaType.

Parameters

The **ReservationRequestCriteriaType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| version | M | The version number assigned by the RA to this reservation instance. If not specified in the initial reservation request, the new reservation will default to one for the first version; however, an initial request can specify any positive integer except zero. Each further reservation request on an existing reservation (a modify operation), will be assigned a linear increasing number, either specified by the RA, or assigned by the PA if not specified. |
| schedule | M | Time parameters specifying the life of the service. |
| bandwidth | M | Bandwidth of the service in Mb/s. |
| serviceAttributes | M | Technology specific attributes relating to the service. |
| path | M | The source and destination end points of the service. Can optionally provide additional path segments to guide path computation. |

#### ReserveConfirmedType

Type definition for the reserveConfirmed message. A Provider NSA sends this positive reserveRequest response to the Requester NSA that issued the original request message.

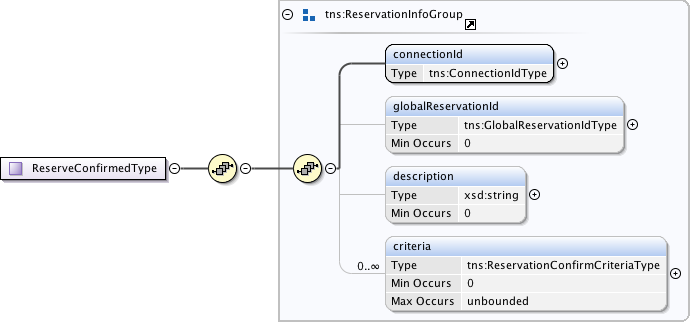


Figure – ReserveConfirmedType.

Parameters

The **ReserveConfirmedType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |
| globalReservationId | O | An optional global reservation id that can be used to correlate individual related service reservations through the network. This must be populated with a Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 | ISO/IEC 9834-8:2005 and IETF RFC 4122. |
| description | O | An optional description for the service reservation. |
| criteria | O | A set of versioned reservation criteria information. |

#### ReserveResponseType

Type definition for the reserveResponse message. A Provider NSA sends this reserveResponse message immediately after receiving the reservation request to inform the Requester NSA of the connectionId allocated to their reservation request. This connectionId can then be used to query reservation progress.



Figure – ReserveResponseType.

Parameters

The **ReserveResponseType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. |

#### ReserveTimeoutRequestType

Type definition for the reserve timeout notification message. This is an autonomous message issued from a Provider NSA to a Requester NSA when a timeout on an existing reserve request occurs and uncommitted resources have been freed.

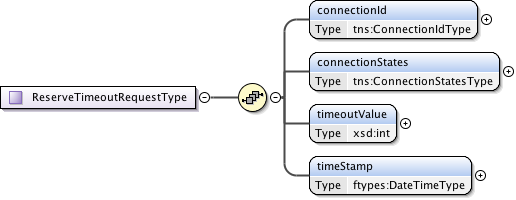


Figure – ReserveTimeoutRequestType.

Parameters

The **ReserveTimeoutRequestType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | M | The Provider NSA assigned connectionId for the reservation experiencing the timeout. |
| connectionStates | M | Current connection state for the reservation after the timeout. |
| timeoutValue | M | The timeout value in seconds that expired this reservation. |
| timeStamp | M | Time the event was generated. |

#### ReserveType

A type definition modeling the reserve message that allows a Requester NSA to reserve network resources for a connection between two STP's constrained by a certain service parameters. This operation allows a Requester NSA to check the feasibility of connection reservation or a modification to an existing reservation. Any resources associated with the reservation or modification will be allocated and held until commit is received or timeout occurs.

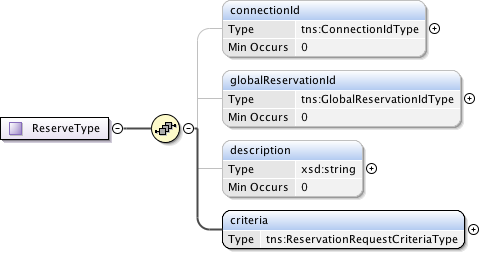


Figure – ReserveType.

Parameters

The **ReserveType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| connectionId | O | The Provider NSA assigned connectionId for this reservation. This value will be unique within the context of the Provider NSA. Provided in reserve request only when an existing reservation is being modified. |
| globalReservationId | O | An optional global reservation id that can be used to correlate individual related service reservations through the network. This must be populated with a Universally Unique Identifier (UUID) URN as per ITU-T Rec. X.667 | ISO/IEC 9834-8:2005 and IETF RFC 4122. |
| description | O | An optional description for the service reservation. |
| criteria | M | Reservation request criteria including start and end time, service attributes, and requested path for the service. |

#### ScheduleType

A type definition modeling the reservation schedule start and end time parameters.

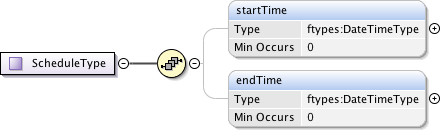


Figure – ScheduleType.

Parameters

The **ScheduleType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| startTime | O | Reservation start time. If not specified then immediate reservation. |
| endTime | O | Reservation end time. If endTime is not specified then the schedule end is indefinite. |

#### StpListType

A simple ordered list type of Service Termination Point (STP). List order is determined by the integer order attribute in the orderedSTP element.

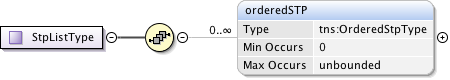


Figure – StpListType.

Parameters

The **StpListType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| orderedSTP | O | A list of STP ordered 0..n by their integer order attribute. |

#### StpType

The Service Termination Point (STP) type used for path selection.

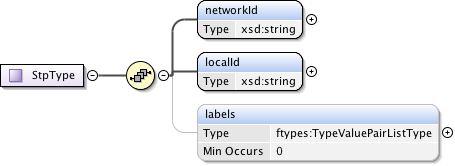


Figure – StpType.

Parameters

The **StpType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| networkId | M | A globally unique identifier (URN) that identifies the Network. Rather than forcing parsing of an STP to determine the Network, a separate Network object is defined to allow an intermediate NSA to forward the message to the target Network without needing to know about the STPs within that domain. |
| localId | M | A locally unique identifier for the STP within the associated network. |
| labels | O | Technology specific attributes associated with the Service Termination Point. |

#### SummaryPathType

This type is used to model a connection reservation's summary path information. The structure provides the next level of connection information but not state.

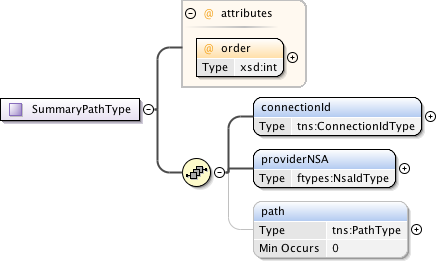


Figure – SummaryPathType.

Parameters

The **SummaryPathType** has the following parameters (M = Mandatory, O = Optional):

|  |  |  |
| --- | --- | --- |
| Parameter | M/O | Description |
| order | M | Specification of ordered path elements. |
| connectionId | M | The connection identifier associated with the reservation and path segment. |
| providerNSA | M | The provider NSA holding the connection information associated with this instance of data. |
| path | O | The summary path information associated with the connection reservation. |

### Simple Types

These simple type definitions are utilized by the CS complex type definitions. Types are listed in alphabetical order.

#### ConnectionIdType

A connectionId is a simple string value that uniquely identifies a reservation segment within the context of a Provider NSA. This value is not globally unique.



Figure – ConnectionIdType.

#### DirectionalityType

Directionality of the requested data service. Possible values are “Bidirectional” for a bidirectional data service, and “Unidirectional” for a unidirectional data service.



Figure – DirectionalityType.

#### EventEnumType

Notification event message types. Possible values are:

* activateFailed – Indicates that the data plane activation related to a reservation has failed, and therefore, there is no data plane connectivity for the reporting uPA.
* deactivateFailed – Indicates that deactivation of the data plane has failed, and as a result, data plane connectivity may still be in place.
* dataplaneError – Indicates that an error has occurred in the data plane and a loss of connectivity may be the result.
* forcedEnd – Indicates that the reservation was administratively terminated by a provider NSA within the network.



Figure – EventEnumType.

#### GlobalReservationIdType

A globalReservationId is a type representing a globally unique identifier for a reservation. This will be populated with a OGF URN (reference artifact 6478 "Procedure for Registration of Subnamespace Identifiers in the URN:OGF Hierarchy") to be used for compatibility with other external systems.



Figure – GlobalReservationIdType.

#### LifecycleStateEnumType

Connection lifecycle state values for the reservation lifecycle state machine. The lifecycle state machine is instantiated when a reservation is committed. Possible state values are:

* Created – A steady state for the lifecycle state machine and the initial state after a reservation has been committed.
* Failed – A steady state for the lifecycle state machine that is reached if a forcedEnd error is received from a uPA.
* Terminating - A transient state modeling the act of terminating the reservation.
* Terminated - A steady state for the lifecycle state machine that is reached when the reservation is terminated by the uRA.



Figure – LifecycleStateEnumType.

#### ProvisionStateEnumType

Connection provisioning state values for modeling the connection services provision state machine.

The Provision State Machine (PSM) is a simple state machine that transits between the Provisioned and the Released state. An instance of the PSM is created when an initial reservation is committed, and at that time it remains in the Released state. The PSM transits states independent of the state of the Reservation State Machine. Note that staying at the Provisioned state is necessary but not sufficient to activate the data plane. The data plane is active if the PSM is in “Provisioned” state AND current\_time is between startTime and endTime.

Possible state values are:

* Released – A steady state for the provision state machine in which data plane resources for this reservation are in a released state, resulting in an inactive data plane.
* Provisioning - A transient state modeling the act of provisioning the reservation’s associated data plane resources.
* Provisioned - A steady state for the provision state machine in which data plane resources for this reservation are in a provisioned state. This state does not imply that data plane resources are active, but it does indicate that a uPA can active the data plane resources if current\_time is between startTime and endTime.
* Releasing - A transient state modeling the act of releasing the reservation’s associated data plane resources.



Figure – ProvisionStateEnumType.

#### ReservationStateEnumType

Connection reservation state values for the connection services reservation state machine. Possible state values are:

* ReserveStart – A steady state for the reservation state machine in which a reservation is created and committed. In the case of the first reservation request this state represents the initial reservation shell has been committed to database.
* ReserveChecking – A transient state modeling the act of checking the feasibility of a new reservation request, or a request to modify an existing reservation.
* ReserveFailed – A steady state for the reservation state machine in which the initial reservation or a subsequent modification request has failed.
* ReserveAborting - A transient state modeling the act of aborting a pending reservation modify request.
* ReserveHeld - A steady state for the reservation state machine in which the initial reservation or a subsequent modification request has successfully had the request resources reserved, but has not yet been committed.
* ReserveCommitting - A transient state modeling the act of committing a held set of reservation resources.
* ReserveTimeout - A steady state for the reservation state machine in which the held resources have been locally timed out on a uPA, resulting in a transition from the ReserveHeld to ReserveTimeout state.



Figure – ReservationStateEnumType.

# Appendix B: State Machine Transition Tables



Table 3. RSM transition table



Table 4. PSM transition table



Table 5. LSM transition table

# Appendix C: Formal statement of coordinator

The following is an attempt to describe the behavior of the Coordinator in relation to the processing of requests and interactions with the various state machines in the NSA. Due to the slight difference in behavior between an AG and uPA, they are describe separately

## Aggregator NSA

### Processing of NSI Requests

**NSI\_rsv.rq(Conn\_ID, Corr\_ID, Ver)** /\* ***from parent NSA*** \*/

if (new Conn\_ID) then

{

create state machines RSM(Conn\_ID), PSM(Conn\_ID), LSM(Conn\_ID)

do path finding -> create entry for all children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

}

send res.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_rsvcommit.rq(Conn\_ID, Corr\_ID, ver)** /\* ***from parent NSA*** \*/

send rsvcommit.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_rsvabort.rq(Conn\_ID, Corr\_ID, ver)** /\* ***from parent NSA*** \*/

send rsvabort.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_prov.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send prov.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_rel.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send rel.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_term.rq** /\* ***from parent NSA*** \*/

send term.rq(Corr\_ID) to LSM(Conn\_ID)

send term.rq to RSM(Conn\_ID), PSM(Conn\_ID) /\* if RSM and PSM exist \*/

**NSI\_rsv.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send res.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsv.fl(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

if request\_list(Conn\_ID, Corr\_ID).Status != fail then

{

set request\_list(Conn\_ID, Corr\_ID).Status = fail

send res.fl(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvcommit.cf(Conn\_ID, Corr\_ID, Ver)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rsvcommit.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvcommit.fl(Conn\_ID, Corr\_ID, Ver)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rsvcommit.fl(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvabort.cf(Conn\_ID, Corr\_ID, Ver)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rsvabort.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_prov.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send prov.cf(Corr\_ID) to PSM(Conn\_ID)

}

**NSI\_rel.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send rel.cf(Corr\_ID) to PSM(Conn\_ID)

}

**NSI\_term.cf(Conn\_ID, Corr\_ID)** /\* ***from child NSA*** \*/

set request\_segment\_list(Conn\_ID, Child\_NSA, Corr\_ID).Status = replied

if all children in request\_segment\_list(Conn\_ID, Child\_NSA,

Corr\_ID).Status == replied then

{

send term.cf(Corr\_ID) to LSM(Conn\_ID)

}

### Requests from State Machines

**rsv.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_rsv.rq(Conn\_ID, Corr\_ID, Ver) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rsvcommit.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_rsvcommit.rq(Conn\_ID, Corr\_ID, Ver) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rsvabort.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_rsvabort.rq(Conn\_ID, Corr\_ID, Ver) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rsv.cf(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsv.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsv.fl(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsv.fl(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvcommit.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsvcommit.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvcommit.fl(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsvcommit.fl(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvabort.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

send NSI\_rsvabort.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**prov.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_prov.rq(Conn\_ID, Corr\_ID) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**rel.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_prov.rq(Conn\_ID, Corr\_ID) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**prov.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_prov.cf(Conn\_ID, Corr\_ID) to the parent

**rel.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_rel.cf(Conn\_ID, Corr\_ID) to the parent

**term.rq(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

create entry for all children in request\_segment\_list(Conn\_ID,

Child\_NSA, Corr\_ID)

send NSI\_term.rq(Conn\_ID, Corr\_ID) to children in

connection\_segment\_list(Conn\_ID, Child\_NSA)

**term.cf(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

clean up everything related to Conn\_ID

send NSI\_term.cf(Conn\_ID, Corr\_ID) to the parent

## Ultimate Provider NSA

### Processing of NSI Requests

**NSI\_rsv.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

if (new Conn\_ID) then

{

create state machines RSM(Conn\_ID), PSM(Conn\_ID), LSM(Conn\_ID)

}

send res.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

if reservation is made by checking the Reservation DB then

{

send res.cf(Corr\_ID, Ver) to RSM(Conn\_ID)

}

else

{

send res.fl(Corr\_ID, Ver) to RSM(Conn\_ID)

}

**NSI\_rsvcommit.rq(Conn\_ID, Corr\_ID, Ver)** /\* ***from parent NSA*** \*/

send rsvcommit.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_rsvabort.rq(Conn\_ID, Corr\_ID, Ver)** /\* ***from parent NSA*** \*/

send rsvabort.rq(Corr\_ID, Ver) to RSM(Conn\_ID)

**NSI\_prov.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send prov.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_rel.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send rel.rq(Corr\_ID) to PSM(Conn\_ID)

**NSI\_term.rq(Conn\_ID, Corr\_ID)** /\* ***from parent NSA*** \*/

send term.rq(Corr\_ID) to LSM(Conn\_ID)

send term.rq to RSM(Conn\_ID), PSM(Conn\_ID), ASM(Conn\_ID)

/\* if RSM, PSM and ASM exist \*/

### Requests from State Machines

**rsv.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

ignore

**rsvcommit.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

ignore

**rsvabort.rq(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

ignore

**rsv.cf(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

set REPLIED(Corr\_ID)

send NSI\_rsv.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsv.fl(Corr\_ID)** /\* ***from RSM(Conn\_ID)*** \*/

set REPLIED(Corr\_ID)

send NSI\_rsv.fl(Conn\_ID, Corr\_ID) to the parent

**rsvcommit.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

commit the reservation(Conn\_ID, Ver)

set REPLIED(Corr\_ID)

send NSI\_rsvcommit.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvcommit.fl(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

commit the reservation(Conn\_ID, Ver)

set REPLIED(Corr\_ID)

send NSI\_rsvcommit.fl(Conn\_ID, Corr\_ID, Ver) to the parent

**rsvabort.cf(Corr\_ID, Ver)** /\* ***from RSM(Conn\_ID)*** \*/

abort the reservation(Conn\_ID, Ver)

set REPLIED(Corr\_ID)

send NSI\_rsvabort.cf(Conn\_ID, Corr\_ID, Ver) to the parent

**prov.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

set prov\_flag(Conn\_ID)

if in\_period\_flag is set then

{

activate data plane according to the latest reservation

send prov.cf(Corr\_ID) to PSM(Conn\_ID)

}

**rel.rq(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

reset prov\_flag(Conn\_ID)

deactivate data plane

send rel.cf(Corr\_ID) to PSM(Conn\_ID)

**prov.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_prov.cf(Conn\_ID, Corr\_ID) to the parent

**rel.cf(Corr\_ID)** /\* ***from PSM(Conn\_ID)*** \*/

send NSI\_rel.cf(Conn\_ID, Corr\_ID) to the parent

**term.rq(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

ignore

**term.cf(Corr\_ID)** /\* ***from LSM(Conn\_ID)*** \*/

clean up everything related to Conn\_ID

send NSI\_term.cf(Conn\_ID, Corr\_ID) to the parent

# Appendix D: Best Practices for NSA implementation

This appendix lists a set of best practices to ensure interoperability between NSA implementations.

## Message transport error handling

Additional error condition handling: The following set of checks is required to pass for messages to be considered vaild, otherwise a message transport layer fault will be returned:

* HTTP authentication – if the message does not have valid credentials it will be rejected with an HTTP 40x message.
* *correlationId* - needed for any acknowledgment, confirmation, or failed message. Must be unique within the context of the providerNSA otherwise the request cannot be accepted.
* *replyTo* - we will send the confirmation, or failed message back to this location. We do not validate the contents of the endpoint, just that it exists.
* *Reservation* – if the reservation parameters are not present then we reject.
* *requesterNSA* and *providerNSA* – must be present and resolve to an *NSnetwork* in topology. Also, the *providerNSA* must be the *NSnetwork* that the NSA is managing or the message will be rejected.
* *connectionId* – this is used as the primary reference attribute for Reservation state machines and must be present.
* If any of these fields are missing or invalid the NSA will return a message transport fault containing the *NSIServiceException* set to an appropriate error message. Typically this will be MISSING\_PARAMETER - "SVC0001", "Invalid or missing parameter" for this generic case and specify attributes identifying the parameter in question.

The following list of parameters should be validated when receiving a reservation message:

|  |  |  |  |
| --- | --- | --- | --- |
| ***errorId*** | ***errorDescription*** | ***text*** | ***variables*** |
| SVC0001 | MISSING\_PARAMETER | Invalid or missing parameter |  |
| SVC0002 | UNSUPPORTED\_OPTION | Parameter provided contains an unsupported value which MUST be processed |  |
| SVC0003 | ALREADY\_EXISTS | Schedule already exists for connectionId |  |
| SVC0004 | DOES\_NOT\_EXIST | Schedule does not exist for connectionId |  |
| SVC0005 | MISSING\_SECURITY | Invalid or missing user credentials |  |
| SVC0006 | TOPOLOGY\_RESOLUTION\_STP | Could not resolve STP in Topology database |  |
| SVC0007 | TOPOLOGY\_RESOLUTION\_STP\_NSA | Could not resolve STP to managing NSA |  |
| SVC0008 | PATH\_COMPUTATION\_NO\_PATH | Path computation failed to resolve route for reservation |  |
| SVC0009 | INVALID\_STATE | Connection state machine is in invalid state for received message |  |
| SVC0010 | INTERNAL\_ERROR | An internal error has caused a message processing failure |  |
| SVC0011 | INTERNAL\_NRM\_ERROR | An internal NRM error has caused a message processing failure |  |
| SVC0012 | STP\_ALREADY\_IN\_USE | Specified STP already in use |  |
| SVC0012 | BANDWIDTH\_NOT\_AVAILABLE | Insufficient bandwidth available for reservation |  |

Table : error messages

\*\*\*We will also need to agree on the format of the message/errorId.

## ACK handling

Delays on the transport layer can result in ACK arriving after the confirm/fail message. The following guidelines are recommended for handling web-service ACKs:

1. For protocol robustness, the NSA should accept any confirm/fail messages even if these are received out-of-order w.r.t. the ACK, i.e. before the associate ACK has been received.
2. The receipt of a confirm/fail message cancels out the need to receive an ACK. So the NSA should not only continue to process the confirm/fail message, but not gate on or wait for the ACK, i.e consequent-messages may be sent without waiting on the receipt of the ACK.
3. The NSA should send the ACK before sending the associated confirm/fail message.
4. The message transport layer takes care of ACK retransmission in case of a packet loss.
5. If the message transport layer is broken, the ACKs will eventually timeout and generate a message transport error that the NSA will need to handle.

## Guidelines on timeouts:

1. Timeouts should be configurable on a per operation basis and set to 2 minutes as a default.
2. Requester side timeouts: It is up to the individual provider to choose appropriate NSA timeouts for their network. As a guide the timeout should be set to 1 minute for reservations to a provider only NSA, and longer for hierarchical requests to aggregator NSAs depending on the number of levels of recursion. Provisioning requests are likely to take longer than Reservation requests. The timeout will need to be tailored to meet the response times of the participating networks.
3. The requester NSA may choose to send queries to check the status of a request rather than terminating at timeout.

## Parallel processing of messages:

The provider NSA should respond to queries even if still working on a response to a request.

## NTP servers

The server running the NSA should use NTP version 4 [8]. This will reduce the risk of clock skew between the NSAs.

## Transport plane failures

Failures in the transport plane can occur at any time, however within the framework of the NSI architecture, there are two time windows in which a transport plane failure is significant:

1. The time between the service Reservation completed and Provisioning start
2. The time between the service Provisioning completed and teardown started

The errors only need to be handled by the NSA if the Data Plane errors affect the user service.

Figure : Local/Remote Failures

Transport failure during the service Reservation and Provisioning: An element in the Data Plane becomes unavailable due to a soft or hard failure causing a Provisioning failure of a confirmed Reservation; the NRM can handle this by either reserving an alternate path as long as it meets the requested service characteristics or applying a forcedEnd to the Reservation. The local policy of a Network provider and availability of resources will determine what recovery action is taken.

Transport failure during Provisioning phase and teardown phase: In case a failure in the Data Plane affects an active Connection, the first recovery mechanisms will be triggered by the protection mechanisms Provisioned with the service. If the Connection Service is unprotected, then the failure notification will be sent to the Domain’s NSA. At that point, NSA will take appropriate action based on service and user policies by either re-routing the Connection within the Network or tearing down the service.

# Appendix E: Tree and chain examples

The CS does not require that NSI messages are forwarded through the same sequence of NSAs/Networks that the Connection transits. As a consequence both tree and chain type architectures are supported, an example of each type is shown in this appendix.

## Connection example managed by an NSA chain

An example of a Connection managed by a NSA chain is shown in the following diagram.

path_chain

Figure : Example of Connection managed by a NSA chain

This example shows a Topology consisting of 3 Networks, one per NSA. Each Network is described as a set of edge points (STPs). This topology would look like this:

Network X: X:a X:b

Network Y: Y:c, Y:d, Y:e

Network Z: Z:f, Z:g

Here the NSAs are connected as a chain:

NSA-X(Requester) to NSA-Y(Provider), NSA-Y(Requester), to NSA-Z(Provider)

Assuming a Request comes from the Application-NSA to NSA-X to reserve a connection X:a to Z:g, then NSA-X will look in the topology and determine that to make this Connection, NSA-X will reserve a local connection from X:a to Xb, and then NSA-X must forward a request for the remainder of the connection to NSA-Y: Y:C to Z:g

NSA-Y gets this request and reserves a connection between Y:c and Y:e and requests a Reservation from NSA-Z for a connection Z:f to Z:g.

## Connection example managed by an NSA tree

An example of a Connection managed by a NSA tree is shown in the following diagram.

path_tree

Figure : Example of a Connection managed by a NSA tree

The topology remains the same as for the previous example:

Network X: X:a X:b

Network Y: Y:c, Y:d, Y:e

Network Z: Z:f, Z:g

In this example, the NSAs are connected as a tree:

NSA-X(Requester) to NSA-Y(Provider) and

NSA-X(Requester) to NSA-Z(Provider)

Assuming a request comes from the Application-NSA to NSA-X to reserve a connection X:a to Z:g, then NSA-X will look in the topology and determine that to make this connection, a connection request is required locally between X:a and X:b. Next NSA-X must forward two requests:

1. To NSA-Y: Y:c to Y:e
2. To NSA-Z: Z:f to Z:g

# References

1. OGF GFD.173: “Network Service Framework v1.0”, <http://www.gridforum.org/documents/GFD.173.pdf>
2. OGF GWD-I Network Service Interface Topology Service Distribution Mechanisms

<https://redmine.ogf.org/dmsf_files/12980?download>=

1. GWD-R-P Network Service Interface Topology Representation <https://redmine.ogf.org/dmsf_files/12981?download>=
2. IETF RFC 5905, Network Time Protocol Version 4: Protocol and Algorithms Specification
3. IETF RFC 4122, A Universally Unique IDdentifier (UUID) URN Namespace
4. ITU-T Rec. X.667 Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components
5. ISO/IEC 9834-8:2005 Information technology -- Open Systems Interconnection -- Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components
6. IETF RFC 4655, "A Path Computation Element (PCE)-Based Architecture", http://www.rfc-editor.org/rfc/rfc4655.txt
7. ISO 8601:2000 “Data elements and interchange formats — Information interchange — Representation of dates and times” or xsd dateTime
8. IETF RFC 5905, “Network Time Protocol Version 4: Protocol and Algorithms Specification”, http://tools.ietf.org/html/rfc5905
9. IETF RFC 6453, “A URN Namespace for the Open Grid Forum (OGF)”, <http://tools.ietf.org/html/rfc6453>
10. OGF GFD-CP.191 "Procedure for Registration of Subnamespace Identifiers in the URN:OGF Hierarchy”, http://www.ogf.org/gf/docs/
11. W3C XML “Schema Definition Language (XSD) 1.1 Part 2: Datatypes”, http://www.w3.org/TR/xmlschema11-2/#anyURI