PacketCable™

ENUM Server Provisioning Specification

PKT-SP-ENUM-PROV-I04-100415

ISSUED

Document Status Sheet

Document Control Number:	PKT-SP-ENUM-PROV-I04-100415			
Document Title:	ENUM Server Provisioning Specification			
Revision History:	I01 - Released 02/15/08 I02 - Released 09/05/08 I03 - Released 06/30/09 I04 - Released 04/15/10			
Date:	April 15, 2010			
Status:	~~Work in Progress~~	~~Draft~~	**Issued**	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/ Member/ Vendor~~	**Public**

Key to Document Status Codes

Work in Progress	An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks

Contents

Figures

Tables

1 INTRODUCTION

1.1 Introduction and Purpose

This specification defines a provisioning protocol for ENUM-SIP addressing servers. It allows cable operators to provision and manage session establishment data used by PacketCable network elements to route SIP sessions to the target destinations which may be served by the cable operator's own internal network or by a session peering partner. The session establishment data is used by the addressing servers to resolve queries sent using the ENUM [RFC 3761] or Session Establishment Protocol (SIP) [RFC 3261].

The purpose of this specification is to define a common mechanism for various sources of session establishment data to provision ENUM-SIP addressing servers. These data sources may reside in an operator's network (intra-domain back-office systems), or in a peer's network in the case of bilateral session peering agreements, or in a session peering registry shared by a group of operators. These data sources advertise the public user identities they serve (SIP user addresses, telephone numbers, and other types of Uniform Resource Identifiers) along with other data elements like the Signaling path Border Elements (SBE) to use to reach those user identities.

This specification defines functional entities (a provisioning client and a provisioning server), two protocols between those entities (one for real-time operations and one for bulk or batch mode operations), and the associated requirements (secure data transport, resolution requirements based on the data model, etc.).

1.2 Protocol Overview and Capabilities

The ENUM Server Provisioning Protocol (ESPP) allows a cable operator to receive session establishment data from various sources: its internal back-office systems for intra-domain routes, a session peer, or a federation registry. A source of session establishment data is logically represented by an ESPP provisioning client (ESPP Client). The cable operator's ENUM-SIP addressing server receives this provisioning data via a functional entity called an ESPP Server.

The data sent by an ESPP Client to an ESPP Server follows a formal data structure based on the data model (Section 6.1) and includes parameters such as a peer's telephone numbers grouped by service area and the Signaling path Border Elements to reach those telephone numbers.

This specification defines the protocol interfaces between a provisioning client (ESPP Client) and a provisioning server function of an ENUM-SIP addressing server (ESPP Server), as shown on Figure 1.

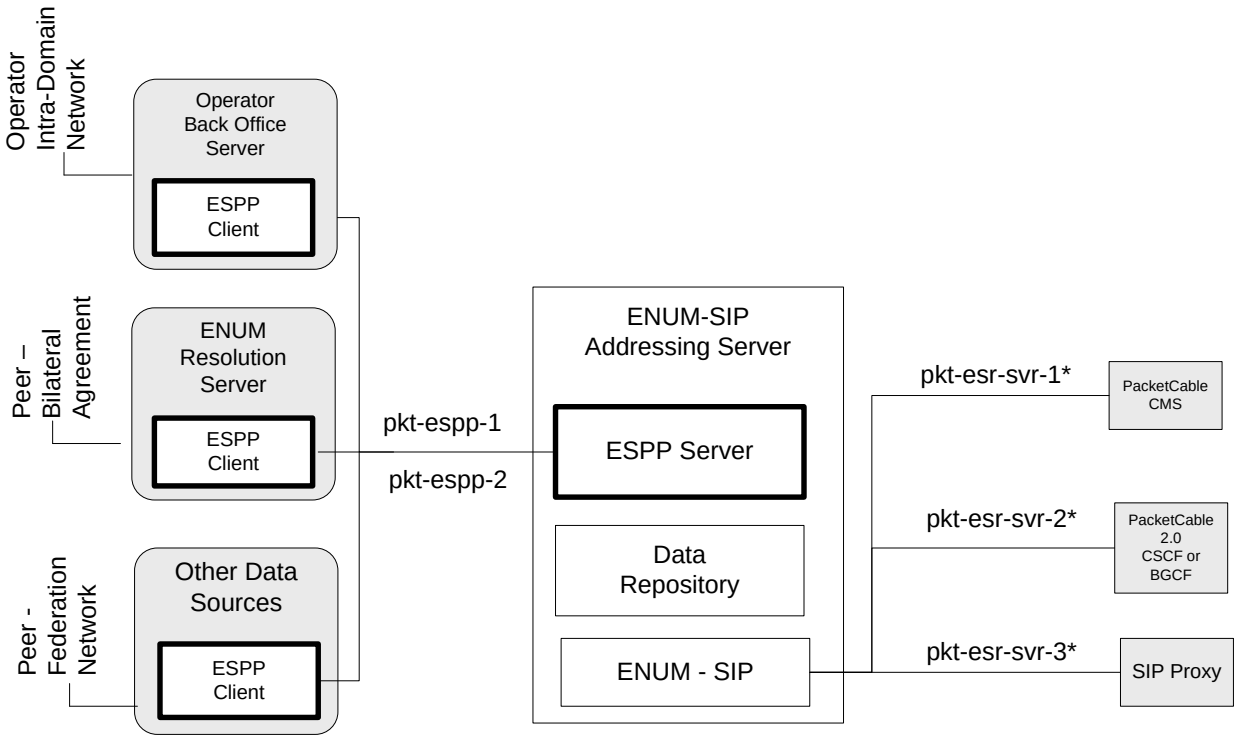

Figure 1 - ENUM Server Provisioning Protocol Reference Architecture

To claim compliance with this specification, an ESPP Client or an ESPP Server MUST implement the pkt-espp-1 and pkt-espp-2 reference points. An ESPP Client or ESPP Server implementing the pkt-espp-1 reference point MUST conform to real-time ESPP protocol requirements defined in Section 6 of this document. An ESPP Client or ESPP Server implementing the pkt-espp-2 reference point MUST conform to ESPP file-based provisioning protocol requirements defined in Section 7 of this document. The reference points pkt-esr-svr-1, pkt-esr-svr-2, pkt-esr-svr-3 are used by various network elements to query the addressing server using ENUM or SIP; they are out of scope of this specification and are denoted with an asterisk (*) in Figure 1.

The guiding design principal behind ESPP is the ability to support a large session addressing space whose size is of the same order of magnitude as the Public Switched Telephone Network (PSTN). Optimizing bandwidth and storage usage is a key characteristic.

The logical data structure implemented by ESPP allows for a more natural modeling of relationships between the entities that comprise the logical topology. Telephone numbers and other types of public user identities are grouped in logical or geographical areas and associated with routes that describe how traffic traverses the nodes of the topology. The process begins by arranging a plurality of telephone numbers and nodes connected by a plurality of routes. Each telephone number, number range, or public user identity is assigned to a service area that represents any contiguous geographic area serviced by the same set of equipment. Through a route assignment each node is associated with one or more service areas. For example, a cable operator could first create service areas for the New York metropolitan area, assigning the telephone numbers of New York subscribers into that service area. The cable operator then defines the call servers that provide digital voice services for New Yorkers and the border elements that connect the New York metropolitan area to peering networks. Finally, through the definition of intra-MSO and inter-MSO routes, that cable operator completes the establishment of session peering data by associating nodes with service areas.

Figure 2 illustrates the link between logical service areas and telephone numbers and the associated routes.

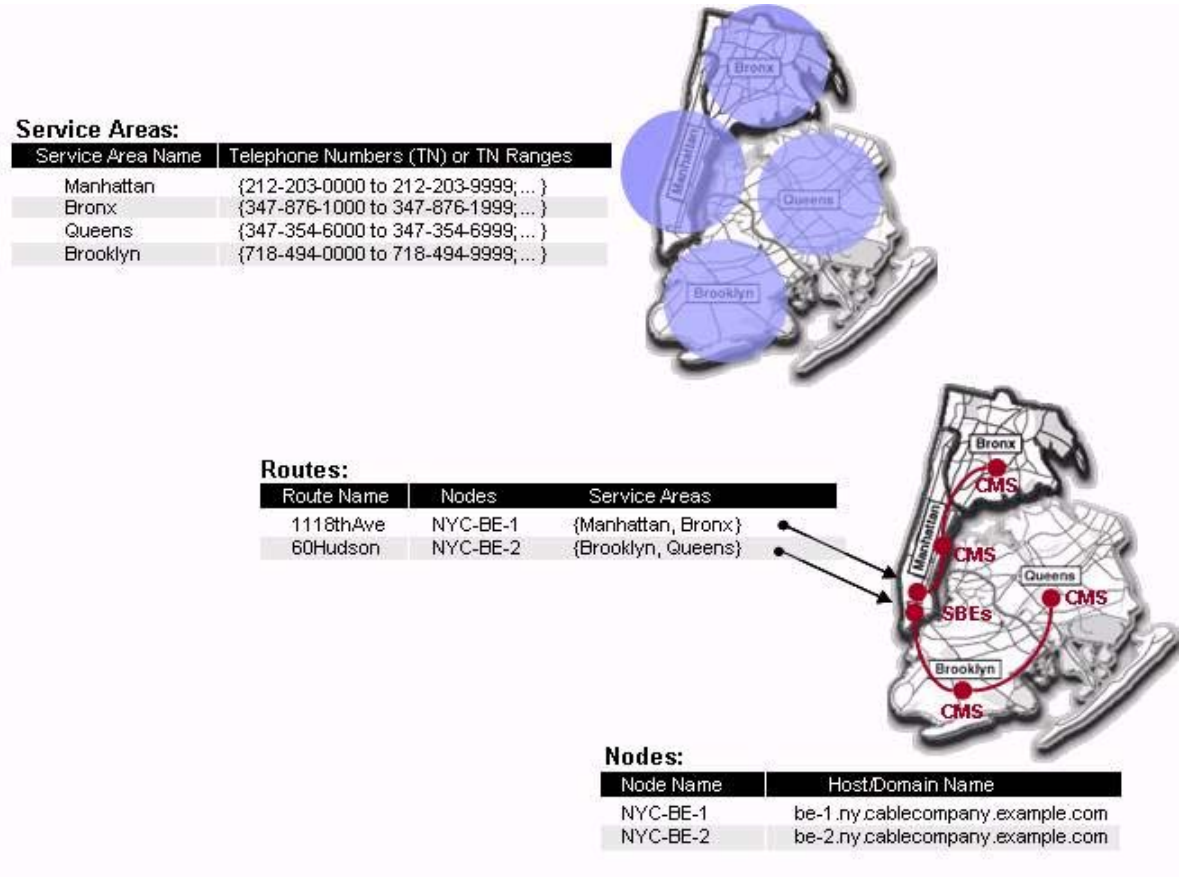

Figure 2 - Example of ESPP Service Areas and Routes

1.3 Requirements

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST" This word means that the item is an absolute requirement of this specification.

"MUST NOT" This phrase means that the item is an absolute prohibition of this specification.

"SHOULD" This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

"SHOULD NOT" This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

"MAY" This word means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

[RFC 2616] IETF RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, June 1999.

[RFC 2617] IETF RFC 2617, HTTP Authentication: Basic and Digest Access Authentication, June 1999.

[RFC 3402] IETF RFC 3402, Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm, October 2002.

[RFC 3403] IETF RFC 3403, Dynamic Delegation Discovery System (DDDS), Part Three: The Domain Name System (DNS) Database, October 2002.

[RFC 3404] IETF RFC 3404, Dynamic Delegation Discovery System (DDDS), Part Four: The Uniform Resource Identifiers (URI), Resolution Application, October 2002.

[RFC 3761] IETF RFC 3761, The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM), April 2004.

[RFC 4346] IETF RFC 4346, The Transport Layer Security (TLS) Protocol Version 1.1, April 2006.

[XML] W3C Recommendation, Extensible Markup Language (XML) 1.0., http://www.w3.org/TR/REC-xml.

[SOAP] W3C Recommendation, http://www.w3.org/TR/soap11/

[WSDL] Web Services Description Language (WSDL) 1.1., http://www.w3.org/TR/wsdl.

2.2 Informative References

This specification uses the following informative references.

[RFC 2821] IETF RFC 2821, Simple Mail Transfer Protocol, April 2001.

[RFC 3261] IETF RFC 3261, SIP: Session Initiation Protocol, June 2002.

[RFC 4114] IETF RFC 4114, E.164 Number Mapping for the Extensible Provisioning Protocol (EPP), June 2005.

2.3 Reference Acquisition

- Internet Engineering Task Force (IETF) Secretariat, 46000 Center Oak Plaza, Sterling, VA 20166, Phone +1-571-434-3500, Fax +1-571-434-3535, http://www.ietf.org.
- World Wide Web Consortium (W3C), 32 Vassar Street, 32-G514, Cambridge, MA 02139 USA, Phone: +1-617-253-2613, Fax: +1-617-258-5999, http://www.w3.org.

3 TERMS AND DEFINITIONS

This specification uses the terms defined in the following table as well as the terms defined in the Data Model Definition, Section 6.1.

Signaling path Border Element (SBE)	Provides signaling-related functions. Typically deployed on a domain's border as a Back-to-Back SIP User Agent (B2BUA), SBEs provide signaling functions such as protocol inter-working, identity and topology hiding, and call admission control.
Ingress SBE	An SBE through which externally originated, incoming session signaling is received.
Egress SBE	An SBE through which internally originated, outbound session signaling is sent.
Point-of-Interconnect (POI)	The geographic location where two networks interconnect and exchange traffic. Typically, one or more SBE(s) are allocated to secure the POI at the session peering level.

4 ABBREVIATIONS AND ACRONYMS

The table below defines the set of terminology abbreviations that must be utilized within the protocol specification (but not necessarily within the textual descriptions of this document). Use and standardization of these abbreviations is designed to disallow multiple abbreviations for the same term, lessen the number of characters that must be transmitted to and from the addressing servers, and enhance the readability of the protocol as excessively long compound type, and element names can degrade readability.

Abbreviation	Meaning
Auth	Authentication
Cred	Credentials
Cmn	Common
Del	Delete
Egr	Egress
eid	Enterprise ID
Entr	Enterprise
ESPP	ENUM Server Provisioning Protocol
Id	unique identifier
LRN	local routing number
Naptr	Name authority pointer
NS	Name Server
Obj	Object
oid	Object Id
Pi	Public Identity
Pref	Preference
Pvt	Private
Pub	Public
Pvi	Private Identity
Regx	regular expression
Rend	Range end

Abbreviation	Meaning
Repl	Replacement
Res	Result
Rn	routing number
Rqst	Request
Rspns	Response
Rstrt	Range start
SBE	signaling path border element
Sa	service area
Schm	Scheme
Svc(s)	service(s)
Tn	telephone number
Trans	Transaction

5 PROTOCOL REQUIREMENTS

This section describes the high level requirements for the ESPP protocol and provides several use cases.

5.1 Connection Oriented Operation

- The protocol MUST support a file-based, bulk delivery mechanism where the ESPP Client writes one or more update requests to one or more files and the file(s) are delivered to and consumed by the ESPP Server.
- All ESPP Clients and Servers MUST use HTTP 1.1 as defined in [RFC 2616] for the transport mechanism.
- All ESPP Clients and Servers SHOULD support HTTP Keep-Alive to allow long lived connections, where multiple request and response pairs are exchanged across a single network connection.

5.2 File Oriented Operation

- The protocol MUST support a file-based mechanism (bulk load) where the ESPP Client writes one or more requests to a file and the file is delivered to and consumed by the ESPP Server.
- The delivery or transmission of bulk files MAY be triggered by a manual process out-of-band of the protocol.
- During bulk loading the ESPP Server SHOULD NOT accept new records through the real-time, connection-oriented interface.
- The maximum size of a bulk load file MUST NOT exceed 500 MB.
- The name of a bulk file SHOULD identify the ESPP Client, ESPP Server, file sequence number, and transaction ID(s) for which the bulk file was generated.
- The bulk load interface MUST be capable of supporting the download of an entire address space of the order of the PSTN.
- The format of the bulk load file MUST be compatible with the XML definitions of the real-time interface.
- ESPP Servers MUST maintain a log that reports response codes with associated transaction IDs and client IDs.
- The errors codes of the bulk load interface SHOULD comply with the error codes of the real-time interface.

5.3 Security Requirements: Authentication, Integrity and Confidentiality

- All ESPP Clients and Servers MUST support Transport Layer Security (TLS) as defined in [RFC 4346] as the secure transport mechanism.
- All ESPP Clients and Servers MUST use HTTP Digest Authentication as defined in [RFC 2617] as the secure authentication mechanism.
- Transfer of bulk files MUST use Secure Copy (SCP), which relies on Secure Shell (SSH) for security, as the secure file transport mechanism

5.4 Data Model Requirements

- The protocol MUST be capable of supporting a large addressing space. For example, if the provisioned data involves telephone numbers, the protocol must be capable of supporting an addressing space of the same magnitude as the PSTN.
- The protocol MUST support the data model defined in Section 6 of this document.

5.5 Data Presentation Requirements

- The protocol MUST utilize SOAP 1.1 [SOAP], WSDL1.1 [WSDL], and XML 1.0 [XML].
- The protocol MUST support efficient transportation of a large number of data model objects from the client to the server.

5.6 Protocol Operations

- The protocol MUST support the ability to add, modify, and delete the objects defined in the protocol data model.
- The protocol MUST support the ability to query for a specific instance of each type of object defined in the data model by using the object identifier.
- The protocol MUST support the ability for multiple ESPP Clients to provision objects into the same ESPP Server.
- The protocol MUST support the ability for ESPP objects created by one ESPP Client to refer to ESPP objects created by another ESPP Client.

5.7 Versioning, Capability Exchange, and Extensibility Requirements

- The protocol MUST support schema versioning such that major version changes are defined as any change that breaks backward compatibility and minor version changes are defined as any change that does not break backward compatibility.
- The protocol MUST allow, but not require, a server to expose multiple concurrent major versions and/or minor versions of the protocol concurrently.
- The protocol MUST make the major version identification of a request message detectable by schema validation and the minor version identification of a request message detectable by the application.
- The protocol MUST be extensible such that new operations and objects can be added to the protocol in a systematic manner.

5.8 Use Cases Examples

5.8.1 Separation of Responsibility

An operator's business practices may impose a separation of roles and responsibilities such that: (a) network engineering and planning personnel are responsible for establishing points-of-interconnect, and (b) telephony personnel are responsible for provisioning telephone numbers, location routing numbers (LRNs) and other forms of resolvable addresses.

For example, two cable operators that respectively service the New York-Long Island and Philadelphia-Camden-Wilmington metropolitan areas agree to peer. In unison, the network engineering personnel of each company establish physical inter-connect in the New York City 60 Hudson Street facility. Each operator commissions redundant Signaling path Border Elements (SBEs) that are used to secure the interconnect located at 60 Hudson Street. Then through the use of the ESPP protocol, the network engineering departments publish IP addressing information to each other's ENUM-SIP addressing server - provisioning NAPTRs for each Ingress SBE and through Route objects associating them with the telephony address space (Service Area) of each metropolitan area. Once the Service Areas and Routes are established, the telephony personnel manage the telephony addressing by adding telephone numbers or LRNs to the respective Services Areas.

5.8.2 File-based Distribution and Bootstrapping

The process of downloading large quantities of session establishment data should be carried out as quickly as possible with minimum resources. It involves the generation and transfer of a bulk file between the client and server. It may be used in cases where the loading a newly commissioned ENUM-SIP addressing server or reloading an existing server data due to a complete shutdown or loss of memory has occurred. The technique designed to bootstrap an ENUM-SIP addressing server should be based on the straightforward bulk file load interface described in Section 7.

For example, a cable operator has decided to opt into a federation of service providers that collectively service over one hundred million (100M) subscribers, choosing to establish interconnections with every member of the federation. Once commissioned the operator's ENUM-SIP addressing server needs to immediately receive with all 100M registered telephone numbers (TNs). Rather than stream the 100M TNs over a network connection in real-time (as defined in Section 6.3), the administrator of the federation registry and operator choose to utilize the file based distribution mechanism (as described in Section 7).

5.8.3 Backward Compatibility to Legacy Switch Translations

The underlying data schema used to provision ENUM-SIP addressing servers should be backward compatible with legacy PSTN or VoIP switch translations. This requirement arises from the fact that many operators wish to utilize the same number translation data employed by their SIP proxy servers or PacketCable Call Management Servers (CMS).

For example, a cable operator's switch translation personnel, who are responsible for managing CMS translations, are given responsibility for managing the operator's ENUM data. Rather than provisioning complete 10-digit numbers to a peer's ENUM server some may choose to provision Location Routing Numbers (LRNs). This decision is, in large part, due to the fact that, in some networks, the trunk selection algorithm of the operator's CMS are based on LRNs (NPA-NXX). The switch translation personnel choose to reuse LRNs rather than taking responsibility for keeping a complete set of the operator's numbers up to date.

6 PROTOCOL DEFINITION

This section first introduces the structure of the data model and the overall technical structure that provide the information framework to which ESPP adheres. An overview of the protocol operations is then provided with a typical deployment scenario. This section concludes with the definition of the operations and data elements that comprise ESPP.

6.1 Logical Structure - Data Model

The data model illustrated and described below defines the logical objects and the relationships between these objects that the ESPP protocol supports. ESPP defines the interface through which an ESPP Client populates an ENUM-SIP addressing server with these logical objects. The protocol may be used by various clients for populating the server. ESPP Clients may include, for example, a back-office system, an ENUM Registry or another ENUM-SIP addressing server.

A key requirement of ESPP is the ability to support an addressing space of the same magnitude as the PSTN. To achieve the above, the data model needs to be designed such that it optimizes bandwidth and storage. The data model needs to support distribution of addressing information in a manner that affords the implementations the opportunity to maximize storage and minimize transport overhead. One way to achieve this is not relying exclusively on distribution of redundant NAPTR records. The use of a direct association between NAPTRs and telephone numbers should be restricted to service scenarios where there is a need to communicate subscriber-specific service attributes other than the subscriber's actual telephone number. This is because the NAPTRs carry a large amount of redundant data when many telephone numbers resolve to the same address.

As illustrated in the figure below, the logical structure that underlies ESPP is designed around a simplified version of the logical entities that comprise session peering data. This logical structure provides a consistent terminology and design to the constructs that make up the protocol.

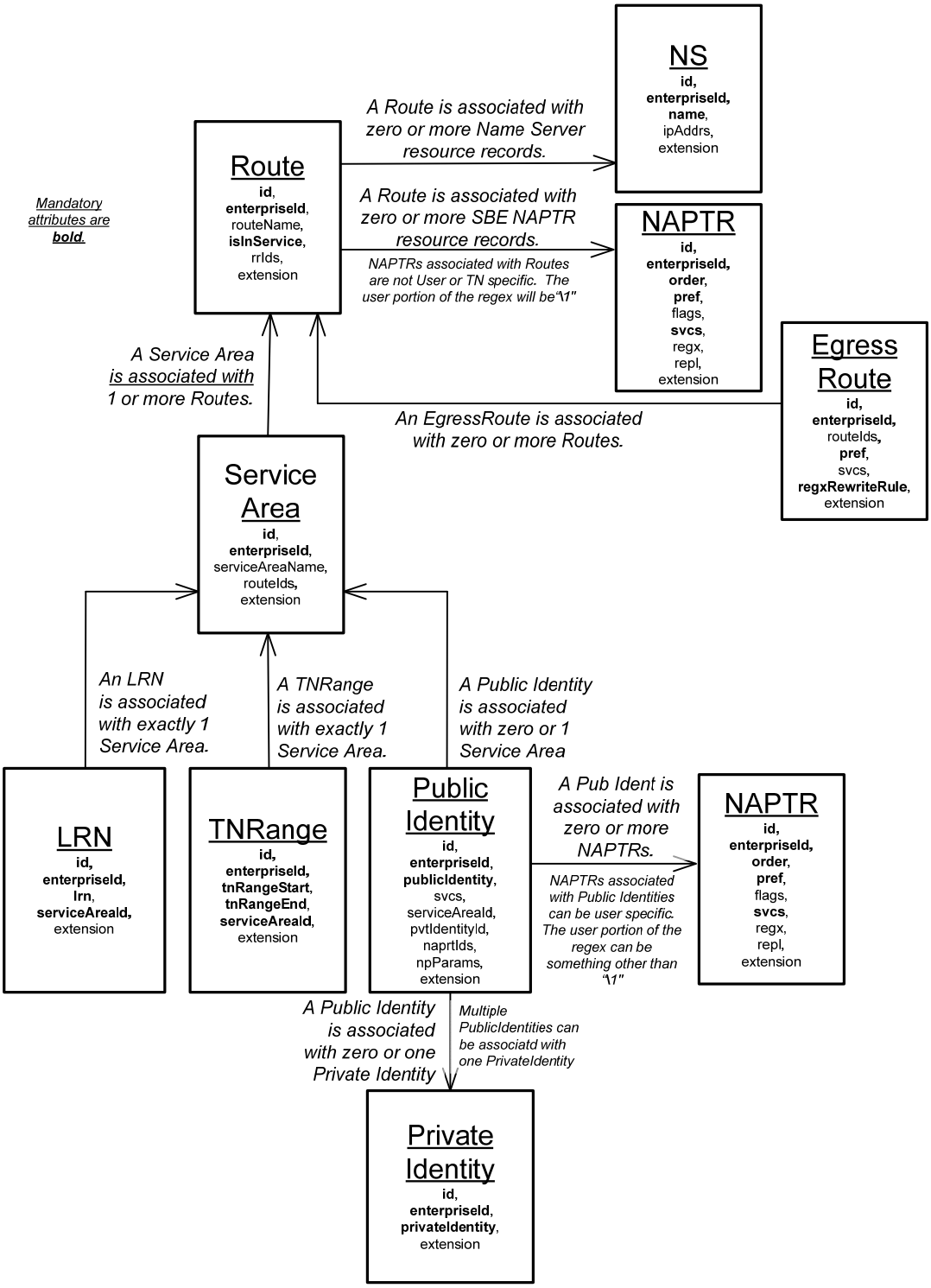

Figure 3 - Logical Structure of ESPP Objects

The objects that comprise the logical domain model can be described as follows:

- Public Identity (PubId): A string of numbers or characters that serves as a public identity. A Public Identity may be a telephone number, an email address, or other identity as deemed appropriate. The Public Identity object

may be associated with a Service Area which serves as a logical grouping of public identities and indirectly defines the signaling path used to Route to that Public Identity. A Public Identity may optionally be associated with zero or more individual NAPTRs. This ability for a Public Identity to be directly associated with a set of NAPTRs, as opposed to being associated with a Service Area, supports the use cases where the NAPTR may contain data specifically tailored to an individual Public Identity. A Public Identity may, optionally, be associated with zero or one Private Identity.

- Private Identity (PvtId): A string of numbers or characters that serves to associate, or group together, a set of related Public Identities. A Private Identity may, for example, represent a user who has more than one Public Identity instances. When a resolution request for a public identity arrives at an addressing server, the addressing server may locate the Public Identity that matches the queried public identity, discover the Private Identity associated with that Public Identity, and then gather and return all the NAPTRs directly or indirectly (via a Service Area) associated with the Public Identities that are associated with that Private Identity.
- Route: A set of related Signaling path Border Elements (SBEs). SBEs are generally ingress or egress points within, or at the edge of, an operator's network. A Route object is associated with zero or more SBEs. An SBE's properties are housed within the constructs of a NAPTR object modeled after a DNS NAPTR Record. At runtime (i.e., during the resolution of a SIP or ENUM query to an addressing server), the SBEs associated with the route to the Service Area of a public identity will be returned as part of the NAPTR response. In case of multiple SBEs, the ENUM response will have multiple NAPTR records, each referring to a particular SBE. This indirection between a Public Identity and its ENUM NAPTRs allows the modification of signaling path border elements without having to modify a large number of NAPTRs.
- Egress Route: Optional rules for re-writing (modifying) the regular expressions housed in the NAPTRs associated with a Route. This is primarily used by the addressing server to dynamically add a SIP Route header as part of a SIP resolution, and to dynamically indicate into an NAPTR's regular expression what SIP Route to use. This allows a NAPTR to define the local egress SBE or the remote ingress SBE of a signaling path.
- Service Area: A collection of zero or more Public Identities, Telephone Number ranges (TNRanges), and Location Routing Numbers (LRNs) that are related by virtue of their common signaling path and a set of zero or more Route relationships. The grouping of Public Identities, TNRanges, and LRNs into related groups and the indirect association of these groups to their signaling path elements significantly enhances the manageability of routing data for a large number of Public Identities. This allows the addressing server to apply routing changes to a set of Public Identities, TNRanges, and LRNs without having to make changes to each individual Public Identities, TNRange, or LRN.
- Location Routing Number (LRN): A 10-digit number that identifies the switching port for a local telephone exchange. LRN is used to support Local Number Portability in North America. An LRN is associated with a Service Area.
- TNRange: An object that represents a range of telephone numbers. The TNRange object must be associated with a Service Area which indirectly defines the signaling path to reach the TNs in that range.
- Enterprise ID: The enterprise ID data element that resides within each object type and uniquely identifies an operational entity within an ESPP Server. The operational entity is responsible for, or owns, that object from a business perspective. This does not identify the software entity that provisioned the object, which is the purpose of the client ID. Refer to Section 6.4 for further detail.
- ID: Each object is uniquely identifiable using its object ID. An object ID is guaranteed to be unique within an ESPP Server and ESPP Client, regardless of the number of ESPP Clients populating objects into a given ESPP Server. Refer to Section 6.4 for further detail.
- NAPTR: The data structure used to house the properties of signaling path border elements, and any other information appropriately held within the data structures offered by a NAPTR.
- NS: The data structure used to house the properties of a Name Server.

6.2 Technical Structure - SOAP/XML Type Hierarchy

In addition to employing the logical structure described in the previous section, ESPP also employs a consistent overall technical structure. This technical structure is typically referred to as the "Document Literal Wrapped" style of designing SOAP based APIs. This style is generally regarded as an optimal approach that enhances maintainability, comprehension, portability, and, to a certain extent, performance. The following figure illustrates this high level technical structure.

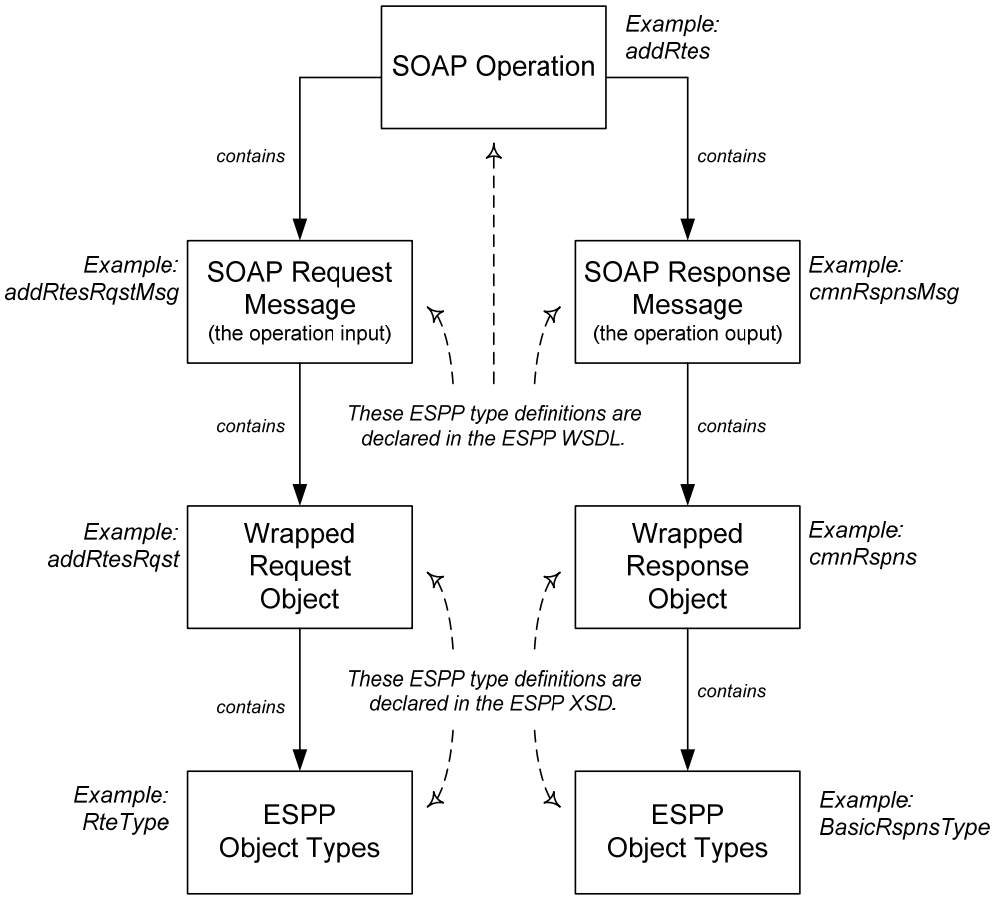

Figure 4 - Technical Structure of the ESPP Protocol

The SOAP operations supported by ESPP (e.g., `addRtes`) are normatively defined in Section 9.1. Each SOAP operation defines a request/input message and a response/output message. Each request and response message then contains a single object that wraps the data elements that comprise the inputs and the outputs, respectively, of the operation.

Section 6.4.2, Protocol Operation Descriptions, of this document defines how each operation utilizes this technical structure and the logical structure defined in the previous section.

6.3 Overview of Protocol Operations

This section provides an overview of ESPP and its protocol operations by describing a usage scenario and examples. It is divided into two sub-sections. The first sub-section defines a usage scenario and provides a set of example ESPP request messages that apply to the scenario. The second sub-section provides examples of a generic set of ESPP request/response pairs, one for each protocol operation.

6.3.1 Deployment Scenario and ESPP Provisioning

A typical deployment scenario for ESPP involves multiple cable operators wanting to exchange multimedia traffic including Voice over IP (VoIP) calls and Instant Messaging (IM) sessions. Each operator first advertises the public user identities (e.g., telephone numbers and instant messaging addresses) it serves on its network along with the routes to its peers.

Various models can be chosen by operators to determine if a target address is served by a peer. In one model, for each session initiation, an out-of-network query could be made to each peer individually, or to a federation entity representing the peers. In another model, an operator who has deployed ENUM-SIP addressing servers for session routing may prefer in-network queries to its addressing servers to resolve target addresses. In this second case, the operator's addressing server could be provisioned with peer data: this is the purpose of the ESPP protocol.

The remaining of this section describes a deployment scenario in more details, by covering both the provisioning activities using ESPP and the query resolutions to resolve target addresses using ENUM or SIP.
We first describe the actors of the scenario (cable operators or enterprises participating in multimedia session exchanges), and then expand on the ESPP operations for the provisioning of peers, Session Establishment Data (SED), and the public identities or Address of Records to which session may be established.

6.3.1.1 Actors

6.3.1.1.1 Enterprises Description

Three operating companies have agreed to exchange Voice over IP traffic and Instant Messages via their IP networks:

- GlobalMSO:

  GlobalMSO is a large operator with service footprints in Boston and Arizona and several PacketCable servers capable of processing VoIP traffic. These PacketCable Call Management Servers (CMS) are logically linked to several Signaling path Border Elements (SBEs) that hide the topology of the operator's network while providing specific access points for other peers. GlobalMSO does also control its egress routes by choosing the SBE that egress calls to a particular peer.

  Each CMS is also associated with one or more Location Routing Number(s) (LRN) due to local number portability support. The telephone numbers served by GlobalMSO are distributed among the CMS servers as well as the PSTN switches.

- MidwestMSO:

  MidwestMSO operates a network similar to GlobalMSO with services in the Chicago area (subscriber base only located in one state). Two PacketCable CMSes and only one SBE are deployed.

- NewYorkMSO:

  NewYorkMSO offers services in the region of New York City including voice and IM. Four SBEs have been deployed for voice, one SBE for IM. IM subscribers are known by a public identity.

Operators take advantage of ESPP to provision session establishment data. They agreed to use a common Registry function with a Registrar component that implements the client-side of the ESPP protocol. Note that ESPP also supports a deployment model where each operator has bilateral relationships with a number of peers and use ESPP directly with each peer.

Each operator is connected to the Registrar and implements an instance of the ENUM SIP addressing server (ESPP Server) for the server-side of the protocol. Note that GlobalMSO is connected to the Registrar from one of their Service Area that distributes data to the Boston Service Area.

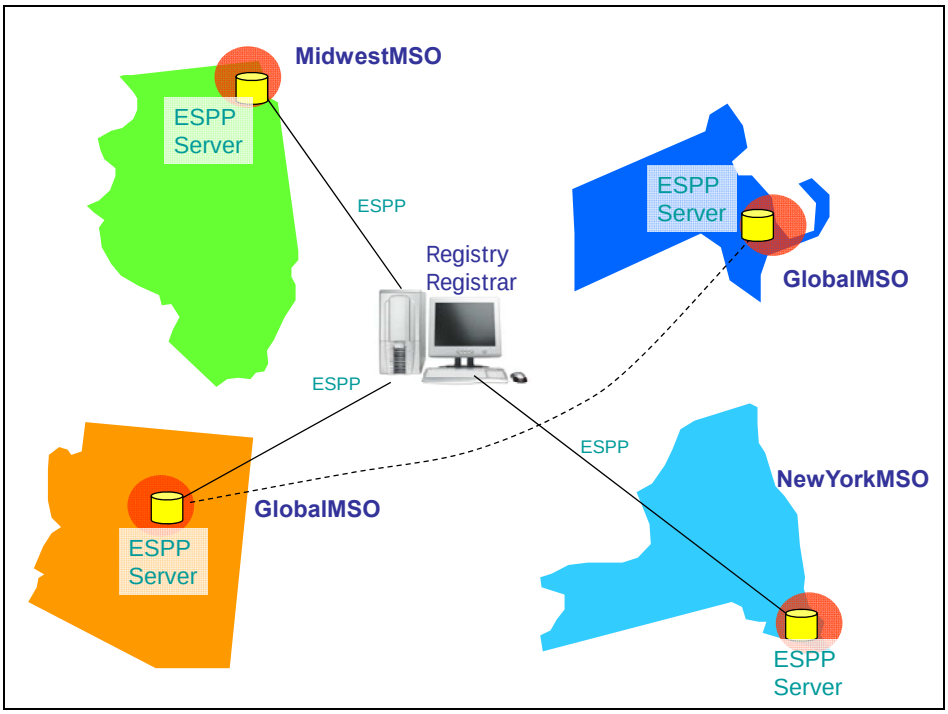

Figure 5 - Example of ESPP Deployment Scenario

Each cable operator is allocated an identifier or Enterprise ID:

MSO	Enterprise ID
GlobalMSO	76543
NewYorkMSO	76544
MidwestMSO	76545

After these three operators have successfully exchanged session establishment data (SED) for some time, a new operator called NewMSO wants to exchange calls and IM with the other three operators. To facilitate the process of getting the data from all three operators in a limited amount of time, the NewMSO server will receive a complete copy of the data the operators wish to share with NewMSO using a bulk file.

MSO	Enterprise ID
NewMSO	76546

In this scenario, each cable operator operates an ESPP Server that houses the ESPP Server's SED and end point data from an ESPP Client (a Registrar or a Registry). The ESPP Client initiates connections and, using the supported ESPP operations, pushes data down to the ESPP Servers. SED and end point objects are stored in the ESPP Server.

Each ESPP Server receives the data its operator is allowed to access and each addressing server has a "unique" view of its peer's data. Note that a cable operator may also choose to store internal or private routes using ESPP. For example, the data for routing sessions out of the operator's network is different from the data used to route sessions within the operators' own Service Areas.

6.3.1.1.2 Enterprises: ESPP Provisioning Operations

As these partner relationships are established, each operator sends SED data to its peers via the ESPP Client(s). In order to identify the peer the SED data belong to, an Enterprise ID is provided along with the list of Service Areas, SBEs, etc.

ESPP is an XML SOAP protocol. An ESPP Client uses the ESPP protocol's `addEntr` operation to create these enterprises in the ESPP Server for each participating operator. To do so, it sends a request to create three enterprise identifiers (`eid`) at an ESPP Server. This operation is sent to the ESPP Server operated by the three cable operators, MidwestMSO, GlobalMSO, and NewYorkMSO.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <addEntrRqst>
    <basicRqst>
     <clientId>7845601</transId>
     <transId>1000000</transId>
     <minorVer>1</minorVer>
    </basicRqst>
    <eid>76543</eid>
    <eid>76544</eid>
    <eid>76545</eid>
   </addEntrRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For brevity and to improve the readability of examples, the `basicRqst` element of the SOAP message will be omitted from the remaining examples. With the exception of the `transId` element incrementing upward, the content of the `basicRqst` element is identical for all examples in this section.

The enterprise ID for NewMSO is added later. At that time, the central registry via its ESPP Client uses the `addEntr` operation to add the NewMSO's enterprise ID to the ESPP Servers of the operators NewMSO wishes to exchange traffic with.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <addEntrRqst>
    <basicRqst> [snip] </basicRqst>
    <eid>76546</eid>
   </addEntrRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `addEntr` operation is an infrequent occurrence; it is used when new enterprises agree to exchange traffic.

6.3.1.2 Session Establishment Data (Routes, SBEs, and Service Areas)

6.3.1.2.1 Description of Session Establishment Data

As part of establishing their traffic exchange policies, each operator defines its points of session interconnect for their service areas, which may or may not vary for each peering partner. This information is part of the data required to establish a session and is called session establishment data (SED). As described in the introductory sections of this document, SED in ESPP includes Routes, SBEs, and Service Areas. SED is usually provisioned once for each cable operator with occasional subsequent updates as interconnect points are added or changed. It is provisioned independently from the provisioning of the elements contained in Service Areas (TNs, TN Ranges, LRN, IM IDs). This allows the rare process of provisioning SED to be distinctly separate from the continuous process of adding subscribers. The service areas supported by each cable operator are illustrated in Figure 6 below.

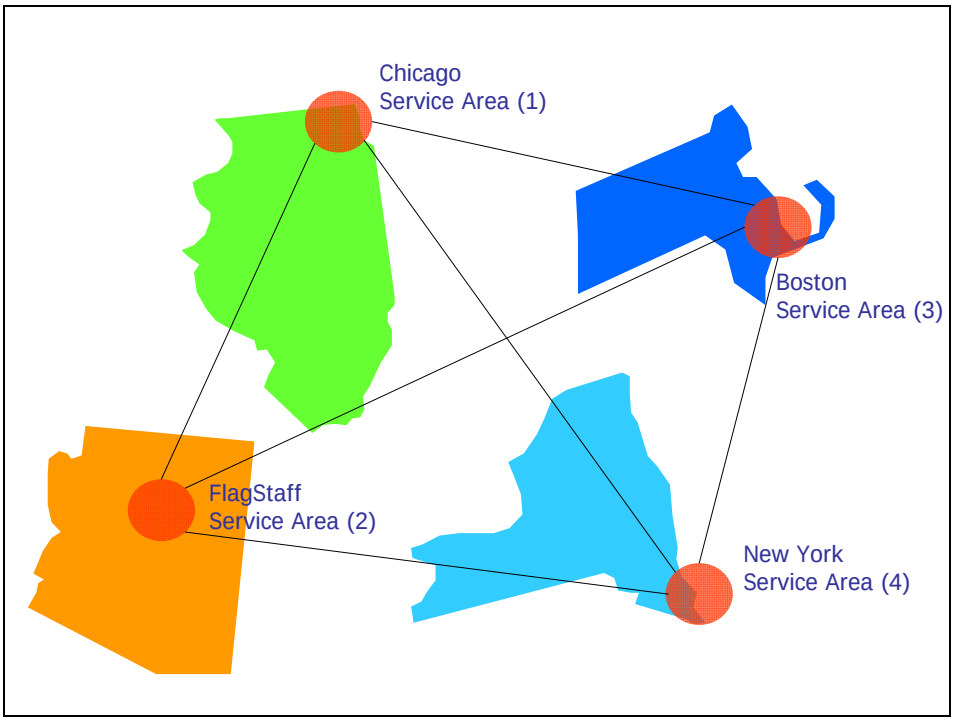

Figure 6 - Sample Scenario and Service Areas

The three initial cable operators offer voice and IM services. GlobalMSO has defined two Service Areas and MidwestMSO and NewYorkMSO have each defined one Service Area. Each Service Area is reachable via a Route associated with a set of SBEs.

The Figure 7 below further illustrates the details of the relationship between the Flagstaff and New York service areas operated by the GlobalMSO and NewYorkMSO respectively. The examples in this section focus on the relationship between GlobalMSO's Flagstaff service area and the NewYorkMSO's New York service area.

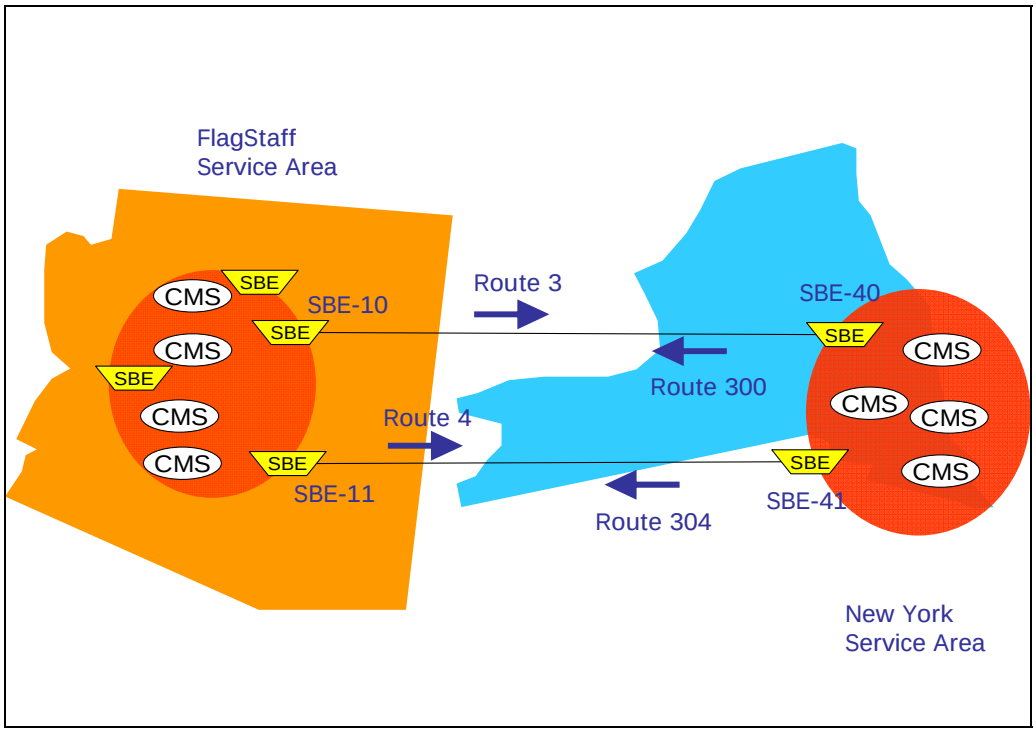

Figure 7 - Sample Scenario - Service Areas and SBEs

Each Route illustrated above is further defined in the Route table (Table 1 below) to contain one or more SBEs that serve as the ingress points for that Route. An MSO may wish to route messages, destined for a given peer's ingress route(s), out of its network through a specific egress host (aka "egress route") (see Table 1 below). There is more than one valid approach to encode this egress route information in a NAPTR. Valid approaches are – trunk group identifier in an ENUM NAPTR and route header in a SIP URI. The following example illustrates the former case, a trunk group identifier in an ENUM NAPTR. More specifically, the example illustrates an approach to specify an "egress regular expression rewrite" that can be applied to the regular expression from a destination peer's ingress NAPTR. The result of applying this "egress regular expression rewrite" to the peer's ingress NAPTR's regular expression is a new NAPTR regular expression that directs the egress routing out through the appropriate egress host.

Table 1 - Examples of Routes to SBEs Associations

Routes	SBE NAPTRs
RTFlag-NYC1 Ingress route to reach the New York Service Area from Flagstaff.	Order= 10, Pref=10, Svcs=E2U+SIP, Regx = "!^(.*)$!sip:\1@sbe40-newyork.newyorkmso.example1.com!"
RTFlag-NYC1-Egress Egress route to reach the New York Service Area from Flagstaff.	Order= 10, Pref=10, Svcs=E2U+SIP, RegxRewrite = "!^(.*@).(.*)$!sip:\1sbe10-globalmso.example2.com;tgrp=sbe40-newyork;trunk-context=example1.com!"
RTFlag-NYC2 Ingress route to reach the New York Service Area from Flagstaff.	Order= 10, Pref=10, Svcs= E2U+IM, Regx = "!^(.*)$!im:\1@sbe41-newyork.newyorkmso.example1.com!"

Routes	SBE NAPTRs
RTFlag-NYC2-Egress Egress route to reach the New York Service Area from Flagstaff.	Order= 10, Pref=10, Svcs=E2U+IM, RegxRewrite = "!^(.*@).(.*)$!im:\1sbe11-globalmso.example2.com;tgrp=sbe41-newyork;trunk-context=example1.com!"
RTNYC-Flag1 Ingress route to reach the Flagstaff Service Area from New York.	Order= 10, Pref=10, Svcs= E2U+SIP, Regx = "!^(.*)$!sip:\1@sbe10-globalmso.example2.com!"
RTNYC-Flag2 Ingress route to reach the Flagstaff Service Area from New York.	Order= 10, Pref=10, Svcs= E2U+IM, Regx = "!^(.*)$!im:\1@sbe11-globalmso.example2.com!"

The Routes defined above can then be associated with the service areas that they support. This allocation is laid out in the Table 2 below.

Table 2 - Example of Service Areas to Routes Associations

Service Areas To From↓	GlobalMSO Flagstaff	GlobalMSO Boston	MidwestMSO Chicago	NewYorkMSO New York
GlobalMSO Flagstaff		RTFlag-Boston	RTFlag-Chicago	RTFlag-NYC1 (and RTFlag-NYC1-Egress) RTFlag-NYC2 (and RTFlag-NYC2-Egress)
GlobalMSO Boston	RTBoston-Flag		RTBoston-Chic	RTBoston-NYC
MidwestMSO Chicago	RTChicago-Flag	RTChicago-Boston		RTChicago-NYC
NewYorkMSO New York	RTNYC-Flag1 RTNYC-Flag2	RTNYC-Boston	RTNYC-Chic	

6.3.1.2.2 ESPP Provisioning of Session Establishment Data

The Route, SBE, and Service Area data is provisioned in the appropriate ESPP Server of each participating cable operator using the `addRtes`, `addSvcAreas`, and `addNAPTRs` operations supported by the ESPP protocol. An optional approach is to combine some of the operations into a single ESPP request using the `batchUpdate` operation. The following example SOAP/XML messages provision the Routes, SBE NAPTRs, and Service Areas for the example above.

Using the `batchUpdate` operation, the ESPP Client provisions the **Global MSOs** Flagstaff SBE **NAPTRs, Routes, and Service Areas** in the NewYorkMSO's ESPP Server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <batchUpdateRqst>
    <basicRqst> [snip] </basicRqst>
```

```
    <batchUpdate>
     <op>
      <naptrAdd>
       <oid>7845601000012345610</oid>
       <eid>76543</eid>
       <order>10</order>
       <pref>10</pref>
       <flags>u</flags>
       <svcs>E2U+SIP</svcs>
       <regx>!^(.*)$!sip:\1@sbe10-globalmso.example2.com!</regx>
      </naptrAdd>
      <naptrAdd>
       <oid>7845601000012345611</oid>
       <eid>76543</eid>
       <order>10</order>
       <pref>10</pref>
       <flags>u</flags>
       <svcs>E2U+IM</svcs>
       <regx>!^(.*)$!im:\1@sbe11-globalmso.example2.com!</regx>
     </naptrAdd>
     <rteAdd>
      <oid>7845601000012345620</oid>
      <eid>76543</eid>
      <rteName>RTNYC-Flag1</rteName>
      <rrId>7845601000012345610</rrId>
      <isInSvc>true</isInSvc>
     </rteAdd>
     <rteAdd>
      <oid>7845601000012345621</oid>
      <eid>76543</eid>
      <rteName>RTNYC-Flag2</rteName>
      <rrId>7845601000012345611</rrId>
      <isInSvc>true</isInSvc>
     </rteAdd>
     <svcAreaAdd>
      <oid>7845601000012345630</oid>
      <eid>76543</eid>
      <saName>FlagstaffSA</saName>
      <rteId>7845601000012345620</rteId>
      <rteId>7845601000012345621</rteId>
     </svcAreaAdd>
    <op>
  </batchUpdateRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Using the alternative approach of a single ESPP operation for each object type (as opposed to the `batchUpdate` operation), the ESPP Client provisions the NewYorkMSO's **NAPTRs, Routes, and Service Areas** in the GlobalMSO's ESPP Server.

Provision the **New York MSO's** New York **NAPTRs** in the Global MSOs Flagstaff SBEs ESPP Server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <addNAPTRsRqst>
    <basicRqst> [snip] </basicRqst>
    <naptr>
     <oid>7845601000012345612</oid>
     <eid>76544</eid>
     <order>10</order>
     <pref>10</pref>
     <flags>u</flags>
     <svcs>E2U+SIP</svcs>
     <regx>!^(.*)$!sip:\1@sbe40-newyork.newyorkmso.example1.com!</regx>
    </naptr>
```

```
    <naptr>
     <oid>7845601000012345613</oid>
     <eid>76544</eid>
     <order>10</order>
     <pref>10</pref>
     <flags>u</flags>
     <svcs>E2U+IM</svcs>
     <regx>!^(.*)$!im:\1@sbe41-newyork.newyorkmso.example1.com!</regx>
   </naptr>
  </addNAPTRsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The ESPP Client provisions the **New York MSO's** New York **Routes** in the Global MSO's ESPP Server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <addRtesRqst>
    <basicRqst> [snip] </basicRqst>
    <rte>
     <oid>7845601000012345622</oid>
     <eid>76544</eid>
     <rteName>RTFlag-NYC1</rteName>
     <rrId>7845601000012345612</rrId>
     <isInSvc>true</isInSvc>
    </rte>
    <rte>
     <oid>7845601000012345623</oid>
     <eid>76544</eid>
     <rteName>RTFlag-NYC2</rteName>
     <rrId>7845601000012345613</rrId>
     <isInSvc>true</isInSvc>
    </rte>
   </addRtesRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The ESPP Client provisions the **New York MSO's** New York **Service Areas** in the Global MSO's ESPP Server.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <addSvcAreasRqst>
    <basicRqst> [snip] </basicRqst>
    <svcArea>
     <oid>7845601000012345631</oid>
     <eid>76544</eid>
     <saName>NewYorkSA</saName>
     <rteId>7845601000012345622</rteId>
     <rteId>7845601000012345623</rteId>
    </svcArea>
   </addSvcAreasRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

6.3.1.3 Public Identities, TN Ranges, and LRNs

6.3.1.3.1 Description of Public and Private Identities, TN Ranges, and LRNs

With the enterprises and SED data in place, operators have established the Service Areas they wish to exchange traffic to. Public identities such as telephone numbers (TN), ranges of TNs, LRNs, IM identifiers can now be provisioned into Service Areas. This scenario illustrates the provisioning of TNs, TN Ranges and LRNs for voice traffic exchanges, as well as the provisioning of TNs and IM identifiers for IM service.

A group of friends wishes to take advantage of their voice and IM services by having their calls within the group always use the most efficient technology. These friends are spread out between Boston, New York, Chicago and Phoenix and across three operators. Their public identities (TNs and IM IDs) are defined below. To simplify the scenario, the delivery of Instant Messages uses the same SBEs as the voice traffic. As messaging grows, there may be a need to separate the SBEs into voice and data portals which would create new SBE NAPTR records and, optionally, new Routes. Note that this is a technical example for the purpose of illustrating a protocol specification; end-user privacy considerations must apply.

Subscriber	Number	Service Area	Operator	IM Identity
Pat	928-774-5555	FlagstaffSA	GlobalMSO	imPat@globalmso.example2.com
Ashley	312-746-5555	ChicagoSA	MidwestMSO	imAshley@midwestmso.example3.com
Vince	718-330-5555	NewYorkSA	NewYorkMSO	imVince@newyorkmso.example1.com
Carl	617-414-5555	BostonSA	GlobalMSO	imCarl@globalmso.example2.com

These four subscribers have authorized their friends to see their public identities. The operators have also defined TN Ranges associated with the new VoIP services as shown below.

Service Area Name/ID	TN Range	
FlagstaffSA	928-774-5000	928-774-5999
ChicagoSA	312-746-5500	312-746-5600
NewYorkSA	718-330-5250	718-330-5590
	718-330-4000	718-330-4999
BostonSA	617-414-5555	617-414-5560

To support number portability, Location Routing Numbers (LRNs) have identified each of the CMSes within the network. Although there may be an LRN associated with each CMS, we show only the relevant sub-set applicable to this example:

CMS	Service Area	LRN
CMS-center	FlagstaffSA	928-774-1000
CMS-lakeStreet	ChicagoSA	312-746-1000
CMS-bronx	NewYorkSA	718-330-1000
CMS-fenway	BostonSA	617-414-1000

6.3.1.3.2 ESPP Provisioning of TNs, TN Ranges, LRNs and other types of identities

The Public Identity, TN Range and LRN data can then be provisioned in the appropriate ESPP Server of each MSO using the `addPvtIds, addPubIds, addTNRs,` and `addLRNs` operations supported by the ESPP protocol. An optional approach is to combine some of the operations into a single request using the `batchUpdate` operation. The following example SOAP/XML messages provision the data described above.

Using the `batchUpdate` operation, the ESPP Client provisions four **GlobalMSO** elements in the Flagstaff service area. This command is sent to the NewYorkMSO's ESPP Server and to any other applicable peer servers.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <batchUpdateRqst>
    <basicRqst> [snip] </basicRqst>
    <batchUpdate>
     <op>
      <pviAdd>
        <oid>7845601000012345650</oid>
        <eid>76544</eid>
        <pvtId>patsPrivateIdentifier</pvtId>
      </pviAdd>
      <piAdd>
        <oid>7845601000012345640</oid>
        <eid>76543</eid>
        <pubId>9287745555</pubId>
        <saId>7845601000012345630</saId>
        <pvtId>7845601000012345650</pvtId>
      </piAdd>
      <piAdd>
        <oid>7845601000012345641</oid>
        <eid>76543</eid>
        <pubId>imPat@globalmso.example2.com</pubId>
        <svcs>E2U+IM</svcs>
        <saId>7845601000012345630</saId>
        <pvtId>7845601000012345650</pvtId>
      </piAdd>
      <tnRAdd>
        <oid>7845601000012345642</oid>
        <eid>76543</eid>
        <tnRStrt>9287745000</tnRStrt>
        <tnREnd>9287745999</tnREnd>
        <saId>7845601000012345630</saId>
      </tnRAdd>
      <lrnAdd>
        <oid>7845601000012345643</oid>
        <eid>76543</eid>
        <rn>9287741000</rn>
        <saId>7845601000012345630</saId>
      </lrnAdd>
    </op>
   </batchUpdate>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Using the `batchUpdate` operation, the ESPP Client provisions four **NewYorkMSO elements** in the New York service area. This command is sent to the GlobalMSO's ESPP Server (and to any other applicable servers).

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
 <SOAP-ENV:Body>
   <batchUpdateRqst>
    <basicRqst> [snip] </basicRqst>
    <batchUpdate>
     <op>
      <pviAdd>
        <oid>7845601000012345651</oid>
        <eid>76544</eid>
        <pvtId>vincesPrivateIdentifier</pubId>
      </pviAdd>
      <piAdd>
        <oid>7845601000012345644</oid>
        <eid>76544</eid>
        <pubId>7183305555</pubId>
        <saId>7845601000012345631</saId>
        <pvtId>7845601000012345651</pvtId>
      </piAdd>
      <piAdd>
```

```
        <oid>7845601000012345645</oid>
        <eid>76544</eid>
        <pubId>imVince@newyorkmso.example1.com</pubId>
        <svcs>E2U+IM</svcs>
        <saId>7845601000012345631</saId>
        <pvtId>7845601000012345651</pvtId>
      </piAdd>
      <tnRAdd>
        <oid>7845601000012345646</oid>
        <eid>76544</eid>
        <tnRStrt>7183305250</tnRStrt>
        <tnREnd>7183305590</tnREnd>
        <saId>7845601000012345631</saId>
      </tnRAdd>
      <lrnAdd>
        <oid>7845601000012345647</oid>
        <eid>76544</eid>
        <rn>7183301000</rn>
        <saId>7845601000012345631</saId>
      </lrnAdd>
    </op>
   </batchUpdate>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

6.3.2 Deployment Scenario and Resolution

6.3.2.1 Initiating an Outgoing Call

Initiating a call from Pat to Vince, the CMS hosting Pat launches an ENUM query to the GlobalMSO addressing server. The addressing server takes the E.164 address and examines the Public Identities to see if the specific number is set in the server.

Vince's number is found in the New York Service Area. This Service Area is associated with the Routes RTFlag-NYC1 and RTFlag-NYC2 that link Flagstaff with New York City. Each of these Routes has a single SBE NAPTR record. The addressing server would respond to the CMS with both NAPTR records. The CMS then attempts to complete the call to the NewYorkMSO by sending the call setup to the address(es) found in the NAPTR record. The detailed steps involved in this process are:

1. The CMS normalizes the TN to an e.164 format and queries the GlobalMSO ESPP Server, which results in the following response:

   ```
   $ORIGIN 5.5.5.5.0.3.3.8.1.7.1.mso-example.
         NAPTR 10 100 "u" "E2U+SIP" "!^(.*)$!sip:\1@sbe10-
   globalmso.example2.com;tgrp=sbe40-newyorkmso;trunk-context=example1.com!"
         NAPTR 10 101 "u" "E2U+IM" "!^.*$!im:imvince@sbe11-
   globalmso.example2.com;tgrp=sbe41-newyorkmso;trunk-context=example1.com!"
   ```

2. Assuming that the CMS is interested in the SIP service, as it is initiating a telephone call, the CMS applies the regular expression rules to the NAPTR obtained in step 1 and places the result into the SIP Request URI:

   ```
   INVITE sip:17183305555@sbe10-globalmso.example2.com;tgrp=sbe-40-newyork;trunk-
   context=example1.com SIP/2.0
   ```

   (Other SIP INVITE content not included for clarity purposes.)

3. CMS sends the SIP INVITE to Vince via the its egress SBE node sbe10-globalmso.example2.com

4. Now this SBE based on its destination trunk group mapping, gets the ingress SBE in the peer network as sbe40-newyork.newyorkmso.example1.com. The SBE now executes a NAPTR query to the DNS server for "sbe40-newyork.newyorkmso.example1.com " resulting in:

   ```
   sbe40-newyork.newyorkmso.example1.com
     order pref flgs  service  regx  replacement
   ```

```
IN NAPTR 100  50  "s"    "SIP+D2U" ""   _sip._udp.sbe40-
newyork.newyorkmso.example1.com
```

5. The SBE now sends a DNS SRV query for _sip._udp.sbe40-newyork.newyorkmso.example1.com which results in:

```
$ORIGIN _sip._udp.sbe40-newyork.newyorkmso.example1.com
```

(Some content not included for clarity purposes.)

```
IN SRV  0  1   5060   sbe40-newyork.newyorkmso.example1.com
;; ANSWER SECTION
sbe40-newyork.newyorkmso.example1.com   A   192.0.2.17
sbe40-newyork.newyorkmso.example1.com   A   192.0.2.18
```

6. Now the SBE rewrites the SIP Request URI to:

```
INVITE sip:17183305555@sbe40-newyork.newyorkmso.example1.com SIP/2.0
```

(Other SIP INVITE content not included for clarity purposes.)

7. The SBE sends the SIP INVITE to Vince via the ingress SBE with associated IPv4 address 192.0.2.17 to port 5060 using UDP as the transport protocol.

6.3.2.2 Receiving an Incoming Call

Taking the call scenario described above, once the call request arrives in NewYorkMSO's network the SBE within NewYorkMSO queries its own ESPP Server and finds a match for Mary's number in the TN Range records (718-330-5250 to 718-330-5590). The NewYorkMSO ESPP Server has been provisioned with intra-domain routes so that the SBE can locate the proper CMS. The NewYorkMSO ESPP Server contains the New York Service Area that is associated with a intra-domain Route that is not shared with the other operators. This allows the internal routing topology of an operator to remain private to the operator. There is only one route for this range of phone numbers with only one NAPTR record. The data used to resolve the query looks like:

```
TN Range:     718-330-5250 to 718-330-5590
     Service Area: New York
         Route: RTNYC-Private
      NAPTR 10 101 "u" "E2U+SIP" "!^(.*)$!sip:\1@cms-bronx.newyorkmso.example1.com!"
```

6.3.2.2.1 Query using the Location Routing Number

If the number has been ported, there are a number of cases where the query may also contain an Location Routing Number. The Location Routing Number is a reference number for one or more telephone switches where previously the first 6 digits of the phone number clearly identified the switch.

Once the call request arrives in NewYorkMSO's network the SBE within NewYorkMSO queries its own ESPP Server and finds a match for the LRN (718-330-1000) in the New York Service Area. The rest of the routing lookup is similar to what is described for the telephone number above.

```
LRN:     718-330-1000
     Service Area: New York
         Route: RTNYC-Private
      NAPTR 10 101 "u" "E2U+SIP" "!^(.*)$!sip:\1@cms-bronx.newyorkmso.example1.com!"
```

6.3.2.2.2 Growth of the Operator Group

When NewMSO is ready to exchange traffic, their ESPP Server receives a batch download the latest data from all its peers via the Registrar. The Registrar takes an image of the current data that each peer has allowed NewMSO to view and performs a secure file transfer to the NewMSO addressing server. Following the download, the NewMSO ESPP Server connects to the Registrar and begin processing real-time updates to ensure that any changes made since creating the batch download files are made in the addressing server.

6.3.3 Generic Examples

Additional examples are available in Appendix I, Generic ESPP Examples.

6.4 Protocol Specification

This section describes the general concepts that apply to the protocol's operations. It also describes each individual protocol operation, its input and output objects, and its functional characteristics. The complete normative specification of the protocol (the WSDL and XSD definitions) can be found in Section 9, Formal API Definition, in this document. Note that if any specification conflicts exist, the normative definitions contained in Section 9 take precedence over the validation rules defined in Section 10 which take precedence over this section.

6.4.1 General Protocol Concepts

The following sub-sections describe the object structures and concepts that are applicable to many of the protocol operations.

6.4.1.1 Basic Request Object

All ESPP operations that may potentially modify persistent protocol objects require a `BasicRqst` object to be passed in, in addition to other object types that are specific to each operation. The basic request object contains a client ID, a transaction ID, the minor version of the protocol, and an optional extension element. Each of the values is described in the subsequent sub-sections.

```
<complexType name="BasicRqstType">
	<sequence>
		<element name="clientId" type="est:ClientIdType"/>
		<element name="transId" type="est:TransIdType"/>
		<element name="minorVer" type="est:VerType"/>
		<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
	</sequence>
</complexType>

<simpleType name="ClientIdType">
	<restriction base="int"/>
</simpleType>

<simpleType name="VerType">
	<restriction base="unsignedLong"/>
</simpleType>
```

6.4.1.2 Basic Query Object

All ESPP "get" operations require a `BasicQueryRqst` object to be passed in, in addition to other object types that are specific to each get operation. The basic query object contains a client ID, the minor version of the protocol, and an optional extension element. Each of the values is described in the subsequent sub-sections. Notice that no transaction ID is necessary for query operations.

```
<complexType name="BasicQueryType">
	<sequence>
		<element name="clientId" type="est:ClientIdType"/>
		<element name="minorVer" type="est:VerType"/>
		<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
	</sequence>
</complexType>
```

6.4.1.3 Authentication

For each ESPP connection, the ESPP Client and ESPP Server MUST use HTTP Digest based authentication as defined in [RFC 2617]. The HTTP Digest authentication SHOULD be performed once per connection. To further support authentication and operational activities, such as auditing, each ESPP operation requires a client ID as input.

6.4.1.4 Transaction ID

Each ESPP Client maintains a transaction ID counter for each ESPP Server that it is configured to update. Similarly, each ESPP Server maintains the transaction ID for each ESPP Client from which it receives updates. These transaction IDs are maintained in persistent storage on both client and server. The protocol data element that contains the transaction ID for each operation is declared as follows:

```
<element name="transId" type="est:TransIdType"/>
       <simpleType name="TransIdType">
              <restriction base="unsignedLong"/>
       </simpleType>
```

A transaction ID is sent with each update request to an ESPP Server. The transaction ID MUST be present in a protocol request operation whose purpose is to modify data within an ESPP Server. The ESPP Client and ESPP Server must implement the transaction ID as a 64-bit value.

The initial value for the transaction ID on the ESPP Client MUST either be chosen randomly, start at zero, or be configured by the operator before protocol processing or bootstrap file generation begins.

When generating an API based update request destined for an ESPP Server, the ESPP Client MUST use the current transaction ID for the target server, which is one increment greater than the previous successful transaction ID (or the first transaction ID in the counter if this is the first request generated for the given ESPP Client/server pair). After receiving a successful response from an API base request, the client increments the transaction ID for that server by one. When the `'isFullResync'` flag is set in the `batchFileUpdateRqst`, the ESPP Server MUST set the transaction ID for that client to the value received in the `batchFileUpdateRqst`.

When receiving a transaction ID in a request, the ESPP Server compares it to its current transaction ID value. If the transaction ID from the client is one greater than the transaction ID from the server, the request is processed normally. If the request succeeds on the server, the ESPP Server MUST update its transaction ID to be the one received from the client. Once all database changes have been synchronized to persistent storage, the server MUST respond with a successful result code. When the client receives the response from the server indicating that the request has succeeded, the client MUST increment its transaction ID by one.

If the server receives a request from a client with a transaction ID that is the same as server's current transaction ID for that client, the server compares the content of the request received from the client to the content of the request most recently received from that client. If the content is identical, this indicates that the client failed to receive the response to the previous transaction. The client is assumed to be retrying that transaction that the server has already successfully processed. In this event the server SHOULD NOT modify its database. The ESPP Server MUST simply return the identical response that it returned in response to the clients first attempt to process the request/response.

In the case that the transaction ID is the same but the request content is different, or the transaction ID from the client is different but is not one greater than the transaction ID for that client on the server, the server and client are out of sync. In this event, the ESPP Server SHOULD return the appropriate error code (Transaction ID out of sequence). The ESPP Client MUST automatically suspend protocol updates to that server and raise a critical alarm. Manual intervention is likely required at this point to analyze and clear the error, perhaps by initiating a full re-synchronization of the client's data store, and then re-enabling real-time protocol communication to that ESPP Server.

In cases where the server determines the transaction ID to be correct, there are two general categories of errors: system errors and business errors. In the event a system error occurs (e.g., disk space has been exhausted) the ESPP Server MUST return the appropriate error message (System Unavailable Error or Authentication Error), and the ESPP Client SHOULD suspend real-time updates to the server for some period of time configured by the operator, and then resume after that time has elapsed. In this case the ESPP Client MUST not update the transaction ID as transaction IDs are only consumed/updated by the client on successful responses to a given API request or when a file based update is successfully generated by the client.

In the event a business error occurs (e.g., a data element fails the regular expression validation defined in the Data Element Validation Constraints, Section 10, of this document), the ESPP Server MUST return the appropriate error code for the given error and operation. In response to this type of error response the client MUST, again, not consume that transaction ID, because the transaction did not complete successfully. The client MUST suspend real-time protocol communication with that ESPP Server and generate an internal alarm. Manual intervention is required to analyze and rectify the erroneous data that caused the error and then re-enable real-time protocol communication with the ESPP Server.

6.4.1.5 Version Identification

The version of the ESPP protocol in use is defined by the combination of the major version ID embedded in the namespace of the XSD and the minor version ID that is passed in with each request as an element of the `BasicRqstType` object.

A major version ID will change when a change is made to the protocol that is not backward compatible. The minor version ID may change when a change is made to the protocol that is backward compatible. The version ID passed in by the ESPP Client MUST indicate the version that the client supports; it SHOULD be the version that the client believes that the ESPP Server supports. When a message arrives at the server, the server MUST evaluate the version indicator passed in by the ESPP Client and it MUST return the appropriate error code and message if the version is not one that is supported by the server. Refer to the list of response codes for the appropriate code to return in this circumstance.

6.4.1.6 Basic Response Object

The request/input object type is distinct for each operation. However, most operations of the ESPP protocol return the same response/output object type, `CmnRspnsMsg`. In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of this response message is declared as follows:

```
<wsdl:message name="cmnRspnsMsg">
      <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>

<complexType name="BasicRspnsType">
      <sequence>
            <element name="resCode" type="int"/>
            <element name="resMsg" type="string"/>
            <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
      </sequence>
</complexType>
```

The resCode element is a response code and the `resMsg` is the corresponding response message. The BasicRspnsType may also be extended using the ext element. Each of these elements is further described in subsequent sections.

6.4.1.7 Object ID and Enterprise ID

As described in the Logical Structure section of this document, each object instance in ESPP has a unique identifier, its `oid`, and the identifier of its "owning" operational entity, its `eid`. These two data elements are defines as follows:

```
<simpleType name="OIdType">
      <restriction base="unsignedLong"/>
</simpleType>

<simpleType name="EIdType">
      <restriction base="string"/>
</simpleType>
```

These two fields are further defined as follows:

- `oid:` Each object is uniquely identifiable using its object ID. An object ID is guaranteed to be unique within an ESPP Server and ESPP Client, regardless of the number of ESPP Clients populating objects into a given ESPP Server. An ESPP Client MUST create and assign the Object IDs it provisions. This approach has significant performance advantages. The management of the unique name space for ESPP object IDs and the approach for creation and assignment of object IDs is defined as follows:
  - ESPP Clients and Servers MUST implement the oid attribute as an unsigned 64-bit value, unsigned long. Unsigned longs support up to 20 digits, but refer to the Section 10 where the maximum allowable value of an unsigned long is documented.
  - The eight most significant digits of the object ID MUST contain a client ID that is guaranteed to be globally unique within an ESPP federation. This client ID MUST be right justified in this 8 digit field, leaving any unused most significant digits empty. The client ID MUST be the combination of two parts, a one to six digit organization ID and a two digit suffix.
    - The organization ID portion of the client ID MUST be a number acquired from the IANA Private Enterprise Number (PEN) registry, http://www.iana.org/assignments/enterprise-numbers. The upper limit on the value for organization ID is limited to 184,466 to align with the maximum allowed value for the upper 6 digits in a 64-bit unsigned integer.
    - The two digit suffix MUST be used to further uniquely identify an ESPP Client software within the organization represented by the organization ID. The 2 digit suffix is necessary in cases where more than one ESPP Client, within the same ESPP federation, are from the same organization and, therefore, would have the same organization ID. The suffix MUST be within the range of 00 and 99, inclusive. Prior to introducing an ESPP Client into an ESPP federation, the organization that operates that ESPP Client acquires an organization ID, adds an appropriate two digit suffix. The resulting client ID will then be used to formulate the object Ids for the ESPP objects created by that client software.
  - The remaining, right hand, twelve digits of the object ID MUST contain an object ID that is guaranteed to be unique within the client identified by the client ID.
  - As an example, a valid object ID would be 123401000012345678. In this example, 1234 is the organization ID of the organization that manages the ESPP Client implementation, 01 is the client ID suffix, and 000012345678 is the object ID that is unique within that ESPP client. Notice that the two most significant digit fields are not used and are not necessary in this example. The result is an object ID that is unique within the ESPP federation.
- `eid:` The enterprise ID data element that resides as a data element within each ESPP object uniquely identifies an operational entity within an ESPP federation that is responsible for, or owns, that object from a business perspective. This is distinct from the client ID, which identifies the piece of software that performance the mechanics of provisioning an ESPP object within an ESPP client. The globally unique name space for the assignment of an enterprise ID is the IANA Private Enterprise Number (PEN) registry, http://www.iana.org/assignments/enterprise-numbers. Prior to contributing its addresses and routing information to an ESPP federation an enterprise acquires a PEN, which will then reside in the eid field of each ESPP object that and ESPP Client provisions on behalf of that enterprise.

6.4.1.8 Response Codes and Messages

The response to each protocol operation includes exactly one response code and response message. The response codes that each operation is permitted to return are listed in each sub-section that describes each operation. And the consolidated list of response codes and messages are listed in the "Response Codes and Messages" section of this document.

All real-time API operations of the ESPP protocols MUST be atomic, all aspects of the operation will fail or all will succeed. An operation should never result in a "partial success". As a result, the response code returned by any given operation call indicates whether the operation succeeded or failed. Each failure reason has a unique failure response code and message, and only one may be returned in response to any given operation call. The protocol object that houses the response code and message is called `BasicRspnsType`.

6.4.1.9 Extensibility

Extensibility allows a given ESPP Client or ESPP Server to supplement ESPP messages with additional object or data elements that may be appropriate for a given ESPP deployment or operational scenario. ESPP is implicitly and explicitly extensible. Implicit extensibility is supported by the fact that XSD and WSDL definitions may be extended without breaking backward compatibility. New WSDL operations can be added and new XSD data types can be added without breaking backward compatibility. Explicit extensibility is supported through the use of the XSD "any" type. Each object type definition in ESPP contains an additional attribute called "ext", which is of type `ExtAnyType`, which contains one or more an XSD "any" elements. Furthermore, the `BasicRqstType` and `BasicRspnsType` also contain an "ext" element to allow the addition of objects of any type to be added to the request and response objects that are used for each ESPP operation. And ESPP Server may dynamically evaluate the version ID of any given request and choose to accept or return these optional attributes from or to an ESPP Client.

```
<complexType name="ExtAnyType">
      <sequence>
            <any namespace="##other" maxOccurs="unbounded"/>
      </sequence>
</complexType>
```

6.4.2 Protocol Operation Descriptions

The following sub-sections describe each individual protocol operation in detail, its input and output objects, and its functional characteristics.

6.4.2.1 Operation Name: addRtes

As described in the introductory sections, a Route represents a collection of session-level peering points named Signaling path Border Elements (SBEs). Routes are linked to Service Areas to establish the link between a set of public identities and their SBEs. It is this indirect linking of public identities to SBEs that significantly improves the scalability and manageability of the peering data. Additions and changes to session-level peering points are reduced to a single data distribution operation of a Route or Service Area in an addressing server, rather than millions of data distribution updates to public identity records that individually contain their peering point data.

The `addRtes` operation creates or overwrites one or more Routes in the addressing server. If a Route with the given Route ID does not exist, then the ESPP Server MUST create the route. If a Route with the given Route ID does exist, then the ESPP Server MUST replace the current properties of the Route with the properties passed into the `addRtes` operation. If a Route with the given ID does exist, but was created by a client other than the one calling the `addRtes` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addRtes">
      <wsdl:input message="addRtesRqstMsg"/>
      <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addRtesRqstMsg">
      <wsdl:part name="rqst" element="est:addRtesRqst"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `addRtesRqst` object, which is declared as follows:

```
<element name="addRtesRqst">
      <complexType>
            <sequence>
                  <element name="basicRqst" type="est:BasicRqstType"/>
                  <element name="rte" type="est:RteType"
maxOccurs="unbounded"/>
            </sequence>
      </complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. One or more `RteType` objects are also required. Any limitation on the maximum number of routes that may be passed into a call to the `addRtes` operation is a policy decision and is not limited by the protocol. The `RteType` object structure is declared as follows:

```
<complexType name="RteType">
      <sequence>
            <element name="oid" type="est:OIdType"/>
            <element name="eid" type="est:EIdType"/>
            <element name="rteName" type="string"/>
            <element name="rrId " type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
            <element name="isInSvc" type="boolean"/>
            <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
      </sequence>
</complexType>
```

The `RteType` object is composed of the following data elements:

- `oid:` Exactly one object ID to uniquely identify the object instance. This identifier is also an input to the `delRtes, getRtes,` and `addSvcAreas` operations to identify a given route.
- `eid:` Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for, or "owns" this object.
- `rteName:` Exactly one human readable name of the Route.
- `rrId:` Zero or more Resource Record identifiers. This is the set of Resource Record identifiers such as `NAPTRType` objects or NSType objects that are contained within the Route.
- `isInSvc:` Toggle-able boolean value that sets the Route in service or out of service. The ESPP Server MUST NOT include the NAPTR record(s) associated with an out-of-service route in the response to a resolution request for a telephone number that is associated with the Route that contains that border element.
- `ext`: Point of extensibility described in Section 6.4.1.9 of this document.

The following Table 3 defines the result codes and messages that the `addRtes` operation SHOULD return.

Table 3 - addRtes Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.2 Operation Name: addSvcAreas

As described in the introductory sections, a Service Area represents a collection of public identities that are linked to a set of Routes. Routes are linked to Service Areas to establish the link between a set of public identities and their session-level peering points. The `addSvcAreas` operation creates or overwrites one or more Service Areas in the addressing server. If a service area with the given service area ID does not exist, then the ESPP Server MUST create the service area. If a service area with the given identity does exist, then the ESPP Server MUST replace the current properties of that service area with the properties passed into the `addSvcAreas` operation. If a service area with the given ID does exist, but was created by a client other than the one calling the `addSvcAreas` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addSvcAreas">
       <wsdl:input message="addSvcAreasRqstMsg"/>
       <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addSvcAreasRqstMsg">
       <wsdl:part name="rqst" element="est:addSvcAreasRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
       <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the addSvcAreasRqst object, which is declared as follows:

```
<element name="addSvcAreasRqst">
	<complexType>
		<sequence>
			<element name="basicRqst" type="est:BasicRqstType"/>
			<element name="svcArea" type="est:SvcAreaType"
maxOccurs="unbounded"/>
		</sequence>
	</complexType>
</element>
```

As with all update operations, the basicRqst data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more SvcAreaType objects. Any limitation on the maximum number of service areas that may be passed into a call to the addSvcAreas operation is a policy decision and is not limited by the protocol.

The SvcArea object structure is declared as follows:

```
<complexType name="SvcAreaType">
	<sequence>
		<element name="oid" type="est:OIdType"/>
		<element name="eid" type="est:EIdType"/>
		<element name="saName" type="string"/>
		<element name="rteId" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
		<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
	</sequence>
</complexType>
```

The data elements of the SvcAreaType are described as follows:

- oid: Exactly one object ID that provides the identity of the object. This identifier is also an input to other operations that must refer to a Service area, such as delSvcAreas, getSvcAreas, and addPubIds operations to identify a given service area.
- eid: Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for, or "owns" this object.
- saName: Exactly one human readable name of the service area.
- rteID: The identity of zero or more routes that the Public Identities, LRNs, and TN Ranges in this service area can be routed through.
- ext: Point of extensibility described in Section 6.4.1.9 of this document.

The ESPP Server MUST reject any addServiceAreas operation call that attempts to create a service area with a route ID that does not exist within the context of the addressing server.

The following Table 4 defines the result codes and messages that the addSvcAreas operation SHOULD return.

Table 4 - addSvcAreas Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.

Result Code	Text
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.3 Operation Name: addPubIds

As described in the introductory sections, a Public Identity object represents a public addressable identity (such as a telephone number, email address, or some other addressable string). It is linked to SBEs and NAPTRs (indirectly via a Service Area ID), and, alternatively, to directly associated NAPTRs. The ESPP protocol allows a Public Identity to have a relationship to a Service Area and/or a set of NAPTRs.

A Public Identity may also be associated with a Private Identity object. The `addPubIds` operation creates or overwrites one or more PubId objects in the addressing server. If a PubId object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the PubId object. If such an object does exist within the context of the addressing server, then the ESPP Server MUST replace the current properties of that object with the values passed into the `addPubIds` operation. If a Public Identity with the given ID does exist, but was created by a client other than the one calling the `addPubIds` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addPubIds">
	<wsdl:input message="esp:addPubIdsRqstMsg"/>
	<wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addPubIdsRqstMsg">
	<wsdl:part name="rqst" element="est:addPubIdsRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
	<wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `addPubIdsRqst` object, which is declared as follows:

```
<element name="addPubIdsRqst">
	<complexType>
		<sequence>
			<element name="basicRqst" type="est:BasicRqstType"/>
			<element name="pi" type="est:PubIdType"
maxOccurs="unbounded"/>
		</sequence>
	</complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more `PubIdType` objects. Any limitation on the maximum number of Public Identities that may be passed into a call to the `addPubIds` operation is a policy decision and is not limited by the protocol.

The `PubIdType` object structure is declared as follows:

```
<complexType name="PubIdType">
      <sequence>
            <element name="oid" type="est:OIdType"/>
            <element name="eid" type="est:EIdType"/>
            <element name="pubId" type="string"/>
            <element name="svcs" type="string" minOccurs="0"/>
            <element name="saId" type="est:OIdType" minOccurs="0"/>
            <element name="naptrId" type="est:OIdType" minOccurs="0"
      maxOccurs="unbounded"/>
            <element name="pvtId" type="est:OIdType" minOccurs="0"/>
            <element name="npParams" type="est:NPParamsType" minOccurs="0"/>
      <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
      </sequence>
</complexType>

<complexType name="NPParamsType">
      <sequence>
            <element name="npdi" type="boolean"/>
            <element name="rn" type="string" minOccurs="0"/>
      </sequence>
</complexType>
```

The `PubIdType` is comprised of the following data elements:

- `oid:` Exactly one object ID that provides the identity of the object.
- `eid:` Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for or "owns" this object.
- `pubId:` Exactly one public identity that is a resolvable identity within the context of the addressing server. This identity may, for example, be a TN, an email address, or some other resolvable identity.
- `svcs:` Zero or one ENUM service type associated with the public identity. This field's value must be of the form specified in [RFC 3761] (e.g., E2U+pstn:sip+sip). The allowable values are a matter of policy and not limited by this protocol.
- `saId:` Zero or one object ID that identifies the Service Area this public identity object instance resides within. The Service Area's relationship to one or more Routes can then define the peering points of Public Identities. The ESPP Server MUST reject, with the appropriate error response code, an attempt to create a public identity object containing a Service Area ID that does not exist within the appropriate context of the addressing server.
- `naptrId:` Zero or more NAPTRs. NAPTR objects directly associated with a public identity can be a lower level, more data distribution intensive approach to associating a telephone number with its peering points.
- `pvtId:` Zero or one object ID that optionally identified a private identity to which this public identity is associated. This provides the ability to create a "related" set of public identities. The ESPP Server MUST return an error response if the Private Identity object identified by this PvtId does not exist within the context of the addressing server.
- `npParams:` Zero or one object of type NPParamsType. The npdi element of the NPParamsType is a boolean that indicates whether or not a number portability dip has been performed on this public identity. If a number portability dip has been performed and an RN was discovered, then the "rn" field of the NPParamsType contains that RN. If the NPParamsType object is not present within the PubIdType then the ESPP server SHOULD assume that the number portability dip has not been performed on that public identity.
- `ext:` Point of extensibility described in Section 6.4.1.9 of this document.

The ESPP protocol does not specifically disallow the creation of Public Identities that have no associated NAPTRs and no associated Service Area.

The following Table 5 defines the result codes and messages that the `addPubIds` operation SHOULD return.

Table 5 - addPubIds Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.4 Operation Name: addPvtIds

As described in the introductory sections, a Private Identity object represents a string of numbers or characters that serves to associate, or group together, a set of related Public Identities. The `addPvtIds` operation creates or overwrites one or more PvtId objects in the addressing server. If a PvtId object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the PvtId object. If such an object does exist within the context of the addressing server, then the ESPP Server MUST replace the current properties of that object with the values passed into the operation. If a Private Identity with the given ID does exist, but was created by a client other than the one calling the `addPvtIds` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addPvtIds">
       <wsdl:input message="esp:addPvtIdsRqstMsg"/>
       <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addPvtIdsRqstMsg">
       <wsdl:part name="rqst" element="est:addPvtIdsRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
       <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `addPvtIdsRqst` object, which is declared as follows:

```
<element name="addPvtIdsRqst">
       <complexType>
              <sequence>
                     <element name="basicRqst" type="est:BasicRqstType"/>
```

```
                        <element name="pvi" type="est:PvtIdType"
maxOccurs="unbounded"/>
                    </sequence>
                </complexType>
        </element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more `PvtIdType` objects. Any limitation on the maximum number of PvtIds that may be passed into a call to the operation is a policy decision and is not limited by the protocol.

The `PvtIdType` object structure is declared as follows:

```
        <complexType name="PvtIdType">
                <sequence>
                        <element name="oid" type="est:OIdType"/>
                        <element name="eid" type="est:EIdType"/>
                        <element name="pvtId" type="string"/>
                        <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
                </sequence>
        </complexType>
```

The `PvtIdType` is comprised of the following data elements:

- `oid:` Exactly one object ID that provides the identity of the object.
- `eid:` Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for or "owns" this object.
- `pvtId:` Exactly one private identity that serves to associate, or group together, a set of related Public Identities.
- `ext:` Point of extensibility described in Section 6.4.1.9 of this document.

The following Table 6 defines the result codes and messages that the `addPvtIds` operation SHOULD return.

Table 6 - addPvtIds Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.5 Operation Name: addLRNs

As described in the introductory sections, an LRN object represents a routing number and its indirect relationship to one or more SBEs (via a Service Area ID).

The `addLRNs` operation creates or overwrites one or more LRN objects in the addressing server. If a LRN object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the LRN object. If an LRN object with the same object ID does exist within the context of the addressing server, then the ESPP Server MUST replace the current properties of that LRN with the properties passed into the `addLRNs` operation. If an LRN with the given ID does exist, but was created by a client other than the one calling the `addLRNs` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addLRNs">
      <wsdl:input message="addLRNsRqstMsg"/>
      <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addLRNsRqstMsg">
      <wsdl:part name="rqst" element="est:addLRNsRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
      <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `addLRNsRqst` object, which is declared as follows:

```
<element name="addLRNsRqst">
      <complexType>
            <sequence>
                  <element name="basicRqst" type="est:BasicRqstType"/>
                  <element name="lrn" type="est:LRNType"
maxOccurs="unbounded"/>
            </sequence>
      </complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more `LRNType` objects. Any limitation on the maximum number of LRNs that may be passed into a call to the `addLRNs` operation is a policy decision and is not limited by the protocol.

The LRN object structure is declared as follows:

```
<complexType name="LRNType">
      <sequence>
            <element name="oid" type="est:OIdType"/>
            <element name="eid" type="est:EIdType"/>
            <element name="rn" type="string"/>
            <element name="saId" type="est:OIdType"/>
            <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
      </sequence>
</complexType>
```

`LRNType` is comprised of the following data elements:

- `oid:` Exactly one object ID that provides the identity of the object. This identifier is also an input to the delLRNs, getLRNs operations to identify a given LRN.
- `eid:` Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for or "owns" this object.
- `rn:` One local routing number.
- `saId:` Exactly one object ID that identifies the Service Area in which this LRN resides. The Service Area's relationship to one or more Routes can then define the LRNs peering points. The ESPP Server MUST reject, with the appropriate error response code, an attempt to create an LRN object containing a Service Area ID that does not exist within the appropriate context of the addressing server.
- `ext:` Point of extensibility described in Section 6.4.1.9 of this document.

The following Table 7 defines the result codes and messages that the `addLRNs` operation SHOULD return.

Table 7 - addLRNs Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.6 Operation Name: addTNRs

As described in the introductory sections, a `TNRType` is a range of telephone numbers which are provisioned as a group, reducing the number of objects that must be independently provisioned and managed. This object contains the indirect relationship to a set of peering point NAPTRs (via a Service Area ID). The `addTNRs` operation creates or overwrites one or more TNR objects in the addressing server. If a TNR object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the TNR object. If such an object does exist within the context of the addressing server, then the ESPP Server MUST replace the current properties of that object with the values passed into the `addTNRanges` operation. If a TNRange with the given ID does exist, but was created by a client other than the one calling the `addTNRs` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
            <wsdl:operation name="addTNRs">
            <wsdl:input message="esp:addTNRsRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
      </wsdl:operation>

<wsdl:message name="addTNRsRqstMsg">
            <wsdl:part name="rqst" element="est:addTNRsRqst"/>
      </wsdl:message>

      <wsdl:message name="cmnRspnsMsg">
            <wsdl:part name="rspns" element="est:cmnRspns"/>
      </wsdl:message>
```

Inputs to the operation are wrapped in the `addTNRsRqst` object, which is declared as follows:

```
      <element name="addTNRsRqst">
            <complexType>
                  <sequence>
                        <element name="basicRqst" type="est:BasicRqstType"/>
                        <element name="tnR" type="est:TNRType"
maxOccurs="unbounded"/>
                  </sequence>
            </complexType>
      </element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more `TNRType` objects. Any limitation on the maximum number of TNRs that may be passed into a call to the operation is a policy decision and is not limited by the protocol.

The `TNRType` object structure is declared as follows:

```
      <complexType name="TNRType">
            <sequence>
                  <element name="oid" type="est:OIdType"/>
                  <element name="eid" type="est:EIdType"/>
                  <element name="tnRStrt" type="string"/>
                  <element name="tnREnd" type="string"/>
                  <element name="saId" type="est:OIdType"/>
                  <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
            </sequence>
      </complexType>
```

`TNRType` is comprised of the following data elements:

- `oid:` Exactly one object ID that provides the identity of the object.
- `eid:` Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for or "owns" this object.
- `tnRStrt:` A required element that represents either the telephone number being added or the first telephone number in a continuous range of telephone numbers being added.
- `tnREnd:` A required element that represents the last telephone number in a continuous range of telephone numbers being added. The ESPP Server MUST validate that the range is a valid range of continuous numbers (e.g., that the `tnREnd` is greater than the `tnRStrt`). If not, then the ESPP Server MUST return the appropriate error code to the ESPP Client.

- `saId:` Exactly one object ID that identifies the Service Area this TN Range object instance resides within. The Service Area's relationship to one or more Routes can then define the telephone numbers' peering points. The ESPP Server MUST reject, with the appropriate error response code, an attempt to create a TNR object containing a Service Area ID that does not exist within the appropriate context of the addressing server.
- `ext:` Point of extensibility described in Section 6.4.1.9 of this document.

The following Table 8 defines the result codes and messages that the `addTNRs` operation SHOULD return.

Table 8 - addTNRs Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.7 Operation Name: addNAPTRs

As described in the introductory sections, a NAPTR object represents a peering point for routing and/or other data. NAPTRs are associated to Routes and TNs. The `addNAPTRs` operation creates or overwrites one or more NAPTR objects in the addressing server. If a NAPTR object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the NAPTR object. If a NAPTR object with the same object ID does exist within the context of the addressing server, then the ESPP Server MUST replace the current properties of that NAPTR with the properties passed into the `addNAPTRs` operation. If a NAPTR with the given ID does exist, but was created by a client other than the one calling the `addNAPTRs` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addNAPTRs">
       <wsdl:input message="addNAPTRsRqstMsg"/>
       <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addNAPTRsRqstMsg">
       <wsdl:part name="rqst" element="est:addNAPTRsRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
       <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the addNAPTRsRqst object, which is declared as follows:

```
<element name="addNAPTRsRqst">
	<complexType>
		<sequence>
			<element name="basicRqst" type="est:BasicRqstType"/>
			<element name="naptr" type="est:NAPTRType"
maxOccurs="unbounded"/>
		</sequence>
	</complexType>
</element>
```

As with all update operations, the basicRqst data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more NAPTRType objects. Any limitation on the maximum number of NAPTRs that may be passed into a call to the addNAPTRs operation is a policy decision and is not limited by the protocol.

A NAPTR object is declared as follows:

```
<complexType name="NAPTRType">
	<sequence>
		<element name="oid" type="est:OIdType"/>
		<element name="eid" type="est:EIdType"/>
		<element name="order" type="unsignedShort"/>
		<element name="pref" type="unsignedShort"/>
		<element name="flags" type="string" minOccurs="0"/>
		<element name="svcs" type="string"/>
		<element name="regx" type="string" minOccurs="0"/>
		<element name="repl" type="string" minOccurs="0"/>
		<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
	</sequence>
</complexType>
```

The NAPTRType object is comprised of the data elements that are necessary for a NAPTR that represents a peering port within the Route. Each data element is described as follows:

- oid: Exactly one object ID to uniquely identify the object instance. This identifier is also an input to the delNAPTRs, getNAPTRs, and addRtes operations to identify a given NAPTR.
- eid: Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for or "owns" this object.
- order: Relative order value in an ENUM NAPTR.
- pref: Relative preference value in an ENUM NAPTR.
- svcs: ENUM service(s) that are served by the SBE. This field's value must be of the form specified in RFC 3761 (e.g., E2U+pstn:sip+sip). The allowable values are a matter of policy and not limited by this protocol.
- regx: NAPTR’s regular expression field. If this is not included then the Repl field must be included.
- repl: NAPTR replacement field, should only be provided if the Regex field is not provided, otherwise it will be ignored by the addressing server.
- ext: Point of extensibility described in Section 6.4.1.9 of this document.

The following Table 9 defines the result codes and messages that the `addNAPTRs` operation SHOULD return.

Table 9 - addNAPTRs Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.8 Operation Name: addEgrRtes

As described in the introductory sections, an Egress Route houses a rule that can be used for re-writing (modifying) the regular expressions housed in the NAPTRs associated with a Route object. The `addEgrRtes` operation creates or overwrites one or more Egress Route objects in the addressing server. If an Egress Route object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the Egress route object. If an Egress Route object with the same object ID does exist within the context of the addressing server, then the ESPP Server MUST replace the current properties of that Egress Route with the properties passed into the `addEgrRtes` operation. If an Egress Routes with the given ID does exist, but was created by a client other than the one calling the `addEgrRtes` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addEgrRtes">
       <wsdl:input message="addEgrRtesRqstMsg"/>
       <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addEgrRtesRqstMsg">
       <wsdl:part name="rqst" element="est:addEgrRtesRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
       <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `addEgrRtesRqst` object, which is declared as follows:

```
<element name="addEgrRtesRqst">
	<complexType>
		<sequence>
			<element name="basicRqst" type="est:BasicRqstType"/>
			<element name="egrRte" type="est:EgrRteType"
maxOccurs="unbounded"/>
		</sequence>
	</complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more `EgrRteType` objects. Any limitation on the maximum number of Egress Routes that may be passed into a call to the `addEgrRtes` operation is a policy decision and is not limited by the protocol.

An Egress Route object is declared as follows:

```
<complexType name="EgrRteType">
	<sequence>
		<element name="oid" type="est:OIdType"/>
		<element name="eid" type="est:EIdType"/>
		<element name="rteId" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
		<element name="pref" type="unsignedShort"/>
		<element name="svcs" type="string" minOccurs="0"/>
		<element name="regxRewriteRule" type="string"/>
		<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
	</sequence>
</complexType>
```

The `EgrRteType` object is comprised of the data elements that are necessary to for a an Egress Route. Each data element is described as follows:

- `oid:` Exactly one object ID to uniquely identify the object instance. This identifier is also an input to the `delEgrRtes` and `getEgrRtes` operations to identify a given Egress Route.
- `eid:` Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for or "owns" this object.
- `rteId:` Zero or more Routes to which this Egress Route applies. The ESPP Server MUST reject, with the appropriate error response code, an attempt to create an Egress Route object containing a Route ID that does not exist within the appropriate context of the addressing server.
- `pref:` Relative preference value of the Egress Route with respect to other Egress Routes associated with the same Route object.
- `svcs:` ENUM service(s) to which this Egress Route's `regxRewriteRule` should be applied. This field's value must be of the form specified in RFC 3761 (e.g., E2U+pstn:sip+sip).
- `regxRewriteRule:` A regular expression based, rewrite rule and substitution expression. The structure and content of this rewrite rule adheres to the same rules specified by the regular expression and substitution expression syntax specification defined for the regular expression field of a NAPTR record. Refer to [RFC 3402] and [RFC 3761]. An Egress Route's rewrite rule is applied to the regex field of all NAPTR objects associated with the same Route object, whose svcs field also contains any one of the services contained in the svcs field of that Egress Route. This application of that Egress Route's regex rewrite rule to those NAPTR object regex fields results in the NAPTR object's regex field pointing to the desired point of egress, rather than the desired point of ingress.
- `ext:` Point of extensibility described in Section 6.4.1.9 of this document.

The following Table 10 defines the result codes and messages that the `addEgrRtes` operation SHOULD return.

Table 10 - addEgrRtes Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.9 Operation Name: addNSs

As described in the introductory sections, a NS object represents a Name Server that should be returned in response to resolution requests for Public Identities within Service Areas that are associated with Routes that contain the NS object. NSs are associated with Routes.

The `addNSs` operation creates or overwrites one or more NS objects in the addressing server. If an NS object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the NS object. If an NS object with the same object ID does exist within the context of the addressing server, then the ESPP Server MUST replace the current properties of that NS with the properties passed into the `addNSs` operation. If an NS with the given ID does exist, but was created by a client other than the one calling the `addNSs` operation then the ESPP Server SHOULD reject the request with the appropriate error code, 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addNSs">
       <wsdl:input message="addNSsRqstMsg"/>
       <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addNSsRqstMsg">
       <wsdl:part name="rqst" element="est:addNSsRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
       <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Any inputs to the operation are wrapped in the `addNSsRqst` object, which is declared as follows:

```
<element name="addNSsRqst">
       <complexType>
              <sequence>
                     <element name="basicRqst" type="est:BasicRqstType"/>
                     <element name="ns" type="est:NSType" maxOccurs="unbounded"/>
```

```
                </sequence>
            </complexType>
        </element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more `NSType` objects. Any limitation on the maximum number of NSs that may be passed into a call to the `addNSs` operation is a policy decision and is not limited by the protocol.

An NS object is declared as follows:

```
        <complexType name="NSType">
            <sequence>
                <element name="oid" type="est:OIdType"/>
                <element name="eid" type="est:EIdType"/>
                <element name="name" type="string"/>
                <element name="ipAddr" type="est:IPAddrType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
            </sequence>
        </complexType>

        <complexType name=" IPAddrType ">
            <sequence>
                <element name="addr" type="string"/>
                <element name="type" type="est:IPType"/>
            </sequence>
        </complexType>

        <simpleType name="IPType">
            <restriction base="token">
                <enumeration value="IPv4"/>
                <enumeration value="IPv6"/>
            </restriction>
        </simpleType>
```

The `NSType` object is comprised of the data elements that are necessary for an NS record. Each data element is described as follows:

- `oid`: Exactly one object ID to uniquely identify the object instance. This identifier is also an input to the delNSs, getNSs, and `addRtes` operations to identify a given NS.
- `eid`: Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for or "owns" this object.
- `name`: Exactly one fully qualified host name of the Name Server.
- `ipAddr`: Zero or more objects of type IpAddrType. Each object holds an IP Address and the IP Address type, IPv4 or IP v6.
- `ext:` Point of extensibility described in Section 6.4.1.9 of this document.

The following Table 11 defines the result codes and messages that the `addNSs` operation SHOULD return.

Table 11 - addNSs Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.

Result Code	Text
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.10 Operation Name: delRtes, delSvcAreas, delPubIds, delPvtIds, delTNRs, delLRNs, delNAPTRs, delEgrRtes, delNSs

This sub-section defines the delete operations. All delete operations are of the same design and the delRtes operation is used here for illustration. The delete operations delete one or more existing objects from an addressing server. If all the indicated objects, with the given ID(s) exist, then the ESPP Server MUST delete those objects and return a response code of 1000. If one or more of the objects do not exist then the ESPP Server MUST return a response code of 1001, i.e., the ESPP Server will not regard this as an error condition. If the ESPP server returns a response code of 1001 the ESPP server SHOULD log the objectIDs of non-existent objects that were requested to be deleted and the ESPP Client that requested the deletion. If one or more of the objects with the given IDs does exist but was created by a client other than the one calling the delete operation then the ESPP Server SHOULD reject the request with an error code of 2106.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as:

```
<wsdl:operation name="delRtes">
      <wsdl:input message="delRtesRqstMsg"/>
      <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="delRtesRqstMsg">
      <wsdl:part name="rqst" element="est:delRtesRqst"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `delRtesRqst` object, which is declared as follows:

```
<element name="delRtesRqst">
      <complexType>
            <sequence>
                  <element name="basicRqst" type="est:BasicRqstType"/>
                  <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
            </sequence>
      </complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more object IDs that identify the objects that the addressing server is to delete. Any limitation on the maximum number object IDs that may be passed into this operation is a policy decision and not limited by the protocol.

It is the responsibility of the ESPP Server to maintain referential integrity when an object is deleted. The following rules apply to the delete operations:

- With the exception of cascading Service Area deletions described below, an ESPP Server SHOULD NOT allow an object created by a client having a given client ID to be deleted by a client having a different client ID.
- The ESPP Server MUST delete all LRNs and TNRanges within a Service Area that is being deleted, even if those LRNs or TNRanges were created by another ESPP client ID. The ESPP Server MUST delete all Public Identity instances within a Service Area, but only if a Public Identity instance is not directly associated with one or more NAPTR objects. As with LRNs and TNRanges, Service Areas and Public Identities MUST be deleted even if they were created by another ESPP client ID.
- When an object is deleted within an ESPP Server, the ESPP Server must automatically remove any references to that object that may exist within objects created by any client IDs within that ESPP Server. For example, if a Route object "ABC" that was created by Client ID "123" is referred to by Egress Route "GHI" that was created by client ID "678", then when Route "ABC" is deleted by client "123" the ESPP Server MUST also automatically remove the reference to Route "ABC" from Egress Route "GHI", or any other Egress Routes. This rule applies to the relationships between Routes and Service Areas, Egress Routes and Routes, Routes and NAPTRs, Public Identities and Private Identities, and Public Identities and NAPTRs.

The following Table 12 defines the result codes and messages that these delete operations SHOULD return.

Table 12 - delRtes, delSvcAreas, delPubIds, delPvtIds, delTNRs, delLRNs, delNAPTRs, delEgrRtes, delNSs Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
1001	Request Succeeded. Deleted object(s) did not exist.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

6.4.2.11 Operation Name: getRtes, getSvcAreas, getPubIds, getPvtIds, getTNRs, getLRNs, getNAPTRs, getEgrRtes, getNSs

This sub-section defines the get, or query, operations. All get operations are of the same design and the getRtes operation is used here for illustration. If an object of the indicated type with the given ID exists, then the ESPP Server MUST return that object. If such an object does not exist, the ESPP Server MUST NOT return an object. An empty result set MUST NOT result in an error code response.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the `getRtes` operation is as follows:

```
<wsdl:operation name="getRtes">
```

```
            <wsdl:input message="getRtesRqstMsg"/>
            <wsdl:output message="getRtesRspnsMsg"/>
      </wsdl:operation>

      <wsdl:message name="getRtesRqstMsg">
            <wsdl:part name="rqst" element="est:getRtesRqst"/>
      </wsdl:message>

      <wsdl:message name="getRtesRspnsMsg">
            <wsdl:part name="rspns" element="est:getRtesRspns"/>
      </wsdl:message>
```

Inputs to the operation are wrapped in the `getRtesRqst` object, which is declared as follows:

```
      <element name="getRtesRqst">
            <complexType>
                  <sequence>
                        <element name="basicRqst" type="est:BasicQueryType"/>
                        <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                  </sequence>
            </complexType>
      </element>

      <complexType name="BasicQueryType">
            <sequence>
                  <element name="clientId" type="est:ClientIdType"/>
                  <element name="minorVer" type="est:VerType"/>
                  <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
            </sequence>
      </complexType>
```

The required input is one `BasicQueryType` object and one or more object IDs that the addressing server is to return and/or one or more enterprise IDs for which the addressing server should return all objects of the given type. The list of object IDs and enterprise IDs MUST be considered "or" criteria, find objects of the given type that have these object IDs or that belong to the given enterprise IDs. Stated another way, the addressing server is to return the objects that have any of the given object IDs and return the objects that have any of the given enterprise IDs. Any limitation on the maximum number of objects that may be passed into or returned by this operation is a policy decision and not limited by the protocol. Notice that the get operations are the only operations that do not accept a `BasicRqstType` object as input. Rather, a `BasicQueryType` object is required. This results from the fact that passing a transaction ID into a get operation is not necessary and would be inefficient. The `BasicQueryType` object only contains the mandatory version identifier and the optional extensibility element.

Results of the operation are wrapped in the `getRtesRspns` object, which is declared as follows:

```
      <element name="getRtesRspns">
            <complexType>
                  <sequence>
                        <element name="rte" type="est:RteType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                  </sequence>
            </complexType>
      </element>
```

The response includes the standard `BasicRspnsType` object and zero or more `RteType` objects, one for each object ID that matches a Route in the addressing server.

The following Table 13 defines the result codes and messages that these get operations SHOULD return.

Table 13 - getRtes, getSvcAreas, getPubIds, getPvtIds, getTNRs, getLRNs, getNAPTRs, getEgrRtes, getNSs Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]

6.4.2.12 Operation Name: addEntr

Prior to attempting to add an object under a given an enterprise ID to an ESPP Server, the ESPP Client MUST create that enterprise ID in that ESPP Server using the addEntr operation.

The addEntr operation creates an enterprise ID in the addressing server. If an enterprise ID value of an addEntr operation does not exist, the ESPP Server MUST create the enterprise ID. If an enterprise ID value of an addEntr operation already exists, the ESPP Server MUST return a success response.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="addEntr">
       <wsdl:input message="esp:addEntrRqstMsg"/>
       <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addEntrRqstMsg">
       <wsdl:part name="rqst" element="est:addEntrRqst"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `addEntrRqst` object, which is declared as follows:

```
<element name="addEntrRqst">
       <complexType>
              <sequence>
                     <element name="basicRqst" type="est:BasicRqstType"/>
                     <element name="eid" type="est:EIdType"/>
              </sequence>
       </complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one enterprise ID.

The following Table 14 defines the result codes and messages that the `addEntr` operation SHOULD return.

Table 14 - addEntr Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]

6.4.2.13 Operation Name: modEntr

Modification of an enterprise ID by one ESPP Client must not impact objects created by another ESPP Client. In other words only objects created by the client modifying the enterprise ID MUST be moved to the new enterprise ID as part of processing the `modEntr` operation. Any objects created by a different ESPP Client must remain in their existing enterprise ID.

The `modEntr` operation modifies an existing enterprise ID in the addressing server. If the "old" enterprise ID being modified does exist then the ESPP Server MUST replace the current enterprise ID with the "new" enterprise ID. If the "old" enterprise ID does not exist then the ESPP Server MUST return the appropriate error code.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="modEntr">
       <wsdl:input message="esp:modEntrRqstMsg"/>
       <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="modEntrRqstMsg">
       <wsdl:part name="rqst" element="est:modEntrRqst"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `modEntrRqst` object, which is declared as follows:

```
<element name="modEntrRqst">
       <complexType>
              <sequence>
                     <element name="basicRqst" type="est:BasicRqstType"/>
                     <element name="newEId" type="est:EIdType"/>
                     <element name="oldEId" type="est:EIdType"/>
              </sequence>
       </complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one "new" enterprise ID and one "old" enterprise ID.

The following Table 15 defines the result codes and messages that the `modEntr` operation SHOULD return.

Table 15 - modEntr Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]

6.4.2.14 Operation Name: delEntr

The `delEntr` operation deletes the enterprise ID and all objects from an addressing server that have the given enterprise ID and that were created by the client that is deleting the enterprise ID. If the enterprise ID does not exist in the addressing server, then the ESPP Server MUST return the response code 1001, i.e., the ESPP Server will not regard this as an error condition. If the ESPP server returns a response code of 1001 the ESPP server SHOULD log the non-existent enterpriseID and the ESPP Client that requested the deletion. If the enterprise ID did exist and the operation was successful then the ESPP Server MUST return the response code 1000.

Deletion of an enterprise ID by one ESPP Client MUST NOT impact objects created by another ESPP Client. In other words only objects created by the client deleting the enterprise ID MUST be deleted. Any objects created by a different ESPP Client MUST NOT be deleted and MUST remain in their existing enterprise ID.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="delEntr">
       <wsdl:input message="delEntrRqstMsg"/>
       <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="delEntrRqstMsg">
       <wsdl:part name="rqst" element="est:delEntrRqst"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `delEntrRqst` object, which is declared as follows:

```
<element name="delEntrRqst">
       <complexType>
              <sequence>
                     <element name="basicRqst" type="est:BasicRqstType"/>
                     <element name="eid" type="est:EIdType"/>
              </sequence>
       </complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one enterprise ID.

The following Table 16 defines the result codes and messages that the `delEntr` operation SHOULD return.

Table 16 - delEntr Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
1001	Request Succeeded. Deleted object(s) did not exist.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]

6.4.2.15 Operation Name: batchUpdate

The `batchUpdate` operation is a single operation with a single transaction ID that may contain one or more update actions. This approach significantly speeds processing time under some circumstances by effectively allowing multiple operations to be transmitted and performed under a single transaction ID and a single connection.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="batchUpdate">
       <wsdl:input message="batchUpdateRqstMsg"/>
       <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="batchUpdateRqstMsg">
       <wsdl:part name="rqst" element="est:batchUpdateRqst"/>
</wsdl:message>

<wsdl:message name="cmnRspnsMsg">
       <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `batchUpdateRqst` object, which is declared as follows:

```
<element name="batchUpdateRqst">
       <complexType>
              <sequence>
                     <element name="basicRqst" type="est:BasicRqstType"/>
                     <element name="batchUpdate" type="est:BatchUpdateType"/>
              </sequence>
       </complexType>
</element>

<complexType name="BatchUpdateType">
       <sequence>
              <element name="op" type="est:BatchOpType" maxOccurs="unbounded"/>
       </sequence>
</complexType>

<complexType name="BatchOpType">
       <sequence>
              <element name="naptrDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
              <element name="naptrAdd" type="est:NAPTRType" minOccurs="0"
```

```
maxOccurs="unbounded"/>
                        <element name="nsDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="nsAdd" type="est:NSType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="egrRteDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="egrRteAdd" type="est:EgrRteType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="rteDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="rteAdd" type="est:RteType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="saDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="saAdd" type="est:SvcAreaType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="pviDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="pviAdd" type="est:PvtIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="piDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="piAdd" type="est:PubIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="tnRDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="tnRAdd" type="est:TNRType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="lrnDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="lrnAdd" type="est:LRNType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="entrDel" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="entrAdd" type=" est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="entrMod" minOccurs="0" maxOccurs="unbounded">
                                <complexType>
                                        <sequence>
                                                <element name="newEId" type="est:EIdType"/>
                                                <element name="oldEId" type="est:EIdType"/>
                                        </sequence>
                                </complexType>
                        </element>
                </sequence>
        </complexType>
```

The `BatchUpdateType` object can contain one or more `BatchOpType` objects. The structure is nested here to allow multiple `BatchUpdateType` objects to be included in any order that is appropriate to achieve the desired end goal of the batch operation. More specifically, this structure works around the strict ordering rules imposed on the XSD "sequence" data structure, allowing adds and dels to occur in any order. The ESPP Server MUST process the sub-elements of the `batchUpdate` and the `op` elements in the order in which they appear in the `batchUpdateRqst` element.

The ESPP Server MUST return the appropriate response code when the operation is successfully completed, i.e., 1000 or 1001. If one or more of the objects do not exist, then the ESPP Server MUST return a response code of 1001, i.e., the ESPP Server will not regard this as an error condition. If the ESPP server returns a response code of 1001 the ESPP server SHOULD log the objectIDs of non-existent objects that were requested to be deleted and the ESPP Client that requested the deletion.

Upon encountering the first error condition, the ESPP Server MUST stop processing and return the appropriate error response code, rolling back any changes that may have been made for the current request.

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 6.4.1, General Protocol Concepts, for a detailed description of this data element. Each of the remaining elements are simply zero or

more instances of the top level data structures that are wrapped in each of the other update operations described in previous sections.

The following Table 17 defines the result codes and messages that the batchUpdate operation SHOULD return.

Table 17 - batchUpdate Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
1001	Request Succeeded. Deleted object(s) did not exist.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]
2105	Object does not exist: [attribute name]:[objectType-objectId]
2106	Object status or ownership does not allow for request: [request name]:[attribute name]:[objectType-objectId]

6.4.2.16 Operation Name: getSvcMenu

The `getSvcMenu` operation returns the status of the ESPP server (in service or out of service) and a list of operations supported by the ESPP Server and the protocol versions supported by the ESPP Server. The list of operations is comprised of a list of URNs, one for each supported operation. This operation can be used by ESPP Clients to dynamically discover the status of the ESPP server and which operations and versions are supported by the ESPP Server.

In keeping with the Document Literal Wrapped design approach described in the introductory sections, the WSDL declaration of the operation is as follows:

```
<wsdl:operation name="getSvcMenu">
       <wsdl:input message="esp:getSvcMenuRqstMsg"/>
       <wsdl:output message="esp:getSvcMenuRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="getSvcMenuRqstMsg">
       <wsdl:part name="rqst" element="est:getSvcMenuRqst"/>
</wsdl:message>

<wsdl:message name="getSvcMenuRspnsMsg">
       <wsdl:part name="rspns" element="est:getSvcMenuRspns"/>
</wsdl:message>
```

Inputs to the operation are wrapped in the `getSvcMenuRqst` object, which is declared as follows:

```
<element name="getSvcMenuRqst">
       <complexType>
              <sequence>
                     <element name="basicRqst" type="est:BasicQueryType"/>
              </sequence>
       </complexType>
</element>
```

The required input is one BasicQueryType and the results of the operation are wrapped in the getRtesRspns object, which is declared as follows:

```
<element name="getSvcMenuRspns">
	<complexType>
		<sequence>
			<element name="svcMenu" type="est:SvcMenuType"/>
			<element name="basicResult" type="est:BasicRspnsType"/>
		</sequence>
	</complexType>
</element>

<complexType name="SvcMenuType">
	<sequence>
		<element name="serverStatus" type="est:ServerStatusType"/>
		<element name="majMinVersion" type="string"
maxOccurs="unbounded"/>
		<element name="objURI" type="anyURI" maxOccurs="unbounded"/>
		<element name="extURI" type="anyURI" minOccurs="0"
maxOccurs="unbounded"/>
	</sequence>
</complexType>

<simpleType name="ServerStatusType">
	<restriction base="token">
		<enumeration value="inService"/>
		<enumeration value="outOfService"/>
	</restriction>
</simpleType>
```

The response includes the standard BasicRspnsType object and one SvcMenuType object. The SvcMenuType contains the following elements:

- ServerStatus: Exactly one status attribute that indicates whether the ESPP server is in-service or out-of-service. The server MUST allow an ESPP client to definitively determine whether the ESPP server is in a usable state via this attribute.
- MajMinVersion: One or more version identifiers in dotted notation, where each version identifier represents a fully qualified version identifier that is supported by the ESPP Server. For example, 2.0 represents a major version of "2" and a minor version of "0." Note that the major version is identified within the namespace URN, following the character 'v.'
- ObjUri: One or more URI(s) where each URI identifies one ESPP operation that is supported by the ESPP Server. URIs take the following form: urn:cablelabs:packetcable:espp:api-1.0:est:addRtesRqst
- ExtUri: One or more URI(s) where each URI identifies an operation that is not part of the formal ESPP specification but is supported by the ESPP Server. This may arise where an ESPP Server has added non-standard operations to its ESPP implementation.

The following Table 18 defines the result codes and messages that the getSvcMenu operation SHOULD return.

Table 18 - getSvcMenu Operation Result Codes and Messages

Result Code	Text
1000	Request Succeeded.
2001	Request syntax invalid.
2301	System temporarily unavailable.
2104	Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId]

7 FILE-BASED PROVISIONING PROTOCOL

To further support the high data distribution requirements of the target reference architecture, the ESPP specification defines a file based operational scheme. Some aspects of this scheme are policy driven and are, therefore, outside the scope of this document.

The file-based provisioning protocol requirements defined in this section MUST be supported by an ESPP Client and an ESPP Server to claim compliance with this specification. They must be used in the following manner to support file base data distribution.

7.1 Overview of the File-based Provisioning Operations

Under certain circumstances, such as bootstrapping a new ENUM-SIP addressing server with its initial data set, or establishing a new session peering relationship between two peers, it may be advisable to distribute peering data in the form of a file. To accomplish this, the ESPP Client MUST have the ability to, on command, generate a set of files containing all the session establishment data residing within the ESPP Client view for a given session peering partner. Similarly, the ESPP Server MUST support the ability to read, parse, and load the content of that file into the appropriate context of the ENUM-SIP addressing server.

The process of retrieving that file from the ESPP Client (acting as a file server) and then delivering the file to the proper location on the ENUM-SIP addressing server is a matter of policy and procedure and it is not specified in this document. The process of suspending real-time provisioning operations (non-file-based) during this file-based provisioning process is also a matter of policy and procedure.

7.2 File Structure

An ESPP compliant distribution file MUST be comprised of exactly one `batchUpdateFileRqst` element which contains one `BasicFileRqstType` object and one or more `BatchUpdateType` objects, which is described in a previous section of this document. This approach is taken to simplify the implementation effort for this feature as the ESPP Client and the ESPP Server have the ability to process the data structures that comprise the `batchUpdateFileRqst`. To ensure data consistency, the ESPP Client must generate batch files using the same transaction identifier sequence as the real time protocol. It is, therefore, necessary to disable real-time propagation while the file-based provisioning process is active.

```
<element name="batchUpdateFileRqst">
       <complexType>
              <sequence>
                     <element name="basicFileRqst" type="est:BasicFileRqstType"/>

                     <element name="batchUpdate" type="est:BatchUpdateType"
maxOccurs="unbounded"/>
              </sequence>
       </complexType>
</element>
```

The `BasicFileRqstType` is the file header that specifies some basic information about where and when the batch update file(s) were generated. A given batch update may span multiple files, with the `BasicFileRqstType` object in each file.

```
<complexType name="BasicFileRqstType">
       <sequence>
              <element name="clientId" type="est:ClientIdType"/>
              <element name="serverId" type="string"/>
              <element name="transId" type="est:TransIdType"/>
              <element name="minorVer" type="est:VerType"/>
              <element name="isFullResync" type="boolean"/>
              <element name="creationTimestamp" type="dateTime"/>
              <element name="sequenceNumber" type="string"/>
              <element name="isEndOfSequence" type="boolean"/>
              <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
       </sequence>
</complexType>
```

- `clientId:` The integer that identifies the ESPP Client that generated the batch file(s).
- `serverId:` A string that identifies the ESPP Server for which the batch file(s) were generated.
- `transId:` A string that identifies the transaction ID of this batch file sequence.
- `minorVer:` The minor version of the protocol.
- `isFullResync:` Indicates if this sequence of files is a full refresh of all the data that should reside within an addressing server. If not, then the file sequence represents an incremental update. The addressing server MUST properly apply the data based on the value of this attribute. This may entail dropping the current data in the addressing server.
- `creationTimestamp:` The creation timestamp for the batch load file sequence MUST follow the format defined by the dateTime datatype of XML Schema Part 2 and MUST contain the time zone information. For example, a valid format can be represented as YYYY-MM-DDTHH:MM:SS('+'|'-')hh':'mm).
- `sequenceNumber:` The sequence number for this batch file within the batch update. An ESPP Client may generate multiple batch files that, as a set, represent the entire batch update. The `sequenceNumber` is a sequential number, starting with 1 and incremented by 1 for each subsequent file.
- `isEndOfSequence:` A boolean value indicating whether or not this is that last file in the sequence of files that comprise the batch update.
- `ext:` Point of extensibility described in Section 6.4.1.9 of this document.

The ESPP Server SHOULD log error codes that correspond to the codes used in the real time protocol as defined in Section 8.

8 RESPONSE CODES AND MESSAGES

This section contains the consolidated listing of the response codes and their corresponding messages.

The response code numbering scheme generally adheres to the theory formalized in section 4.2.1 of [RFC 2821]:

- The first digit of the response code can only be 1 or 2: 1 = a positive result, 2 = a negative result.
- The second digit of the response code indicates the category: 0 = Protocol Syntax, 1 = Implementation Specific Business Rule, 2 = Security, 3 = Server System
- The third and fourth digits of the response code indicate the individual message event within the category defines by the first two digits.

Table 19 - Response Codes Numbering Scheme and Messages

Result Code	Text
1000	Request Succeeded.
1001	Request Succeeded. Deleted object(s) did not exist.
2001	Request syntax invalid.
2002	Request too large.
2003	Version not supported.
2301	System temporarily unavailable.
2103	Transaction ID out of sequence: [transaction ID received]:[transaction ID expected]
2104	Attribute value invalid: [attributeName]:[attribute value]:[objectType-objectId]
2105	Object does not exist: [attributeName]:[objectType-objectId]
2106	Object status/ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId]

Some response messages are "parameterized" with one or more of the following parameters:

- "attribute name"
- "attribute value"
- "objectType-objectId"
- "operation name"

The use of these parameters MUST adhere to the following rules:

- All parameters within a response message are mandatory and MUST be present. Parameters within a response message MUST NOT be left empty.
- Any value provided for the "attribute name" parameter MUST be an exact element name of the protocol data element that the response message is referring to. For example, allowable values for "attribute name" are "`tnRStrt`", "`rteName`", etc.
- Any value provided for the "request name" parameter MUST be an exact request object name that the response message is referring to. For example, an allowable value for "request object name" is "`delRtesRqst`".

- The value for "attribute value" MUST be the value of the data element to which the preceding "attribute name" refers.
- Result codes 2104 and 2105 MUST NOT be used interchangeably. Response code 2105 SHOULD be returned when the data element(s) used to uniquely identify a pre-existing object do not exist. If the data elements used to uniquely identify an object are malformed, then response code 2104 SHOULD be returned.
- Result code 2104 MUST be used whenever an element value does not adhere to the data validation rules defined in the Data Element Validation Constraints section of this document. It MUST also be used for circumstances where an optional data element was expected but not received.
- Result code 2301 MUST be returned whenever an ESPP addressing server encounters an unexpected internal system or server error or application bug.

9 FORMAL API DEFINITION

9.1 WSDL Specification

```
<wsdl:definitions xmlns:esp="urn:cablelabs:packetcable:espp:api-v4"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:est="urn:cablelabs:packetcable:espp:types-v4"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns="http://schemas.xmlsoap.org/soap/encoding/"
targetNamespace="urn:cablelabs:packetcable:espp:api-v4"
xsi:schemaLocation="urn:cablelabs:packetcable:espp:types-v4">
       <wsdl:types>
              <xsd:schema targetNamespace="urn:cablelabs:packetcable:espp:api-v4"
xmlns:esp="urn:cablelabs:packetcable:espp:api-v4"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
                     <xsd:import namespace="urn:cablelabs:packetcable:espp:types-v4"/>
              </xsd:schema>
       </wsdl:types>
       <wsdl:message name="addRtesRqstMsg">
              <wsdl:part name="rqst" element="est:addRtesRqst"/>
       </wsdl:message>
       <wsdl:message name="addSvcAreasRqstMsg">
              <wsdl:part name="rqst" element="est:addSvcAreasRqst"/>
       </wsdl:message>
       <wsdl:message name="addPubIdsRqstMsg">
              <wsdl:part name="rqst" element="est:addPubIdsRqst"/>
       </wsdl:message>
       <wsdl:message name="addTNRsRqstMsg">
              <wsdl:part name="rqst" element="est:addTNRsRqst"/>
       </wsdl:message>
       <wsdl:message name="addPvtIdsRqstMsg">
              <wsdl:part name="rqst" element="est:addPvtIdsRqst"/>
       </wsdl:message>
       <wsdl:message name="addLRNsRqstMsg">
              <wsdl:part name="rqst" element="est:addLRNsRqst"/>
       </wsdl:message>
       <wsdl:message name="addNAPTRsRqstMsg">
              <wsdl:part name="rqst" element="est:addNAPTRsRqst"/>
       </wsdl:message>
       <wsdl:message name="addNSsRqstMsg">
              <wsdl:part name="rqst" element="est:addNSsRqst"/>
       </wsdl:message>
       <wsdl:message name="addEgrRtesRqstMsg">
              <wsdl:part name="rqst" element="est:addEgrRtesRqst"/>
       </wsdl:message>
       <wsdl:message name="addEntrRqstMsg">
              <wsdl:part name="rqst" element="est:addEntrRqst"/>
       </wsdl:message>
       <wsdl:message name="batchUpdateRqstMsg">
              <wsdl:part name="rqst" element="est:batchUpdateRqst"/>
       </wsdl:message>
       <wsdl:message name="getRtesRqstMsg">
              <wsdl:part name="rqst" element="est:getRtesRqst"/>
       </wsdl:message>
       <wsdl:message name="getSvcAreasRqstMsg">
              <wsdl:part name="rqst" element="est:getSvcAreasRqst"/>
       </wsdl:message>
       <wsdl:message name="getPubIdsRqstMsg">
              <wsdl:part name="rqst" element="est:getPubIdsRqst"/>
       </wsdl:message>
```

```
<wsdl:message name="getTNRsRqstMsg">
       <wsdl:part name="rqst" element="est:getTNRsRqst"/>
</wsdl:message>
<wsdl:message name="getPvtIdsRqstMsg">
       <wsdl:part name="rqst" element="est:getPvtIdsRqst"/>
</wsdl:message>
<wsdl:message name="getLRNsRqstMsg">
       <wsdl:part name="rqst" element="est:getLRNsRqst"/>
</wsdl:message>
<wsdl:message name="getNAPTRsRqstMsg">
       <wsdl:part name="rqst" element="est:getNAPTRsRqst"/>
</wsdl:message>
<wsdl:message name="getNSsRqstMsg">
       <wsdl:part name="rqst" element="est:getNSsRqst"/>
</wsdl:message>
<wsdl:message name="getEgrRtesRqstMsg">
       <wsdl:part name="rqst" element="est:getEgrRtesRqst"/>
</wsdl:message>
<wsdl:message name="getSvcMenuRqstMsg">
       <wsdl:part name="rqst" element="est:getSvcMenuRqst"/>
</wsdl:message>
<wsdl:message name="delRtesRqstMsg">
       <wsdl:part name="rqst" element="est:delRtesRqst"/>
</wsdl:message>
<wsdl:message name="delSvcAreasRqstMsg">
       <wsdl:part name="rqst" element="est:delSvcAreasRqst"/>
</wsdl:message>
<wsdl:message name="delPubIdsRqstMsg">
       <wsdl:part name="rqst" element="est:delPubIdsRqst"/>
</wsdl:message>
<wsdl:message name="delTNRsRqstMsg">
       <wsdl:part name="rqst" element="est:delTNRsRqst"/>
</wsdl:message>
<wsdl:message name="delPvtIdsRqstMsg">
       <wsdl:part name="rqst" element="est:delPvtIdsRqst"/>
</wsdl:message>
<wsdl:message name="delLRNsRqstMsg">
       <wsdl:part name="rqst" element="est:delLRNsRqst"/>
</wsdl:message>
<wsdl:message name="delNAPTRsRqstMsg">
       <wsdl:part name="rqst" element="est:delNAPTRsRqst"/>
</wsdl:message>
<wsdl:message name="delNSsRqstMsg">
       <wsdl:part name="rqst" element="est:delNSsRqst"/>
</wsdl:message>
<wsdl:message name="delEgrRtesRqstMsg">
       <wsdl:part name="rqst" element="est:delEgrRtesRqst"/>
</wsdl:message>
<wsdl:message name="delEntrRqstMsg">
       <wsdl:part name="rqst" element="est:delEntrRqst"/>
</wsdl:message>
<wsdl:message name="modEntrRqstMsg">
       <wsdl:part name="rqst" element="est:modEntrRqst"/>
</wsdl:message>
<wsdl:message name="cmnRspnsMsg">
       <wsdl:part name="rspns" element="est:cmnRspns"/>
</wsdl:message>
<wsdl:message name="getRtesRspnsMsg">
       <wsdl:part name="rspns" element="est:getRtesRspns"/>
</wsdl:message>
<wsdl:message name="getSvcAreasRspnsMsg">
       <wsdl:part name="rspns" element="est:getSvcAreasRspns"/>
</wsdl:message>
<wsdl:message name="getPubIdsRspnsMsg">
       <wsdl:part name="rspns" element="est:getPubIdsRspns"/>
</wsdl:message>
<wsdl:message name="getTNRsRspnsMsg">
```

```
        <wsdl:part name="rspns" element="est:getTNRsRspns"/>
    </wsdl:message>
    <wsdl:message name="getPvtIdsRspnsMsg">
        <wsdl:part name="rspns" element="est:getPvtIdsRspns"/>
    </wsdl:message>
    <wsdl:message name="getLRNsRspnsMsg">
        <wsdl:part name="rspns" element="est:getLRNsRspns"/>
    </wsdl:message>
    <wsdl:message name="getNAPTRsRspnsMsg">
        <wsdl:part name="rspns" element="est:getNAPTRsRspns"/>
    </wsdl:message>
    <wsdl:message name="getNSsRspnsMsg">
        <wsdl:part name="rspns" element="est:getNSsRspns"/>
    </wsdl:message>
    <wsdl:message name="getEgrRtesRspnsMsg">
        <wsdl:part name="rspns" element="est:getEgrRtesRspns"/>
    </wsdl:message>
    <wsdl:message name="getSvcMenuRspnsMsg">
        <wsdl:part name="rspns" element="est:getSvcMenuRspns"/>
    </wsdl:message>
    <wsdl:portType name="ESPPPortType">
        <wsdl:operation name="addRtes">
            <wsdl:input message="esp:addRtesRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="delRtes">
            <wsdl:input message="esp:delRtesRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="getRtes">
            <wsdl:input message="esp:getRtesRqstMsg"/>
            <wsdl:output message="esp:getRtesRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="addSvcAreas">
            <wsdl:input message="esp:addSvcAreasRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="delSvcAreas">
            <wsdl:input message="esp:delSvcAreasRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="getSvcAreas">
            <wsdl:input message="esp:getSvcAreasRqstMsg"/>
            <wsdl:output message="esp:getSvcAreasRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="addPubIds">
            <wsdl:input message="esp:addPubIdsRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="delPubIds">
            <wsdl:input message="esp:delPubIdsRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="getPubIds">
            <wsdl:input message="esp:getPubIdsRqstMsg"/>
            <wsdl:output message="esp:getPubIdsRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="addTNRs">
            <wsdl:input message="esp:addTNRsRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="delTNRs">
            <wsdl:input message="esp:delTNRsRqstMsg"/>
            <wsdl:output message="esp:cmnRspnsMsg"/>
        </wsdl:operation>
        <wsdl:operation name="getTNRs">
            <wsdl:input message="esp:getTNRsRqstMsg"/>
```

```
        <wsdl:output message="esp:getTNRsRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="addPvtIds">
        <wsdl:input message="esp:addPvtIdsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="delPvtIds">
        <wsdl:input message="esp:delPvtIdsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="getPvtIds">
        <wsdl:input message="esp:getPvtIdsRqstMsg"/>
        <wsdl:output message="esp:getPvtIdsRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="addLRNs">
        <wsdl:input message="esp:addLRNsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="delLRNs">
        <wsdl:input message="esp:delLRNsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="getLRNs">
        <wsdl:input message="esp:getLRNsRqstMsg"/>
        <wsdl:output message="esp:getLRNsRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="addNAPTRs">
        <wsdl:input message="esp:addNAPTRsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="delNAPTRs">
        <wsdl:input message="esp:delNAPTRsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="getNAPTRs">
        <wsdl:input message="esp:getNAPTRsRqstMsg"/>
        <wsdl:output message="esp:getNAPTRsRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="addNSs">
        <wsdl:input message="esp:addNSsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="delNSs">
        <wsdl:input message="esp:delNSsRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="getNSs">
        <wsdl:input message="esp:getNSsRqstMsg"/>
        <wsdl:output message="esp:getNSsRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="addEgrRtes">
        <wsdl:input message="esp:addEgrRtesRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="delEgrRtes">
        <wsdl:input message="esp:delEgrRtesRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="getEgrRtes">
        <wsdl:input message="esp:getEgrRtesRqstMsg"/>
        <wsdl:output message="esp:getEgrRtesRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="batchUpdate">
        <wsdl:input message="esp:batchUpdateRqstMsg"/>
        <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="addEntr">
```

```
                  <wsdl:input message="esp:addEntrRqstMsg"/>
                  <wsdl:output message="esp:cmnRspnsMsg"/>
            </wsdl:operation>
            <wsdl:operation name="modEntr">
                  <wsdl:input message="esp:modEntrRqstMsg"/>
                  <wsdl:output message="esp:cmnRspnsMsg"/>
            </wsdl:operation>
            <wsdl:operation name="delEntr">
                  <wsdl:input message="esp:delEntrRqstMsg"/>
                  <wsdl:output message="esp:cmnRspnsMsg"/>
            </wsdl:operation>
            <wsdl:operation name="getSvcMenu">
                  <wsdl:input message="esp:getSvcMenuRqstMsg"/>
                  <wsdl:output message="esp:getSvcMenuRspnsMsg"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="ESPPSoapBinding" type="esp:ESPPPortType">
            <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="addRtes">
                  <soap:operation soapAction="addRtes" style="document"/>
                  <wsdl:input>
                        <soap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="delRtes">
                  <soap:operation soapAction="delRtes" style="document"/>
                  <wsdl:input>
                        <soap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="getRtes">
                  <soap:operation soapAction="getRtes" style="document"/>
                  <wsdl:input>
                        <soap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="addSvcAreas">
                  <soap:operation soapAction="addSvcAreas" style="document"/>
                  <wsdl:input>
                        <soap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="delSvcAreas">
                  <soap:operation soapAction="delSvcAreas" style="document"/>
                  <wsdl:input>
                        <soap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="getSvcAreas">
                  <soap:operation soapAction="getSvcAreas" style="document"/>
                  <wsdl:input>
```

```
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="addPubIds">
            <soap:operation soapAction="addPubIds" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="delPubIds">
            <soap:operation soapAction="delPubIds" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getPubIds">
            <soap:operation soapAction="getPubIds" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="addTNRs">
            <soap:operation soapAction="addTNRs" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="delTNRs">
            <soap:operation soapAction="delTNRs" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getTNRs">
            <soap:operation soapAction="getTNRs" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="addPvtIds">
            <soap:operation soapAction="addPvtIds" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
```

```
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="delPvtIds">
        <soap:operation soapAction="delPvtIds" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getPvtIds">
        <soap:operation soapAction="getPvtIds" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="addLRNs">
        <soap:operation soapAction="addLRNs" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="delLRNs">
        <soap:operation soapAction="delLRNs" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getLRNs">
        <soap:operation soapAction="getLRNs" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="addNAPTRs">
        <soap:operation soapAction="addNAPTRs" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="delNAPTRs">
        <soap:operation soapAction="delNAPTRs" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getNAPTRs">
        <soap:operation soapAction="getNAPTRs" style="document"/>
```

```
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="addNSs">
        <soap:operation soapAction="addNSs" style="document"/>
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="delNSs">
        <soap:operation soapAction="delNSs" style="document"/>
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getNSs">
        <soap:operation soapAction="getNSs" style="document"/>
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="addEgrRtes">
        <soap:operation soapAction="addEgrRtes" style="document"/>
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="delEgrRtes">
        <soap:operation soapAction="delEgrRtes" style="document"/>
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getEgrRtes">
        <soap:operation soapAction="getEgrRtes" style="document"/>
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal"/>
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="batchUpdate">
        <soap:operation soapAction="batchUpdate" style="document"/>
        <wsdl:input>
                <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
```

```
                        <soap:body use="literal"/>
                    </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="addEntr">
                    <soap:operation soapAction="addEntr" style="document"/>
                    <wsdl:input>
                        <soap:body use="literal"/>
                    </wsdl:input>
                    <wsdl:output>
                        <soap:body use="literal"/>
                    </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="modEntr">
                    <soap:operation soapAction="modEntr" style="document"/>
                    <wsdl:input>
                        <soap:body use="literal"/>
                    </wsdl:input>
                    <wsdl:output>
                        <soap:body use="literal"/>
                    </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="delEntr">
                    <soap:operation soapAction="delEntr" style="document"/>
                    <wsdl:input>
                        <soap:body use="literal"/>
                    </wsdl:input>
                    <wsdl:output>
                        <soap:body use="literal"/>
                    </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="getSvcMenu">
                    <soap:operation soapAction="getSvcMenu" style="document"/>
                    <wsdl:input>
                        <soap:body use="literal"/>
                    </wsdl:input>
                    <wsdl:output>
                        <soap:body use="literal"/>
                    </wsdl:output>
                </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="ESPPService">
                <wsdl:port name="ESPPPort" binding="esp:ESPPSoapBinding">
                    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

9.2 XSD Types Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:est="urn:cablelabs:packetcable:espp:types-v4"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="urn:cablelabs:packetcable:espp:types-v4"
targetNamespace="urn:cablelabs:packetcable:espp:types-v4"
elementFormDefault="qualified" xml:lang="EN">
        <annotation>
                <documentation>
                ------------------ Object Type Definitions ------------------------------
--------
                </documentation>
        </annotation>
        <complexType name="RteType">
                <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
```

```
                    <element name="rteName" type="string"/>
                    <element name="rrId" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    <element name="isInSvc" type="boolean"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="SvcAreaType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="saName" type="string"/>
                    <element name="rteId" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="PubIdType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="pubId" type="string"/>
                    <element name="svcs" type="string" minOccurs="0"/>
                    <element name="saId" type="est:OIdType" minOccurs="0"/>
                    <element name="naptrId" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    <element name="pvtId" type="est:OIdType" minOccurs="0"/>
                    <element name="npParams" type="est:NPParamsType" minOccurs="0"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="PvtIdType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="pvtId" type="string"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="TNRType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="tnRStrt" type="string"/>
                    <element name="tnREnd" type="string"/>
                    <element name="saId" type="est:OIdType"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="LRNType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="rn" type="string"/>
                    <element name="saId" type="est:OIdType"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="NAPTRType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="order" type="unsignedShort"/>
                    <element name="pref" type="unsignedShort"/>
                    <element name="flags" type="string" minOccurs="0"/>
                    <element name="svcs" type="string"/>
                    <element name="regx" type="string" minOccurs="0"/>
```

```
                    <element name="repl" type="string" minOccurs="0"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="NSType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="name" type="string"/>
                    <element name="ipAddr" type="est:IPAddrType" minOccurs="0"
maxOccurs="unbounded"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="EgrRteType">
             <sequence>
                    <element name="oid" type="est:OIdType"/>
                    <element name="eid" type="est:EIdType"/>
                    <element name="rteId" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    <element name="pref" type="unsignedShort"/>
                    <element name="svcs" type="string" minOccurs="0"/>
                    <element name="regxRewriteRule" type="string"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <simpleType name="OIdType">
             <restriction base="unsignedLong"/>
      </simpleType>
      <simpleType name="EIdType">
             <restriction base="string"/>
      </simpleType>
      <simpleType name="ClientIdType">
             <restriction base="int"/>
      </simpleType>
      <simpleType name="TransIdType">
             <restriction base="unsignedLong"/>
      </simpleType>
      <simpleType name="minorVerType">
             <restriction base="unsignedLong"/>
      </simpleType>
      <complexType name="BasicRspnsType">
             <sequence>
                    <element name="resCode" type="int"/>
                    <element name="resMsg" type="string"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="BasicRqstType">
             <sequence>
                    <element name="clientId" type="est:ClientIdType"/>
                    <element name="transId" type="est:TransIdType"/>
                    <element name="minorVer" type="est:minorVerType"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="BasicQueryType">
             <sequence>
                    <element name="clientId" type="est:ClientIdType"/>
                    <element name="minorVer" type="est:minorVerType"/>
                    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
             </sequence>
      </complexType>
      <complexType name="IPAddrType">
             <sequence>
                    <element name="addr" type="string"/>
                    <element name="type" type="est:IPType"/>
```

```
                <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
            </sequence>
        </complexType>
        <simpleType name="IPType">
            <restriction base="token">
                <enumeration value="IPv4"/>
                <enumeration value="IPv6"/>
            </restriction>
        </simpleType>
        <complexType name="BatchUpdateType">
            <sequence>
                <element name="op" type="est:BatchOpType" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
        <complexType name="BatchOpType">
            <sequence>
                <element name="naptrDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="naptrAdd" type="est:NAPTRType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="nsDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="nsAdd" type="est:NSType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="egrRteDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="egrRteAdd" type="est:EgrRteType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="rteDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="rteAdd" type="est:RteType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="saDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="saAdd" type="est:SvcAreaType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="pviDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="pviAdd" type="est:PvtIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="piDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="piAdd" type="est:PubIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="tnRDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="tnRAdd" type="est:TNRType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="lrnDel" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="lrnAdd" type="est:LRNType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="entrDel" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="entrAdd" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="entrMod" minOccurs="0" maxOccurs="unbounded">
                    <complexType>
                        <sequence>
                            <element name="newEId" type="est:EIdType"/>
                            <element name="oldEId" type="est:EIdType"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
        <complexType name="BasicFileRqstType">
```

```
            <sequence>
                  <element name="clientId" type="est:ClientIdType"/>
                  <element name="serverId" type="string"/>
                  <element name="transId" type="est:TransIdType"/>
                  <element name="minorVer" type="est:minorVerType"/>
                  <element name="isFullResync" type="boolean"/>
                  <element name="creationTimestamp" type="dateTime"/>
                  <element name="sequenceNumber" type="unsignedLong"/>
                  <element name="isEndOfSequence" type="boolean"/>
                  <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
            </sequence>
      </complexType>
      <complexType name="SvcMenuType">
            <sequence>
                  <element name="serverStatus" type="est:ServerStatusType"/>
                  <element name="majMinVersion" type="string"
maxOccurs="unbounded"/>
                  <element name="objURI" type="anyURI" maxOccurs="unbounded"/>
                  <element name="extURI" type="anyURI" minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
      </complexType>
      <simpleType name="ServerStatusType">
            <restriction base="token">
                  <enumeration value="inService"/>
                  <enumeration value="outOfService"/>
            </restriction>
      </simpleType>
      <complexType name="NPParamsType">
            <sequence>
                  <element name="npdi" type="boolean"/>
                  <element name="rn" type="string" minOccurs="0"/>
            </sequence>
      </complexType>
      <complexType name="ExtAnyType">
            <sequence>
                  <any namespace="##other" maxOccurs="unbounded"/>
            </sequence>
      </complexType>
      <annotation>
            <documentation>
            -------------- Wrapped Rqst Message Definitions ------------------------
--------
            </documentation>
      </annotation>
      <element name="addRtesRqst">
            <complexType>
                  <sequence>
                        <element name="basicRqst" type="est:BasicRqstType"/>
                        <element name="rte" type="est:RteType"
maxOccurs="unbounded"/>
                  </sequence>
            </complexType>
      </element>
      <element name="delRtesRqst">
            <complexType>
                  <sequence>
                        <element name="basicRqst" type="est:BasicRqstType"/>
                        <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                  </sequence>
            </complexType>
      </element>
      <element name="getRtesRqst">
            <complexType>
                  <sequence>
                        <element name="basicRqst" type="est:BasicQueryType"/>
```

```
                        <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
        <element name="addSvcAreasRqst">
            <complexType>
                <sequence>
                    <element name="basicRqst" type="est:BasicRqstType"/>
                    <element name="svcArea" type="est:SvcAreaType"
maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
        <element name="delSvcAreasRqst">
            <complexType>
                <sequence>
                    <element name="basicRqst" type="est:BasicRqstType"/>
                    <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
        <element name="getSvcAreasRqst">
            <complexType>
                <sequence>
                    <element name="basicRqst" type="est:BasicQueryType"/>
                    <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
        <element name="addPubIdsRqst">
            <complexType>
                <sequence>
                    <element name="basicRqst" type="est:BasicRqstType"/>
                    <element name="pi" type="est:PubIdType"
maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
        <element name="delPubIdsRqst">
            <complexType>
                <sequence>
                    <element name="basicRqst" type="est:BasicRqstType"/>
                    <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
        <element name="getPubIdsRqst">
            <complexType>
                <sequence>
                    <element name="basicRqst" type="est:BasicQueryType"/>
                    <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
        <element name="addPvtIdsRqst">
            <complexType>
```

```
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="pvi" type="est:PvtIdType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="delPvtIdsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="getPvtIdsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicQueryType"/>
                          <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                          <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="addTNRsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="tnR" type="est:TNRType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="delTNRsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="getTNRsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicQueryType"/>
                          <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                          <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="addLRNsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="lrn" type="est:LRNType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="delLRNsRqst">
             <complexType>
```

```
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="getLRNsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicQueryType"/>
                          <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                          <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="addNAPTRsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="naptr" type="est:NAPTRType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="delNAPTRsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="getNAPTRsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicQueryType"/>
                          <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                          <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="addNSsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="naptr" type="est:NSType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="delNSsRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="getNSsRqst">
             <complexType>
```

```
                    <sequence>
                          <element name="basicRqst" type="est:BasicQueryType"/>
                          <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                          <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="addEgrRtesRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="egrRte" type="est:EgrRteType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="delEgrRtesRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="oid" type="est:OIdType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="getEgrRtesRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicQueryType"/>
                          <element name="oid" type="est:OIdType" minOccurs="0"
maxOccurs="unbounded"/>
                          <element name="eid" type="est:EIdType" minOccurs="0"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="batchUpdateRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="batchUpdate" type="est:BatchUpdateType"/>
                    </sequence>
             </complexType>
      </element>
      <element name="batchUpdateFileRqst">
             <complexType>
                    <sequence>
                          <element name="basicFileRqst" type="est:BasicFileRqstType"/>
                          <element name="batchUpdate" type="est:BatchUpdateType"
maxOccurs="unbounded"/>
                    </sequence>
             </complexType>
      </element>
      <element name="addEntrRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
                          <element name="eid" type="est:EIdType"/>
                    </sequence>
             </complexType>
      </element>
      <element name="modEntrRqst">
             <complexType>
                    <sequence>
                          <element name="basicRqst" type="est:BasicRqstType"/>
```

```
                        <element name="newEId" type="est:EIdType"/>
                        <element name="oldEId" type="est:EIdType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="delEntrRqst">
                <complexType>
                    <sequence>
                        <element name="basicRqst" type="est:BasicRqstType"/>
                        <element name="eid" type="est:EIdType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getSvcMenuRqst">
                <complexType>
                    <sequence>
                        <element name="basicRqst" type="est:BasicQueryType"/>
                    </sequence>
                </complexType>
        </element>
        <annotation>
                <documentation>
                -------- Wrapped Rspns Message Definitions ------------------------------
--
                </documentation>
        </annotation>
        <element name="getRtesRspns">
                <complexType>
                    <sequence>
                        <element name="rte" type="est:RteType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getSvcAreasRspns">
                <complexType>
                    <sequence>
                        <element name="svcArea" type="est:SvcAreaType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getPubIdsRspns">
                <complexType>
                    <sequence>
                        <element name="pi" type="est:PubIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getPvtIdsRspns">
                <complexType>
                    <sequence>
                        <element name="pvi" type="est:PvtIdType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getTNRsRspns">
                <complexType>
                    <sequence>
                        <element name="tnR" type="est:TNRType" minOccurs="0"
maxOccurs="unbounded"/>
```

```
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getLRNsRspns">
                <complexType>
                    <sequence>
                        <element name="lrn" type="est:LRNType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getNAPTRsRspns">
                <complexType>
                    <sequence>
                        <element name="naptr" type="est:NAPTRType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getNSsRspns">
                <complexType>
                    <sequence>
                        <element name="ns" type="est:NSType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getEgrRtesRspns">
                <complexType>
                    <sequence>
                        <element name="egrRte" type="est:EgrRteType" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="getSvcMenuRspns">
                <complexType>
                    <sequence>
                        <element name="svcMenu" type="est:SvcMenuType"/>
                        <element name="basicResult" type="est:BasicRspnsType"/>
                    </sequence>
                </complexType>
        </element>
        <element name="cmnRspns">
                <complexType>
                    <sequence>
                        <element name="rspns" type="est:BasicRspnsType"
nillable="false"/>
                    </sequence>
                </complexType>
        </element>
</schema>
```

10 DATA ELEMENT VALIDATION CONSTRAINTS

This section contains formal validation rules for the data elements defined in ESPP.

An ESPP Client and Server SHOULD conform to the following data validation constraints for the data elements that are passed over the ESPP interface. When these validation constraints are violated, the ESPP Server SHOULD return the appropriate error code defined in the error code table for each operation.

Table 20 - Validation Rules for BasicRqstType Attributes

Attribute	Constraints	Notes
clientId	int	
transId	unsignedLong (64-bit) min: 0	Monotonically increasing
minorVer	unsignedLong (64-bit) min: 0 max: 99	The minor version for the data models contained within this revision of the specification is '0'.

Table 21 - Validation Rules for BasicQueryType Attributes

Attribute	Constraints	Notes
minorVer	unsignedLong (64-bit) min: 0 max: 99	The minor version for the data models contained within this revision of the specification is '0'.

Table 22 - Validation Rules for BasicRspnsType Attributes

Attribute	Constraints	Notes
resCode	int	See Section 8, Response Codes and Messages
resMsg	string	See Table 19 - Response Codes Numbering Scheme and Messages

Table 23 - Validation Rules for Id Types Attributes

Attribute	Constraints	Notes
oid	unsignedLong (64-bit) min: 0 max: 18,446,744,073,709,551,615	

Attribute	Constraints	Notes
eid	string min length: 1 max length: 32 regexp: .*	The element identifier must exist. It is created by the addEntrRqst or modEntrRqst requests.

Table 24 - Validation Rules for RteType Attributes

Attribute	Constraints	Notes
rteName	string min length: 1 max length: 255 regexp: .*	
drrId	0 or more of `OIdType`	Each resource record in the list must exist.

Table 25 - Validation Rules for SvcAreaType Attributes

Attribute	Constraints	Notes
saName	string min length: 1 max length: 255 regexp: .*	
rteId	0 or more of `OIdType`	Each Route in the list must exist.

Table 26 - Validation Rules for PubIdType Attributes

Attribute	Constraints	Notes
pubId	string min length: 1 max length: 255 regexp: .*	
svcs	string min length: 1 max length: 255 regexp: ^E2U([+](X-)?[A-Za-z0-9]{1,32}(\:(X-)?[A-Za-z0-9]{1,32})*)+$	Case Insensitive See [RFC 3761]

Attribute	Constraints	Notes
saId	0 or 1 of `OIdType`	Service Area must exist if present.
naptrId	0 or more of `OIdType`	Each NAPTR in the list must exist.
pvtId	0 or 1 of `OIdType`	Private Id must exist if present.

Table 27 - Validation Rules for PvtIdType Attributes

Attribute	Constraints	Notes
pvtId	string min length: 1 max length: 255 regexp: .*	

Table 28 - Validation Rules for TNRType Attributes

Attribute	Constraints	Notes
tnRStrt	string min length: 1 max length: 16 regexp: [0-9]+	The value of tnRStrt must be reducible to an unsignedLong.
tnREnd	string min length: 1 max length: 16 regexp: [0-9]+	The value of tnREnd must be reducible to an unsignedLong and must be greater or equal to the value of tnRStrt.
saId	0 or 1 of `OIdType`	The Service Area must exist if present.

Table 29 - Validation Rules for LRNType Attributes

Attribute	Constraints	Notes
rn	string min length: 1 max length: 16 regexp: [0-9]+	
saId	`OIdType`	The Service Area must exist.

Table 30 - Validation Rules for NAPTRType Attributes

Attribute	Constraints	Notes
flags	string regexp: [A-Z0-9]*	Case insensitive
svcs	string min length: 1 max length: 255 regexp: ^E2U([+](X-)?[A-Za-z0-9]{1,32}(\:(X-)?[A-Za-z0-9]{1,32})*)+$	See [RFC 3761]
regx	string min length: 1 max length: 255 regexp: .*	See [RFC 3402] and [RFC 3403].
repl	string min length: 1 max length: 255 regexp: .*	See [RFC 3403]**.** It must be FQDN

Table 31 - Validation Rules for EgrRteType Attributes

Attribute	Constraints	Notes
rteId	0 or more of `OIdType`	Each Route in the list must exist.
svcs	string min length: 1 max length: 255 regexp: ^E2U([+](X-)?[A-Za-z0-9]{1,32}(\:(X-)?[A-Za-z0-9]{1,32})*)+$	Case insensitive See [RFC 3761]
regxRewriteRule	string min length: 1 max length: 255 regexp: .*	

Table 32 - Validation Rules for BasicFileRqstType Attributes

Attribute	Constraints	Notes
clientId	int	
serverId	string min length: 1 max length: 32 regexp: .*	
transId	unsignedLong (64-bit) min: 0	Monotonically increasing
minorVer	unsignedLong (64-bit) min: 0 max: 99	The minor version for the data models contained within this revision of the specification is '0'.
creationTimestamp	dateTime	The value of creationTimestamp MUST include time zone information.
sequenceNumber	unsignedLong (64-bit) min: 0	

Appendix I Generic ESPP Examples

This section contains generic example SOAP and XML messages for each operation. These examples follow the approach of prefixing each type with its appropriate namespace, however, this is not the required approach. The standard XML rules for namespace prefixes apply. Therefore, it is permissible within the ESPP protocol to identify the namespace in the header and not prefix the namespace to each type within the XML body.

This section contains generic example SOAP and XML messages for each operation. These examples follow the approach of prefixing each type with its appropriate namespace, however, this is not the required approach. The standard XML rules for namespace prefixes apply. Therefore, it is permissible within the ESPP protocol to identify the namespace in the header and not prefix the namespace to each type within the XML body.

I.1 Operation: addRtes

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addRtesRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>38</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:rte>
      <est:oid>7845601000000003100</est:oid>
      <est:eid>76543</est:eid>
     <est:rteName>RT1-SBE1-1</est:rteName>
     <est:sbeNaprtId>7845601000000001000</est:sbeNaprtId>
     <est:isInSvc>false</est:isInSvc>
    </est:rte>
   </est:addRtesRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:cmnRspns>
   <est:rspns>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:rspns>
  </est:cmnRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.2 Operation: addSvcAreas

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addSvcAreasRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>40</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:svcArea>
     <est:oid>7845601000000005000</est:oid>
     <est:eid>76543</est:eid>
     <est:saName>SA1-CMS1-2</est:saName>
     <est:rteId>7845601000000003200</est:rteId>
    </est:svcArea>
   </est:addSvcAreasRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>


Response: See above.
```

I.3 Operation: addTNRs

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addTNRsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>44</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:tnR>
     <est:oid>7845601000000006000</est:oid>
      <est:eid>76543</est:eid>
     <est:tnRStrt>4001000000</est:tnRStrt>
     <est:tnREnd>4001000499</est:tnREnd>
     <est:saId>7845601000000005000</est:saId>
    </est:tnR>
   </est:addTNRsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response: See above.
```

I.4 Operation: addEgrRtes

```
Request:
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addEgrRtesRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>30</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:egrRte>
     <est:oid>7845601000000004000</est:oid>
     <est:eid>76543</est:eid>
     <est:rteId>7845601000000003100</est:rteId>
     <est:pref>1</est:pref>
     <est:svcs>E2U+SIP</est:svcs>
     <est:regxRewriteRule>!^.*$!\\1@cms1-1.mso-aEgressRoute.com</est:regxRewriteRule>
    </est:egrRte>
   </est:addEgrRtesRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response: See above.

I.5 Operation: addLRNs

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addLRNsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>32</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:lrn>
     <est:oid>7845601000000006100</est:oid>
     <est:eid>76543</est:eid>
     <est:rn>9194605000</est:rn>
     <est:saId>7845601000000005000</est:saId>
    </est:lrn>
   </est:addLRNsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response: See above.

I.6 Operation: addNAPTRs

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
```

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addNAPTRsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>33</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:naptr>
     <est:oid>7845601000000001100</est:oid>
     <est:eid>76543</est:eid>
     <est:order>10</est:order>
     <est:pref>10</est:pref>
     <est:flags>u</est:flags>
     <est:svcs>E2U+SIP</est:svcs>
     <est:regx>!^(.*)$!sip:\1@sbe1-1-Inter.mso-a.example.com!</est:regx>
    </est:naptr>
   </est:addNAPTRsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response: See above.

I.7 Operation: addPubIds

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addPubIdsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>36</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:pi>
     <est:oid>7845601000000006200</est:oid>
     <est:eid>76543</est:eid>
     <est:pubId>9194605555</est:pubId>
     <est:svcs>E2U+SIP</est:svcs>
     <est:saId>7845601000000005000</est:saId>
     <est:naptrId>7845601000000001500</est:naptrId>
     <est:pvtId>7845601000000002000</est:pvtId>
    </est:pi>
   </est:addPubIdsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response: See above.
```

I.8 Operation: addPvtIds

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:addPvtIdsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>37</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:pvi>
     <est:oid>7845601000000002100</est:oid>
     <est:eid>76543</est:eid>
     <est:pvtId>9194613000</est:pvtId>
    </est:pvi>
   </est:addPvtIdsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>


Response: See above.
```

I.9 Operation: batchUpdate

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:batchUpdateRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>51</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:batchUpdate>
     <est:op>
      <est:naptrToDel>7845601000000001000</est:naptrToDel>
      <est:naptrToAdd>
       <est:oid>7845601000000001000</est:oid>
       <est:eid>76543</est:eid>
       <est:order>10</est:order>
       <est:pref>10</est:pref>
       <est:flags>u</est:flags>
       <est:svcs>E2U+SIP</est:svcs>
       <est:regx>!^(.*)$!sip:\1@sbe1-1-Inter.mso-a.example.com!</est:regx>
      </est:naptrToAdd>
      <est:egrRteToDel>7845601000000001000</est:egrRteToDel>
      <est:egrRteToAdd>
       <est:oid>7845601000000001000</est:oid>
       <est:eid>76543</est:eid>
       <est:rteId>7845601000000002000</est:rteId>
       <est:pref>1</est:pref>
```

```
       <est:svcs>E2U+SIP</est:svcs>
       <est:regxRewriteRule>!^.*$!\\1@cms1-1.msoEgressRoute.com!</est:regxRewriteRule>
      </est:egrRteToAdd>
      <est:rteToDel>7845601000000003000</est:rteToDel>
      <est:rteToAdd>
       <est:oid>7845601000000003001</est:oid>
       <est:eid>76543</est:eid>
       <est:rteName>RT1-SBE1-1</est:rteName>
       <est:sbeNaprtId>7845601000000005000</est:sbeNaprtId>
       <est:isInSvc>false</est:isInSvc>
      </est:rteToAdd>
      <est:saToDel>7845601000000001100</est:saToDel>
      <est:saToAdd>
       <est:oid>7845601000000001100</est:oid>
       <est:eid>76543</est:eid>
       <est:saName>SA1-CMS1-2</est:saName>
       <est:rteId>7845601000000001000</est:rteId>
      </est:saToAdd>
      <est:pvtIdToDel>7845601000000003215</est:pvtIdToDel>
      <est:pvtIdToAdd>
       <est:oid>7845601000000009000</est:oid>
       <est:eid>76543</est:eid>
       <est:pvtId>7845601009194613000</est:pvtId>
      </est:pvtIdToAdd>
      <est:pubIdToDel>7845601000000005000</est:pubIdToDel>
      <est:pubIdToAdd>
       <est:oid>7845601000006523489</est:oid>
       <est:eid>76543</est:eid>
       <est:pubId>9194605555</est:pubId>
       <est:saId>7845601000000006000</est:saId>
       <est:naptrId>7845601000000003000</est:naptrId>
       <est:pvtId>7845601000000008000</est:pvtId>
      </est:pubIdToAdd>
      <est:tnRToDel>12000</est:tnRToDel>
      <est:tnRToAdd>
       <est:oid>7845601000000012001</est:oid>
       <est:eid>76543</est:eid>
       <est:tnRStrt>4001000000</est:tnRStrt>
       <est:tnREnd>4001000499</est:tnREnd>
       <est:saId>7845601000000003000</est:saId>
      </est:tnRToAdd>
      <est:lrnToDel>7845601000003264782</est:lrnToDel>
      <est:lrnToAdd>
       <est:oid>7845601000000003000</est:oid>
       <est:eid>76543</est:eid>
       <est:rn>9194605000</est:rn>
       <est:saId>7845601000000004000</est:saId>
      </est:lrnToAdd>
     </est:op>
    </est:batchUpdate>
   </est:batchUpdateRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response: See above.
```

I.10 Operation: delRtes

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
```

```
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:delRtesRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>38</est:transId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
     <est:oid>7845601000000003000</est:oid>
   </est:delRtesRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response: See above.
```

I.11 Operation: getRtes

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:getRtesRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
     <est:oid>7845601000000005000</est:oid>
     <est:eid>76543</est:eid>
   </est:getRtesRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:getRtesRspns>
   <est:rte>
    <est:oid>7845601000000003000</est:oid>
    <est:eid>76543</est:eid>
    <est:rteName>RT1-SBE1-1</est:rteName>
    <est:sbeNaprtId>5000</est:sbeNaprtId>
    <est:isInSvc>false</est:isInSvc>
   </est:rte>
   <est:basicResult>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:basicResult>
  </est:getRtesRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.12 Operation: getSvcAreas

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:getSvcAreasRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
     <est:oid>7845601000000011000</est:oid>
     <est:eid>76543</est:eid>
   </est:getSvcAreasRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:getSvcAreasRspns>
   <est:svcArea>
    <est:oid>7845601000000011000</est:oid>
    <est:eid>76543</est:eid>
    <est:saName>SA1-CMS1-2</est:saName>
    <est:rteId>78456010000000001001</est:rteId>
   </est:svcArea>
   <est:basicResult>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:basicResult>
  </est:getSvcAreasRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.13 Operation: delTNRs

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:delTNRsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:transId>58</est:transId>
     <est:minorVer>1</est:minorVer>
```

```
    </est:basicRqst>
     <est:oid>7845601000000012000</est:oid>
   </est:delTNRsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>


Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:cmnRspns>
   <est:rspns>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:rspns>
  </est:cmnRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.14 Operation: getEgrRtes

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:getEgrRtesRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
     <est:oid>7845601000000001000</est:oid>
     <est:eid>76543</est:eid>
   </est:getEgrRtesRqst>

 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>


Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:getEgrRtesRspns>
   <est:egrRte>
     <est:oid>7845601000000001000</est:oid>
```

```
     <est:eid>76543</est:eid>
    <est:rteId>7845601000000002000</est:rteId>
    <est:pref>1</est:pref>
    <est:svcs>E2U+SIP</est:svcs>
    <est:regxRewriteRule>!^.*$!\\1@cms1-1.mso-a-EgressRoute.com!</est:regxRewriteRule>
   </est:egrRte>
   <est:basicResult>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:basicResult>
  </est:getEgrRtesRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.15 Operation: getLRNs

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:getLRNsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
    <est:oid>7845601000000003000</est:oid>
   </est:getLRNsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>


Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:getLRNsRspns>
    <est:lrn>
     <est:oid>7845601000000003000</est:oid>
     <est:eid>76543</est:eid>
     <est:rn>9194605000</est:rn>
     <est:saId>7845601000000005000</est:saId>
    </est:lrn>
   <est:basicResult>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:basicResult>
  </est:getLRNsRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.16 Operation: getNAPTRs

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:getNAPTRsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
     <est:oid>7845601000000000103</est:oid>
   </est:getNAPTRsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:getNAPTRsRspns>
   <est:naptr>
     <est:oid>7845601000000000103</est:oid>
     <est:eid>76543</est:eid>
    <est:order>10</est:order>
    <est:pref>10</est:pref>
    <est:flags>u</est:flags>
    <est:svcs>E2U+SIP</est:svcs>
    <est:regx>!^(.*)$!sip:\1@sbe1-1-Inter.mso-a.example.com!</est:regx>
   </est:naptr>
   <est:basicResult>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:basicResult>
  </est:getNAPTRsRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.17 Operation: getPubIds

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:getPubIdsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
```

```
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
     <est:oid>78456010000000005000</est:oid>
   </est:getPubIdsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:cmnRspns>
   <est:rspns>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:rspns>
  </est:cmnRspns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.18 Operation: getPvtIds

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
   <est:getPvtIdsRqst>
    <est:basicRqst>
     <est:clientId>7845601</est:clientId>
     <est:minorVer>1</est:minorVer>
    </est:basicRqst>
     <est:oid>78456010000000009000</est:oid>
   </est:getPvtIdsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response:

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:esp="urn:cablelabs:packetcable: espp:api-1.0"
 xmlns:est="urn:cablelabs:packetcable: espp:types-1.0">
 <SOAP-ENV:Body>
  <est:cmnRspns>
   <est:rspns>
    <est:resCode>1000</est:resCode>
    <est:resMsg>Success</est:resMsg>
   </est:rspns>
  </est:cmnRspns>
```

```
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I.19 Operation: batchUpdateFileRqst

```
Request:
<?xml version="1.0" encoding="UTF-8"?>
<est:batchUpdateFileRqst>
     <est:basicFileRqst>
      <est:clientId>123456</est:clientId>
      <est:serverId>ESPP-SERVER-1</est:serverId>
      <est:transId>78</est:transId>
      <est:minorVer>1</est:minorVer>
      <est:isFullResync>false</est:isFullResync>
      <est:creationTimestamp> Wed Dec 05 21:35:14 EST 2007</ est:creationTimestamp>
      <est:sequenceNumber>1</est:sequenceNumber>
      <est:isEndOfSequence>true</est:isEndOfSequence>
      </est:ext>
     </est:basicFileRqst>
     <est:batchUpdate>
      <est:op>
       <est:naptrToDel>1000</est:naptrToDel>
       <est:naptrToAdd>
        <est:oid>1000</est:oid>
        <est:eid>76543</est:eid>
        <est:order>10</est:order>
        <est:pref>10</est:pref>
        <est:flags>u</est:flags>
        <est:svcs>E2U+sip</est:svcs>
        <est:regx>!^(.*)$!sip:\1@sbe-1-1-Inter.mso-a.example.com!</est:regx>
       </est:naptrToAdd>
      <est:egrRteToDel>1000</est:egrRteToDel>
      <est:egrRteToAdd>
       <est:oid>1000</est:oid>
       <est:eid>76543</est:eid>
       <est:rteId>2000</est:rteId>
       <est:pref>10</est:pref>
       <est:svcs>E2U+sip</est:svcs>
<est:regxRewriteRule>!^.*$!\\1@cms1-1.msoEgressRoute.com!</est:regxRewriteRule>
      </est:egrRteToAdd>
      <est:rteToDel>3000</est:rteToDel>
      <est:rteToAdd>
       <est:oid>3000</est:oid>
       <est:eid>76543</est:eid>
       <est:rteName>RT1-SBE1-1</est:rteName>
       <est:sbeNaprtId>5000</est:sbeNaprtId>
       <est:isInSvc>false</est:isInSvc>
      </est:rteToAdd>
      <est:saToDel>11000</est:saToDel>
      <est:saToAdd>
       <est:oid>11000</est:oid>
       <est:eid>76543</est:eid>
       <est:saName>SA1-CMS1-2</est:saName>
       <est:rteId>1000</est:rteId>
      </est:saToAdd>
      <est:pvtIdToDel>9000</est:pvtIdToDel>
      <est:pvtIdToAdd>
       <est:oid>9000</est:oid>
       <est:eid>76543</est:eid>
       <est:pvtId>9194613000</est:pvtId>
      </est:pvtIdToAdd>
      <est:pubIdToDel>5000</est:pubIdToDel>
      <est:pubIdToAdd>
       <est:oid>5000</est:oid>
       <est:eid>76543</est:eid>
       <est:pubId>9194605555</est:pubId>
       <est:saId>6000</est:saId>
       <est:naptrId>3000</est:naptrId>
       <est:pvtId>8000</est:pvtId>
      </est:pubIdToAdd>
```

```
       <est:tnRToDel>12000</est:tnRToDel>
       <est:tnRToAdd>
        <est:oid>12000</est:oid>
        <est:eid>76543</est:eid>
        <est:tnRStrt>4001000000</est:tnRStrt>
        <est:tnREnd>4001000499</est:tnREnd>
        <est:saId>3000</est:saId>
       </est:tnRToAdd>
       <est:lrnToDel>3000</est:lrnToDel>
       <est:lrnToAdd>
        <est:oid>3000</est:oid>
        <est:eid>76543</est:eid>
        <est:rn>9194605000</est:rn>
        <est:saId>4000</est:saId>
       </est:lrnToAdd>
      </est:op>
     </est:batchUpdate>
    </est:batchFileUpdateRqst>

Response: N/A
```

Appendix II Acknowledgements

CableLabs wishes to thank the following participants for various contributions and efforts that led to the development of this specification:

Jack Burton - Cablevision

Paul Natale - Charter

Tom Creighton - Comcast

Costas Gavrilidis - Cox Communications

Matt Cannon - Time Warner Cable

James Brister, Ted Lemon, Vivian Neou - Nominum

Mark McBride, Tim Cody, Sean Leach, Gene Lew, Rich Shockey, Mark Teodoro - NeuStar

Robby Benedyk, Steve Dimig - Tekelec

Ken Cartwright, Ajay Gupta, Sean Kent, Tom Kershaw, Manjul Maharishi, Yasir Saleem, Sanjeev Chauhan, Gaurav Sharma - Verisign

Vikas Sarawat, Daryl Malas, Jean-François Mulé - CableLabs

Sumanth Channabasappa (CableLabs)

Appendix III Revision History

The following Engineering Change Notices were incorporated in PKT-SP-ENUM-PROV-I02-080903.

ECN	ECN Date	Summary
ENUM-PROV-N-08.0519-4	5/5/2008	New Success Response Code 1001,
ENUM-PROV-N-08.0520-3	5/27/2008	Update Transaction Id section to handle 'isFullResync', and update XML tags,
ENUM-PROV-N-08.0526-3	8/18/2008	Updates to svcs regexp and addition of BasicFileRqst object,
ENUM-PROV-N-08.0527-4	8/25/2008	Misc clarificaitons to oid, Internal Server Error, Service Area deletion, and eid.

The following Engineering Change Notices were incorporated in PKT-SP-ENUM-PROV-I03-090630.

ECN	ECN Date	Summary
ENUM-PROV-N-09.0562-2	6/8/2009	Add the ability to send Name Server Records from the ESPP client to the ESPP server.
ENUM-PROV-N-09.0563-3	6/15/2009	Add the ability to allow an ESPP client to ping an ESPP server to discover whether or not it is in service.
ENUM-PROV-N-09.0564-1	6/8/2009	Editorial fixes.
ENUM-PROV-N-09.0565-2	6/8/2009	Added example for using egress routes and regular expression corrections.

The following Engineering Change Notices were incorporated in PKT-SP-ENUM-PROV-I04-100415.

ECN	ECN Date	Summary
ENUM-PROV-N-09.0613-3	12/28/2009	NPDI and RN on a Public Identity object.
ENUM-PROV-N-09.0619-2	12/28/2009	Clarification on the handling of Egress Route object.