A Provisioning Protocol for ENUM-SIP Addressing Servers
draft-mule-peppermint-espp-protocol-02

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on January 15, 2009.
Abstract

This document defines a provisioning protocol for ENUM-SIP addressing servers.
An ENUM-SIP addressing server is a host that acts as Lookup Function in session peering to determine the target domain of a given SIP request and it may also act as a Location Routing Function to develop the location of the SIP signaling entity in that target domain.
This protocol allows SIP service providers to provision and manage session establishment data used by SIP network elements to route SIP sessions to the target destinations which may be served by the SIP service provider’s own internal network or by a session peering partner. The data provisioned into an ENUM-SIP addressing server is queried by SIP entities using ENUM or SIP.

This version of the protocol integrates comments received on the IETF peppermint and drinks mailing lists before July 2008. This document is an Internet-Draft and the protocol it describes is subject to technical changes that may make this version incompatible with future versions defined in Internet-Drafts. It is expected that the authors will continue to update this protocol based on the drinks working group requirements on the session establishment data.
Table of Contents

1. Introduction ........................................... 4
2. Terminology ............................................ 6
3. Protocol Requirements ................................. 7
4. Protocol Definition .................................... 11
   4.1. Logical Structure - Data Model .................. 11
   4.2. Technical Structure - SOAP/XML Type Hierarchy ... 15
   4.3. Overview of Protocol Operations ................. 16
       4.3.1. Deployment Scenario and ESPP Provisioning ... 17
       4.3.2. Deployment Scenario and Resolution .......... 33
       4.3.3. Generic Examples .......................... 35
   4.4. Protocol Specification ............................ 35
       4.4.1. General Protocol Concepts .................. 36
       4.4.2. Protocol Operation Descriptions ............ 43
5. File-based ESPP provisioning protocol ................. 86
   5.1. Overview of the File-based Provisioning Operations ... 86
   5.2. File Structure ................................... 86
6. Response Codes and Messages ............................ 89
7. IANA Considerations .................................. 91
8. Formal API Definition ................................ 92
   8.1. WSDL Specification ............................ 92
   8.2. XSD Types Specification ....................... 103
9. Security Considerations .............................. 116
10. Acknowledgments .................................... 117
11. References ........................................ 118
    11.1. Normative References ........................ 118
    11.2. Informative References ....................... 118
Authors’ Addresses ..................................... 120
Intellectual Property and Copyright Statements ........ 121
1. Introduction

This document defines a provisioning protocol for ENUM-SIP addressing servers. It allows SIP service providers to provision data related to their SIP route management. The data is used by ENUM-SIP addressing servers to return part of the session establishment data to SIP network elements so that they can route SIP sessions to the proper target destinations.

An ENUM-SIP addressing server is a session routing server that implements a Lookup Function and may also implement a Location Routing Function as defined in [I-D.ietf-speermint-terminology]. An ENUM-SIP addressing server typically takes a user address that can be used to establish a SIP session as the input to the lookup query (SIP address of record, a telephone number or any type of public user addresses). It resolves the address into one or more Uniform Resource Identifiers (URIs) based on various rules and routing logic. Optionally, it may also develop the responses to include information on how to route the SIP session to the requesting entity. The data provisioned into an ENUM-SIP addressing server is queried by SIP entities using protocols such as ENUM [RFC3761] or Session Establishment Protocol (SIP) [RFC3261]. While the protocols used to query the provisioned data may impose limitations on the data modeling, they are considered out of scope of this document.

The use cases and protocol requirements for the ENUM-SIP Server Provisioning Protocol (ESPP) are described in [I-D.espp-requirements]. For reference, the original version of this protocol is published as a CableLabs specification, the PacketCable ENUM Server Provisioning Specification ([CableLabs-ESPP]). A number of vendors have client and server implementations of this original protocol and three interoperability testing events have been conducted to date.

This document is organized as follows:

- **Section 3** describes the protocol’s functional entities and provides a pointer to the use cases and protocol requirements maintained in a separate document,

- **Section 4** and **Section 5** define the real-time and filed-based operations of the ESPP provisioning protocol including the data model (Section 4.1), the SOAP/XML Type Hierarchy (Section 4.2), and an overview of the protocol operations by walking through a real-life deployment scenario showing examples of ESPP operations and how the provisioned data can be used to resolve ENUM or SIP queries,(Section 4.3),
Section 8 provides the complete normative WSDL and XSD definitions of the protocol.
2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document also reuses the SIP terminology defined in [RFC3261]. The Lookup Function (LUF), Location Routing Functions (LRF) and other session peering terms are defined [I-D.ietf-speermint-terminology].
3. Protocol Requirements

The ENUM Server Provisioning Protocol (ESPP) allows a SIP Service Provider (SSP) to receive session establishment-related data from various sources: its internal back-office systems for intra-domain routes, a session peer or a federation registry. A source of provisioning data is logically represented by an ESPP provisioning client (ESPP Client). The SSP Service Provider’s addressing server receives this provisioning data via a functional entity called an ESPP Server. This data is then used by the server’s Lookup and Location routing Functions to return session establishment data to querying entities.

The data sent by an ESPP Client to an ESPP Server follows a formal data structure based on the data model defined in Section 4.1. The data elements include parameters such as a peer’s target addresses grouped by destination groups 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.
To claim compliance with this specification, an ESPP Client and an ESPP Server MUST conform to the real-time and file-based provisioning protocol requirements defined in Section 4, Section 5 and Section 8 of this document. The protocols used by SIP entities and various network elements to query the addressing server are typically 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 destination group. Through a route assignment each node is associated with one or more destination groups. For example, a SIP Service Provider could first create destination groups for the New York metropolitan area, assigning the telephone numbers of New York subscribers into that destination group. The SIP Service Provider 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-SSP and inter-SSP routes, that SIP Service Provider completes the establishment of session peering data by associating nodes with destination groups.
Figure 2 illustrates the link between logical destination groups and telephone numbers and the associated routes.

### Destination Groups:

<table>
<thead>
<tr>
<th>Destination Group Name</th>
<th>TN or TN Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manhattan</td>
<td>212-203-0000 -&gt; 212-203-9999</td>
</tr>
<tr>
<td></td>
<td>347-876-1000 -&gt; 347-876-1999</td>
</tr>
<tr>
<td></td>
<td>347-354-6000 -&gt; 347-354-6999</td>
</tr>
</tbody>
</table>

### Routes:

<table>
<thead>
<tr>
<th>Route Name</th>
<th>Nodes</th>
<th>Dest. Groups</th>
<th>(Manhattan)</th>
</tr>
</thead>
<tbody>
<tr>
<td>118th Ave</td>
<td>NYC-SBE-1</td>
<td>Manhattan/Bronx</td>
<td>--&gt;(SBE)</td>
</tr>
<tr>
<td>60 Hudson</td>
<td>NYC-SBE-2</td>
<td>Queens</td>
<td>‘’--’’ (Queens)</td>
</tr>
</tbody>
</table>

### Nodes:

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Host/Domain Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NYC-SBE-1</td>
<td>sbe-1.nyc.example.com</td>
</tr>
<tr>
<td>NYC-SBE-2</td>
<td>sbe-2.nyc.example.com</td>
</tr>
</tbody>
</table>

Example of ESPP Destination Groups and Routes

Figure 2

A number of design considerations and requirements (transport, data model, security, etc.) can be found in [I-D.espp-requirements].
4. 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.

4.1. Logical Structure - Data Model

The data model illustrated and described in Figure 3 defines the logical objects and the relationships between these objects that the ESPP protocol supports. ESPP defines the protocol operations through which an ESPP Client populates an ENUM-SIP addressing server with these logical objects. Various clients may use the protocol for populating the server’s data. ESPP Clients may include, for example, a back-office system, a federation 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. This is because the NAPTRs carry a large amount of redundant data when many telephone numbers resolve into a URI where the host part is the same Internet 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.
A Route is associated with zero or more SBE NAPTRs.

Route: id,enterpriseId, routeName, isLnService, sbeNAPTRs, extension

An EgressRoute is associated with zero or more Routes.

A Destination Group is associated with 1 or more Routes.

An RN is associated with only 1 Destination Group.

A TNRange is associated with only 1 Destination Group.

A Public Identity is associated with zero or 1 Private Identity
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 Destination Group 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 Destination Group, 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 Destination Group) associated with the Public Identities that are associated with that Private Identity.
Identity.

Route:
A set of related Signaling path Border Elements (SBEs). SBEs are generally ingress or egress points within, or at the edge of, a SIP service provider’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 Destination Group 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 a 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.

Destination Group:
A collection of zero or more Public Identities, Telephone Number ranges (TNRanges), and Routing Numbers (RNs) 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 RNs 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 RNs without having to make changes to each individual Public Identities, TNRange, or RN.

Routing Number (RN):
A routing number prefixed for a routing purpose (as defined in the ITU-T Recommendation e.164). The length of a routing number may vary from country to country. The RN is sometimes called a prefix as it can symbolize a global telephone number prefix. The RN can indeed represent the first digits of a number block handled by an SSP to which the session should be routed. In some countries, the RN value contains one or more digits not used as first digits in the national numbering plan to indicate a ported number (e.g. in
North American where the RN is used for Local Number Portability). In the rest of this document, we use the RN as a synonym of prefix. An RN is associated with a Destination Group.

**TNRange:**
An object that represents a range of telephone numbers. The TNRange object must be associated with a Destination Group 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 4.4.1.7 for more details.

**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 4.4.1.7 for more details.

**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.

### 4.2. 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. Figure 4 illustrates this high level technical structure.
The SOAP operations supported by ESPP (e.g., addRtes) are normatively defined in Section 8. 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 4.4.2 of this document defines how each operation utilizes this technical structure and the logical structure defined in the previous section.

4.3. 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.

4.3.1. Deployment Scenario and ESPP Provisioning

A typical deployment scenario for ESPP involves multiple SIP Service Providers wanting to exchange multimedia traffic including Voice over IP (VoIP) calls and Instant Messaging (IM) sessions. Each service provider 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. Note that the process by which a SIP service provider exchanges data such as an instant messaging address may be subject to end-user privacy and a subscriber-triggered authorization step may be required.

Various models can be chosen by SSPs 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 SSP 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 service provider’s addressing server could be provisioned with peer data: this is the purpose of the ESPP protocol. Both models can also co-exist: in some cases and depending on the SSP policy, the resolution of certain types of addresses may require an out-of-network query.

The remaining of this section assumes that SSPs have deployed ENUM-SIP addressing servers and prefer in-network queries. A deployment scenario is described in details covering both the provisioning activities using ESPP and the query resolutions to resolve target addresses using ENUM or SIP. We first introduce the actors of the scenario (SIP Service Providers 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.

4.3.1.1. Actors

4.3.1.1.1. Enterprises Description

Three SIP service Providers have agreed to exchange Voice over IP traffic and Instant Messages via their IP networks:
GlobalSSP:

GlobalSSP is a large SIP service provider with service footprints in Boston and Arizona and several SIP servers capable of processing VoIP traffic. The SIP servers are logically linked to several Signaling path Border Elements (SBEs) that hide the topology of the SIP service provider’s network while providing specific access points for other peers. GlobalSSP does also control its egress routes by choosing the SBE that egress calls to a particular peer.

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

MidwestSSP:

MidwestSSP operates a network similar to GlobalSSP with services in the Chicago area (subscriber base only located in one state). Two Call Management Servers and only one SBE are deployed.

NewYorkSSP:

NewYorkSSP 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.

SIP service providers 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 SIP service provider has bilateral relationships with a number of peers and use ESPP directly with each peer.

Each SIP service provider 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 GlobalSSP is connected to the Registrar from one of their Destination Group that distributes data to the Boston Destination Group.

Each SIP service provider is allocated an identifier or Enterprise ID:
After these three SSPs have successfully exchanged session establishment data (SED) for some time, a new SSP called NewSSP wants to exchange calls and IM with the other three. To facilitate the process of getting the data from all three SIP service providers in a limited amount of time, the NewSSP server will receive a complete copy of the data the SSPs wish to share with NewSSP using a bulk file.

### Table 2: Example NewSSP and Enterprise ID

<table>
<thead>
<tr>
<th>SSP</th>
<th>Enterprise ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewSSP</td>
<td>76546</td>
</tr>
</tbody>
</table>

In this scenario, each SIP Service Provider operates an ESPP Server that houses the ESPP Server’s SED and endpoint 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 endpoint objects are stored in the ESPP Server.

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

4.3.1.1.2. Enterprises: ESPP Provisioning Operations

As these partner relationships are established, each SSP 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 Destination Groups, 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 SSP. 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 SIP Service Providers, MidwestSSP, GlobalSSP, and NewYorkSSP.

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

Figure 5

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 NewSSP is added later. At that time, the central registry via its ESPP Client uses the addEntr operation to add the NewSSP’s enterprise ID to the ESPP Servers of the peers NewSSP wishes to exchange traffic with.

```xml
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope [snip] >
  <SOAP-ENV:Body>
    <addEntrRqst>
      <basicRqst>
        <eId>76546</eId>
      </basicRqst>
    </addEntrRqst>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Figure 6

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

4.3.1.2. Session Establishment Data

4.3.1.2.1. Description of Session Establishment Data

As part of establishing their traffic exchange policies, each SSP defines its points of session interconnect for their destination groups, 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 Destination Groups. SED is usually provisioned once for each SIP Service Provider with occasional subsequent updates as interconnect points are added or changed. It is provisioned independently from the provisioning of the elements contained in Destination Groups (TNs, TN Ranges, RNs, and IM IDs). This allows the rare process of provisioning SED to be distinctly separate from the continuous process of adding subscribers. The destination groups supported by each SIP Service Provider are illustrated in Figure 7 below.

```
+----------------+                 +----------------+
|    Chicago     |_________________|    Boston      |
|Dest. Group  (1)|                 |Dest. Group (3) |
+----------------+                +----------------+
                  |                      |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  |                        |
                  +----------------+.-'           '-.+----------------+
|   Flagstaff    |_________________|   New York     |
|Dest. Group (2) |                 |Dest. Group (4) |
+----------------+                 +----------------+
```

Sample Scenario and Destination Groups

Figure 7

The three initial SIP Service Providers offer voice and IM services. GlobalSSP has defined two Destination Groups and MidwestSSP and NewYorkSSP have each defined one Destination Group. Each Destination Group is reachable via a Route associated with a set of SBEs.
Figure 8 below further illustrates the details of the relationship between the Flagstaff and New York destination groups operated by the GlobalSSP and NewYorkSSP respectively. The examples in this section focus on the relationship between GlobalSSP’s Flagstaff destination group and the NewYorkSSP’s New York destination group.

Sample Scenario - Destination Groups and SBEs

Each Route illustrated above is further defined in the Route table (Table 3 below) to contain one or more SBEs that serve as the ingress points for that Route.
<table>
<thead>
<tr>
<th>Routes</th>
<th>SBE NAPTRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTFlag-NYC1</td>
<td>Order = 10, Pref=10, Svcs=E2U+SIP, Regx = &quot;!^.*$!sip:\<a href="mailto:1@sbe40-newyork.newyorkssp.example1.com">1@sbe40-newyork.newyorkssp.example1.com</a>!&quot;</td>
</tr>
<tr>
<td>Ingress</td>
<td>route to reach the New York Destination Group from Flagstaff.</td>
</tr>
<tr>
<td>RTFlag-NYC2</td>
<td>Order= 10, Pref=10, Svcs= E2U+IM, Regx = &quot;!^.*$!im:\<a href="mailto:1@sbe41-newyork.newyorkssp.example1.com">1@sbe41-newyork.newyorkssp.example1.com</a>!&quot;</td>
</tr>
<tr>
<td>Ingress</td>
<td>route to reach the New York Destination Group from Flagstaff.</td>
</tr>
<tr>
<td>RTNYC-Flag1</td>
<td>Order= 10, Pref=10, Svcs= E2U+SIP, Regx = &quot;!^.*$!sip:\<a href="mailto:1@sbe10-globalssp.example2.com">1@sbe10-globalssp.example2.com</a>!&quot;</td>
</tr>
<tr>
<td>Ingress</td>
<td>route to reach the Flagstaff Destination Group from New York.</td>
</tr>
<tr>
<td>RTNYC-Flag2</td>
<td>Order = 10, Pref=10, Svcs= E2U+IM, Regx = &quot;!^.*$!im:\<a href="mailto:1@sbe11-globalssp.example2.com">1@sbe11-globalssp.example2.com</a>!&quot;</td>
</tr>
<tr>
<td>Ingress</td>
<td>route to reach the Flagstaff Destination Group from New York.</td>
</tr>
</tbody>
</table>

Table 3: Examples of Routes to SBEs Associations
The Routes defined above can then be associated with the destination groups that they support. This allocation is laid out in the Table 4 below.

<table>
<thead>
<tr>
<th>Destination Groups</th>
<th>GlobalSSP Flagstaff</th>
<th>GlobalSSP Boston</th>
<th>MidwestSSP Chicago</th>
<th>NewYorkSSP New York</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalSSP Flagstaff</td>
<td>RTFlag-Bos</td>
<td>RTFlag-Chi</td>
<td>RTFlag-NYC1</td>
<td></td>
</tr>
<tr>
<td>GlobalSSP Boston</td>
<td></td>
<td>RTFlag-Chi</td>
<td></td>
<td>RTBos-NYC</td>
</tr>
<tr>
<td>MidwestSSP Chicago</td>
<td>RTChi-Flag</td>
<td>RTChic-Bos</td>
<td></td>
<td>RTChi-NYC</td>
</tr>
<tr>
<td>NewYorkSSP New York</td>
<td>RTNYC-Flag1</td>
<td>RTNYC-Bos</td>
<td>RTNYC-Chi</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Example of Destination Groups to Routes Associations

**4.3.1.2.2. ESPP Provisioning of Session Establishment Data**

The Route, SBE, and Destination Group data is provisioned in the appropriate ESPP Server of each participating SIP Service Provider using the addRtes, addDestGroups, 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 Destination Groups for the example above.

Using the batchUpdate operation, the ESPP Client provisions the Global SSPs Flagstaff SBE NAPTRs, Routes, and Destination Groups in the NewYorkSSP’s ESPP Server.

```xml
<?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>!^.*$!sip:\1@sbe10-globalssp.example2.com!</regx>
      </naptrAdd>
      <naptrAdd>
        <oid>7845601000012345611</oid>
        <eid>76543</eid>
        <order>10</order>
        <pref>10</pref>
        <flags>u</flags>
        <svcs>E2U+IM</svcs>
        <regx>!^.*$!im:\1@sbe11-globalssp.example2.com!</regx>
      </naptrAdd>
      <rteAdd>
        <oid>7845601000012345620</oid>
        <eid>76543</eid>
        <rteName>RTNYC-Flag1</rteName>
        <sbeNAPTRId>7845601000012345610</sbeNAPTRId>
        <isInSvc>true</isInSvc>
      </rteAdd>
      <rteAdd>
        <oid>7845601000012345621</oid>
        <eid>76543</eid>
        <rteName>RTNYC-Flag2</rteName>
        <sbeNAPTRId>7845601000012345611</sbeNAPTRId>
        <isInSvc>true</isInSvc>
      </rteAdd>
      <destGroupAdd>
        <oid>7845601000012345630</oid>
        <eid>76543</eid>
        <destGroupName>FlagstaffDG</destGroupName>
        <rteId>7845601000012345620</rteId>
        <rteId>7845601000012345621</rteId>
      </destGroupAdd>
    </op>
  </batchUpdate>
</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 NewYorkSSP’s NAPTRs, Routes, and Destination Groups in the GlobalSSP’s ESPP Server.

Provision the New York SSP’s New York NAPTRs in the Global SSPs 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>!^.*$!sip:\1@sbe40-newyork.newyorkssp.example1.com!</regx>
   </naptr>
   <naptr>
    <oid>7845601000012345613</oid>
    <eid>76544</eid>
    <order>10</order>
    <pref>10</pref>
    <flags>u</flags>
    <svcs>E2U+IM</svcs>
    <regx>!^.*$!im:\1@sbe41-newyork.newyorkssp.example1.com!</regx>
   </naptr>
  </addNAPTRsRqst>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

The ESPP Client provisions the New York SSP’s New York Routes in the Global SSP’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>
        <sbeNAPTRId>7845601000012345612</sbeNAPTRId>
        <isInSvc>true</isInSvc>
      </rte>
      <rte>
        <oid>7845601000012345623</oid>
        <eid>76544</eid>
        <rteName>RTFlag-NYC2</rteName>
        <sbeNAPTRId>7845601000012345613</sbeNAPTRId>
        <isInSvc>true</isInSvc>
      </rte>
    </addRtesRqst>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Figure 11

The ESPP Client provisions the New York SSP’s New York Destination Groups in the Global SSP’s ESPP Server.

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [snip] >
  <SOAP-ENV:Body>
    <addDestGroupsRqst>
      <basicRqst> [snip] </basicRqst>
      <destGroup>
        <oid>7845601000012345631</oid>
        <eid>76544</eid>
        <destGroupName>NewYorkDG</destGroupName>
        <rteId>7845601000012345622</rteId>
        <rteId>7845601000012345623</rteId>
      </destGroup>
    </addDestGroupsRqst>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Figure 12
4.3.1.3. Public Identities, TN Ranges, and RNs

4.3.1.3.1. Description of Identities, TN Ranges, and RNs

With the enterprises and SED data in place, SSPs have established the Destination Groups they wish to exchange traffic to. Public identities such as telephone numbers (TN), ranges of TNs, RNs, IM identifiers can now be provisioned into Destination Groups. This scenario illustrates the provisioning of TNs, TN Ranges and RNs 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 SSPs. 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.

<table>
<thead>
<tr>
<th>Subscriber</th>
<th>Number</th>
<th>Destination Group</th>
<th>SSP</th>
<th>IM Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pat</td>
<td>928-774-555</td>
<td>FlagstaffDG</td>
<td>GlobalSSP</td>
<td><a href="mailto:imPat@globalssp.example2.com">imPat@globalssp.example2.com</a></td>
</tr>
<tr>
<td>Ashley</td>
<td>312-746-555</td>
<td>ChicagoDG</td>
<td>MidwestSSP</td>
<td><a href="mailto:imAshley@midwestssp.example3.com">imAshley@midwestssp.example3.com</a></td>
</tr>
<tr>
<td>Vince</td>
<td>718-330-555</td>
<td>NewYorkDG</td>
<td>NewYorkSSP</td>
<td><a href="mailto:imVince@newyorkssp.example1.com">imVince@newyorkssp.example1.com</a></td>
</tr>
<tr>
<td>Carl</td>
<td>617-414-555</td>
<td>BostonDG</td>
<td>GlobalSSP</td>
<td><a href="mailto:imCarl@globalssp.example2.com">imCarl@globalssp.example2.com</a></td>
</tr>
</tbody>
</table>

Table 5

These four subscribers have authorized their friends to see their public identities. The SSPs have also defined TN Ranges associated with the new VoIP services as shown below.
To support number portability in some countries, Routing Numbers (RNs) have identified each of the CMSes within the network. Although there may be an RN associated with each CMS, we show only the relevant sub-set applicable to this example:

<table>
<thead>
<tr>
<th>Proxy</th>
<th>Destination Group</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy-center</td>
<td>FlagstaffDG</td>
<td>928-774-1000</td>
</tr>
<tr>
<td>Proxy-lakestreet</td>
<td>ChicagoDG</td>
<td>312-746-1000</td>
</tr>
<tr>
<td>Proxy-bronx</td>
<td>NewYorkDG</td>
<td>718-330-1000</td>
</tr>
<tr>
<td>Proxy-fenway</td>
<td>BostonDG</td>
<td>617-414-1000</td>
</tr>
</tbody>
</table>

Table 7

4.3.1.3.2. ESPP Provisioning of TNs, RNs, and identities

The Public Identity, TN Range and RN data can then be provisioned in the appropriate ESPP Server of each SSP using the addPvtIds, addPubIds, addTNRs, and addRNs 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 GlobalSSP elements in the Flagstaff destination group. This command is sent to the NewYorkSSP’s ESPP Server and to any other applicable
peer servers.

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

Figure 13
Using the batchUpdate operation, the ESPP Client provisions four NewYorkSSP elements in the New York destination group. This command is sent to the GlobalSSP’s ESPP Server (and to any other applicable servers).
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <batchUpdateRqst>
      <basicRqst>
      </basicRqst>
      <batchUpdate>
        <op>
          <pviAdd>
            <oid>7845601000012345651</oid>
            <eid>76544</eid>
            <pvtId>vincesPrivateIdentifier</pvtId>
          </pviAdd>
          <piAdd>
            <oid>7845601000012345644</oid>
            <eid>76544</eid>
            <pubId>7183305555</pubId>
            <dgId>7845601000012345631</dgId>
            <pvtId>7845601000012345651</pvtId>
          </piAdd>
          <piAdd>
            <oid>7845601000012345645</oid>
            <eid>76544</eid>
            <pubId>imVince@newyorkssp.example1.com</pubId>
            <svcs>E2U+IM</svcs>
            <dgId>7845601000012345631</dgId>
            <pvtId>7845601000012345651</pvtId>
          </piAdd>
          <tnRAdd>
            <oid>7845601000012345646</oid>
            <eid>76544</eid>
            <tnRStrt>7183305250</tnRStrt>
            <tnREnd>7183305590</tnREnd>
            <dgId>7845601000012345631</dgId>
          </tnRAdd>
          <rnAdd>
            <oid>7845601000012345647</oid>
            <eid>76544</eid>
            <rn>7183301000</rn>
            <dgId>7845601000012345631</dgId>
          </rnAdd>
        </op>
      </batchUpdate>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

Figure 14
4.3.2. Deployment Scenario and Resolution

4.3.2.1. Initiating an Outgoing Call

Initiating a call from Pat to Vince, the CMS hosting Pat launches an ENUM query to the GlobalSSP 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 Destination Group. This Destination Group 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 NewYorkSSP 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 GlobalSSP ESPP Server, which results in the following response:

   $ORIGIN 5.5.5.5.0.3.3.8.1.7.1.ssp-example.
   NAPTR 10 100 "u" "E2U+SIP" "!^.*$
   !sip:\1@sbe40-newyorkssp.newyorkssp.example1.com!".
   NAPTR 10 101 "u" "E2U+IM" "!^.*$
   !im:imvince@sbe41-newyorkssp.newyorkssp.example1.com!".

   Figure 15

2. Assuming that the CMS is interested in the SIP service, as it is initiating a telephone call, the CMS now executes a NAPTR query to the DNS server for "sbe40-newyorkssp.example1.com" resulting in:

   sbe40-newyorkssp.example1.com.
   order pref flgs service regx    replacement
   IN NAPTR 100  50   "s"   "SIP+D2U" ""   _sip._udp.sbe40-newyorkssp
   .newyorkssp.example1.com

   Figure 16

3. The CMS now sends a DNS SRV query for _sip._udp.sbe40-newyorkssp.
   newyorkssp.example1.com which results in:
$ORIGIN _sip._udp.sbe40-newyorkssp.newyorkssp.example1.com.
(Some content not included for clarity purposes.)
IN SRV 0 1 5060 sbe40-newyorkssp.example1.com

;; ANSWER SECTION
sbe40-newyorkssp.newyorkssp.example1.com A 192.0.2.17
sbe40-newyorkssp.newyorkssp.example1.com A 192.0.2.18

Figure 17

4. Now 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@sbe40-newyorkssp.newyorkssp.example1.com
   SIP/2.0
(Other SIP INVITE content not included for clarity purposes.)

Figure 18

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

4.3.2.2. Receiving an Incoming Call

Taking the call scenario described above, once the call request arrives in NewYorkSSP’s network the SBE within NewYorkSSP 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 NewYorkSSP ESPP Server has been provisioned with intra-domain routes so that the SBE can locate the proper CMS. The NewYorkSSP ESPP Server contains the New York Destination Group that is associated with a intra-domain Route that is not shared with the other SSPs. This allows the internal routing topology of an SSP to remain private to the SSP. 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
Destination Group: New York
Route: RTNYC-Private
NAPTR 10 101 "u" "E2U+SIP" "!^.*$
   !sip:\1@cms-bronx.newyorkssp.example1.com!" .

Figure 19
4.3.2.2.1. Query using the Routing Number

If the number has been ported, there are a number of cases where the query may also contain a Routing Number. The 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 NewYorkSSP’s network the SBE within NewYorkSSP queries its own ESPP Server and finds a match for the RN or prefix (718-330-1000) in the New York Destination Group. The rest of the routing lookup is similar to what is described for the telephone number above.

RN:     718-330-1000
Destination Group: New York
Route: RTNYC-Private
NAPTR 10 101 "u" "E2U+SIP" "!^.*$
    !sip:\\1@cms-bronx.newyorkssp.example1.com!".

Figure 20

4.3.2.2.2. Growth of the SIP Service Provider Group

When NewSSP 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 NewSSP to view and performs a secure file transfer to the NewSSP addressing server. Following the download, the NewSSP 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.

4.3.3. Generic Examples

Additional examples are available in Appendix I of [CableLabs-ESPP].

4.4. 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 8, Formal API Definition, in this document. Note that if any specification conflicts exist between this section and Section 8, the normative definitions contained in Section 8 take precedence.
The protocol utilizes SOAP 1.1 [SOAP], WSDL1.1 [WSDL], and XML 1.0 [XML].

4.4.1. General Protocol Concepts

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

4.4.1.1. 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.

```xml
<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>

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

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

4.4.1.2. 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>

4.4.1.3. Authentication

For each ESPP connection, the ESPP Client and ESPP Server MUST use HTTP Digest based authentication as defined in [RFC2617]. 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.

4.4.1.4. 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 SIP Service Provider before protocol processing or bootstrap file generation begins.

When generating an API-based update request or a file based batchFileUpdateRqst 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 successfully generating a batchFileUpdateRqst structure or 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 client’s 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 SIP Service Provider, 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), 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.

4.4.1.5. 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.

4.4.1.6. 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.

4.4.1.7. 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 defined 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 data validation appendix of this document where the maximum allowable value of an unsigned long is documented.

The 8 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 1 to 6 digit organization ID and a 2 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 2 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 2 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, 12 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 00012345678 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 performs the mechanics of provisioning an ESPP object within an
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.

The ESPP Client and ESPP Server MUST implement the oId attribute as an unsigned 64-bit value.

4.4.1.8. 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 BasicRqstType.

4.4.1.9. 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
4.4.2. Protocol Operation Descriptions

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

4.4.2.1. 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 Destination Groups 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 Destination Group 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:
Inputs to the operation are wrapped in the addRtesRqst object, which is declared as follows:

```xml
<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 4.4.1, 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="sbeNaptrId" 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 comprised 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 addDestGroups 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.

- **SBENaptrId**: Zero or more NAPTR IDs.

- **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 4.4.1.9.

Table 8 defines the result codes and messages that the addRtes operation SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2103</td>
<td>Transaction ID out of sequence: [transaction ID received]:[ transaction ID expected].</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
<tr>
<td>2106</td>
<td>Object status or ownership does not allow for request: [request name]:[ attributeName]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 8: addRtes Operation Result Codes and Messages

4.4.2.2. Operation Name: addDestGroups

As described in the introductory sections, a Destination Group represents a collection of public identities that are linked to a set of Routes. Routes are linked to Destination Groups to establish the link between a set of public identities and their session-level peering points. The addDestGroups operation creates or overwrites one or more Destination Groups in the addressing server. If a destination group with the given destination group ID does not exist, then the ESPP Server MUST create the destination group. If a destination group with the given identity does exist, then the ESPP Server MUST replace the current properties of that destination group with the properties passed into the addDestGroups operation. If a destination group with the given ID does exist, but was created by a client other than the one calling the addDestGroups 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:
Inputs to the operation are wrapped in the addDestGroupsRqst object, which is declared as follows:

```
<element name="addDestGroupsRqst">
  <complexType>
    <sequence>
      <element name="basicRqst" type="est:BasicRqstType"/>
      <element name="destGroup" type="est:DestGroupType" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
```

As with all update operations, the basicRqst data element is a required input. Refer to Section 4.4.1, General Protocol Concepts, for a detailed description of this data element. Also required is one or more DestGroupType objects. Any limitation on the maximum number of destination groups that may be passed into a call to the addDestGroups operation is a policy decision and is not limited by the protocol.

The DestGroup object structure is declared as follows:

```
<complexType name="DestGroupType">
  <sequence>
    <element name="oid" type="est:OIdType"/>
    <element name="eid" type="est:EIdType"/>
    <element name="destGroupName" 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 DestGroupType 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 Destination Group, such as delDestGroups, getDestGroups, and addPubIds operations to identify a given destination group.

- **Eid**: Exactly one enterprise ID that uniquely identifies the operational entity within the call routing community that is responsible for, or "owns" this object.

- **destGroupName**: Exactly one human readable name of the destination group.

- **RteID**: The identity of zero or more routes that the Public Identities, RNs, and TN Ranges in this destination group can be routed through.

- **Ext**: Point of extensibility described in Section 4.4.1.9.

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

Table 9 defines the result codes and messages that the addDestGroups operation SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2103</td>
<td>Transaction ID out of sequence: [transaction ID received]:[ transaction ID expected].</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
<tr>
<td>2105</td>
<td>Object does not exist: [attribute name]:[objectType-objectId].</td>
</tr>
<tr>
<td>2106</td>
<td>Object status or ownership does not allow for request: [request name]:[ attributeName]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 9: addDestGroups Operation Result Codes and Messages

4.4.2.3. 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 SBIs and NAPTRs (indirectly via a Destination Group ID), and, alternatively, to directly associated NAPTRs. The ESPP protocol allows a Public Identity to have a relationship to a Destination Group 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:

```xml
<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:

```xml
<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 4.4.1, 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"/>
    <element name="dgId" type="est:OIdType" minOccurs="0" maxOccurs="unbounded"/>
    <element name="naptrId" type="est:OIdType" minOccurs="0" maxOccurs="unbounded"/>
    <element name="pvtId" type="est:OIdType" minOccurs="0" maxOccurs="unbounded"/>
    <element name="ext" type="est:ExtAnyType" minOccurs="0" maxOccurs="unbounded"/>
  </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**: Exactly 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.
- **DgId**: Zero or one object ID that identifies the Destination Group this public identity object instance resides within. The Destination Group’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 Destination Group ID that does not exist within the appropriate context of the addressing server.
- **NAPTR**: 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.

Ext: Point of extensibility described in Section 4.4.1.9.

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

Table 10 defines the result codes and messages that the addPubIds operation SHOULD return.

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

Table 10: addPubIds Operation Result Codes and Messages
4.4.2.4. 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:

```xml
<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:

```xml
<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 4.4.1, 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 PvtIdType object structure is declared as follows:

```xml
<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 4.4.1.9.

Table 11 defines the result codes and messages that the addPvtIds operation SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2103</td>
<td>Transaction ID out of sequence: [transaction ID received]:[ transaction ID expected].</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
<tr>
<td>2105</td>
<td>Object does not exist: [attribute name]:[objectType-objectId].</td>
</tr>
<tr>
<td>2106</td>
<td>Object status or ownership does not allow for request: [request name]:[ attributeName]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 11: addPvtIds Operation Result Codes and Messages

4.4.2.5. Operation Name: addRNs

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

The addRNs operation creates or overwrites one or more RN objects in the addressing server. If an RN object with the same object ID does not exist within the context of the addressing server, then the ESPP Server MUST create the RN object. If an RN 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 RN with the properties passed into the addRNs operation. If an RN with the given ID does exist, but was created by a client other than the one calling the addRN 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:

```xml
<wsdl:operation name="addRNs">
  <wsdl:input message="addRNsRqstMsg"/>
  <wsdl:output message="cmnRspnsMsg"/>
</wsdl:operation>

<wsdl:message name="addRNsRqstMsg">
  <wsdl:part name="rqst" element="est:addRNsRqst"/>
</wsdl:message>

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

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

```xml
<element name="addRNsRqst">
  <complexType>
    <sequence>
      <element name="basicRqst" type="est:BasicRqstType"/>
      <element name="rn" type="est:RNType" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
```

As with all update operations, the `basicRqst` data element is a required input. Refer to the Section 4.4.1 General Protocol Concepts for a detailed description of this data element. Also required is one or more `RNType` objects. Any limitation on the maximum number of RNs that may be passed into a call to the `addRNs` operation is a policy decision and is not limited by the protocol.

The RN object structure is declared as follows:
<complexType name="RNType">
  <sequence>
    <element name="oid" type="est:OIdType"/>
    <element name="eid" type="est:EIdType"/>
    <element name="rn" type="string"/>
    <element name="dgId" type="est:OIdType"/>
    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
  </sequence>
</complexType>

RNType 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 delRNs, getRNs operations to identify a given RN.

- **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.

- **dgId**: Exactly one object ID that identifies the Destination Group in which this RN resides. The Destination Group’s relationship to one or more Routes can then define the RNs peering points. The ESPP Server MUST reject, with the appropriate error response code, an attempt to create an RN object containing a Destination Group ID that does not exist within the appropriate context of the addressing server.

- **Ext**: Point of extensibility described in Section 4.4.1.9 of this document.

Table 12 defines the result codes and messages that the addRNs operation SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2103</td>
<td>Transaction ID out of sequence: [transaction ID received]:[ transaction ID expected].</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
<tr>
<td>2105</td>
<td>Object does not exist: [attribute name]:[objectType-objectId].</td>
</tr>
<tr>
<td>2106</td>
<td>Object status or ownership does not allow for request: [request name]:[ attributeName]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 12: addRNs Operation Result Codes and Messages

4.4.2.6. Operation Name: addTNRs

As described in the introductory sections, a TNRTtype 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 Destination Group 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:

```xml
<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:

```xml
<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 4.4.1, 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:
TNRTyp 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**: An optional 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.
- **dgId**: Exactly one object ID that identifies the Destination Group this TN Range object instance resides within. The Destination Group’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 Destination Group ID that does not exist within the appropriate context of the addressing server.
- **Ext**: Point of extensibility described in Section 4.4.1.9 of this document.

Table 13 defines the result codes and messages that the addTNRs operation SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2103</td>
<td>Transaction ID out of sequence: [transaction ID received]:[ transaction ID expected].</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
<tr>
<td>2105</td>
<td>Object does not exist: [attribute name]:[objectType-objectId].</td>
</tr>
<tr>
<td>2106</td>
<td>Object status or ownership does not allow for request: [request name]:[ attributeName]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 13: addTNRs Operation Result Codes and Messages

4.4.2.7. 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 4.4.1 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.
- **Regex**: 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 4.4.1.9 of this document.

Table 14 defines the result codes and messages that the addNAPTRs operation SHOULD return.
Table 14: addNAPTRs Operation Result Codes and Messages

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

4.4.2.8. 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 4.4.1 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:
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 "own" this object.

- **Order**: Relative order value in an ENUM NAPTR.

- **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 that is applied to the Regx field of a NAPTR.

- **Ext**: Point of extensibility described in Section 4.4.1.9 of this document.
Table 15 defines the result codes and messages that the addEgrRtes operation SHOULD return.

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

Table 15: addEgrRtes Operation Result Codes and Messages

4.4.2.9. Delete Operations

This sub-section defines the delete operations (i.e. delRtes, delDestGroups, delPubIds, delPvtIds, delTNRs, delRNs, delNAPTRs, delEgrRtes). 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 the addressing server. If an object of the indicated type with the given ID exists, then the operation must delete that object. If such an object does not exist, then the ESPP Server MUST return the appropriate error response code. If an object with the given ID 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 the appropriate error code, 2106. 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:

```xml
<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:

```xml
<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 4.4.1 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:

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 Destination Group 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 RNs and TNRanges within a Destination Group that is being deleted, even if those RNs or TNRanges were created by another ESPP client ID. The ESPP Server MUST delete all Public Identity instances within a Destination Group, but only if a Public Identity instance is not directly associated with one or more NAPTR objects. As with RNs and TNRanges, Destination Groups 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 Destination Groups, Egress Routes and Routes, Routes and NAPTRs, Public Identities and Private Identities, and Public Identities and NAPTRs.

Table 16 defines the result codes and messages that these delete operations SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>1001</td>
<td>Request Succeeded. Deleted object(s) did not exist.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2103</td>
<td>Transaction ID out of sequence: [transaction ID received]:[transaction ID expected].</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
<tr>
<td>2105</td>
<td>Object does not exist: [attribute name]:[objectType-objectId].</td>
</tr>
<tr>
<td>2106</td>
<td>Object status or ownership does not allow for request: [request name]:[attributeName]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 16: delRtes, delDestGroups, delPubIds, delPvtIds, delTNRs, delRNs, delNAPTRs, delEgrRtes Operation Result Codes and Messages

4.4.2.10. Get Operations

This sub-section defines the get, or query, operations (i.e. getRtes, getDestGroups, getPubIds, getPvtIds, getTNRs, getRNs, getNAPTRs, getEgrRtes). 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:
Inputs to the operation are wrapped in the getRtesRqst object, which is declared as follows:

```xml
<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>
      <!-- ... -->
    </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.

Table 17 defines the result codes and messages that these get operations SHOULD return.
### Table 17: getRtes, getDestGroups, getPubIds, getPvtIds, getTNRs, getRNs, getNAPTRs, getEgrRtes Operation Result Codes and Messages

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

#### 4.4.2.11. 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:

```xml
<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:

```xml
<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 Section 4.4.1, for a detailed description of this data element. Also required is one enterprise ID.

Table 18 defines the result codes and messages that the addEntr operation SHOULD return.

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

Table 18: addEntr Operation Result Codes and Messages
4.4.2.12. 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:

```xml
<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:

```xml
<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 Section 4.4.1, for a detailed description of this data element. Also required is one "new" enterprise ID and one "old" enterprise ID.

Table 19 defines the result codes and messages that the modEntr operation SHOULD return.

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

Table 19: modEntr Operation Result Codes and Messages

4.4.2.13. 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:

```xml
<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:

```xml
<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 Section 4.4.1, for a detailed description of this data element. Also required is one enterprise ID.

Table 20 defines the result codes and messages that the delEntr operation SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>1001</td>
<td>Request Succeeded. Deleted object(s) did not exist.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2002</td>
<td>Request too large.</td>
</tr>
<tr>
<td>2003</td>
<td>Version not supported.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2103</td>
<td>Transaction ID out of sequence: [transaction ID received]:[ transaction ID expected].</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 20: delEntr Operation Result Codes and Messages

4.4.2.14. Operation Name: batchUpdate

The batchUpdate operation is a single operation, with a single transaction ID that may contain one or more of update actions. This approach can significantly speed 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="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"/>
    </sequence>
</complexType>
<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:DestGroupType"
    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="rnDel" type="est:OIdType"
    minOccurs="0" maxOccurs="unbounded"/>
<element name="rnAdd" type="est:RNType"
    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>
</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 Section 4.4.1, 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.

Table 21 defines the result codes and messages that the batchUpdate operation SHOULD return.
Table 21: batchUpdate Operation Result Codes and Messages

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

4.4.2.15. Operation Name: getSvcMenu

The getSvcMenu operation returns 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 or URNs, one for each supported operation. This feature can be used by ESPP Clients to dynamically discover which operations and versions are supported by the ESPP Servers.

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

- **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.

- **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:example.com:TBD-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.

Table 22 defines the result codes and messages that the getSvcMenu operation SHOULD return.
<table>
<thead>
<tr>
<th>Result Code</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Request Succeeded.</td>
</tr>
<tr>
<td>2001</td>
<td>Request syntax invalid.</td>
</tr>
<tr>
<td>2301</td>
<td>System temporarily unavailable.</td>
</tr>
<tr>
<td>2104</td>
<td>Attribute value invalid: [attribute name]:[attribute value]:[objectType-objectId].</td>
</tr>
</tbody>
</table>

Table 22: getSvcMenu Operation Result Codes and Messages
5. File-based ESPP 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.

5.1. 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.

5.2. File Structure

An ESPP compliant distribution file MUST be comprised of exactly one batchUpdateFileRqst element which contains one BatchUpdateFilePropsType 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.
The BatchUpdateFilePropsType 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 BatchUpdateFilePropsType object in each file.

- 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.
- 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.

The ESPP Server SHOULD log error codes that correspond to the codes used in the real time protocol as defined in Section 4.4.1.8.
6. 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 [RFC2821]:

- 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 defined by the first two digits.

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

Table 23: Response Codes Numbering Scheme and Messages

Some response messages are "parameterized" with one or more of the following parameters: "attribute name", "attribute value", "objectType-objectId", and "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 SHOULD be used whenever an element value does not adhere to data validation rules. It MUST be used for circumstances where an optional data element was expected but not received.
7. IANA Considerations

Should there be interest in working on a protocol like ESPP in IETF, a formal IANA consideration section should be drafted with proper registrations for the protocol namespace(s), versions, etc.
8. Formal API Definition

Note that the formal API definition is currently provided as defined in the PacketCable specifications for the XML namespace and protocol versioning conventions.

8.1. WSDL Specification

```xml
<wSDL:definitions xmlns:esp="urn:example.com:TBD-api-1.0"
xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/"
xmlns:est="urn:example.com:TBD-types-1.0"
targetNamespace="urn:example.com:TBD-api-1.0">
  <wSDL:types>
    <xsd:schema targetNamespace="urn:example.com:TBD-api-1.0"
xmlns:esp="urn:example.com:TBD-api-1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/"
xmlns:est="urn:example.com:TBD-types-1.0"
schemaLocation="ietfesppptypes.xsd"/>
  </wSDL:types>
  <wSDL:message name="addRtesRqstMsg">
    <wSDL:part name="rqst" element="est:addRtesRqst"/>
  </wSDL:message>
  <wSDL:message name="addDestGroupsRqstMsg">
    <wSDL:part name="rqst" element="est:addDestGroupsRqst"/>
  </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="addRNsRqstMsg">
    <wSDL:part name="rqst" element="est:addRNsRqst"/>
  </wSDL:message>
  <wSDL:message name="addNAPTRsRqstMsg">
    <wSDL:part name="rqst" element="est:addNAPTRsRqst"/>
  </wSDL:message>
  <wSDL:message name="addEgrRtesRqstMsg">
    <wSDL:part name="rqst" element="est:addEgrRtesRqst"/>
  </wSDL:message>
</wSDL:definitions>
```
</wsdl:message>
<wsdl:message name="addEntrRqstMsg">
  <wsdl:part name="rqst" element="est:addEntrRqst"/>
</wsdl:message>
</wsdl:message>
</wsdl:message>
<wsdl:message name="batchUpdateRqstMsg">
  <wsdl:part name="rqst" element="est:batchUpdateRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getRtesRqstMsg">
  <wsdl:part name="rqst" element="est:getRtesRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getDestGroupsRqstMsg">
  <wsdl:part name="rqst" element="est:getDestGroupsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getPubIdsRqstMsg">
  <wsdl:part name="rqst" element="est:getPubIdsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getTNRsRqstMsg">
  <wsdl:part name="rqst" element="est:getTNRsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getPvtIdsRqstMsg">
  <wsdl:part name="rqst" element="est:getPvtIdsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getRNsRqstMsg">
  <wsdl:part name="rqst" element="est:getRNsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getNAPTRsRqstMsg">
  <wsdl:part name="rqst" element="est:getNAPTRsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getEgrRtesRqstMsg">
  <wsdl:part name="rqst" element="est:getEgrRtesRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="getSvcMenuRqstMsg">
  <wsdl:part name="rqst" element="est:getSvcMenuRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="delRtesRqstMsg">
  <wsdl:part name="rqst" element="est:delRtesRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="delDestGroupsRqstMsg">
  <wsdl:part name="rqst" element="est:delDestGroupsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="delPubIdsRqstMsg">
  <wsdl:part name="rqst" element="est:delPubIdsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="delTNRsRqstMsg">
  <wsdl:part name="rqst" element="est:delTNRsRqst"/>
</wsdl:message>
</wsdl:message>
<wsdl:message name="delPvtIdsRqstMsg">
  <wsdl:part name="rqst" element="est:delPvtIdsRqst"/>
</wsdl:message>
<wsdl:message name="delRNsRqstMsg">
  <wsdl:part name="rqst" element="est:delRNsRqst"/>
</wsdl:message>
<wsdl:message name="delNAPTRsRqstMsg">
  <wsdl:part name="rqst" element="est:delNAPTRsRqst"/>
</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="getDestGroupsRspnsMsg">
  <wsdl:part name="rspns" element="est:getDestGroupsRspns"/>
</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="getRNsRspnsMsg">
  <wsdl:part name="rspns" element="est:getRNsRspns"/>
</wsdl:message>
<wsdl:message name="getNAPTRsRspnsMsg">
  <wsdl:part name="rspns" element="est:getNAPTRsRspns"/>
</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:operation>
</wsdl:portType>
<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="addDestGroups">
<wsdl:input message="esp:addDestGroupsRqstMsg"/>
<wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="delDestGroups">
<wsdl:input message="esp:delDestGroupsRqstMsg"/>
<wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="getDestGroups">
<wsdl:input message="esp:getDestGroupsRqstMsg"/>
<wsdl:output message="esp:getDestGroupsRspnsMsg"/>
</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: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="addRNs">
  <wsdl:input message="esp:addRNsRqstMsg"/>
  <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="delRNs">
  <wsdl:input message="esp:delRNsRqstMsg"/>
  <wsdl:output message="esp:cmnRspnsMsg"/>
</wsdl:operation>
<wsdl:operation name="getRNs">
  <wsdl:input message="esp:getRNsRqstMsg"/>
  <wsdl:output message="esp:getRNsRspnsMsg"/>
</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="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:binding>
<wsdl:output>
  <soap:body use="literal"/>
</wsdl:output>

<wsdl:operation name="addDestGroups">
  <soap:operation soapAction="addDestGroups" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="delDestGroups">
  <soap:operation soapAction="delDestGroups" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getDestGroups">
  <soap:operation soapAction="getDestGroups" 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:input>
  <soap:body use="literal"/>
</wsdl:input>

<wsdl:output>
  <soap:body use="literal"/>
</wsdl:output>

<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="addRNs">
  <soap:operation soapAction="addRNs" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wSDL:operation name="delRNs">
  <soap:operation soapAction="delRNs" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wSDL:operation name="getRNs">
  <soap:operation soapAction="getRNs" 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:input>
<wsdl:output>
  <soap:body use="literal"/>
</wsdl:output>

<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="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>
<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>

8.2. XSD Types Specification

<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:est="urn:example.com:TBD-types-1.0"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:example.com:TBD-types-1.0"
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="sbeNaptrId" type="est:OIdType" minOccurs="0" maxOccurs="unbounded"/>
<element name="isInSvc" type="boolean"/>
<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
</sequence>
</complexType>
<complexType name="DestGroupType">
<sequence>
<element name="oid" type="est:OIdType"/>
<element name="eid" type="est:EIdType"/>
<element name="destGroupName" 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"/>
<element name="dgId" type="est:OIdType" minOccurs="0"/>
</sequence>
</complexType>
<element name="naptrId" type="est:OIdType" minOccurs="0" maxOccurs="unbounded"/>
  <element name="pvtId" type="est:OIdType" 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="dgId" type="est:OIdType"/>
    <element name="ext" type="est:ExtAnyType" minOccurs="0"/>
  </sequence>
</complexType>
<complexType name="RNType">
  <sequence>
    <element name="oid" type="est:OIdType"/>
    <element name="eid" type="est:EIdType"/>
    <element name="rn" type="string"/>
    <element name="dgId" 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="EgrRteType">
  <sequence>
  </sequence>
</complexType>
<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="regexRewriteRule" type="string"/>
<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
</sequence>
</complexType>
<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:VerType"/>
<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:VerType"/>
<element name="ext" type="est:ExtAnyType" minOccurs="0"/>
</sequence>
</complexType>
<complexType name="BatchUpdateType">
<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="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:DestGroupType" 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="rnToDel" type="est:OIdType" minOccurs="0" maxOccurs="unbounded"/>
    <element name="rnAdd" type="est:RNType" 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="BatchUpdateFilePropsType">
<sequence>
    <element name="clientId" type="est:ClientIdType"/>
    <element name="serverId" type="string"/>
    <element name="isFullResync" type="boolean"/>
    <element name="creationTimestamp" type="dateTime"/>
    <element name="sequenceNumber" type="unsignedLong"/>
    <element name="isEndOfSequence" type="boolean"/>
</sequence>
</complexType>
<complexType name="SvcMenuType">
<sequence>
    <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>
<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>
<element name="basicRqst" type="est:BasicRqstType"/>
<element name="oid" type="est:OIdType" maxOccurs="unbounded"/>
</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="addDestGroupsRqst">
<complexType>
<sequence>
<element name="basicRqst" type="est:BasicRqstType"/>
<element name="destGroup" type="est:DestGroupType" maxOccurs="unbounded"/>
</sequence>
</complexType>
</element>
<element name="delDestGroupsRqst">
<complexType>
<sequence>
<element name="basicRqst" type="est:BasicRqstType"/>
<element name="oid" type="est:OIdType" maxOccurs="unbounded"/>
</sequence>
</complexType>
</element>
<element name="getDestGroupsRqst">
<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="oid" type="est:OIdType" minOccurs="0" maxOccurs="unbounded"/>
<element name="eid" type="est:EIdType" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
</complexType>
</element>
<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"
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"
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" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<element name="addRNsRqst">
  <complexType>
    <sequence>
      <element name="basicRqst" type="est:BasicRqstType"/>
      <element name="rn" type="est:RNType" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

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

<element name="getRNsRqst">
  <complexType>
    <sequence>
      <element name="basicRqst" type="est:BasicQueryType"/>
      <element name="oid" type="est:OIdType" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
<complexType>
  <sequence>
    <element name="basicRqst" type="est:BasicQueryType"/>
    <element name="oid" type="est:OIdType" 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" 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>
<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="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="basicRqst" type="est:BasicRqstType"/>
      <element name="batchUpdateFileProps" type="est:BatchUpdateFilePropsType"/>
      <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="getDestGroupsRspns">
  <complexType>
    <sequence>
      <element name="destGroup" type="est:DestGroupType" 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="getRNsRspns">
  <complexType>
    <sequence>
      <element name="rn" type="est:NTNType" 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="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>
    </sequence>
  </complexType>
</element>
<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>
9. Security Considerations

This document specifies a new provisioning protocol.

Provisioning data and other configuration information in scope of ESPP include public identities, telephone number ranges, RNs, signaling path border elements, NAPTRs and Egress Routes. This information is sensitive and its transmission in the clear and without integrity checking leaves servers exposed to eavesdropping attacks. Compliant implementations must therefore satisfy the requirements defined in Section 4.4.1.3.

If the object values for the Public Identity, Route, Destination Group, RN, SBE, MAPTRs are set maliciously, it may result in sessions being misrouted and service disruption. It may also result in an over-allocation of signaling resources in an attempt to create denial of service attacks (all routes of all destination groups directed towards a particular SBE).

Additional security considerations must also apply given the provisioning data exchanged within and between Trust Domains as defined in [RFC3325].

An initial set of security requirements that must be met by compliant ESPP Clients and ESPP Servers is defined in [I-D.espp-requirements].
10. Acknowledgments

This document is based on the work of participants in the CableLabs PacketCable ENUM Server vendor focus team.

The authors wish to thank the following participants for their contributions and comments:
11. References

11.1. Normative References


11.2. Informative References


Authors’ Addresses

Kenneth Cartwright
VeriSign
21355 Ridgetop Circle
Dulles, VA  20166
USA

Email: kcartwright@verisign.com

Stephen M. Dimig
Tekelec
5200 Paramount Parkway
Morrisville, NC  27560
USA

Email: stephen.dimig@tekelec.com

Mark Teodoro
NeuStar
46000 Center Oak Plaza
Sterling, VA  20166
USA

Email: mark.teodoro@neustar.com

Jean-Francois Mule
CableLabs
858 Coal Creek Circle
Louisville, CO  80027
USA

Email: jfm@cablelabs.com