DRINKS
Internet-Draft
Intended status: Standards Track
Expires: May 18, 2012

K. Cartwright
V. Bhatia
TNS

November 15, 2011

SPPP Over SOAP and HTTP
draft-ietf-drinks-sppp-over-soap-07

Abstract

The Session Peering Provisioning Protocol (SPPP) is an XML protocol that exists to enable the provisioning of session establishment data into Session Data Registries or SIP Service Provider data stores. Sending XML data structures over Simple Object Access Protocol (SOAP) and HTTP(s) is a widely used, de-facto standard for messaging between elements of provisioning systems. Therefore the combination of SOAP and HTTP(s) as a transport for SPPP is a natural fit. The obvious benefits include leveraging existing industry expertise, leveraging existing standards, and a higher probability that existing provisioning systems can be more easily integrated with this protocol. This document describes the specification for transporting SPPP XML structures over SOAP and HTTP(s).

Table of Contents

1. Introduction

SPPP, defined in [I-D.draft-ietf-drinks-spprov], is best supported by a transport and messaging infrastructure that is connection oriented, request-response oriented, easily secured, supports propagation through firewalls in a standard fashion, and that is easily integrated into back-office systems. This is due to the fact that the client side of SPPP is likely to be integrated with organizations' operational support systems that facilitate transactional provisioning of user addresses and their associated session establishment data. While the server side of SPPP is likely to reside in a separate organization's network, resulting the SPPP provisioning transactions traversing the Internet as they are propagated from the SPPP client to the SPPP server. Given the current state of industry practice and technologies, SOAP and HTTP(s) are well suited for this type of environment. This document describes the specification for transporting SPPP XML structures over SOAP and HTTP(s).

The specification in this document for transporting SPPP XML structures over SOAP and HTTP(s) is primarily comprised of five subjects: (1) a description of any applicable SOAP features, (2) any applicable HTTP features, (3) security considerations, and perhaps most importantly, (4) the Web Services Description Language (WSDL) definition for SPPP over SOAP, and (5) "transport" specific XML schema type definitions

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].

3. SOAP Features and Protocol Layering

The list of SOAP features that are explicitly used and required for SPPP are limited. Most SOAP features are not necessary for SPPP. SPPP primarily uses SOAP simply as a standard message envelope technology. The SOAP message envelope is comprised of the SOAP header and body. As described in the SOAP specifications, the SOAP header can contain optional, application specific, information about the message. The SOAP body contains the SPPP message itself, whose structure is defined by the combination of one of the WSDL operations defined in this document and the SPPP XML data structures defined in this document and the SPPP protocol document. SPPP does not rely on any data elements in the SOAP header. All relevant data elements are defined in the SPPP XML schema described in [I-D.draft-ietf-drinks-spprov] and the SPPP WSDL types specification described in this document.

WSDL is a widely standardized and adopted technology for defining the top-level structures of the messages that are transported within the body of a SOAP message. The WSDL definition for the SPPP SOAP messages is defined later in this document, which imports by reference the XML data types contained in the SPPP schema. The IANA registry where the SPPP schema resides is described in The IETF XML Registry [RFC3688].

There are multiple structural styles that SOAP WSDL allows. But the best practice for this type of application is what is sometimes referred to as the Document Literal Wrapped style of designing SOAP WSDL. This style is generally regarded as an optimal approach that enhances maintainability, comprehension, portability, and, to a certain extent, performance. It is characterized by setting the soapAction binding style as _document_, the soapAction encoding style as _literal_, and then defining the SOAP messages to simply contain a single data element that _wraps_ a data structure containing all the required input or output data elements. The figure below illustrates this high level technical structure as conceptual layers 3 through 6.

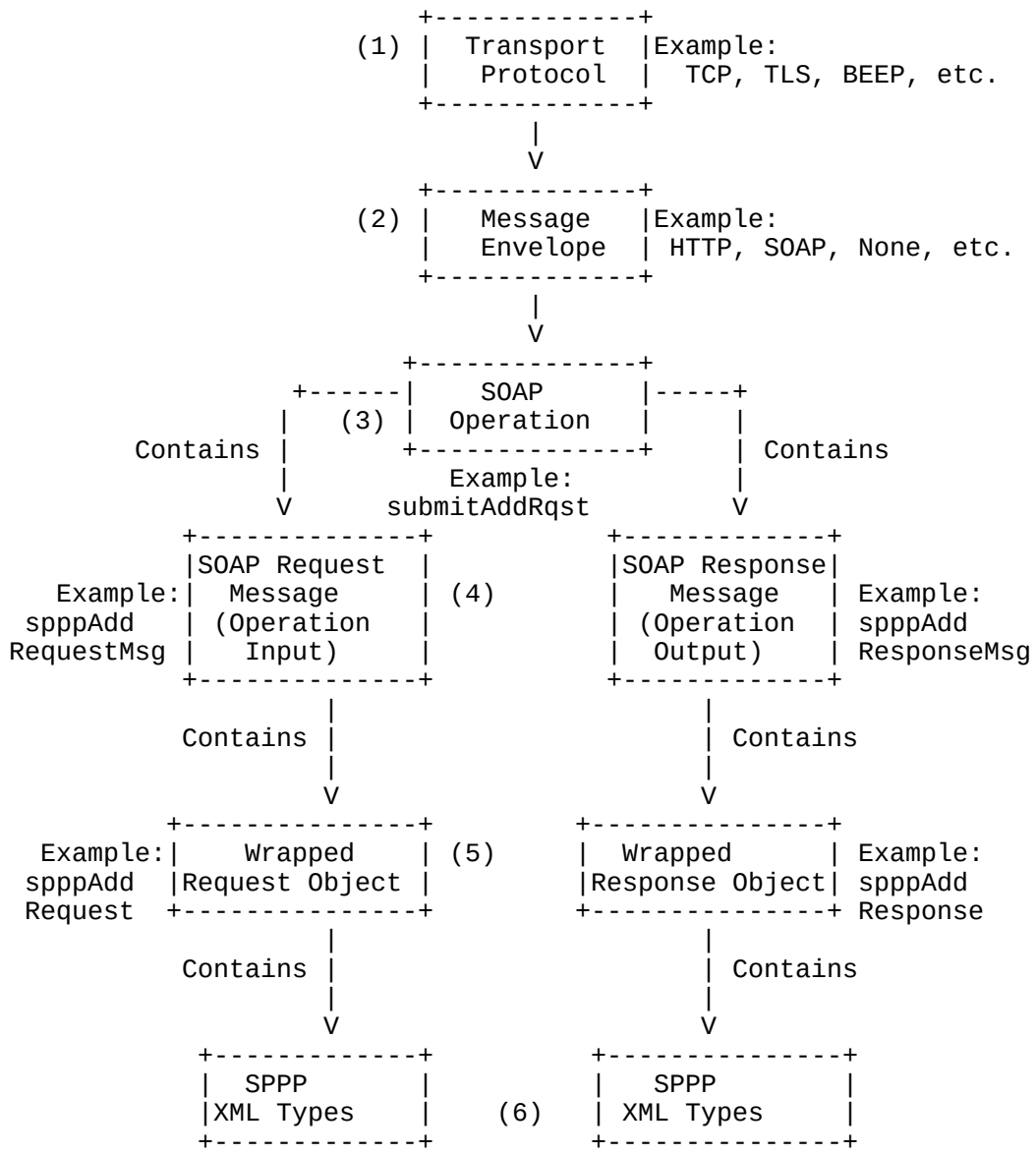


Figure 1: Layering and Technical Structure of the SPPP SOAP Messages

The SOAP operations supported by SPPP are normatively defined later in this document. Each SOAP operation defines a request/input message and a response/output message. Each such request and response message then contains a single object that wraps the SPPP XML data types that comprise the inputs and the outputs,

respectively, of the SOAP operation.

SOAP faults are not used by the SPPP SOAP mapping. All SPPP success and error responses are specified in the "Response Codes and Messages" section of this document. However, if a SOAP fault were to occur, perhaps due to failures in the SOAP message handling layer of a SOAP library, the client application should capture and handle the fault. Specifics on how to handle such SOAP faults, if they should occur, will be specific to the chosen SOAP implementation.

SOAP 1.2 [SOAPREF] or higher and WSDL 1.1 [WSDLREF] or higher SHOULD be used.

SPPP is a request/reply protocol that allows a client application to submit provisioning data and query requests to a server. The SPPP data structures are designed to be protocol agnostic. Concerns regarding encryption, non-repudiation, and authentication are beyond the scope of this document. For more details, please refer to the "Transport Protocol Requirements" section in the protocol document.

As illustrated in the previous diagram, SPPP can be viewed as a set of layers that collectively define the structure of an SPPP request and response. Layers 1 and 2 represent the transport, envelope, and authentication technologies. This document defines layers 3, 4, 5, and 6 below.

1. Layer 1: The transport protocol layer represents the communication mechanism between the client and server. SPPP can be layered over any transport protocol that provides a set of basic requirements defined in the Transport Protocol Requirements section. But this document specifies the required mechanism.

2. Layer 2: The message envelope layer is optional, but can provide features that are above the transport technology layer but below the application messaging layer. Technologies such as HTTP and SOAP are examples of messaging envelope technologies. This document specifies the required envelope technology.

3. Layers 3,4,5,6: The operation and message layers provides an envelope-independent and transport-independent wrapper for the SPPP data model objects that are being acted on (created, modified, queried).

4. HTTP(s) Features and SPPP

SOAP is not tied to HTTP(s), however, for reasons described in the introduction, HTTP(s) is a good choice as the transport mechanism for the SPPP SOAP messages. HTTP 1.1 includes the "persistent connection" feature, which allows multiple HTTP request/response pairs to be transported across a single HTTP connection. This is an important performance optimization feature, particularly when the connections is an HTTPS connection where the relatively time consuming SSL handshake has occurred. Persistent connections SHOULD be used for the SPPP HTTP connections.

HTTP 1.1 [RFC2616] or higher SHOULD be used.

5. Authentication and Session Management

To achieve integrity and privacy, conforming SPPP SOAP Clients and Servers MUST support SOAP over HTTP over TLS [RFC5246] as the secure transport mechanism. This combination of HTTP and TLS is referred to as HTTPS. And to accomplish authentication, conforming SOAP SPPP Clients and Servers MUST use HTTP Digest Authentication as defined in [RFC2617]. As a result, the communication session is established through the initial HTTP connection setup, the digest authentication, and the TLS handshake. When the HTTP connection is broken down, the communication session ends.

6. SPPP SOAP Data Structures

SPPP over SOAP uses a set of XML based data structures for all the supported operations and any parameters that those operations are applied to. As also mentioned earlier in this document, these XML structures are envelope-independent and transport-independent. Refer the "Protocol Operations" section of document for a description of all the operations that MUST be supported.

The following sections describe the definition all the XML data structures.

6.1. Concrete Object Key Types

Certain SPPP operations require an object key that uniquely identifies the object on which a given operation needs to be performed. The following sub-sections define the various types of concrete object key types used in certain operations:

6.1.1. Generic Object Key

Most objects in SPPP are unqiuely identified by the attributes in the concrete ObjKeyType. The definition of ObjKeyType is as below:

```
<complexType name="ObjKeyType">
 <complexContent>
  <extension base="spppb:ObjKeyType">
   <sequence>
    <element name="rant" type="spppb:OrgIdType"/>
    <element name="name" type="spppb:ObjNameType"/>
    <element name="type" type="spppb:ObjKeyTypeEnum"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

The ObjKeyType has the data elements as described below:

- rant: The identifier of the registrant organization that owns the object.

- name: The character string that contains the name of the object.

- type: The enumeration vaue that represents the type of SPPP object.

6.1.2. Public Identity Object Key

Public Identity type objects can further be of various sub-types like a TN, RN, TN Prefix, or a TN Range and cannot be cleanly identified with the attributes in the generic ObjKeyType. The definition of PubIdKeyType is as below:

```
<complexType name="PubIdKeyType">
 <complexContent>
  <extension base="spppb:PubIdKeyType">
   <sequence>
    <element name="rant" type="spppb:OrgIdType"/>
    <element name="dgName" type="spppb:ObjNameType" minOccurs="0"/>
    <choice>
     <element name="number"
     type="spppb:NumberType"/>
     <element name="range"
      type="spppb:NumberRangeType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

The PubIdKeyType has the data elements as described below:

- rant: The identifier of the registrant organization that owns the object.

- dgName: The name of the Destination Group that a Public Identifier is member of. Note that this is an optional attribute of the key as Public Identifiers may or may not be provisioned as members of a Destination Group.

- number: An element of type NumberType (refer protocol document) that contains the value and type of a the number .

- range: An element of type NumberRangeType (refer protocol document) that contains a rage of numbers.

It is MUST that only one of the "number" and "range" elements appears in a PubIdKeyType instance.

6.1.3. Route Group Offer Key

In addition to the attributes in the generic ObjKeyType, a Route Group Offer object is uniquely identified by the organization ID of the organization to whom an Route Group has been offered. The definition of RteGrpOfferKeyType is as below:

```
<complexType name="RteGrpOfferKeyType">
 <complexContent>
  <extension base="spppb:RteGrpOfferKeyType">
   <sequence>
    <element name="rteGrpKey" type="sppps:ObjKeyType"/>
    <element name="offeredTo" type="spppb:OrgIdType"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

The RteGrpOfferKeyType has the data elements as described below:

- rteGrpKey: Identifies the Route Group that was offered.

- offeredTo: The organization ID of the organization that was offered the Route Group object identified by the rteGrpKey.

6.2. Operation Request and Response Structures

An SPPP client interacts with an SPPP server by using one of the supported transport mechanisms to send one or more requests to the server and receive corresponding replies from the server. The basic set of operations that an SPPP client can submit to an SPPP server and the semantics of those operations are defined in the "Protocol Operations" section of the protocol document. The following sub-sections describe the XML data structures that are used for each of those types of operations for a SOAP based SPPP implementation.

6.2.1. Add Operation Structure

In order to add (or modify) an object in the registry, an authorized entity can send the spppAddRequest to the registry.

An SPPP Add request is wrapped within the <spppAddRequest> element while an SPPP Add response is wrapped within an <spppAddResponse> element. The following sub-sections describe the spppAddRequest and spppAddResponse elements. Refer the "SPPP SOAP Examples" section of this document for an example of Add operation on each type of SPPP object.

6.2.1.1. Add Request

An SPPP Add request definition is contained within the generic <spppAddRequest> element.

```
<element name="spppAddRequest">
 <complexType>
  <sequence>
   <element name="clientTransId"
    type="spppb:TransIdType" minOccurs="0"/>
   <element name="minorVer"
    type="spppb:MinorVerType" minOccurs="0"/>
   <element name="obj" type="spppb:BasicObjType"
   maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>

<simpleType name="TransIdType">
  <restriction base="string"/>
</simpleType>

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

The data elements within the <spppAddRequest> element are described as follows:

- clientTransId: Zero or one client-generated transaction ID that, within the context of the SPPP client, identifies this request. This value can be used at the discretion of the SPPP client to track, log or correlate requests and their responses. SPPP server MUST echo back this value to the client in the corresponding response to the incoming request. SPPP server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using the SPPP server menu operation described later in the document.

- obj: One or more elements of abstract type BasicObjType (defined in the protocol document). Each element contains all the attributes of an SPPP object that that the client is requesting the SPPP server to add. Refer the "Protocol Data Model Objects" section of the protocol document for the XML structure of all concrete types, for various SPPP objects, that extend from abstract BasicObjType and hence are eligible to be passed into this element. The elements are processed by the SPPP server in the order in which they are included in the request. With respect to handling of error conditions, it is a matter of policy whether the objects are processed in a "stop and rollback" fashion or in a "stop and commit" fashion. In the "stop and rollback" scenario, the SPPP server would stop processing BasicObjType elements in the request at the first error and roll back any BasicObjType elements that had already been processed for that add request. In the "stop and commit" scenario the SPPP server would stop processing BasicObjType elements in the request at the first error but commit any BasicObjType elements that had already been processed for that add request.

6.2.1.2. Add Response

An SPPP add response object is contained within the generic <spppAddResponse> element. This response structure is used for all types of SPPP objects that are provisioned by the SPPP client.

```
<element name="spppAddResponse">
  <complexType>
    <sequence>
      <element name="clientTransId" type="spppb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="spppb:TransIdType"/>
      <element name="overallResult" type="spppb:ResultCodeType"/>
      <element name="dtlResult" type="sppps:ObjResultCodeType"
      minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<complexType name="ResultCodeType">
  <sequence>
     <element name="code" type="int"/>
     <element name="msg" type="string"/>
  </sequence>
</complexType>

 <complexType name="ObjResultCodeType">
  <complexContent>
   <extension base="sppps:ResultCodeType">
    <sequence>
     <element name="obj" type="spppb:BasicObjType"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

An <spppAddResponse> contains the elements necessary for the SPPP client to precisely determine the overall result of the request, and if an error occurred, it provides information about the specific object(s) that caused the error.

The data elements within the SPPP Add response are described as follows:

- clientTransId: Zero or one client transaction ID. This value is simply an echo of the client transaction ID that SPPP client passed into the SPPP update request. When included in the request, the SPPP server MUST return it in the corresponding response message.

- serverTransId: Exactly one server transaction ID that identifies this request for tracking purposes. This value MUST be unique

for a given SPPP server.

- overallResult: Exactly one response code and message pair that explicitly identifies the result of the request. See the Response Code section for further details.

- dtlResult: An optional response code, response message, and BasicObjType (as defined in the protocol document) triplet. This element will be present only if an object level error has occurred. It indicates the error condition and the exact request object that contributed to the error. The response code will reflect the exact error. See the Response Code section for further details.

6.2.2. Delete Operation Structure

In order to remove an object from the registry, an authorized entity can send the spppDelRequest into the registry. An SPPP Del request is wrapped within the <spppDelRequest> element while a SPPP Del response is wrapped within the generic <spppDelResponse> element. The following sub-sections describe the spppDelRequest and spppDelResponse elements. Refer the "SPPP SOAP Examples" section of this document for an example of Delete operation on each type of SPPP object.

6.2.2.1. Delete Request

An SPPP Del request definition is contained within the generic <spppDelRequest> element.

```
<element name="spppDelRequest">
 <complexType>
  <sequence>
   <element name="clientTransId"
   type="spppb:TransIdType" minOccurs="0"/>
   <element name="minorVer"
   type="spppb:MinorVerType" minOccurs="0"/>
  <element name="objKey" type="spppb:ObjKeyType"
   maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

The data elements within the <spppDelRequest> element are described as follows:

- clientTransId: Zero or one client-generated transaction ID that, within the context of the SPPP client, identifies this request. This value can be used at the discretion of the SPPP client to track, log or correlate requests and their responses. SPPP server MUST echo back this value to the client in the corresponding response to the incoming request. SPPP server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using the SPPP server menu operation described later in the document.

- objKey: One or more elements of abstract type ObjKeyType (as defined in the protocol document). Each element contains attributes that uniquely identify the object that the client is requesting the server to delete. Refer the "Concrete Object Keys" section of this document for a description of all concrete object key types, for various SPPP objects, which are eligible to be passed into this element. The elements are processed by the SPPP server in the order in which they are included in the request. With respect to handling of error conditions, it is a matter of policy whether the objects are processed in a "stop and rollback" fashion or in a "stop and commit" fashion. In the "stop and rollback" scenario, the SPPP server would stop processing ObjKeyType elements in the request at the first error and roll back any ObjKeyType elements that had already been processed for that delete request. In the "stop and commit" scenario the SPPP server would stop processing ObjKeyType elements in the request at the first error but commit any KeyParamType elements that had already been processed for that delete request.

6.2.2.2. Delete Response

An SPPP delete response object is contained within the generic <sppDeleteResponse> element. This response structure is used for a delete request on all types of SPPP objects that are provisioned by the SPPP client.

```
<element name="spppDelResponse">
  <complexType>
    <sequence>
      <element name="clientTransId" type="spppb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="spppb:TransIdType"/>
      <element name="overallResult" type="spppb:ResultCodeType"/>
      <element name="dtlResult" type="sppps:ObjKeyResultCodeType"
      minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<complexType name="ResultCodeType">
  <sequence>
     <element name="code" type="int"/>
     <element name="msg" type="string"/>
  </sequence>
</complexType>

 <complexType name="ObjKeyResultCodeType">
  <complexContent>
   <extension base="sppps:ResultCodeType">
    <sequence>
     <element name="objKey" type="spppb:ObjKeyType"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

An <spppDelResponse> contains the elements necessary for the SPPP client to precisely determine the overall result of the request, and if an error occurred, it provides information about the specific object key(s) that caused the error.

The data elements within the SPPP Delete response are described as follows:

- clientTransId: Zero or one client transaction ID. This value is simply an echo of the client transaction ID that SPPP client passed into the SPPP update request. When included in the request, the SPPP server MUST return it in the corresponding response message.

- serverTransId: Exactly one server transaction ID that identifies this request for tracking purposes. This value MUST be unique

for a given SPPP server.

- overallResult: Exactly one response code and message pair that explicitly identifies the result of the request. See the Response Code section for further details.

- dtlResult: An optional response code, response message, and ObjKeyType (as defined in the protocol document) triplet. This element will be present only if an specific object key level error has occurred. It indicates the error condition and the exact request object key that contributed to the error. The response code will reflect the exact error. See the Response Code section for further details.

6.2.3. Accept Operation Structure

In SPPP, a Route Group Offer can be accepted or rejected by, or on behalf of, the registrant to whom the Route Group has been offered (refer "Protocol Data Model Objects" section of the protocol document for a description of the Route Group Offer object). The Accept operation is used to accept such Route Group Offers by, or on behalf of, the Registrant. The request structure for an SPPP Accept operation is wrapped within the <spppAcceptRequest> element while an SPPP Accept response is wrapped within the generic <spppAcceptResponse> element. The following sub-sections describe the spppAcceptRequest and spppAcceptResponse elements. Refer the "SPPP SOAP Examples" section of this document for an example of Accept operation on a Route Group Offer.

6.2.3.1. Accept Request Structure

An SPPP Accept request definition is contained within the generic <sppAcceptRequest> element.

```
<element name="spppAcceptRequest">
 <complexType>
  <sequence>
   <element name="clientTransId"
   type="spppb:TransIdType" minOccurs="0"/>
   <element name="minorVer"
   type="spppb:MinorVerType" minOccurs="0"/>
   <element name="rteGrpOfferKey"
   type="sppps:RteGrpOfferKeyType"
   maxOccurs="unbounded"/>
  </sequence>
```

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

The data elements within the <spppAcceptRequest> element are described as follows:

- clientTransId: Zero or one client-generated transaction ID that, within the context of the SPPP client, identifies this request. This value can be used at the discretion of the SPPP client to track, log or correlate requests and their responses. SPPP server MUST echo back this value to the client in the corresponding response to the incoming request. SPPP server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using the SPPP server menu operation described later in the document.

- rteGrpOfferKey: One or more elements of type RteGrpOfferKeyType (as defined in this document). Each element contains attributes that uniquely identify a Route Group Offer that the client is requesting the server to accept. The elements are processed by the SPPP server in the order in which they are included in the request. With respect to handling of error conditions, it is a matter of policy whether the objects are processed in a "stop and rollback" fashion or in a "stop and commit" fashion. In the "stop and rollback" scenario, the SPPP server would stop processing RteGrpOfferKeyType elements in the request at the first error and roll back any RteGrpOfferKeyType elements that had already been processed for that accept request. In the "stop and commit" scenario the SPPP server would stop processing RteGrpOfferKeyType elements in the request at the first error but commit any RteGrpOfferKeyType elements that had already been processed for that accept request.

6.2.3.2. Accept Response

An SPPP accept response structure is contained within the generic <sppAcceptResponse> element. This response structure is used for an Accept request on a Route Group Offer.

```
<element name="spppAcceptResponse">
  <complexType>
    <sequence>
      <element name="clientTransId" type="spppb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="spppb:TransIdType"/>
      <element name="overallResult" type="spppb:ResultCodeType"/>
      <element name="dtlResult"
      type="sppps:RteGrpOfferKeyResultCodeType"
      minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<complexType name="ResultCodeType">
  <sequence>
     <element name="code" type="int"/>
     <element name="msg" type="string"/>
  </sequence>
</complexType>

<complexType name="RteGrpOfferKeyResultCodeType">
  <complexContent>
   <extension base="sppps:ResultCodeType">
    <sequence>
     <element name="rteGrpOfferKey" type="sppps:RteGrpOfferKeyType"/>
    </sequence>
   </extension>
  </complexContent>
</complexType>
```

An <spppAcceptResponse> contains the elements necessary for the SPPP client to precisely determine the overall result of the request, and if an error occurred, it provides information about the specific Route Group Offer key(s) that caused the error.

The data elements within the SPPP Accept response are described as follows:

- clientTransId: Zero or one client transaction ID. This value is simply an echo of the client transaction ID that SPPP client passed into the SPPP update request. When included in the request, the SPPP server MUST return it in the corresponding response message.

- serverTransId: Exactly one server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPP server.

- overallResult: Exactly one response code and message pair that explicitly identifies the result of the request. See the Response Code section for further details.

- dtlResult: An optional response code, response message, and RteGrpOfferKeyType (as defined in this document) triplet. This element will be present only if any specific Route Group Offer key level error has occurred. It indicates the error condition and the exact request Route Group Offer key that contributed to the error. The response code will reflect the exact error. See the Response Code section for further details.

6.2.4. Reject Operation Structure

In SPPP, Route Group Offer can be accepted or rejected by, or on behalf of, the registrant to whom the Route Group has been offered (refer "Protocol Data Model Objects" section of this document for a description of the Route Group Offer object). The Reject operation is used to reject such Route Group Offers by, or on behalf of, the Registrant. The request structure for an SPPP Reject operation is wrapped within the <spppRejectRequest> element while an SPPP Reject response is wrapped within the generic <spppRejecResponse> element. The following sub-sections describe the spppRejectRequest and spppRejecResponse elements. Refer the "SPPP SOAP Examples" section of this document for an example of Reject operation on a Route Group Offer.

6.2.4.1. Reject Request

An SPPP Reject request definition is contained within the generic <spppRejectRequest> element.

```
<element name="spppRejectRequest">
 <complexType>
  <sequence>
   <element name="clientTransId"
   type="spppb:TransIdType" minOccurs="0"/>
   <element name="minorVer"
   type="spppb:MinorVerType" minOccurs="0"/>
   <element name="rteGrpOfferKey"
   type="sppps:RteGrpOfferKeyType"
   maxOccurs="unbounded"/>
 </complexType>
</element>
```

The data elements within the <spppRejectRequest> element are described as follows:

- clientTransId: Zero or one client-generated transaction ID that, within the context of the SPPP client, identifies this request. This value can be used at the discretion of the SPPP client to track, log or correlate requests and their responses. SPPP server MUST echo back this value to the client in the corresponding response to the incoming request. SPPP server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using the SPPP server menu operation described later in the document.

- rteGrpOfferKey: One or more elements of type RteGrpOfferKeyType (as defined in this document). Each element contains attributes that uniquely identify a Route Group Offer that the client is requesting the server to reject. The elements are processed by the SPPP server in the order in which they are included in the request. With respect to handling of error conditions, it is a matter of policy whether the objects are processed in a "stop and rollback" fashion or in a "stop and commit" fashion. In the "stop and rollback" scenario, the SPPP server would stop processing RteGrpOfferKeyType elements in the request at the first error and roll back any RteGrpOfferKeyType elements that

had already been processed for that reject request. In the "stop and commit" scenario the SPPP server would stop processing RteGrpOfferKeyType elements in the request at the first error but commit any RteGrpOfferKeyType elements that had already been processed for that reject request.

6.2.4.2. Reject Response

An SPPP reject response structure is contained within the generic <sppRejectResponse> element. This response structure is used for an Reject request on a Route Group Offer.

```
<element name="spppRejectResponse">
  <complexType>
    <sequence>
      <element name="clientTransId" type="spppb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="spppb:TransIdType"/>
      <element name="overallResult" type="spppb:ResultCodeType"/>
      <element name="dtlResult"
      type="sppps:RteGrpOfferKeyResultCodeType"
      minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<complexType name="ResultCodeType">
  <sequence>
     <element name="code" type="int"/>
     <element name="msg" type="string"/>
  </sequence>
</complexType>

<complexType name="RteGrpOfferKeyResultCodeType">
  <complexContent>
   <extension base="sppps:ResultCodeType">
    <sequence>
     <element name="rteGrpOfferKey" type="sppps:RteGrpOfferKeyType"/>
    </sequence>
   </extension>
  </complexContent>
</complexType>
```

An <spppRejectResponse> contains the elements necessary for the SPPP

client to precisely determine the overall result of the request, and if an error occurred, it provides information about the specific Route Group Offer key(s) that caused the error.

The data elements within the SPPP Reject response are described as follows:

- clientTransId: Zero or one client transaction ID. This value is simply an echo of the client transaction ID that SPPP client passed into the SPPP update request. When included in the request, the SPPP server MUST return it in the corresponding response message.

- serverTransId: Exactly one server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPP server.

- overallResult: Exactly one response code and message pair that explicitly identifies the result of the request. See the Response Code section for further details.

- dtlResult: An optional response code, response message, and RteGrpOfferKeyType (as defined in this document) triplet. This element will be present only if any specific Route Group Offer key level error has occurred. It indicates the error condition and the exact request Route Group Offer key that contributed to the error. The response code will reflect the exact error. See the Response Code section for further details.

6.2.5. Batch Operation Structure

An SPPP Batch request XML structure allows the SPPP client to send any of of Add, Del, Accept or Reject operations together in one single request. This gives an SPPP Client the flexibility to use one single request structure to perform more than operations (verbs). The batch request structure is wrapped within the <spppBatchRequest> element while a SPPP Batch response is wrapped within the <spppBatchResponse> element. This following sub-sections describe the spppBatchRequest and spppBatchResponse elements. Refer the "SPPP SOAP Examples" section of this document for an example of a batch operation.

6.2.5.1. Batch Request Structure

An SPPP Batch request definition is contained within the generic <spppBatchRequest> element.

```
      <element name="spppBatchRequest">
      <complexType>
       <sequence>
        <element name="clientTransId"
        type="spppb:TransIdType" minOccurs="0"/>
        <element name="minorVer"
        type="spppb:MinorVerType" minOccurs="0"/>
         <choice minOccurs="1" maxOccurs="unbounded">
          <element name="addObj" type="spppb:BasicObjType"/>
          <element name="delObj" type="spppb:ObjKeyType"/>
          <element name="acceptRteGrpOffer"
          type="sppps:RteGrpOfferKeyType"/>
          <element name="rejectRteGrpOffer"
          type="sppps:RteGrpOfferKeyType"/>
         </choice>
       </sequence>
      </complexType>
     </element>
```

The data elements within the <sppBatchRequest> element are described as follows:

- clientTransId: Zero or one client-generated transaction ID that, within the context of the SPPP client, identifies this request. This value can be used at the discretion of the SPPP client to track, log or correlate requests and their responses. SPPP server MUST echo back this value to the client in the corresponding response to the incoming request. SPPP server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using the SPPP server menu operation described later in the document.

- addObj: One or more elements of abstract type BasicObjType where each element identifies an object that needs to be added.

- delObj: One or more elements of abstract type ObjKeyType where each element identifies a key for the object that needs to be deleted .

- acceptRteGrpOffer: One or more elements of type RteGrpOfferKeyType where each element identifies a Route Group Offer that needs to be accepted.

- rejectRteGrpOffer: One or more elements of type RteGrpOfferKeyType where each element identifies a Route Group Offer that needs to be rejected.

With respect to handling of error conditions, it is a matter of policy whether the batch operation processed in a "stop and rollback" fashion or in a "stop and commit" fashion. In the "stop and rollback" scenario, the SPPP server would stop processing elements in the request at the first error and roll back any elements that had already been processed for that batch request. In the "stop and commit" scenario the SPPP server would stop processing elements in the request at the first error but commit any elements that had already been processed for that batch request.

6.2.5.2. Batch Response

An SPPP batch response structure is contained within the generic <sppBatchResponse> element. This response structure is used for an Batch request that contains many different types of SPPP operations.

```
<element name="spppBatchResponse">
  <complexType>
    <sequence>
      <element name="clientTransId" type="spppb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="spppb:TransIdType"/>
      <element name="overallResult" type="spppb:ResultCodeType"/>
      <choice minOccurs="0" maxOccurs="unbounded">
         <element name="addResult"
         type="sppps:ObjResultCodeType"/>
         <element name="delResult"
         type="sppps:ObjKeyResultCodeType"/>
         <element name="acceptResult"
         type="sppps:RteGrpOfferKeyResultCodeType"/>
         <element name="rejectResult"
         type="sppps:RteGrpOfferKeyResultCodeType"/>
       </choice>
    </sequence>
  </complexType>
</element>
```

An <spppBatchResponse> contains the elements necessary for an SPPP client to precisely determine the overall result of various operations in the request, and if an error occurred, it provides information about the specific objects or keys in the request that caused the error.

The data elements within the SPPP Batch response are described as follows:

- clientTransId: Zero or one client transaction ID. This value is simply an echo of the client transaction ID that SPPP client passed into the SPPP update request. When included in the request, the SPPP server MUST return it in the corresponding response message.

- serverTransId: Exactly one server transaction ID that identifies this request for tracking purposes. This value MUST be unique for a given SPPP server.

- overallResult: Exactly one response code and message pair that explicitly identifies the result of the request. See the Response Code section for further details.

- addResult: One or more elements of type ObjResultCodeType where each element identifies the result code, result message and the specific object that the result relates to.

- delResult: One or more elements of type ObjKeyResultCodeType where each element identifies the result code, result message and the specific object key that the result relates to.

- acceptResult: One or more elements of type RteGrpOfferKeyResultCodeType where each element identifies the result code, result message and the specific Route Group Offer key that the result relates to.

- rejectResult: One or more elements of type RteGrpOfferKeyResultCodeType where each element identifies the result code, result message and the specific Route Group Offer key that the result relates to.

6.2.6. Get Operation Structure

In order to query the details of an object from the Registry, an authorized entity can send the spppGetRequest to the registry with a GetRqstType XML data structure containing one or more object keys that uniquely identify the object whose details are being queried. The request strcuture for an SPPP Get operation is contained within

the generic <spppGetRequest> element while an SPPP Get response is wrapped within the generic <spppGetResponse> element. The following sub-sections describe the spppGetRequest and spppGetResponse element. Refer the examples section for an example of Get operation on each type of SPPP object

6.2.6.1. Get Request

```
<element name="spppGetRequest">
 <complexType>
  <sequence>
   <element name="minorVer"
   type="spppb:MinorVerType" minOccurs="0"/>
   <element name="objKey"
   type="spppb:ObjKeyType"
   maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

The data elements within the <spppGetRequest> element are described as follows:

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using the SPPP server menu operation described later in the document.

- objKey: One or more elements of abstract type ObjKeyType (as defined in the protocol document). Each element contains attributes that uniquely identify the object that the client is requesting the server to query. Refer the "Concrete Object Keys" section of this document for a description of all concrete object key types, for various SPPP objects, which are eligible to be passed into this element.

6.2.6.2. Get Response

The spppGetResponse element is described later in section titled "Generic Query Response".

6.2.7. Get Route Group Offers Operation Structure

In addition to the ability to query the details of one or more Route Group offers using an a Route Group Offer key in the spppGetRequest, this operation also provides an additonal, more flexible, structure to query for Route Group Offer objects. This additional structure is contained within the <getRteGrpOffersRequest> element while the response is wrapped within the generic <spppGetResponse> element. The following sub-sections describe the getRteGrpOffersRequest and spppGetResponse elements.

6.2.7.1. Get Route Group Offers Request

Using the details passed into this structure, the server will attempt to find Route Group Offer objects that satisfy all the criteria passed into the request. If no criteria is passed in then the server will return the list of Route Group Offer objects that belongs to the registrant. If there are no matching Route Group Offers found then an empty result set will be returned.

```
<element name="getRteGrpOffersRequest">
<complexType>
 <sequence>
  <element name="minorVer" type="spppb:MinorVerType"
  minOccurs="0"/>
 <element name="offeredBy" type="spppb:OrgIdType"
 minOccurs="0" maxOccurs="unbounded"/>
 <element name="offeredTo" type="spppb:OrgIdType"
 minOccurs="0" maxOccurs="unbounded"/>
 <element name="status" type="spppb:RteGrpOfferStatusType"
  minOccurs="0"/>
 <element name="rteGrpOfferKey" type="sppps:RteGrpOfferKeyType"
 minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
</element>
```

The data elements within the <getRteGrpOffersRequest> element are described as follows:

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using the SPPP server menu operation described later in the document.

- offeredBy: Zero or more organization IDs. Only offers that are offered to the organization IDs in this list should be included in the result set. The result set is also subject to other query criteria in the request.

- offeredTo: Zero or more organization IDs. Only offers that are offered by the organization IDs in this list should be included in the result set. The result set is also subject to other query criteria in the request.

- status: The status of the offer, offered or accepted. Only offers in the specified status should be included in the result set. If this element is not present then the status of the offer should not be considered in the query. The result set is also subject to other query criteria in the request.

- rteGrpOfferKey: Zero or more Route Group Offer Keys. Only offers having one of these keys should be included in the result set. The result set is also subject to other query criteria in the request.

6.2.7.2. Get Route Group Offers Response

The spppGetResponse element is described later in section titled "Generic Query Response".

6.2.8. Generic Query Response

An SPPP query response object is contained within the generic <spppGetResponse> element.

```
<element name="spppGetResponse">
 <complexType>
  <sequence>
   <element name="overallResult"
   type="sppps:ResultCodeType"/>
   <element name="resultObj"
   type="spppb:BasicObjType"
   minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

An <spppGetResponse> contains the elements necessary for the SPPP client to precisely determine the overall result of the query, and details of any SPPP objects that matched the criteria in the request.

The data elements within the SPPP query response are described as follows:

- overallResult: Exactly one response code and message pair that explicitly identifies the result of the request. See the Response Code section for further details.

- resultObj: The set of zero or more objects that matched the query criteria. If no objects matched the query criteria then the result object(s) MUST be empty and the overallResult value MUST indicate success (if no matches are found for the query criteria, the response is considered a success).

6.2.9. Get Server Details Operation Structure

In order to query certain details of the SPPP server, like the SPPP server's status and the major/minor version supported by the server, the Server Details operation structure SHOULD be used. This structure is contained within the <spppServerStatusRequest> element while a SPPP server status response is wrapped within the <spppServerStatusResponse> element. This following sub-sections describe the spppServerStatusRequest and spppServerStatusResponse elements.

6.2.9.1. Get Server Details Request

```
<element name="spppServerStatusRequest">
 <complexType>
  <sequence>
   <element name="minorVer"
   type="spppb:MinorVerType" minOccurs="0"/>
  </sequence>
 </complexType>
</element>
```

The data elements within the <spppServerStatusRequest> element are described as follows:

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPP API that the client is attempting to use. This is used in conjunction with the major version identifier in the XML namespace to identify the version of SPPP that the client is using. If the element is not present, the server assumes that the client is using the latest minor version supported by the SPPP server for the given major version. The versions supported by a given SPPP server can be retrieved by the client using this same spppServerStatusRequest without passing in the minorVer element.

6.2.9.2. Get Server Details Response

An SPPP server details response structure is contained within the generic <spppServerStatusResponse> element.

```
<element name="spppServerStatusResponse">
 <complexType>
  <sequence>
   <element name="overallResult" type="sppps:ResultCodeType"/>
   <element name="svcMenu" type="spppb:SvcMenuType"/>
  </sequence>
 </complexType>
</element>
```

The data elements within the <spppServerStatusResponse> element are described as follows:

- overallResult: Exactly one response code and message pair that explicitly identifies the result of the request. See the Response Code section for further details.

- svcMenu: Exactly one element of type SvcMenuType which in turn contains the elements to return the server status and major/ minor version of the SPPP protocol supported by the SPPP server (refer protocol document for definition of SvcMenuType) .

6.3. Response Codes and Messages

This section contains the listing of response codes and their corresponding human-readable text. These response codes are in conformance with the response types defined in the section "Response Message Types" of the protocol document.

The response code numbering scheme generally adheres to the theory formalized in section 4.2.1 of [RFC5321]:

- The first digit of the response code can only be 1 or 2: 1 = a positive result, 2 = a negative result.

- The second digit of the response code indicates the category: 0 = Protocol Syntax, 1 = Implementation Specific Business Rule, 2 = Security, 3 = Server System.

- The third and fourth digits of the response code indicate the individual message event within the category defines by the first two digits.

The response codes are also categorized as to whether they are overall response codes that may only be returned in the "overallResult" data element in SPPP responses, or object level response codes that may only be returned in the "dtlResult" element of the SPPP responses.

Result Code	Result Message	Overall or Object Level
1000	Request Succeeded.	Overall Response Code
2001	Request syntax invalid.	Overall Response Code
2002	Request too large.	Overall Response Code
2003	Version not supported.	Overall Response Code
2103	Command invalid.	Overall Response Code
2301	System temporarily unavailable.	Overall Response Code
2302	Unexpected internal system or server error.	Overall Response Code
2104	Attribute value invalid. AttrName:[AttributeName] AttrVal:[AttributeValue]	Object Level Response Code
2105	Object does not exist. AttrName:[AttributeName] AttrVal:[AttributeValue]	Object Level Response Code
2106	Object status or ownership does not allow for operation. AttrName:[AttributeName] AttrVal:[AttributeValue]	Object Level Response Code

Table 1: Response Codes Numbering Scheme and Messages

Each of the object level response messages are "parameterized" with the following parameters: "AttributeName" and "AttributeValue".

The use of these parameters MUST adhere to the rules defined in "Response Message Types" section of the protocol document.

7. Protocol Operations

Refer the "Protocol Operations" section of the protocol document for a description of all SPPP operations, and any necessary semantics that MUST be adhered to in order to conform with the SPPP protocol specification.

8. SPPP SOAP WSDL Definition

The SPPP WSDL and data types are defined below. The WSDL design approach is commonly referred to as _Generic WSDL_. It is generic in the sense that there is not a specific WSDL operation defined for each object type that is supported by the SPPP protocol. There is a single WSDL structure for each type of SPPP operation. Each such WSDL structure contains exactly one input structure and one output structure that wraps any data elements that are part of the incoming request and the outgoing response respectively. The spppSOAPBinding in the WSDL defines the binding style as _document_ and the encoding as _literal_. It is this combination of _wrapped_ input and output data structures, _document_ binding style, and _literal_ encoding that characterize the Document Literal Wrapped style of WSDL specifications.

Note: The following WSDL has been formatted (e.g., tabs, spaces) to meet I-D requirements.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:spppb="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:sppps="urn:ietf:params:xml:ns:sppp:soap:1"
targetNamespace="urn:ietf:params:xml:ns:sppp:soap:1">
 <wsdl:types>
  <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sppps="urn:ietf:params:xml:ns:sppp:soap:1"
  targetNamespace="urn:ietf:params:xml:ns:sppp:soap:1">
   <annotation>
    <documentation>
                                        ---- Import base schema ----
                                </documentation>
   </annotation>
   <import namespace="urn:ietf:params:xml:ns:sppp:base:1"
   schemaLocation="spppbase.xsd"/>
   <annotation>
    <documentation>
                                        ---- Key type(s) extended
                                        from base schema. ----
                                </documentation>
   </annotation>
   <complexType name="ObjKeyType">
    <complexContent>
```

```
        <extension base="spppb:ObjKeyType">
         <sequence>
          <element name="rant" type="spppb:OrgIdType"/>
          <element name="name" type="spppb:ObjNameType"/>
          <element name="type" type="spppb:ObjKeyTypeEnum"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="RteGrpOfferKeyType">
       <complexContent>
        <extension base="spppb:RteGrpOfferKeyType">
         <sequence>
          <element name="rteGrpKey"
          type="sppps:ObjKeyType"/>
          <element name="offeredTo"
          type="spppb:OrgIdType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="PubIdKeyType">
       <complexContent>
        <extension base="spppb:PubIdKeyType">
         <sequence>
          <element name="rant" type="spppb:OrgIdType"/>
          <element name="dgName"
          type="spppb:ObjNameType" minOccurs="0"/>
          <choice>
           <element name="number"
           type="spppb:NumberType"/>
           <element name="range"
           type="spppb:NumberRangeType"/>
          </choice>
         </sequence>
        </extension>
       </complexContent>
      </complexType>

      <annotation>
       <documentation>
                                          ---- Generic Request and
                                          Response Definitions ----
                                 </documentation>
      </annotation>
      <element name="spppAddRequest">
       <complexType>
        <sequence>
```

```
          <element name="clientTransId"
          type="spppb:TransIdType" minOccurs="0"/>
          <element name="minorVer"
          type="spppb:MinorVerType" minOccurs="0"/>
          <element name="obj" type="spppb:BasicObjType"
          maxOccurs="unbounded"/>
         </sequence>
        </complexType>
       </element>
       <element name="spppDelRequest">
        <complexType>
         <sequence>
          <element name="clientTransId"
          type="spppb:TransIdType" minOccurs="0"/>
          <element name="minorVer"
          type="spppb:MinorVerType" minOccurs="0"/>
          <element name="objKey"
          type="spppb:ObjKeyType" maxOccurs="unbounded"/>
         </sequence>
        </complexType>
       </element>
       <element name="spppAcceptRequest">
        <complexType>
         <sequence>
          <element name="clientTransId"
          type="spppb:TransIdType" minOccurs="0"/>
          <element name="minorVer"
          type="spppb:MinorVerType" minOccurs="0"/>
          <element name="rteGrpOfferKey"
          type="sppps:RteGrpOfferKeyType"
          maxOccurs="unbounded"/>
         </sequence>
        </complexType>
       </element>
       <element name="spppRejectRequest">
        <complexType>
         <sequence>
          <element name="clientTransId"
          type="spppb:TransIdType" minOccurs="0"/>
          <element name="minorVer"
          type="spppb:MinorVerType" minOccurs="0"/>
          <element name="rteGrpOfferKey"
          type="sppps:RteGrpOfferKeyType"
          maxOccurs="unbounded"/>
         </sequence>
        </complexType>
       </element>
       <element name="spppGetRequest">
```

```
       <complexType>
        <sequence>
         <element name="minorVer"
         type="spppb:MinorVerType" minOccurs="0"/>
         <element name="objKey"
         type="spppb:ObjKeyType"
         maxOccurs="unbounded"/>
        </sequence>
       </complexType>
      </element>
      <element name="spppBatchRequest">
       <complexType>
        <sequence>
         <element name="clientTransId"
         type="spppb:TransIdType" minOccurs="0"/>
         <element name="minorVer"
         type="spppb:MinorVerType" minOccurs="0"/>
          <choice minOccurs="1" maxOccurs="unbounded">
           <element name="addObj" type="spppb:BasicObjType"/>
           <element name="delObj" type="spppb:ObjKeyType"/>
           <element name="acceptRteGrpOffer"
           type="sppps:RteGrpOfferKeyType"/>
           <element name="rejectRteGrpOffer"
           type="sppps:RteGrpOfferKeyType"/>
          </choice>
        </sequence>
       </complexType>
      </element>
      <element name="spppServerStatusRequest">
       <complexType>
        <sequence>
         <element name="minorVer"
         type="spppb:MinorVerType" minOccurs="0"/>
        </sequence>
       </complexType>
      </element>
      <element name="getRteGrpOffersRequest">
       <complexType>
        <sequence>
         <element name="minorVer"
         type="spppb:MinorVerType" minOccurs="0"/>
        <element name="offeredBy"
        type="spppb:OrgIdType" minOccurs="0"
        maxOccurs="unbounded"/>
        <element name="offeredTo" type="spppb:OrgIdType"
        minOccurs="0" maxOccurs="unbounded"/>
        <element name="status"
        type="spppb:RteGrpOfferStatusType" minOccurs="0"/>
```

```
        <element name="rteGrpOfferKey"
        type="sppps:RteGrpOfferKeyType"
        minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
       </complexType>
      </element>
      <element name="spppAddResponse">
       <complexType>
        <sequence>
         <element name="clientTransId"
         type="spppb:TransIdType" minOccurs="0"/>
         <element name="serverTransId"
         type="spppb:TransIdType"/>
         <element name="overallResult"
         type="sppps:ResultCodeType"/>
         <element name="dtlResult"
         type="sppps:ObjResultCodeType"
         minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
       </complexType>
      </element>
      <element name="spppDelResponse">
       <complexType>
        <sequence>
         <element name="clientTransId"
         type="spppb:TransIdType" minOccurs="0"/>
         <element name="serverTransId"
         type="spppb:TransIdType"/>
         <element name="overallResult"
         type="sppps:ResultCodeType"/>
         <element name="dtlResult"
         type="sppps:ObjKeyResultCodeType"
         minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
       </complexType>
      </element>
         <element name="spppAcceptResponse">
       <complexType>
        <sequence>
         <element name="clientTransId"
         type="spppb:TransIdType" minOccurs="0"/>
         <element name="serverTransId"
         type="spppb:TransIdType"/>
         <element name="overallResult"
         type="sppps:ResultCodeType"/>
         <element name="dtlResult"
         type="sppps:RteGrpOfferKeyResultCodeType"
         minOccurs="0" maxOccurs="unbounded"/>
```

```
        </sequence>
       </complexType>
      </element>
            <element name="spppRejectResponse">
       <complexType>
        <sequence>
         <element name="clientTransId"
         type="spppb:TransIdType" minOccurs="0"/>
         <element name="serverTransId"
         type="spppb:TransIdType"/>
         <element name="overallResult"
         type="sppps:ResultCodeType"/>
         <element name="dtlResult"
         type="sppps:RteGrpOfferKeyResultCodeType"
         minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
       </complexType>
      </element>
       <element name="spppBatchResponse">
       <complexType>
        <sequence>
         <element name="clientTransId"
         type="spppb:TransIdType" minOccurs="0"/>
         <element name="serverTransId"
         type="spppb:TransIdType"/>
         <element name="overallResult"
         type="sppps:ResultCodeType"/>
          <choice minOccurs="0" maxOccurs="unbounded">
                  <element name="addResult"
                  type="sppps:ObjResultCodeType"/>
                  <element name="delResult"
                  type="sppps:ObjKeyResultCodeType"/>
                  <element name="acceptResult"
                  type="sppps:RteGrpOfferKeyResultCodeType"/>
                  <element name="rejectResult"
                  type="sppps:RteGrpOfferKeyResultCodeType"/>
          </choice>
        </sequence>
       </complexType>
      </element>
      <element name="spppGetResponse">
       <complexType>
        <sequence>
         <element name="overallResult"
         type="sppps:ResultCodeType"/>
         <element name="resultObj"
         type="spppb:BasicObjType"
         minOccurs="0" maxOccurs="unbounded"/>
```

```
        </sequence>
       </complexType>
      </element>
      <element name="spppServerStatusResponse">
       <complexType>
        <sequence>
         <element name="overallResult"
         type="sppps:ResultCodeType"/>
         <element name="svcMenu"
         type="spppb:SvcMenuType"/>
        </sequence>
       </complexType>
      </element>
      <annotation>
       <documentation>
                                          ---- Operation Result Type
                                          Definitions ----
                                 </documentation>
      </annotation>
      <complexType name="ResultCodeType">
       <sequence>
        <element name="code" type="int"/>
        <element name="msg" type="token"/>
       </sequence>
      </complexType>
      <complexType name="ObjResultCodeType">
       <complexContent>
        <extension base="sppps:ResultCodeType">
         <sequence>
          <element name="obj" type="spppb:BasicObjType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="ObjKeyResultCodeType">
       <complexContent>
        <extension base="sppps:ResultCodeType">
         <sequence>
          <element name="objKey" type="spppb:ObjKeyType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
         <complexType name="RteGrpOfferKeyResultCodeType">
       <complexContent>
        <extension base="sppps:ResultCodeType">
         <sequence>
          <element name="rteGrpOfferKey"
```

```
          type="sppps:RteGrpOfferKeyType"/>
        </sequence>
       </extension>
      </complexContent>
      </complexType>
 </xsd:schema>
</wsdl:types>
<wsdl:message name="spppAddRequestMsg">
 <wsdl:part name="rqst" element="sppps:spppAddRequest"/>
</wsdl:message>
<wsdl:message name="spppDelRequestMsg">
 <wsdl:part name="rqst" element="sppps:spppDelRequest"/>
</wsdl:message>
<wsdl:message name="spppAcceptRequestMsg">
 <wsdl:part name="rqst" element="sppps:spppAcceptRequest"/>
</wsdl:message>
<wsdl:message name="spppRejectRequestMsg">
 <wsdl:part name="rqst" element="sppps:spppRejectRequest"/>
</wsdl:message>
<wsdl:message name="spppBatchRequestMsg">
 <wsdl:part name="rqst" element="sppps:spppBatchRequest"/>
</wsdl:message>
<wsdl:message name="spppGetRequestMsg">
 <wsdl:part name="rqst" element="sppps:spppGetRequest"/>
</wsdl:message>
<wsdl:message name="spppGetRteGrpOffersRequestMsg">
 <wsdl:part name="rqst" element="sppps:getRteGrpOffersRequest"/>
</wsdl:message>
<wsdl:message name="spppAddResponseMsg">
 <wsdl:part name="rspns" element="sppps:spppAddResponse"/>
</wsdl:message>
 <wsdl:message name="spppDelResponseMsg">
 <wsdl:part name="rspns" element="sppps:spppDelResponse"/>
</wsdl:message>
 <wsdl:message name="spppAcceptResponseMsg">
 <wsdl:part name="rspns" element="sppps:spppAcceptResponse"/>
</wsdl:message>
 <wsdl:message name="spppRejectResponseMsg">
 <wsdl:part name="rspns" element="sppps:spppRejectResponse"/>
</wsdl:message>
 <wsdl:message name="spppBatchResponseMsg">
 <wsdl:part name="rspns" element="sppps:spppBatchResponse"/>
</wsdl:message>
<wsdl:message name="spppGetResponseMsg">
 <wsdl:part name="rspns" element="sppps:spppGetResponse"/>
</wsdl:message>
<wsdl:message name="spppServerStatusRequestMsg">
 <wsdl:part name="rqst" element="sppps:spppServerStatusRequest"/>
```

```
    </wsdl:message>
    <wsdl:message name="spppServerStatusResponseMsg">
     <wsdl:part name="rspns" element="sppps:spppServerStatusResponse"/>
    </wsdl:message>
    <wsdl:portType name="spppPortType">
     <wsdl:operation name="submitAddRqst">
      <wsdl:input message="sppps:spppAddRequestMsg"/>
      <wsdl:output message="sppps:spppAddResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitDelRqst">
      <wsdl:input message="sppps:spppDelRequestMsg"/>
      <wsdl:output message="sppps:spppDelResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitAcceptRqst">
      <wsdl:input message="sppps:spppAcceptRequestMsg"/>
      <wsdl:output message="sppps:spppAcceptResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitRejectRqst">
      <wsdl:input message="sppps:spppRejectRequestMsg"/>
      <wsdl:output message="sppps:spppRejectResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitBatchRqst">
      <wsdl:input message="sppps:spppBatchRequestMsg"/>
      <wsdl:output message="sppps:spppBatchResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitGetRqst">
      <wsdl:input message="sppps:spppGetRequestMsg"/>
      <wsdl:output message="sppps:spppGetResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitGetRteGrpOffersRqst">
      <wsdl:input message="sppps:spppGetRteGrpOffersRequestMsg"/>
      <wsdl:output message="sppps:spppGetResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitServerStatusRqst">
      <wsdl:input message="sppps:spppServerStatusRequestMsg"/>
      <wsdl:output message="sppps:spppServerStatusResponseMsg"/>
     </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="spppSoapBinding" type="sppps:spppPortType">
     <soap:binding style="document"
     transport="http://schemas.xmlsoap.org/soap/http"/>
     <wsdl:operation name="submitAddRqst">
      <soap:operation soapAction="submitAddRqst" style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
```

```
      </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="submitDelRqst">
      <soap:operation soapAction="submitDelRqst" style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="submitAcceptRqst">
      <soap:operation soapAction="submitAcceptRqst" style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="submitRejectRqst">
      <soap:operation soapAction="submitRejectRqst" style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="submitBatchRqst">
      <soap:operation soapAction="submitBatchRqst" style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="submitGetRqst">
      <soap:operation soapAction="submitGetRqst" style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="submitGetRteGrpOffersRqst">
```

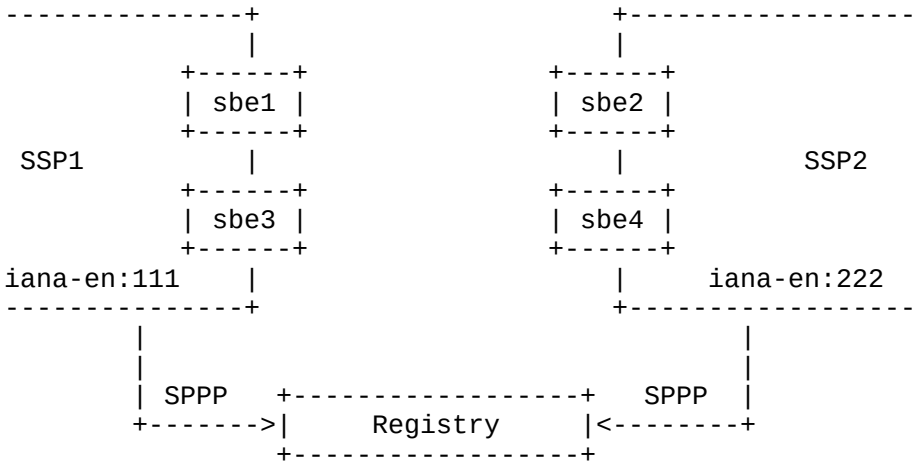
```
      <soap:operation soapAction="submitGetRteGrpOffersRqst"
      style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
     </wsdl:operation>
     <wsdl:operation name="submitServerStatusRqst">
      <soap:operation soapAction="submitServerStatusRqst"
      style="document"/>
      <wsdl:input>
       <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body use="literal"/>
      </wsdl:output>
     </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="spppService">
     <wsdl:port name="spppPort" binding="sppps:spppSoapBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
     </wsdl:port>
    </wsdl:service>
   </wsdl:definitions>
```

Figure 2: WSDL

9. SPPP SOAP Examples

This section shows XML message exchange between two SIP Service Providers (SSP) and a registry. The SPPP messages in this section are valid XML instances that conform to the SOAP based SPPP schema version within this document. This section relies on the XML data structures defined in the base SPPP protocol specification [I-D.draft-ietf-drinks-spprov]. So refer to that document to understand XML object types embedded in these example messages.

In this sample use case scenario, SSP1 and SSP2 provision resource data in the registry and use SPPP constructs to selectively share the route groups. In the figure below, SSP2 has two ingress SBE instances that are associated with the public identities that SSP2 has the retail relationship with. Also, the two SBE instances for SSP1 are used to show how to use SPPP to associate route preferences for the destination ingress routes and exercise greater control on outbound traffic to the peer's ingress SBEs.



9.1. Add Destination Group

SSP2 adds a destination group to the registry for use later. The SSP2 SPPP client sets a unique transaction identifier 'txn_1479' for tracking purposes. The name of the destination group is set to DEST_GRP_SSP2_1

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <!--Optional:-->
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:DestGrpType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

The registry processes the request and return a favorable response confirming successful creation of the named destination group. Also, besides returning a unique server transaction identifier, Registry also returns the matching client transaction identifier from the request message back to the SPPP client.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.2. Add Route Records

SSP2 adds an ingress routes in the registry.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <!--Optional:-->
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:NAPTRType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:rrName>RTE_SSP2_SBE2</urn1:rrName>
            <urn1:order>10</urn1:order>
            <urn1:flags>u</urn1:flags>
            <urn1:svcs>E2U+sip</urn1:svcs>
            <urn1:regx>
               <urn1:ere>^(.*)$</urn1:ere>
               <urn1:repl>sip:\1@sbe2.ssp2.example.com</urn1:repl>
            </urn1:regx>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

The registry returns a success response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.3. Add Route Records -- URIType

SSP2 adds another ingress routes in the registry and makes use of URIType

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:URIType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:rrName>RTE_SSP2_SBE4</urn1:rrName>
            <urn1:ere>^(.*)$</urn1:ere>
            <urn1:uri>sip:\1;npdi@sbe4.ssp2.example.com</urn1:uri>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

The registry returns a success response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.4. Add Route Group

SSP2 creates the grouping of the ingress routes and choses higher precedence for RTE_SSP2_SBE2 by setting a lower number for the "priority" attribute, a protocol agnostic precedence indicator.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:RteGrpType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:rgName>RTE_GRP_SSP2_1</urn1:rgName>
            <urn1:rrRef>
               <urn1:rrKey xsi:type="urn:ObjKeyType">
                  <rant>iana-en:222</rant>
                  <name>RTE_SSP2_SBE2</name>
                                 <type>RteRec</type>
               </urn1:rrKey>
               <urn1:priority>100</urn1:priority>
            </urn1:rrRef>
            <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
            <urn1:isInSvc>true</urn1:isInSvc>
            <urn1:priority>10</urn1:priority>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

To confirm successful processing of this request, registry returns a well-known result code ’1000’ to the SSP2 client.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.5. Add Public Identity -- Successful COR claim

SSP2 activates a TN public identity by associating it with a valid destination group. Further, SSP2 puts forth a claim that it is the carrier-of-record for the TN.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:TNType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
                        <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
            <urn1:tn>+12025556666</urn1:tn>
            <urn1:corInfo>
               <urn1:corClaim>true</urn1:corClaim>
            </urn1:corInfo>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Assuming that the registry has access to TN authority data and it performs the required checks to verify that SSP2 is in fact the service provider of record for the given TN, the request is processed successfully. In the response message, the registry sets the value of <cor> to "true" in order to confirm SSP2 claim as the carrier of record and the <corDate> reflects the time when the carrier of record claim is processed.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
         <dtlResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
            <obj xsi:type="ns2:TNType">
               <ns2:rant>iana-en:222</ns2:rant>
               <ns2:rar>iana-en:223</ns2:rar>
               <ns2:cDate>2010-05-30T09:30:10Z</ns2:cDate>
               <ns2:dgName>DEST_GRP_SSP2_1</ns2:dgName>
               <ns2:tn>+12025556666</ns2:tn>
               <ns2:corInfo>
                  <ns2:corClaim>true</ns2:corClaim>
                  <ns2:cor>true</ns2:cor>
                  <ns2:corDate>2010-05-30T09:30:11Z</ns2:corDate>
               </ns2:corInfo>
            </obj>
         </dtlResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.6. Add LRN

If another entity that SSP2 shares the routes with has access to Number Portability data, it may choose to perform route lookups by routing number. Therefore, SSP2 associates a routing number to a destination group in order to facilitate ingress route discovery.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:RNType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
                        <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
                        <urn1:rn>2025550000</urn1:rn>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response to the SPPP client.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.7. Add TN Range

Next, SSP2 activates a block of ten thousand TNs and associate it to a destination group.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:TNRType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
            <urn1:range>
               <urn1:startTn>+12026660000</urn1:startTn>
               <urn1:endTn>+12026669999</urn1:endTn>
            </urn1:range>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.8. Add TN Prefix

Next, SSP2 activates a block of ten thousand TNs using the TNPType structure and identifying a TN prefix.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:TNPType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
                        <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
                        <urn1:tnPrefix>+1202777</urn1:tnPrefix>
         </obj>
      </urn:spppAddRequest>
```

```
      </soapenv:Body>
   </soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
   <?xml version="1.0" encoding="UTF-8"?>
   <S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns3:spppAddResponse
         xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
         xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
            <clientTransId>txn_1479</clientTransId>
            <serverTransId>tx_12345</serverTransId>
            <overallResult>
               <code>1000</code>
               <msg>Request Succeeded.</msg>
            </overallResult>
         </ns3:spppAddResponse>
      </S:Body>
   </S:Envelope>
```

9.9. Enable Peering -- Route Group Offer

In order for SSP1 to complete session establishment for a destination TN where the target subscriber has a retail relationship with SSP2, it first requires an asynchronous bi-directional handshake to show mutual consent. To start the process, SSP2 initiates the peering handshake by offering SSP1 access to its route group.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:RteGrpOfferType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:rteGrpOfferKey xsi:type="urn:RteGrpOfferKeyType">
               <rteGrpKey xsi:type="urn:ObjKeyType">
                  <rant>iana-en:222</rant>
                  <name>RTE_GRP_SSP2_1</name>
                              <type>RteGrp</type>
               </rteGrpKey>
               <offeredTo>iana-en:111</offeredTo>
            </urn1:rteGrpOfferKey>
            <urn1:status>offered</urn1:status>
            <urn1:offerDateTime>
              2006-05-04T18:13:51.0Z
            </urn1:offerDateTime>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and confirms that the SSP1 will now have the opportunity to weigh in on the offer and either accept or reject it. The registry may employ out-of-band notification mechanisms for quicker updates to SSP1 so they can act faster, though this topic is beyond the scope of this document.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.10. Enable Peering -- Route Group Offer Accept

SSP1 responds to the offer from SSP2 and agrees to have visibility to SSP2 ingress routes.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAcceptRequest>
         <!--Optional:-->
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <rteGrpOfferKey>
            <rteGrpKey>
               <rant>iana-en:222</rant>
               <name>RTE_GRP_SSP2_1</name>
               <type>RteGrp</type>
            </rteGrpKey>
            <offeredTo>iana-en:111</offeredTo>
         </rteGrpOfferKey>
      </urn:spppAcceptRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry confirms that the request has been processed successfully. From this point forward, if SSP1 looks up a public identity through the query resolution server, where the public identity is part of the destination group by way of "RTE_GRP_SSP2_1" route association, SSP2 ingress SBE information will be shared with SSP1.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAcceptResponse
        xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12350</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAcceptResponse>
   </S:Body>
</S:Envelope>
```

9.11. Add Egress Route

SSP1 wants to prioritize all outbound traffic to routes associated with "RTE_GRP_SSP2_1" route group through "sbe1.ssp1.example.com".

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppAddRequest>
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <obj xsi:type="urn1:EgrRteType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:egrRteName>EGR_RTE_01</urn1:egrRteName>
            <urn1:pref>50</urn1:pref>
            <urn1:regxRewriteRule>
               <urn1:ere>^(.*@)(.*)$</urn1:ere>
               <urn1:repl>\1\2?route=sbe1.ssp1.example.com</urn1:repl>
            </urn1:regxRewriteRule>
            <urn1:ingrRteRec xsi:type="urn:ObjKeyType">
               <rant>iana-en:222</rant>
               <name>SSP2_RTE_REC_3</name>
                           <type>RteRec</type>
            </urn1:ingrRteRec>
         </obj>
      </urn:spppAddRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Since peering has already been established, the request to add the egress route has been successfully completed.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppAddResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12345</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppAddResponse>
   </S:Body>
</S:Envelope>
```

9.12. Remove Peering -- Route Group Offer Reject

SSP1 had earlier accepted to have visibility to SSP2 ingress routes. SSP1 now decides to no more maintain this visiblity and hence rejects the Route Group Offer.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppRejectRequest>
         <!--Optional:-->
         <clientTransId>txn_1479</clientTransId>
         <!--1 or more repetitions:-->
         <rteGrpOfferKey>
            <rteGrpKey>
               <rant>iana-en:222</rant>
               <name>RTE_GRP_SSP2_1</name>
               <type>RteGrp</type>
            </rteGrpKey>
            <offeredTo>iana-en:111</offeredTo>
         </rteGrpOfferKey>
      </urn:spppRejectRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry confirms that the request has been processed successfully. From this point forward, if SSP1 looks up a public identity through the query resolution server, where the public identity is part of the destination group by way of "RTE_GRP_SSP2_1" route association, SSP2 ingress SBE information will NOT be shared with SSP1 and hence SSP2 ingress SBE will NOT be returned in the query response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppRejectResponse
      xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <clientTransId>txn_1479</clientTransId>
         <serverTransId>tx_12350</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppRejectResponse>
   </S:Body>
</S:Envelope>
```

9.13. Get Destination Group

SSP2 uses the ’spppGetRequest’ operation to tally the last provisioned record for destination group DEST_GRP_SSP2_1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppGetRequest>
         <!--1 or more repetitions:-->
         <objKey xsi:type="urn:ObjKeyType">
                <rant>iana-en:222</rant>
                <name>DEST_GRP_SSP2_1</name>
                        <type>DestGrp</type>
                </objKey>
      </urn:spppGetRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <S:Body>
      <ns3:spppGetResponse
                xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <overallResult>
            <code>1000</code>
            <msg>success</msg>
         </overallResult>
         <resultObj xsi:type="ns2:DestGrpType">
            <ns2:rant>iana-en:222</ns2:rant>
            <ns2:rar>iana-en:223</ns2:rar>
            <ns2:dgName>DEST_GRP_SSP2_1</ns2:dgName>
         </resultObj>
      </ns3:spppGetResponse>
   </S:Body>
</S:Envelope>
```

9.14. Get Public Identity

SSP2 obtains the last provisioned record associated with a given TN.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppGetRequest>
         <!--1 or more repetitions:-->
        <objKey xsi:type="urn:PubIdKeyType">
            <rant>iana-en:222</rant>
                        <number>
                                <urn1:value>+12025556666</urn1:value>
                                <urn1:type>TN</urn1:type>
                        </number>
                </objKey>
      </urn:spppGetRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <S:Body>
      <ns3:spppGetResponse
        xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <overallResult>
            <code>1000</code>
            <msg>success</msg>
         </overallResult>
         <resultObj xsi:type="ns2:TNType">
            <ns2:rant>iana-en:222</ns2:rant>
            <ns2:rar>iana-en:223</ns2:rar>
            <ns2:dgName>DEST_GRP_SSP2_1</ns2:dgName>
            <ns2:tn>+12025556666</ns2:tn>
            <ns2:corInfo>
               <ns2:corClaim>true</ns2:corClaim>
               <ns2:cor>true</ns2:cor>
               <ns2:corDate>2010-05-30T09:30:10Z</ns2:corDate>
            </ns2:corInfo>
         </resultObj>
      </ns3:spppGetResponse>
   </S:Body>
</S:Envelope>
```

9.15. Get Route Group Request

SSP2 obtains the last provisioned record for the route group RTE_GRP_SSP2_1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppGetRequest>
         <!--1 or more repetitions:-->
         <objKey xsi:type="urn:ObjKeyType">
                <rant>iana-en:222</rant>
                <name>RTE_GRP_SSP2_1</name>
                        <type>RteGrp</type>
                </objKey>
      </urn:spppGetRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <S:Body>
      <ns3:spppGetResponse
       xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
       xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <overallResult>
            <code>1000</code>
            <msg>success</msg>
         </overallResult>
         <resultObj xsi:type="ns2:RteGrpType">
            <ns2:rant>iana-en:222</ns2:rant>
            <ns2:rar>iana-en:223</ns2:rar>
            <ns2:rgName>RTE_GRP_SSP2_1</ns2:rgName>
            <ns2:rrRef>
               <ns2:rrKey xsi:type="ns3:ObjKeyType">
                  <rant>iana-en:222</rant>
                  <name>RTE_SSP2_SBE2</name>
                  <type>RteRec</type>
               </ns2:rrKey>
               <ns2:priority>100</ns2:priority>
            </ns2:rrRef>
            <ns2:rrRef>
               <ns2:rrKey xsi:type="ns3:ObjKeyType">
                  <rant>iana-en:222</rant>
                  <name>RTE_SSP2_SBE4</name>
                  <type>RteRec</type>
               </ns2:rrKey>
               <ns2:priority>101</ns2:priority>
            </ns2:rrRef>
            <ns2:dgName>DEST_GRP_SSP2_1</ns2:dgName>
            <ns2:isInSvc>true</ns2:isInSvc>
            <ns2:priority>10</ns2:priority>
         </resultObj>
      </ns3:spppGetResponse>
   </S:Body>
</S:Envelope>
```

9.16. Get Route Group Offers Request

SSP2 fetches the last provisioned route group offer to the <peeringOrg> SSP1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:getRteGrpOffersRequest>
         <offeredTo>iana-en:111</offeredTo>
      </urn:getRteGrpOffersRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry processes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <S:Body>
      <ns3:spppGetResponse
        xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <overallResult>
            <code>1000</code>
            <msg>success</msg>
         </overallResult>
         <resultObj xsi:type="ns2:RteGrpOfferType">
            <ns2:rant>iana-en:222</ns2:rant>
            <ns2:rar>iana-en:223</ns2:rar>
            <ns2:rteGrpOfferKey
            xsi:type="ns3:RteGrpOfferKeyType">
               <rteGrpKey>
                  <rant>iana-en:222</rant>
                  <name>RTE_GRP_SSP2_1</name>
                  <type>RteGrp</type>
               </rteGrpKey>
               <offeredTo>iana-en:111</offeredTo>
            </ns2:rteGrpOfferKey>
            <ns2:status>offered</ns2:status>
            <ns2:offerDateTime>
             2006-05-04T18:13:51.0Z
            </ns2:offerDateTime>
         </resultObj>
      </ns3:spppGetResponse>
```

```
      </S:Body>
   </S:Envelope>
```

9.17. Get Egress Route

SSP1 wants to verify the last provisioned record for the egress route called EGR_RTE_01.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppGetRequest>
         <!--1 or more repetitions:-->
         <objKey xsi:type="urn:ObjKeyType">
          <rant>iana-en:111</rant>
          <name>EGR_RTE_01</name>
          <type>EgrRte</type>
        </objKey>
      </urn:spppGetRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <S:Body>
      <ns3:spppGetResponse
        xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <overallResult>
            <code>1000</code>
            <msg>success</msg>
         </overallResult>
         <resultObj xsi:type="ns2:EgrRteType">
            <ns2:rant>iana-en:222</ns2:rant>
            <ns2:rar>iana-en:223</ns2:rar>
            <ns2:egrRteName>EGR_RTE_01</ns2:egrRteName>
            <ns2:pref>50</ns2:pref>
            <ns2:regxRewriteRule>
               <ns2:ere>^(.*)$</ns2:ere>
               <ns2:repl>sip:\1@sbe1.ssp1.example.com</ns2:repl>
            </ns2:regxRewriteRule>
            <ns2:ingrRteRec xsi:type="ns3:ObjKeyType">
               <rant>iana-en:222</rant>
               <name>RTE_GRP_SSP2_1</name>
               <type>RteRec</type>
            </ns2:ingrRteRec>
         </resultObj>
      </ns3:spppGetResponse>
   </S:Body>
</S:Envelope>
```

9.18. Delete Destination Group

SSP2 initiates a request to delete the destination group DEST_GRP_SSP2_1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppDelRequest>
         <!--1 or more repetitions:-->
         <objKey xsi:type="urn:ObjKeyType">
                <rant>iana-en:222</rant>
                <name>DEST_GRP_SSP2_1</name>
                        <type>DestGrp</type>
                </objKey>
      </urn:spppDelRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppDelResponse
                xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <serverTransId>tx_12354</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppDelResponse>
   </S:Body>
</S:Envelope>
```

9.19. Delete Public Identity

SSP2 choses to de-activate the TN and remove it from the registry.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppDelRequest>
         <!--1 or more repetitions:-->
        <objKey xsi:type="urn:PubIdKeyType">
            <rant>iana-en:222</rant>
                        <dgName>DEST_GRP_SSP2_1</dgName>
                        <number>
                                <urn1:value>+12025556666</urn1:value>
                                <urn1:type>TN</urn1:type>
                        </number>
                </objKey>
      </urn:spppDelRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppDelResponse
                xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <serverTransId>tx_12354</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppDelResponse>
   </S:Body>
</S:Envelope>
```

9.20. Delete Route Group Request

SSP2 removes the route group called RTE_GRP_SSP2_1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppDelRequest>
         <!--1 or more repetitions:-->
         <objKey xsi:type="urn:ObjKeyType">
                <rant>iana-en:222</rant>
                <name>RTE_GRP_SSP2_1</name>
                        <type>RteGrp</type>
                </objKey>
      </urn:spppDelRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppDelResponse
                xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <serverTransId>tx_12354</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppDelResponse>
   </S:Body>
</S:Envelope>
```

9.21. Delete Route Group Offers Request

SSP2 no longer wants to share route group RTE_GRP_SSP2_1 with SSP1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppDelRequest>
         <!--1 or more repetitions:-->
         <objKey xsi:type="urn:RteGrpOfferKeyType">
            <rteGrpKey>
               <rant>iana-en:222</rant>
               <name>RTE_GRP_SSP2_1</name>
               <type>RteGrp</type>
            </rteGrpKey>
            <offeredTo>iana-en:111</offeredTo>
         </objKey>
      </urn:spppDelRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response. Restoring this resource sharing will require a new route group offer from SSP2 to SSP1 followed by a successful route group accept request from SSP1.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppDelResponse
                xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <serverTransId>tx_12354</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppDelResponse>
```

```
      </S:Body>
   </S:Envelope>
```

9.22. Delete Egress Route

SSP1 decides to remove the egress route with the label EGR_RTE_01.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppDelRequest>
         <!--1 or more repetitions:-->
         <objKey xsi:type="urn:ObjKeyType">
                <rant>iana-en:111</rant>
                <name>EGR_RTE_01</name>
                        <type>EgrRte</type>
                </objKey>
      </urn:spppDelRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppDelResponse
                xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <serverTransId>tx_12354</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppDelResponse>
   </S:Body>
```

```
   </S:Envelope>
```

9.23. Batch Request

Following is an example of how some of the operations mentioned in previous sections MAY be performed by an SPPP client as a batch in one single SOAP based SPPP request.

In the sample request below SSP1 wants to accept a Route Group Offer from SSP3, add a Destination Group, add a NAPTR Route Rec, add a Route Group, add a Route Group Offer, delete a previously provisioned TN type Public Identifier, delete a previously provisioned Route Group, and reject a Route Group Offer from SSP4.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ietf:params:xml:ns:sppp:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppp:base:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:spppBatchRequest>
         <clientTransId>txn_1467</clientTransId>
         <minorVer>1</minorVer>

         <acceptRteGrpOffer>
            <rteGrpKey>
               <rant>iana-en:225</rant>
               <name>RTE_SSP3_SBE1_Offered</name>
               <type>RteGrp</type>
            </rteGrpKey>
            <offeredTo>iana-en:222</offeredTo>
         </acceptRteGrpOffer>

         <addObj xsi:type="urn1:DestGrpType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
         </addObj>

         <addObj xsi:type="urn1:NAPTRType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:rrName>RTE_SSP2_SBE2</urn1:rrName>
```

```
            <urn1:order>10</urn1:order>
            <urn1:flags>u</urn1:flags>
            <urn1:svcs>E2U+sip</urn1:svcs>
            <urn1:regx>
               <urn1:ere>^(.*)$</urn1:ere>
               <urn1:repl>sip:\1@sbe2.ssp2.example.com</urn1:repl>
            </urn1:regx>
         </addObj>

         <addObj xsi:type="urn1:RteGrpType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:rgName>RTE_GRP_SSP2_1</urn1:rgName>
            <urn1:rrRef>
               <urn1:rrKey xsi:type="urn:ObjKeyType">
                  <rant>iana-en:222</rant>
                  <name>RTE_SSP2_SBE2</name>
                  <type>RteRec</type>
               </urn1:rrKey>
               <urn1:priority>100</urn1:priority>
            </urn1:rrRef>
            <urn1:dgName>DEST_GRP_SSP2_1</urn1:dgName>
            <urn1:isInSvc>true</urn1:isInSvc>
            <urn1:priority>10</urn1:priority>
         </addObj>

         <addObj xsi:type="urn1:RteGrpOfferType">
            <urn1:rant>iana-en:222</urn1:rant>
            <urn1:rar>iana-en:223</urn1:rar>
            <urn1:rteGrpOfferKey xsi:type="urn:RteGrpOfferKeyType">
               <rteGrpKey xsi:type="urn:ObjKeyType">
                  <rant>iana-en:222</rant>
                  <name>RTE_GRP_SSP2_1</name>
                  <type>RteGrp</type>
               </rteGrpKey>
               <offeredTo>iana-en:111</offeredTo>
            </urn1:rteGrpOfferKey>
            <urn1:status>offered</urn1:status>
            <urn1:offerDateTime>
             2006-05-04T18:13:51.0Z
            </urn1:offerDateTime>
         </addObj>

         <delObj xsi:type="urn:PubIdKeyType">
            <rant>iana-en:222</rant>
            <dgName>DEST_GRP_SSP2_Previous</dgName>
            <number>
               <urn1:value>+12025556666</urn1:value>
```

```
                  <urn1:type>TN</urn1:type>
               </number>
            </delObj>

            <delObj xsi:type="urn:ObjKeyType">
               <rant>iana-en:222</rant>
               <name>RTE_GRP_SSP2_Previous</name>
               <type>RteGrp</type>
            </delObj>

            <rejectRteGrpOffer>
               <rteGrpKey>
                  <rant>iana-en:226</rant>
                  <name>RTE_SSP4_SBE1_Offered</name>
                  <type>RteGrp</type>
               </rteGrpKey>
               <offeredTo>iana-en:222</offeredTo>
            </rejectRteGrpOffer>

         </urn:spppBatchRequest>
      </soapenv:Body>
   </soapenv:Envelope>
```

Registry completes the request successfully and returns a favorable response.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns3:spppBatchResponse
                xmlns:ns2="urn:ietf:params:xml:ns:sppp:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppp:soap:1">
         <serverTransId>tx_12354</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppBatchResponse>
   </S:Body>
</S:Envelope>
```

10. Security Considerations

SPPP is used to query and update session peering data and addresses, so the ability to access this protocol should be limited to users and systems that are authorized to query and update this data. Because this data is sent in both directions, it may not be sufficient for just the client or user to be authenticated with the server. The identity of the server should also be authenticated by the client, which is often accomplished using the TLS certificate exchange and validation described in [RFC2818]. SPPP data may include sensitive information, routing data, lists of resolvable addresses, etc. So when used in a production setting and across non-secure networks, SPPP should only be used over communications channels that provide strong encryption for data privacy.

10.1. Integrity, Privacy, and Authentication

The SPPP SOAP binding relies on an underlying secure transport for integrity and privacy. Such transports are expected to include TLS/ HTTPS. In addition to the application level authentication imposed by an SPPP server, there are a number of options for authentication within the transport layer and the messaging envelope. These include TLS client certificates, HTTP Digest Access Authentication, and digital signatures within SOAP headers.

At a miniumum, all conforming SPPP over SOAP implementations MUST support HTTPS.

10.2. Vulnerabilities

The above protocols may have various vulnerabilities, and these may be inherited by SPPP over SOAP. And SPPP itself may have vulnerabilities because an authorization model is not explicitly specified in the current specification.

It is important that SPPP implementations implement an authorization model that considers the source of each SPPP query or update request and determines whether it is reasonable to authorize that source to perform that specific query or update.

10.3. Deployment Environment Specifics

Some deployments of SPPP over SOAP could choose to use transports without encryption. This presents vulnerabilities but could be selected for deployments involving closed networks or debugging scenarios. However, the vulnerabilities of such a deployment could be a lack of integrity and privacy of the data transported over SPPP messages in this type of deployment.

11. IANA Considerations

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688].

URN assignments are requested: urn:ietf:params:xml:ns:sppp:soap

12. Acknowledgements

This document is a result of various discussions held by the DRINKS design team, which is comprised of the following individuals, in no specific order: Syed Ali (NeuStar), Sumanth Channabasappa (Cable Labs), David Schwartz (XConnect), Jean-Francois Mule (CableLabs), Kenneth Cartwright (TNS, Inc.), Manjul Maharishi (TNS, Inc.), Alexander Mayrhofer (enum.at GmbH), Vikas Bhatia (TNS, Inc.).

13. References

13.1. Normative References

[I-D.draft-ietf-drinks-spprov]
Mule, J-F., Cartwright, K., Ali, S., Mayrhofer, A., and V. Bhatia, "DRINKS Use cases and Protocol Requirements", draft-ietf-drinks-spprov-12 (work in progress), June 2011.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[SOAPREF] Gudgin, M., Hadley, M., Moreau, J., and H. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation REC-SOAP12-part1-20030624, June 2002.

13.2. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[WSDLREF] Christensen, E., Curbera, F., Meredith, G., and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C Note NOTE-wsdl-20010315, March 2001.

Authors' Addresses

Kenneth Cartwright
TNS
1939 Roland Clarke Place
Reston, VA 20191
USA

Email: kcartwright@tnsi.com

Vikas Bhatia
TNS
1939 Roland Clarke Place
Reston, VA 20191
USA

Email: vbhatia@tnsi.com

DRINKS
Internet-Draft
Intended status: Standards Track
Expires: May 18, 2012

J-F. Mule
CableLabs
K. Cartwright
TNS
S. Ali
NeuStar
A. Mayrhofer
enum.at GmbH
V. Bhatia
TNS

November 15, 2011

Session Peering Provisioning Protocol Data Model
draft-ietf-drinks-spprov-12

Abstract

This document specifies the data model and the overall structure for a protocol to provision session establishment data into Session Data Registries and SIP Service Provider data stores. The protocol is called the Session Peering Provisioning Protocol (SPPP). The provisioned data is typically used by network elements for session peering.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on May 18, 2012.

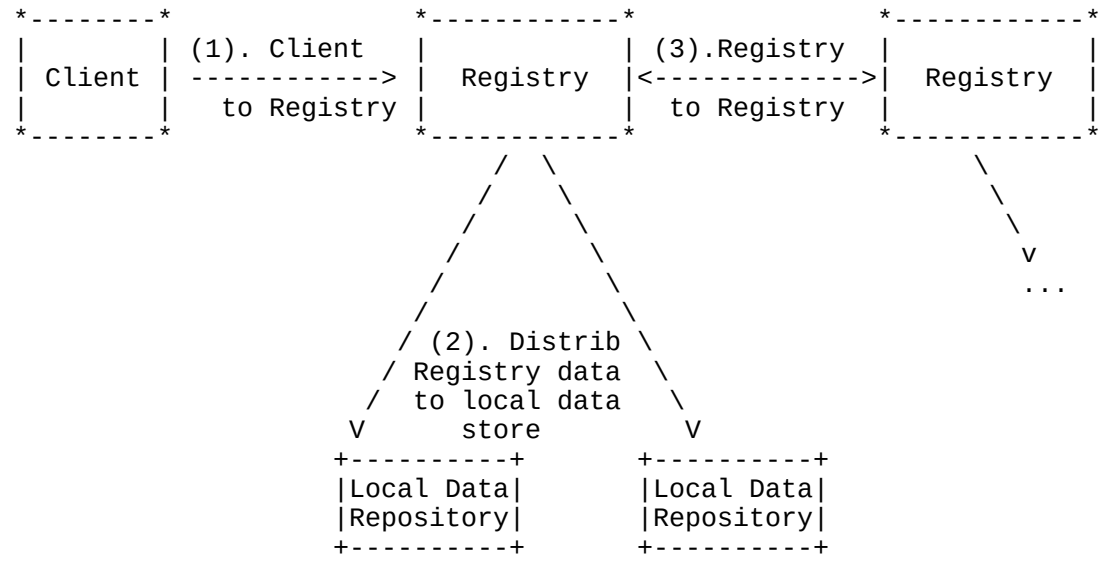
Table of Contents

1. Introduction

Service providers and enterprises use registries to make session routing decisions for Voice over IP, SMS and MMS traffic exchanges. This document is narrowly focused on the provisioning protocol for these registries. This protocol prescribes a way for an entity to provision session-related data into a registry. The data being provisioned can be optionally shared with other participating peering entities. The requirements and use cases driving this protocol have been documented in [I-D.ietf-drinks-usecases-requirements]. The reader is expected to be familiar with the terminology defined in the previously mentioned document.

Three types of provisioning flows have been described in the use case document: client to registry provisioning, registry to local data repository and registry to registry. This document addresses client to registry aspect to fulfill the need to provision Session Establishment Data (SED). The protocol that supports flow of messages to facilitate client to registry provisioning is referred to as Session Peering Provisioning Protocol (SPPP).

Please note that the role of the "client" and the "server" only applies to the connection, and those roles are not related in any way to the type of entity that participates in a protocol exchange. For example, a registry might also include a "client" when such a registry initiates a connection (for example, for data distribution to SSP).



Three Registry Provisioning Flows

Figure 1

The data provisioned for session establishment is typically used by various downstream SIP signaling systems to route a call to the next hop associated with the called domain. These systems typically use a local data store ("Local Data Repository") as their source of session routing information. More specifically, the SED data is the set of parameters that the outgoing signaling path border elements (SBEs) need to initiate the session. See [RFC5486] for more details.

A "terminating" SIP Service Provider (SSP) provisions SED into the registry to be selectively shared with other peer SSPs. Subsequently, a registry may distribute the provisioned data into local data repositories used for look-up queries (identifier -> URI) or for lookup and location resolution (identifier -> URI -> ingress SBE of terminating SSP). In some cases, the registry may additionally offer a central query resolution service (not shown in the above figure).

A key requirement for the SPPP protocol is to be able to accommodate two basic deployment scenarios:

1. A resolution system returns a Look-Up Function (LUF) that comprises of the target domain to assist in call routing (as described in [RFC5486]). In this case, the querying entity may use other means to perform the Location Routing Function (LRF)

which in turn helps determine the actual location of the Signaling Function in that domain.

2. A resolution system returns both a Look-Up function (LUF) and Location Routing Function (LRF) to locate the SED data fully.

In terms of protocol design, SPPP is agnostic to the transport. This document includes the specification of the data model and identifies, but does not specify, the means to enable protocol operations within a request and response structure. That aspect of the specification has been delegated to the "transport" specification for the protocol. To encourage interoperability, the protocol supports extensibility aspects.

Transport requirements are provided in this document to help with the selection of the optimum transport mechanism. ([I-D.ietf-drinks-sppp-over-soap]) identifies a SOAP transport mechanism for SPPP.

This document is organized as follows:

- Section 2 provides the terminology;
- Section 3 provides an overview of SPPP, including the functional entities and data model;
- Section 4 specifies requirements for SPPP transport protocols;
- Section 5 describes the base protocol data structures, the generic response types that MUST be supported by a conforming "transport" specification, and the basic object type most first class objects extend from;
- Section 6 detailed descriptoins of the data model object specifications;
- Section 8 defines XML considerations that XML parsers must meet to conform to this specification;
- Section 11 normatively defines the SPPP protocol using its XML Schema Definition.

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 reuses terms from [RFC3261], [RFC5486], use cases and requirements documented in [I-D.ietf-drinks-usecases-requirements] and the ENUM Validation Architecture [RFC4725].

In addition, this document specifies the following additional terms:

SPPP: Session Peering Provisioning Protocol, the protocol used to provision data into a Registry (see arrow labeled "1." in Figure 1 of [I-D.ietf-drinks-usecases-requirements]). It is the primary scope of this document.

SPDP: Session Peering Distribution Protocol, the protocol used to distribute data to Local Data Repository (see arrow labeled "2." in Figure 1 of [I-D.ietf-drinks-usecases-requirements]).

Client: An application that supports an SPPP client; it is sometimes referred to as a "registry client".

Registry: The Registry operates a master database of Session Establishment Data for one or more Registrants.

A Registry acts as an SPPP server.

Registrant: In this document we extend the definition of a Registrant based on [RFC4725]. The Registrant is the end-user, the person or organization that is the "holder" of the Session Establishment Data being provisioned into the Registry by a Registrar. For example, in [I-D.ietf-drinks-usecases-requirements], a Registrant is pictured as a SIP Service Provider in Figure 2.

Within the confines of a Registry, a Registrant is uniquely identified by a well-known ID.

Registrar: In this document we extend the definition of a Registrar from [RFC4725]. A Registrar is an entity that performs provisioning operations on behalf of a Registrant by interacting with the Registry via SPPP operations. In other words the Registrar is the SPPP Client. The Registrar and Registrant roles are logically separate to allow, but not require, a single Registrar to perform provisioning operations on behalf of more than one Registrant.

Peering Organization: A Peering Organization is an entity to which a Registrant’s Route Groups are made visible using the operations of SPPP.

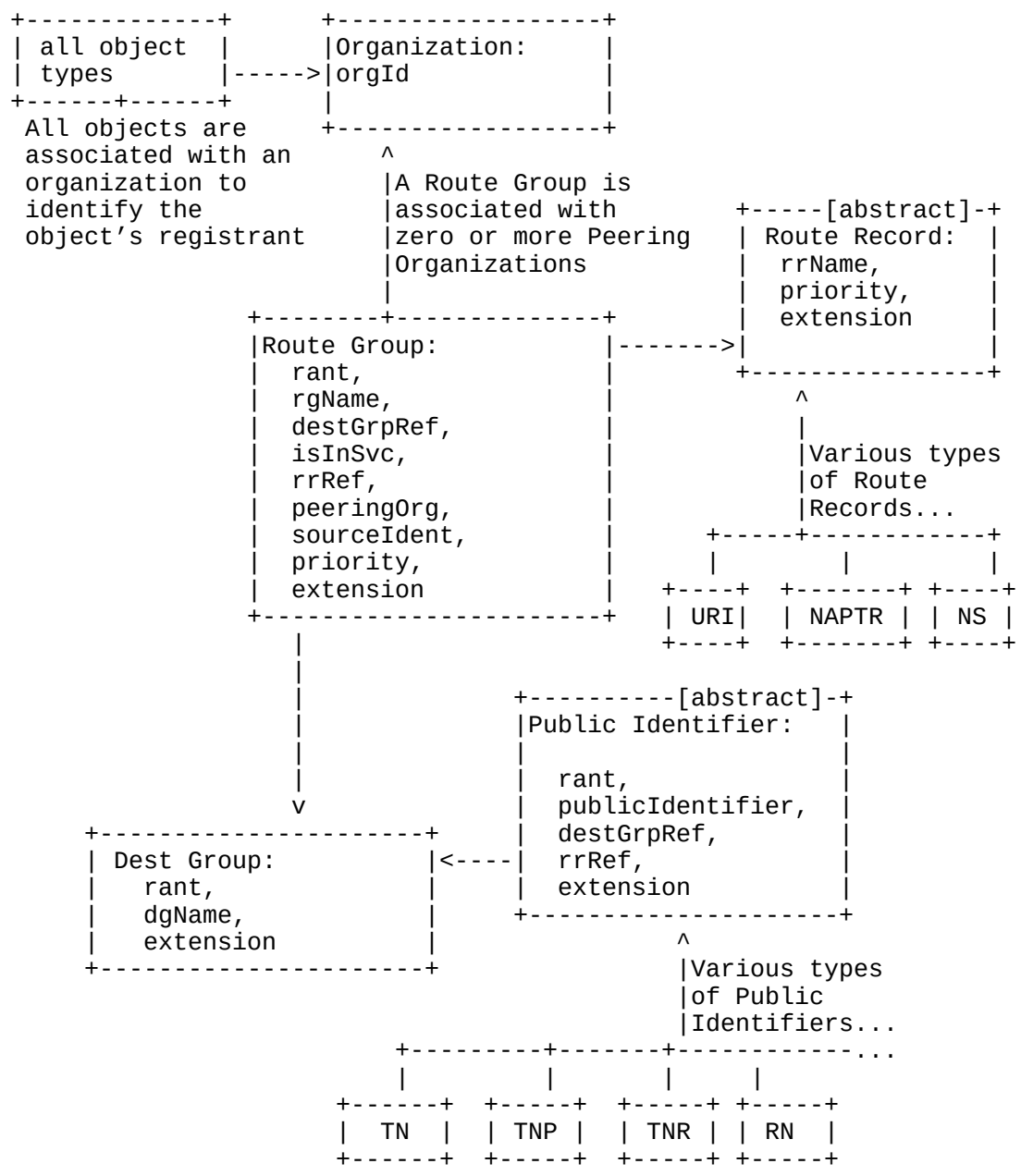
3. Protocol High Level Design

This section introduces the structure of the data model and provides the information framework for the SPPP. An overview of the protocol operations is first provided with a typical deployment scenario. The data model is then defined along with all the objects manipulated by the protocol and their relationships.

3.1. Protocol Data Model

The data model illustrated and described in Figure 2 defines the logical objects and the relationships between these objects that the SPPP protocol supports. SPPP defines the protocol operations through which an SPPP client populates a registry with these logical objects. Various clients belonging to different registrars may use the protocol for populating the registry’s data.

The logical structure presented below is consistent with the terminology and requirements defined in [I-D.ietf-drinks-usecases-requirements].



SPPP Data Model

Figure 2

The objects and attributes that comprise the data model can be described as follows (objects listed from the bottom up):

- Public Identifier:
  From a broad perspective a public identifier is a well-known attribute that is used as the key to perform resolution lookups. Within the context of SPPP, a public identifier object can be a telephone number, a range of telephone numbers, a PSTN Routing Number (RN), or a TN prefix.

  An SPPP Public Identifier is associated with a Destination Group to create a logical grouping of Public Identifiers that share a common set of Routes.

  A TN Public Identifier may optionally be associated with zero or more individual Route Records. This ability for a Public Identifier to be directly associated with a set of Route Records (e.g. target URI), as opposed to being associated with a Destination Group, supports the use cases where the target URI contains data specifically tailored to an individual TN Public Identifier.

- Destination Group:
  A named collection of zero or more Public Identifiers that can be associated with one or more Route Groups for the purpose of facilitating the management of their common routing information.

- Route Group:
  A Route Group contains a set of Route Record references, a set of Destination Group references, and a set of peering organization identifiers. This is used to establish a three part relationships between a set of Public Identifiers, the routing information (SED) shared across the Public Identifiers, and the list of peering organizations whose query responses from the resolution system may include the routing information from a given route group. In addition, the sourceIdent element within a Route Group, in concert with the set of peering organization identifiers, enables fine-grained source based routing. For further details about the Route Group and source based routing, refer to the definitions and descriptions of the Route Group operations found later in this document.

- Route Record:
  A Route Record contains the data that a resolution system returns in response to a successful query for a Public Identifier. Route Records are generally associated with a Route Group when the SED within is not specific to a Public Identifier.
  To support the use cases defined in

[I-D.ietf-drinks-usecases-requirements], SPPP defines three type of Route Records: URIType, NAPTRType, and NSType. These Route Records extend the abstract type RteRecType and inherit the common attribute 'priority' that is meant for setting precedence across the route records defined within a Route Group in a protocol agnostic fashion.

- o Organization:
  An An Organization is an entity that may fulfill any combination of three roles: Registrant, Registrar, and Peering Organization. All SPPP objects are associated with two organization identifiers to identify each object's registrant and registrar. A Route Group object is also associated with a set of zero or more organization identifiers that identify the peering organization(s) whose resolution query responses may include the routing information (SED) defined in the Route Records within that Route Group. A peering organization is an entity that the registrant intends to share the SED data with.

3.2. Time Value

Some SPPP request and response messages include time value(s) defined as type xs:dateTime, a built-in W3C XML Schema Datatype. Use of unqualified local time value is discouraged as it can lead to interoperability issues. The value of time attribute MUST BE expressed in Coordinated Universal Time (UTC) format without the timezone digits.

"2010-05-30T09:30:10Z" is an example of an acceptable time value for use in SPPP messages. "2010-05-30T06:30:10+3:00" is a valid UTC time, but it is not approved for use in SPPP messages.

4. Transport Protocol Requirements

This section provides requirements for transport protocols suitable for SPPP. More specifically, this section specifies the services, features, and assumptions that SPPP delegates to the chosen transport and envelope technologies.

4.1. Connection Oriented

The SPPP follows a model where a client establishes a connection to a server in order to further exchange SPPP messages over such point-to-point connection. A transport protocol for SPPP MUST therefore be connection oriented.

4.2. Request and Response Model

Provisioning operations in SPPP follow the request-response model, where a client sends a request message to initiate a transaction and the server responds with a response. Multiple subsequent request-response exchanges MAY be performed over a single persistent connection.

Therefore, a transport protocol for SPPP MUST follow the request-response model by allowing a response to be sent to the request initiator.

4.3. Connection Lifetime

Some use cases involve provisioning a single request to a network element. Connections supporting such provisioning requests might be short-lived, and may be established only on demand. Other use cases involve either provisioning a large dataset, or a constant stream of small updates, either of which would likely require long-lived connections.

Therefore, a protocol suitable for SPPP SHOULD be able to support both short-lived as well as long-lived connections.

4.4. Authentication

All SPPP objects are associated with a registrant identifier. SPPP Clients provisions SPPP objects on behalf of registrants. An authenticated SPP Client is a registrar. Therefore, the SPPP transport protocol MUST provide means for an SPPP server to authenticate an SPPP Client.

4.5. Authorization

After successful authentication of the SPPP client as a registrar the registry performs authorization checks to determine if the registrar is authorized to act on behalf of the Registrant whose identifier is included in the SPPP request. Refer to the Security Considerations section for further guidance.

4.6. Confidentiality and Integrity

In some deployments, the SPPP objects that an SPPP registry manages can be private in nature. As a result it MAY NOT be appropriate to for transmission in plain text over a connection to the SPPP registry. Therefore, the transport protocol SHOULD provide means for end-to-end encryption between the SPPP client and server.

For some SPPP implementations, it may be acceptable for the data to be transmitted in plain text, but the failure to detect a change in data after it leaves the SPPP client and before it is received at the server, either by accident or with a malicious intent, will adversely affect the stability and integrity of the registry. Therefore, the transport protocol SHOULD provide means for data integrity protection.

4.7. Near Real Time

Many use cases require near real-time responses from the server. Therefore, a DRINKS transport protocol MUST support near real-time response to requests submitted by the client.

4.8. Request and Response Sizes

Use of SPPP may involve simple updates that may consist of small number of bytes, such as, update of a single public identifier. Other provisioning operations may constitute large number of datasets as in adding millions records to a registry. As a result, a suitable transport protocol for SPPP SHOULD accommodate datasets of various sizes.

4.9. Request and Response Correlation

A transport protocol suitable for SPPP MUST allow responses to be correlated with requests.

4.10. Request Acknowledgement

Data transported in the SPPP is likely crucial for the operation of the communication network that is being provisioned. A SPPP client

responsible for provisioning SED to the registry has a need to know if the submitted requests have been processed correctly.

Failed transactions can lead to situations where a subset of public identifiers or even SSPs might not be reachable, or the provisioning state of the network is inconsistent.

Therefore, a transport protocol for SPPP MUST provide a response for each request, so that a client can identify whether a request succeeded or failed.

4.11. Mandatory Transport

At the time of this writing, a choice of transport protocol has been provided in [I-D.ietf-drinks-sppp-over-soap]. To encourage interoperability, the SPPP server MUST provide support for this transport protocol. With time, it is possible that other transport layer choices may surface that agree with the requirements discussed above.

5. Base Protocol Data Structures and Response Codes

SPPP contains some common data structures for most of the supported object types. This section describes these common data structures.

5.1. Basic Object Type and Organization Identifiers

This section introduces the basic object type that most first class objects derive from.

All first class objects extend the basic object type BasicObjType that contains the identifier of the registrant organization that owns this object, the identifier of the registrar organization that created this object, the date and time that the object was created by the server, and the date and time that the object was last modified.

```
<complexType name="BasicObjType" abstract="true">
        <sequence>
                <element name="rant" type="spppb:OrgIdType"/>
                <element name="rar" type="spppb:OrgIdType"/>
                <element name="cDate" type="dateTime"
                   minOccurs="0"/>
                <element name="mDate" type="dateTime"
                   minOccurs="0"/>
                <element name="ext" type="spppb:ExtAnyType"
                   minOccurs="0"/>
        </sequence>
</complexType>
```

The identifiers used for registrants (rant), registrars (rar), and peering organizations (peeringOrg) are instances of OrgIdType. The OrgIdType is defined as a string and all OrgIdType instances SHOULD follow the textual convention: "namespace:value" (for example "iana-en:32473"). See the IANA Consideration section for more details.

5.2. Various Object Key Types

5.2.1. Generic Object Key Type

The SPPP data model contains some object relationships. In some cases these object relationships are established by embedding the unique identity of the related object inside the relating object. In addition, an object’s unique identity is required to Delete or Get the details of an object. The abstract type called ObjKeyType is where this unique identity is housed. Because this object key type is abstract, it MUST be specified in a concrete form in any conforming SPPP "transport" specification.

Most objects in SPPP are uniquely identified by an object key that has the object's name, object's type and its registrant's organization ID as its attributes. Consequently, any concrete representation of the ObjKeyType MUST contain the following:

Object Name: The name of the object.

Registrant Id: The unique organization ID that identifies the Registrant.

Type: The enumeration vaue that represents the type of SPPP object that. This is required as different types of objects in SPPP, that belong to the same registrant, can have the same name.

The structure of abstract ObjKeyType is as follows:

```
<complexType name="ObjKeyType" abstract="true">
        <annotation>
                <documentation>
                        ---- Generic type that represents the
                        key for various objects in SPPP. ----
                </documentation>
        </annotation>
</complexType>
```

The object types in SPPP that MUST adhere to this definition of generic object key are defined as an enumeration in the XML data structure. The structure of the the enumeration is as follows:

```
 <simpleType name="ObjKeyTypeEnum">
   <restriction base="token">
     <enumeration value="RteGrp"/>
     <enumeration value="DestGrp"/>
     <enumeration value="RteRec"/>
     <enumeration value="EgrRte"/>
   </restriction>
 </simpleType>
```

5.2.2. Derived Object Key Types

The SPPP data model contains certain objects that are uniquely identified by attributes, different from or in addition to, the attributes in the generic object key described in previous section. These kind of object keys are derived from the abstract ObjKeyType and defined in there own abstract key types. Because these object key types are abstract, these MUST be specified in a concrete form in any conforming SPPP "transport" specification. These are used in

Delete and Get operations, and may also be used in Accept and Reject operations.

Following are the derived object keys in SPPP data model:

o RteGrpOfferKeyType: This uniquely identifies a Route Group object offer. This key type extends from ObjKeyType and MUST also have the organization ID of the Registrant to whom the object is being offered, as one of its attributes. In addition to the Delete and Get operations, these key types are used in Accept and Reject operations on a Route Group Offer object. The structure of abstract RteGrpOfferKeyType is as follows:

```
<complexType name="RteGrpOfferKeyType"
abstract="true">
        <complexContent>
                <extension base="spppb:ObjKeyType">
                        <annotation>
                                <documentation>
                                ---- Generic type that represents the
                                key for a object offer. ----
                                </documentation>
                        </annotation>
                </extension>
        </complexContent>
</complexType>
```

A Route Group Offer object MUST use RteGrpOfferKeyType. Refer the "Protocol Data Model Objects" section of this document for description of Route Group Offer object.

o PubIdKeyType: This uniquely identifies a Public Identity object. This key type extends from abstract ObjKeyType. Any concrete defintion of PubIdKeyType MUST contain the elements that identify the value and type of Public Identity and also contain the organization ID of the Registrant that is the owner of the Public Identity object. A Public Identity object key in SPPP is uniquely identified by the the registrant’s organization ID, the value of the public identity, and, optionally, the Destination Group name the public identiy belongs to. Consequently, any concrete representation of the ObjKeyType MUST contain the following attributes:

  * Registrant Id: The unique organization ID that identifies the Registrant.

  * Destination Group name: The name of the Destination Group the Public Identity is associated with. This is an

optional attribute.

* Type: The type of Public Identity.

* Value: The value of the Public Identity.

The .PubIdKeyType is used in Delete and Get operations on a Public Identifier object.

o The structure of abstract PubIdKeyType is as follows:

```
<complexType name="PubIdKeyType" abstract="true">
        <complexContent>
                <extension base="spppb:ObjKeyType">
                        <annotation>
                                <documentation>
---- Generic type that represents
the key for a Pub Id. ----
                                </documentation>
                        </annotation>
                </extension>
        </complexContent>
</complexType>
```

A Public Identity object MUST use attributes of PubIdKeyType for its unique identification . Refer the "Protocol Data Model Objects" section of this document for a description of Public Identity object.

5.3. Response Message Types

This section contains the listing of response types that MUST be defined by the conforming "transport" specification and implemented by a conforming SPPP server.

Response Type	Description
Request Succeeded	Any conforming specification MUST define a response to indicate that a given request succeeded.
Request syntax invalid	Any conforming specification MUST define a response to indicate that a syntax of a given request was found invalid.
Request too large	Any conforming specification MUST define a response to indicate that the count of entities in the request is larger than the server is willing or able to process.
Version not supported	Any conforming specification MUST define a response to indicate that the server does not support the version of the SPPP protocol specified in the request.
Command invalid	Any conforming specification MUST define a response to indicate that the operation and/or command being requested by the client is invalid and/or not supported by the server.
System temporarily unavailable	Any conforming specification MUST define a response to indicate that the SPPP server is temporarily not available to serve client request.
Unexpected internal system or server error.	Any conforming specification MUST define a response to indicate that the SPPP server encountered an unexpected error that prevented the server from fulfilling the request.
Attribute value invalid	Any conforming specification MUST define a response to indicate that the SPPP server encountered an attribute or property in the request that had an invalid/bad value. Optionally, the specification MAY provide a way to indicate the Attribute Name and the Attribute Value to identify the object that was found to be invalid.

Object does not exist	Any conforming specification MUST define a response to indicate that an object present in the request does not exist on the SPPP server. Optionally, the specification MAY provide a way to indicate the Attribute Name and the Attribute Value that identifies the non-existent object.
Object status or ownership does not allow for operation.	Any conforming specification MUST define a response to indicate that the operation requested on an object present in the request cannot be performed because the object is in a status that does not allow the said operation or the user requesting the operation is not authorized to perform the said operation on the object. Optionally, the specification MAY provide a way to indicate the Attribute Name and the Attribute Value that identifies the object.

Table 1: Response Types

When the response messages are "parameterized" with the Attribute Name and Attribute Value, then the use of these parameters MUST adhere to the following rules:

- Any value provided for the Attribute Name parameter MUST be an exact XSD element name of the protocol data element that the response message is referring to. For example, valid values for "attribute name" are "dgName", "rgName", "rteRec", etc.

- The value for Attribute Value MUST be the value of the data element to which the preceding Attribute Name refers.

- Response type "Attribute value invalid" SHOULD be used whenever an element value does not adhere to data validation rules.

- Response types "Attribute value invalid" and "Object does not exist" MUST NOT be used interchangeably. Response type "Object does not exist" SHOULD be returned by an Add/Del/Accept/Reject operation 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 type "Attribute value invalid" SHOULD be returned.

6. Protocol Data Model Objects

This section provides a description of the specification of each supported data model object (the nouns) and identifies the commands (the verbs) that MUST be supported for each data model object. However, the specification of the data structures necessary to support each command is delegated to the "transport" specification.

6.1. Destination Group

As described in the introductory sections, a Destination Group represents a set of Public Identifiers with common routing information. The transport protocol MUST support the ability to Create, Modify, Get, and Delete Destination Groups (refer the "Protocol Operations" section of this document for a generic description of various operations).

A Destination Group object MUST be uniquely identified by attributes as defined in the description of "ObjKeyType" in the section "Generic Object Key Type" of this document.

The DestGrpType object structure is defined as follows:

```
<complexType name="DestGrpType">
  <complexContent>
   <extension base="spppb:BasicObjType">
    <sequence>
       <element name="dgName" type="spppb:ObjNameType"/>
    </sequence>
   </extension>
  </complexContent>
</complexType>
```

The DestGrpType object is composed of the following elements:

- base: All first class objects extend BasicObjType that contains the ID of the registrant organization that owns this object, registrar organization that provisioned this object on behalf of the registrant, the date and time that the object was created by the server, and the date and time that the object was last modified. If the client passed in either the created date or the modification date, the server will ignore them. The server sets these two date/time values.

- dgName: The character string that contains the name of the Destination Group.

- ext: Point of extensibility described in a previous section of this document.

6.2. Public Identifier

A Public Identifier is the search key used for locating the session establishment data (SED). In many cases, a Public Identifier is attributed to the end user who has a retail relationship with the service provider or registrant organization. SPPP supports the notion of the carrier-of-record as defined in [RFC5067]. Therefore, the registrant under whom the Public Identity is being created can optionally claim to be a carrier-of-record.

SPPP identifies two types of Public Identifiers: telephone numbers (TN), and the routing numbers (RN). SPPP provides structures to manage a single TN, a contiguous range of TNs, and a TN prefix. The transport protocol MUST support the ability to Create, Modify, Get, and Delete Public Identifiers (refer the "Protocol Operations" section of this document for a generic description of various operations).

A Public Identity object MUST be uniquely identified by attributes as defined in the description of "PubIdKeyType" in the section "Derived Object Key Types" of this document.

The abstract XML schema type definition PubIDType is a generalization for the concrete the Public Identifier schema types. PubIDType element 'dgName' represents the name of the destination group that a given Public Identifier MAY be a member of. The PubIDType object structure is defined as follows:

```
<complexType name="PubIdType" abstract="true">
  <complexContent>
   <extension base="spppb:BasicObjType">
     <sequence>
       <element name="dgName" type="spppb:ObjNameType"
          minOccurs="0"/>
     </sequence>
   </extension>
  </complexContent>
</complexType>
```

A Public Identifier may be provisioned as a member of a Destination Group or provisioned outside of a Destination Group. A Public Identifier that is provisioned as a member of a Destination Group is intended to be associated with its SED through the Route Group(s) that are associated with its containing Destination Group. A Public

Identifier that is not provisioned as a member of a Destination Group is intended to be associated with its SED through the Route Records that are directly associated with the Public Identifier.

A telephone number is provisioned using the TNType, an extension of PubIDType. When a Public Identifier is provisioned as a member of a Destination Group, each TNType object is uniquely identified by the combination of its value contained within <tn> element, and the unique key of its parent Destination Group (dgName and rantId). In other words a given telephone number string may exist within one or more Destination Groups, but must not exist more than once within a Destination Group. A Public Identifier that is not provisioned as a member of a Destination Group is uniquely identified by the combination of its value, and its registrant ID. TNType is defined as follows:

```
 <complexType name="TNType">
  <complexContent>
   <extension base="spppb:PubIdType">
    <sequence>
     <element name="tn"
     type="spppb:NumberValType"/>
     <element name="corInfo"
     type="spppb:CORInfoType" minOccurs="0"/>
     <element name="rrRef"
     type="spppb:RteRecRefType"
     minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>

<simpleType name="NumberValType">
        <restriction base="token">
                <maxLength value="20"/>
                <pattern value="\+?\d\d*"/>
        </restriction>
</simpleType>
```

TNType consists of the following attributes:

o tn: Telephone number to be added to the registry.

o rrRef: Optional reference to route records that are directly associated with the TN Public Identifier. Following the SPPP data model, the route record could be a protocol agnostic

URIType or another type.

- corInfo: corInfo is an optional parameter of type CORInfoType that allows the registrant organization to set forth a claim to be the carrier-of-record (see [RFC5067]). This is done by setting the value of <corClaim> element of the CORInfoType object structure to "true". The other two parameters of the CORInfoType, <cor> and <corDate> are set by the registry to describe the outcome of the carrier-of-record claim by the registrant. In general, inclusion of <corInfo> parameter is useful if the registry has the authority information, such as, the number portability data, etc., in order to qualify whether the registrant claim can be satisfied. If the carrier-of-record claim disagrees with the authority data in the registry, whether the TN add operation fails or not is a matter of policy and it is beyond the scope of this document.

A routing number is provisioned using the RNType, an extension of PubIDType. SSPs that possess the number portability data may be able to leverage the RN search key to discover the ingress routes for session establishment. Therefore, the registrant organization can add the RN and associate it with the appropriate destination group to share the route information. Each RNType object is uniquely identified by the combination of its value inside the <rn> element, and the unique key of its parent Destination Group (dgName and rantId). In other words a given routing number string may exist within one or more Destination Groups, but must not exist more than once within a Destination Group. RNType is defined as follows:

```
 <complexType name="RNType">
  <complexContent>
   <extension base="spppb:PubIdType">
    <sequence>
     <element name="rn"
     type="spppb:NumberValType"/>
     <element name="corInfo"
     type="spppb:CORInfoType" minOccurs="0"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

RNType has the following attributes:

- rn: Routing Number used as the search key.

o corInfo: Optional <corInfo> element of type CORInfoType.

TNRType structure is used to provision a contiguous range of telephone numbers. The object definition requires a starting TN and an ending TN that together define the span of the TN range. Use of TNRType is particularly useful when expressing a TN range that does not include all the TNs within a TN block or prefix. The TNRType definition accommodates the open number plan as well such that the TNs that fall between the start and end TN range may include TNs with different length variance. Whether the registry can accommodate the open number plan semantics is a matter of policy and is beyond the scope of this document. Each TNRType object is uniquely identified by the combination of its value that in turn is a combination of the <startTn> and <endTn> elements, and the unique key of its parent Destination Group (dgName and rantId). In other words a given TN Range may exist within one or more Destination Groups, but must not exist more than once within a Destination Group. TNRType object structure definition is as follows:

```
 <complexType name="TNRType">
  <complexContent>
   <extension base="spppb:PubIdType">
    <sequence>
     <element name="range" type="spppb:NumberRangeType"/>
     <element name="corInfo" type="spppb:CORInfoType" minOccurs="0"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>

  <complexType name="NumberRangeType">
   <sequence>
    <element name="startTn" type="spppb:NumberValType"/>
    <element name="endTn" type="spppb:NumberValType"/>
   </sequence>
  </complexType>
```

TNRType has the following attributes:

o startTn: Starting TN in the TN range

o endTn: The last TN in the TN range

o corInfo: Optional <corInfo> element of type CORInfoType

In some cases, it is useful to describe a set of TNs with the help of the first few digits of the telephone number, also referred to as the

telephone number prefix or a block. A given TN prefix may include TNs with different length variance in support of open number plan. Once again, whether the registry supports the open number plan semantics is a matter of policy and it is beyond the scope of this document. The TNPType data structure is used to provision a TN prefix. Each TNPType object is uniquely identified by the combination of its value in the <tnPrefix> element, and the unique key of its parent Destination Group (dgName and rantId). TNPType is defined as follows:

```
 <complexType name="TNPType">
  <complexContent>
   <extension base="spppb:PubIdType">
    <sequence>
     <element name="tnPrefix"
     type="spppb:NumberValType"/>
     <element name="corInfo"
     type="spppb:CORInfoType" minOccurs="0"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

TNPType consists of the following attributes:

o tnPrefix: The telephone number prefix

o corInfo: Optional <corInfo> element of type CORInfoType.

6.3. Route Group

As described in the introductory sections, a Route Group represents a combined grouping of Route Records that define route information, Destination Groups that contain a set of Public Identifiers with common routing information, and the list of peer organizations that have access to these public identifiers using this route information. It is this indirect linking of public identifiers to their route information that significantly improves the scalability and manageability of the peering data. Additions and changes to routing information are reduced to a single operation on a Route Group or Route Record , rather than millions of data updates to individual public identifier records that individually contain their peering data. The transport protocol MUST support the ability to Create, Modify, Get, and Delete Route Groups (refer the "Protocol Operations" section of this document for a generic description of various operations).

A Route Group object MUST be uniquely identified by attributes as defined in the description of "ObjKeyType" in the section "Generic Object Key Type" of this document.

The RteGrpType object structure is defined as follows:

```
<complexType name="RteGrpType">
  <complexContent>
        <extension base="spppb:BasicObjType">
          <sequence>
                <element name="rgName" type="spppb:ObjNameType"/>
                <element name="rrRef" type="spppb:RteRecRefType"
                  minOccurs="0" maxOccurs="unbounded"/>
                <element name="dgName" type="spppb:ObjNameType"
                   minOccurs="0" maxOccurs="unbounded"/>
                <element name="peeringOrg" type="spppb:OrgIdType"
                   minOccurs="0" maxOccurs="unbounded"/>
                <element name="sourceIdent"
                   type="spppb:SourceIdentType"
                   minOccurs="0" maxOccurs="unbounded"/>
                <element name="isInSvc" type="boolean"/>
                <element name="priority" type="unsignedShort"/>
                <element name="ext" type="spppb:ExtAnyType"
                   minOccurs="0"/>
                </sequence>
         </extension>
  </complexContent>
</complexType>

 <complexType name="RteRecRefType">
   <sequence>
     <element name="rrKey" type="spppb:ObjKeyType"/>
     <element name="priority" type="unsignedShort"/>
     <element name="ext" type="spppb:ExtAnyType"
        minOccurs="0"/>
   </sequence>
 </complexType>
```

The RteGrpType object is composed of the following elements:

o base: All first class objects extend BasicObjType that contains the ID of the registrant organization that owns this object, the date and time that the object was created by the server, and the date and time that the object was last modified. If the client passes in either the created date or the modification date, the server will ignore them. The server sets these two date/time values.

- rgName: The character string that contains the name of the Route Group. It uniquely identifies this object within the context of the registrant ID (a child element of the base element as described above).

- rrRef: Set of zero or more objects of type RteRecRefType that house the unique keys of the Route Records that the RteGrpType object refers to and their relative priority within the context of a given route group. The associated Route Records contain the routing information, sometimes called SED, associated with this Route Group.

- dgName: Set of zero or more names of DestGrpType object instances. Each dgName name, in association with this Route Group's registrant ID, uniquely identifies a DestGrpType object instance whose public identifiers are reachable using the routing information housed in this Route Group. An intended side affect of this is that a Route Group cannot provide routing information for a Destination Group belonging to another registrant.

- peeringOrg: Set of zero or more peering organization IDs that have accepted an offer to receive this Route Group's information. The set of peering organizations in this list is not directly settable or modifiable using the addRteGrpsRqst operation. This set is instead controlled using the route offer and accept operations.

- sourceIdent: Set of zero or more SourceIdentType object instances. These objects, described further below, house the source identification schemes and identifiers that are applied at resolution time as part of source based routing algorithms for the Route Group.

- isInSvc: A boolean element that defines whether this Route Group is in service. The routing information contained in a Route Group that is in service is a candidate for inclusion in resolution responses for public identities residing in the Destination Group associated with this Route Group. The routing information contained in a Route Group that is not in service is not a candidate for inclusion in resolution responses.

- priority: Zero or one priority value that can be used to provide a relative value weighting of one Route Group over another. The manner in which this value is used, perhaps in conjunction with other factors, is a matter of policy.

- ext: Point of extensibility described in a previous section of this document.

As described above, the Route Group contains a set of references to route record objects. A route record object is based on an abstract type: RteRecType. The concrete types that use RteRecType as an extension base are NAPTRType, NSType, and URIType. The definitions of these types are included the Route Record section of this document.

The RteGrpType object provides support for source-based routing via the peeringOrg data element and more granular source base routing via the source identity element. The source identity element provides the ability to specify zero or more of the following in association with a given Route Group: a regular expression that is matched against the resolution client IP address, a regular expression that is matched against the root domain name(s), and/or a regular expression that is matched against the calling party URI(s). The result will be that, after identifying the visible Route Groups whose associated Destination Group(s) contain the lookup key being queried and whose peeringOrg list contains the querying organizations organization ID, the resolution server will evaluate the characteristics of the Source URI, and Source IP address, and root domain of the lookup key being queried. The resolution server then compares these criteria against the source identity criteria associated with the Route Groups. The routing information contained in Route Groups that have source based routing criteria will only be included in the resolution response if one or more of the criteria matches the source criteria from the resolution request. The Source Identity data element is of type SourceIdentType, whose structure is defined as follows:

```
<complexType name="SourceIdentType">
  <sequence>
        <element name="sourceIdentLabel" type="token"/>
        <element name="sourceIdentScheme"
          type="spppb:SourceIdentSchemeType"/>
        <element name="ext" type="spppb:ExtAnyType"
           minOccurs="0"/>
  </sequence>
</complexType>

<simpleType name="SourceIdentSchemeType">
        <restriction base="token">
                <enumeration value="uri"/>
                <enumeration value="ip"/>
                <enumeration value="rootDomain"/>
        </restriction>
</simpleType>
```

The SourceIdentType object is composed of the following data elements:

- sourceIdentScheme: The source identification scheme that this source identification criteria applies to and that the associated sourceIdentRegex should be matched against.

- sourceIdentRegex: The regular expression that should be used to test for a match against the portion of the resolution request that is dictated by the associated sourceIdentScheme.

- ext: Point of extensibility described in a previous section of this document.

6.4. Route Record

As described in the introductory sections, a Route Group represents a combined grouping of Route Records that define route information. However, Route Records need not be created to just serve a single Route Group. Route Records can be created and managed to serve multiple Route Groups. As a result, a change to the properties of a network node used for multiple routes, would necessitate just a single update operation to change the properties of that node. The change would then be reflected in all the Route Groups whose route record set contains a reference to that node. The transport protocol MUST support the ability to Create, Modify, Get, and Delete Route Records (refer the "Protocol Operations" section of this document for a generic description of various operations).

A Route Record object MUST be uniquely identified by attributes as

defined in the description of "ObjKeyType" in the section "Generic Object Key Type" of this document.

The RteRecType object structure is defined as follows:

```
<complexType name="RteRecType" abstract="true">
  <complexContent>
        <extension base="spppb:BasicObjType">
          <sequence>
                <element name="rrName" type="spppb:ObjNameType"/>
                <element name="priority" type="unsignedShort"
                   minOccurs="0"/>
          </sequence>
    </extension>
  </complexContent>
</complexType>
```

The RteRecType object is composed of the following elements:

- base: All first class objects extend BasicObjType that contains the ID of the registrant organization that owns this object, the date and time that the object was created by the server, and the date and time that the object was last modified. If the client passes in either the created date or the modification date, the server will ignore them. The server sets these two date/time values.

- rrName: The character string that contains the name of the Route Record. It uniquely identifies this object within the context of the registrant ID (a child element of the base element as described above).

- priority: Zero or one priority value that can be used to provide a relative value weighting of one Route Record over another. The manner in which this value is used, perhaps in conjunction with other factors, is a matter of policy.

As described above, route records are based on an abstract type: RteRecType. The concrete types that use RteRecType as an extension base are NAPTRType, NSType, and URIType. The definitions of these types are included below. The NAPTRType object is comprised of the data elements necessary for a NAPTR that contains routing information for a Route Group. The NSType object is comprised of the data elements necessary for a DNS name server that points to another DNS server that contains the desired routing information. The NSType is relevant only when the resolution protocol is ENUM. The URIType object is comprised of the data elements necessary to house a URI.

The data provisioned in a registry can be leveraged for many purposes and queried using various protocols including SIP, ENUM and others. It is for this reason that a route record type offers a choice of URI and DNS resource record types. URIType fulfills the need for both SIP and ENUM protocols. When a given URIType is associated to a destination group, the user part of the replacement string <uri> that may require the Public Identifier cannot be preset. As a SIP Redirect, the resolution server will apply <ere> pattern on the input Public Identifier in the query and process the replacement string by substituting any back reference(s) in the <uri> to arrive at the final URI that is returned in the SIP Contact header. For an ENUM query, the resolution server will simply return the value of the <ere> and <uri> members of the URIType in the NAPTR REGEX parameter.

```
<complexType name="NAPTRType">
  <complexContent>
        <extension base="spppb:RteRecType">
          <sequence>
                <element name="order" type="unsignedShort"/>
                <element name="flags" type="spppb:FlagsType"
                   minOccurs="0"/>
                <element name="svcs" type="spppb:SvcType"/>
                <element name="regx" type="spppb:RegexParamType"
                   minOccurs="0"/>
                <element name="repl" type="spppb:ReplType"
                   minOccurs="0"/>
                <element name="ttl" type="positiveInteger"
                   minOccurs="0"/>
                <element name="ext" type="spppb:ExtAnyType"
                   minOccurs="0"/>
          </sequence>
        </extension>
  </complexContent>
</complexType>

<complexType name="NSType">
  <complexContent>
        <extension base="spppb:RteRecType">
          <sequence>
                <element name="hostName" type="token"/>
                <element name="ipAddr" type="spppb:IPAddrType"
                   minOccurs="0" maxOccurs="unbounded"/>
                <element name="ttl" type="positiveInteger"
                   minOccurs="0"/>
                <element name="ext" type="spppb:ExtAnyType"
                   minOccurs="0"/>
          </sequence>
```

```
          </extension>
    </complexContent>
  </complexType>

  <complexType name="IPAddrType">
    <sequence>
          <element name="addr" type="spppb:AddrStringType"/>
          <element name="ext" type="spppb:ExtAnyType"
             minOccurs="0"/>
    </sequence>
    <attribute name="type" type="spppb:IPType"
       default="v4"/>
  </complexType>

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

  <complexType name="URIType">
    <complexContent>
          <extension base="spppb:RteRecType">
            <sequence>
                  <element name="ere" type="token"
                     default="^(.*)$"/>
                  <element name="uri" type="anyURI"/>
                  <element name="ext" type="spppb:ExtAnyType"
                     minOccurs="0"/>
            </sequence>
          </extension>
    </complexContent>
  </complexType>

  <simpleType name="flagsType">
          <restriction base="token">
                  <length value="1"/>
                  <pattern value="[A-Z]|[a-z]|[0-9]"/>
          </restriction>
  </simpleType>
```

The NAPTRType object is composed of the following elements:

o order: Order value in an ENUM NAPTR, relative to other NAPTRType objects in the same Route Group.

- svcs: ENUM service(s) that are served by the SBE. This field's value must be of the form specified in [RFC6116] (e.g., E2U+ pstn:sip+sip). The allowable values are a matter of policy and not limited by this protocol.

- regx: NAPTR's regular expression field. If this is not included then the Repl field must be included.

- repl: NAPTR replacement field, should only be provided if the Regex field is not provided, otherwise the server will ignore it

- ttl: Number of seconds that an addressing server may cache this NAPTR.

- ext: Point of extensibility described in a previous section of this document.

The NSType object is composed of the following elements:

- hostName: Fully qualified host name of the name server.

- ipAddr: Zero or more objects of type IpAddrType. Each object holds an IP Address and the IP Address type, IPv4 or IP v6.

- ttl: Number of seconds that an addressing server may cache this DNS name server.

- ext: Point of extensibility described in a previous section of this document.

The URIType object is composed of the following elements:

- ere: The POSIX Extended Regular Expression (ere) as defined in [RFC3986].

- uri: the URI as defined in [RFC3986]. In some cases, this will serve as the replacement string and it will be left to the resolution server to arrive at the final usable URI.

6.5. Route Group Offer

The list of peer organizations whose resolution responses can include the routing information contained in a given Route Group is controlled by the organization to which a Route Group object belongs (its registrant), and the peer organization that submits resolution requests (a data recipient, also know as a peering organization). The registrant offers access to a Route Group by submitting a Route Group Offer. The data recipient can then accept or reject that

offer. Not until access to a Route Group has been offered and accepted will the data recipient's organization ID be included in the peeringOrg list in a Route Group object, and that Route Group's peering information become a candidate for inclusion in the responses to the resolution requests submitted by that data recipient. The transport protocol MUST support the ability to Create, Modify, Get, Delete, Accept and Reject Route Group Offers (refer the "Protocol Operations" section of this document for a generic description of various operations).

A Route Group Offer object MUST be uniquely identified by attributes as defined in the description of "RteGrpOfferKeyType" in the section "Derived Object Key Types" of this document.

The RteGrpOfferType object structure is defined as follows:

```
   <complexType name="RteGrpOfferType">
     <complexContent>
           <extension base="spppb:BasicObjType">
             <sequence>
                   <element name="rteGrpOfferKey"
                      type="spppb:RteGrpOfferKeyType"/>
                   <element name="status"
                      type="spppb:RteGrpOfferStatusType"/>
                   <element name="offerDateTime" type="dateTime"/>
                   <element name="acceptDateTime" type="dateTime"
                      minOccurs="0"/>
                   <element name="ext" type="spppb:ExtAnyType"
                      minOccurs="0"/>
                   </sequence>
            </extension>
     </complexContent>
   </complexType>

   <complexType name="RteGrpOfferKeyType" abstract="true">
     <annotation>
           <documentation>
           -- Generic type that represents the key for a route
           route group offer. Must be defined in concrete form
           in the transport specificaiton. --
           </documentation>
     </annotation>
   </complexType>

   <simpleType name="RteGrpOfferStatusType">
      <restriction base="token">
         <enumeration value="offered"/>
         <enumeration value="accepted"/>
       </restriction>
     </simpleType>
```

The RteGrpOfferType object is composed of the following elements:

- base: All first class objects extend BasicObjType that contains the ID of the registrant organization that owns this object, the date and time that the object was created by the server, and the date and time that the object was last modified. If the client passed in either the created date or the modification date, the will ignore them. The server sets these two date/time values.

- rteGrpOfferKey: The object that identifies the route that is or has been offered and the organization that it is or has been offered to.

- status: The status of the offer, offered or accepted. The server controls the status. It is automatically set to "offered" when ever a new Route Group Offer is added, and is automatically set to "accepted" if and when that offer is accepted. The value of the element is ignored when passed in by the client.

- offerDateTime: Date and time in UTC when the Route Group Offer was added.

- acceptDateTime: Date and time in UTC when the Route Group Offer was accepted.

6.6. Egress Route

In a high-availability environment, the originating SSP likely has more than one egress paths to the ingress SBE of the target SSP. If the originating SSP wants to exercise greater control and choose a specific egress SBE to be associated to the target ingress SBE, it can do so using the EgrRteType object.

A Egress Route object MUST be uniquely identified by attributes as defined in the description of "ObjKeyType" in the section "Generic Object Key Type" of this document.

Lets assume that the target SSP has offered to share one or more ingress route information and that the originating SSP has accepted the offer. In order to add the egress route to the registry, the originating SSP uses a valid regular expression to rewrite ingress route in order to include the egress SBE information. Also, more than one egress route can be associated with a given ingress route in support of fault-tolerant configurations. The supporting SPPP structure provides a way to include route precedence information to help manage traffic to more than one outbound egress SBE.

The transport protocol MUST support the ability to Add, Modify, Get, and Delete Egress Routes (refer the "Protocol Operations" section of this document for a generic description of various operations). The EgrRteType object structure is defined as follows:

```
<complexType name="EgrRteType">
  <complexContent>
        <extension base="spppb:BasicObjType">
          <sequence>
                <element name="egrRteName" type="spppb:ObjNameType"/>
                <element name="pref" type="unsignedShort"/>
                <element name="regxRewriteRule"
                  type="spppb:RegexParamType"/>
                <element name="ingrRteRec" type="spppb:ObjKeyType"
                   minOccurs="0" maxOccurs="unbounded"/>
                <element name="ext" type="spppb:ExtAnyType"
                  minOccurs="0"/>
          </sequence>
        </extension>
  </complexContent>
</complexType>
```

The EgrRteType object is composed of the following elements:

- base: All first class objects extend BasicObjType that contains the ID of the registrant organization that owns this object, the date and time that the object was created by the server, and the date and time that the object was last modified. If the client passes in either the created date or the modification date, the server will ignore them. The server sets these two date/time values.

- egrRteName: The name of the egress route.

- pref: The preference of this egress route relative to other egress routes that may get selected when responding to a resolution request.

- regxRewriteRule: The regular expression re-write rule that should be applied to the regular expression of the ingress NAPTR(s) that belong to the ingress route.

- ingrRteRec: The ingress route records that the egress route should be used for.

- ext: Point of extensibility described in a previous section of this document.

7. Protocol Operations

7.1. Add Operation

Any conforming "transport" specification MUST provide a definition for the operation that adds one or more SPPP objects into the registry. If the object, as identified by the request attributes that form part of the object's key, does not exist, then the registry MUST create the object. If the object does exist, then the registry MUST replace the current properties of the object with the properties passed in as part of the Add operation.

If the entity that issued the command is not authorized to perform this operation an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" section of the document.

7.2. Delete Operation

Any conforming "transport" specification MUST provide a definition for the operation that deletes one or more SPPP objects from the registry using the object's key.

If the entity that issued the command is not authorized to perform this operation an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" section of the document.

When an object is deleted, any references to that object must of course also be removed as the SPPP server implementation fulfills the deletion request. Furthermore, the deletion of a composite object must also result in the deletion of the objects it contains. As a result, the following rules apply to the deletion of SPPP object types:

- Destination Groups: When a destination group is deleted all public identifiers within that destination group must also be automatically deleted by the SPPP implementation as part of fulfilling the deletion request. And any references between that destination group and any route group must be automatically removed by the SPPP implementation as part of fulfilling the deletion request.

- Route Groups: When a route group is deleted any references between that route group and any destination group must be automatically removed by the SPPP implementation as part of fulfilling the deletion request. Similarly any references between that route group and any route records must be removed

by the SPPP implementation as part of fulfilling the deletion request. Furthermore, route group offers relating that route group must also be deleted as part of fulfilling the deletion request.

- Route Records: When a route record is deleted any references between that route record and any route group must be removed by the SPPP implementation as part of fulfilling the deletion request.

- Public Identifiers: When a public identifier is deleted any references between that public identifier and its containing destination group must be removed by the SPPP implementation as part of fulfilling the deletion request. And any route records contained directly within that Public Identifier must be deleted by the SPPP implementation as part of fulfilling the deletion request.

7.3. Get Operations

At times, on behalf of the registrant, the registrar may need to have access to SPPP objects that were previously provisioned in the registry. A few examples include logging, auditing, and pre-provisioning dependency checking. This query mechanism is limited to aid provisioning scenarios and should not be confused with query protocols provided as part of the resolution system (e.g. ENUM and SIP). Any conforming "transport" specification MUST provide a definition for the operation that queries the details of one or more SPPP objects from the registry using the object's key. If the entity that issued the command is not authorized to perform this operation an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" section of the document.

7.4. Accept Operations

In SPPP, a Route Group Offer can be accepted or rejected by, or on behalf of, the registrant to whom the Route Group has been offered (refer "Protocol Data Model Objects" section of this document for a description of the Route Group Offer object). The Accept operation is used to accept the Route Group Offers. Any conforming "transport" specification MUST provide a definition for the operation to accept Route Group Offers by, or on behalf of the Registrant, using the Route Group Offer object key.

Not until access to a Route Group has been offered and accepted will the registrant's organization ID be included in the peeringOrg list in that Route Group object, and that Route Group's peering

information become a candidate for inclusion in the responses to the resolution requests submitted by that registrant. A Route Group Offer that is in the "offered" status is accepted by, or on behalf of, the registrant to which it has been offered. When the Route Group Offer is accepted the the Route Group Offer is moved to the "accepted" status and adds that data recipient's organization ID into the list of peerOrgIds for that Route Group.

If the entity that issued the command is not authorized to perform this operation an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" section of the document.

7.5. Reject Operations

In SPPP, a Route Group Offer object can be accepted or rejected by, or on behalf of, the registrant to whom the Route Group has been offered (refer "Protocol Data Model Objects" section of this document for a description of the Route Group Offer object). Furthermore, that offer may be rejected, regardless of whether or not it has been previously accepted. The Reject operation is used to reject the Route Group Offers. When the Route Group Offer is rejected that Route Group Offer is deleted, and, if appropriate, the data recipient's organization ID is removed from the list of peeringOrg IDs for that Route Group. Any conforming "transport" specification MUST provide a definition for the operation to reject Route Group Offers by, or on behalf of the Registrant, using the Route Group Offer object key.

If the entity that issued the command is not authorized to perform this operation an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" section of the document.

7.6. Get Server Details Operation

In SPPP, Get Server Details operation can be used to request certain details about the SPPP server that include the SPPP server's current status, the major/minor version of the SPPP protocol supported by the SPPP server..

Any conforming "transport" specification MUST provide a definition for the operation to request such details from the SPPP server. If the entity that issued the command is not authorized to perform this operation an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" section of the document.

8. XML Considerations

XML serves as the encoding format for SPPP, allowing complex hierarchical data to be expressed in a text format that can be read, saved, and manipulated with both traditional text tools and tools specific to XML.

XML is case sensitive. Unless stated otherwise, XML specifications and examples provided in this document MUST be interpreted in the character case presented to develop a conforming implementation.

This section discusses a small number of XML-related considerations pertaining to SPPP.

8.1. Namespaces

All SPPP elements are defined in the namespaces in the IANA Considerations section and in the Formal Protocol Specification section of this document.

8.2. Versioning and Character Encoding

All XML instances SHOULD begin with an <?xml?> declaration to identify the version of XML that is being used, optionally identify use of the character encoding used, and optionally provide a hint to an XML parser that an external schema file is needed to validate the XML instance.

Conformant XML parsers recognize both UTF-8 (defined in [RFC3629]) and UTF-16 (defined in [RFC2781]); per [RFC2277] UTF-8 is the RECOMMENDED character encoding for use with SPPP.

Character encodings other than UTF-8 and UTF-16 are allowed by XML. UTF-8 is the default encoding assumed by XML in the absence of an "encoding" attribute or a byte order mark (BOM); thus, the "encoding" attribute in the XML declaration is OPTIONAL if UTF-8 encoding is used. SPPP clients and servers MUST accept a UTF-8 BOM if present, though emitting a UTF-8 BOM is NOT RECOMMENDED.

Example XML declarations:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

9. Security Considerations

Many SPPP implementations manage data that is considered confidential and critical. Furthermore, SPPP implementations can support provisioning activities for multiple registrars and registrants. As a result any SPPP implementation must address the requirements for confidentiality, authentication, and authorization.

With respect to confidentiality and authentication, the transport protocol requirements section of this document contains security properties that the transport protocol must provide so that authenticated endpoints can exchange data confidentially and with integrity protection. Refer to that section and the resulting transport protocol specification document for the specific solutions to authentication and confidentiality.

With respect to authorization, the SPPP server implementation must define and implement a set of authorization rules that precisely address (1) which registrars will be authorized to create/modify/ delete each SPPP object type for given registrant(s) and (2) which registrars will be authorized to view/get each SPPP object type for given registrant(s). These authorization rules are a matter of policy and are not specified within the context of SPPP. However, any SPPP implementation must specify these authorization rules in order to function in a reliable and safe manner.

In some situations, it may be required to protect against denial of involvement (see [RFC4949]) and tackle non-repudiation concerns in regards to SPPP messages. This type of protection is useful to satisfy authenticity concerns related to SPPP messages beyond the end-to-end connection integrity, confidentiality, and authentication protection that the transport layer provides. This is an optional feature and some SPPP implementations MAY provide support for it.

It is not uncommon for the logging systems to document on-the-wire messages for various purposes, such as, debug, audit, and tracking. At the minimum, the various support and administration staff will have access to these logs. Also, if an unprivileged user gains access to the SPPP deployments and/or support systems, it will have access to the information that is potentially deemed confidential. To manage information disclosure concerns beyond the transport level, SPPP implementations MAY provide support for encryption at the SPPP object level.

Anti-replay protection ensures that a given SPPP object replayed at a later time doesn’t affect the integrity of the system. SPPP provides at least one mechanism to fight against replay attacks. Use of the optional client transaction identifier allows the SPPP client to

correlate the request message with the response and to be sure that it is not a replay of a server response from earlier exchanges. Use of unique values for the client transaction identifier is highly encouraged to avoid chance matches to a potential replay message.

The SPPP client or registrar can be a separate entity acting on behalf of the registrant in facilitating provisioning transactions to the registry. Further, the transport layer provides end-to-end connection protection between SPPP client and the SPPP server. Therefore, man-in-the-middle attack is a possibility that may affect the integrity of the data that belongs to the registrant and/or expose peer data to unintended actors in case well-established peering relationships already exist.

10. IANA Considerations

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [RFC3688].

Two URI assignments are requested.

Registration request for the SPPP XML namespace:
urn:ietf:params:xml:ns:sppp:base:1
Registrant Contact: IESG
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the XML schema:
URI: urn:ietf:params:xml:schema:sppp:1
Registrant Contact: IESG
XML: See the "Formal Specification" section of this document (Section 11).

IANA is requested to create a new SPPP registry for Organization Identifiers that will indicate valid strings to be used for well-known enterprise namespaces.
This document makes the following assignments for the OrgIdType namespaces:

Namespace	OrgIdType namespace string
IANA Enterprise Numbers	iana-en

11. Formal Specification

This section provides the draft XML Schema Definition for SPPP Protocol.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:spppb="urn:ietf:params:xml:ns:sppp:base:1"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:ietf:params:xml:ns:sppp:base:1"
elementFormDefault="qualified" xml:lang="EN">
 <annotation>
  <documentation>
                ---- Generic Object key
                types to be defined by specific
                Transport/Architecture.
                The types defined here can
                be extended by the
                specific architecture to
                define the Object Identifiers ----
  </documentation>
 </annotation>
 <complexType name="ObjKeyType"
  abstract="true">
  <annotation>
   <documentation>
   ---- Generic type that
   represents the key for various
   objects in SPPP. ----
   </documentation>
  </annotation>
 </complexType>
 <complexType name="RteGrpOfferKeyType" abstract="true">
  <complexContent>
   <extension base="spppb:ObjKeyType">
    <annotation>
     <documentation>
     ---- Generic type
     that represents
     the key for a route
     group offer. ----
     </documentation>
    </annotation>
   </extension>
  </complexContent>
 </complexType>
```

```
    <complexType name="PubIdKeyType" abstract="true">
     <complexContent>
      <extension base="spppb:ObjKeyType">
       <annotation>
        <documentation>
         ----Generic type that
         represents the key
         for a Pub Id. ----
        </documentation>
       </annotation>
      </extension>
     </complexContent>
    </complexType>
    <annotation>
     <documentation> ---- Object Type Definitions ---- </documentation>
    </annotation>
    <complexType name="RteGrpType">
     <complexContent>
      <extension base="spppb:BasicObjType">
       <sequence>
        <element name="rgName"
        type="spppb:ObjNameType"/>
        <element name="rrRef"
        type="spppb:RteRecRefType"
        minOccurs="0" maxOccurs="unbounded"/>
        <element name="dgName"
        type="spppb:ObjNameType"
        minOccurs="0" maxOccurs="unbounded"/>
        <element name="peeringOrg"
        type="spppb:OrgIdType"
        minOccurs="0" maxOccurs="unbounded"/>
        <element name="sourceIdent"
         type="spppb:SourceIdentType"
         minOccurs="0" maxOccurs="unbounded"/>
        <element name="isInSvc" type="boolean"/>
        <element name="priority" type="unsignedShort"/>
        <element name="ext"
        type="spppb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="DestGrpType">
     <complexContent>
      <extension base="spppb:BasicObjType">
       <sequence>
        <element name="dgName"
        type="spppb:ObjNameType"/>
```

```
      </sequence>
     </extension>
    </complexContent>
   </complexType>
   <complexType name="PubIdType" abstract="true">
    <complexContent>
     <extension base="spppb:BasicObjType">
      <sequence>
       <element name="dgName"
       type="spppb:ObjNameType" minOccurs="0"/>
      </sequence>
     </extension>
    </complexContent>
   </complexType>
   <complexType name="TNType">
    <complexContent>
     <extension base="spppb:PubIdType">
      <sequence>
       <element name="tn"
       type="spppb:NumberValType"/>
       <element name="corInfo"
       type="spppb:CORInfoType" minOccurs="0"/>
       <element name="rrRef"
       type="spppb:RteRecRefType"
       minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
     </extension>
    </complexContent>
   </complexType>
   <complexType name="TNRType">
    <complexContent>
     <extension base="spppb:PubIdType">
      <sequence>
       <element name="range"
       type="spppb:NumberRangeType"/>
       <element name="corInfo"
       type="spppb:CORInfoType" minOccurs="0"/>
      </sequence>
     </extension>
    </complexContent>
   </complexType>
   <complexType name="TNPType">
    <complexContent>
     <extension base="spppb:PubIdType">
      <sequence>
       <element name="tnPrefix"
       type="spppb:NumberValType"/>
       <element name="corInfo"
```

```
        type="spppb:CORInfoType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="RNType">
     <complexContent>
      <extension base="spppb:PubIdType">
       <sequence>
        <element name="rn"
        type="spppb:NumberValType"/>
        <element name="corInfo"
        type="spppb:CORInfoType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="RteRecType" abstract="true">
     <complexContent>
      <extension base="spppb:BasicObjType">
       <sequence>
        <element name="rrName"
        type="spppb:ObjNameType"/>
        <element name="priority"
        type="unsignedShort" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="NAPTRType">
     <complexContent>
      <extension base="spppb:RteRecType">
       <sequence>
        <element name="order" type="unsignedShort"/>
        <element name="flags" type="spppb:FlagsType" minOccurs="0"/>
        <element name="svcs" type="spppb:SvcType"/>
        <element name="regx" type="spppb:RegexParamType" minOccurs="0"/>
        <element name="repl" type="spppb:ReplType" minOccurs="0"/>
        <element name="ttl" type="positiveInteger" minOccurs="0"/>
        <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="NSType">
     <complexContent>
      <extension base="spppb:RteRecType">
       <sequence>
```

```
        <element name="hostName" type="token"/>
        <element name="ipAddr" type="spppb:IPAddrType"
        minOccurs="0" maxOccurs="unbounded"/>
        <element name="ttl" type="positiveInteger" minOccurs="0"/>
        <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="URIType">
     <complexContent>
      <extension base="spppb:RteRecType">
       <sequence>
        <element name="ere" type="token" default="^(.*)$"/>
        <element name="uri" type="anyURI"/>
        <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="RteGrpOfferType">
     <complexContent>
      <extension base="spppb:BasicObjType">
       <sequence>
        <element name="rteGrpOfferKey" type="spppb:RteGrpOfferKeyType"/>
        <element name="status" type="spppb:RteGrpOfferStatusType"/>
        <element name="offerDateTime" type="dateTime"/>
        <element name="acceptDateTime" type="dateTime" minOccurs="0"/>
        <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="EgrRteType">
     <complexContent>
      <extension base="spppb:BasicObjType">
       <sequence>
        <element name="egrRteName" type="spppb:ObjNameType"/>
        <element name="pref" type="unsignedShort"/>
        <element name="regxRewriteRule" type="spppb:RegexParamType"/>
        <element name="ingrRteRec" type="spppb:ObjKeyType" minOccurs="0"
        maxOccurs="unbounded"/>
        <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <annotation>
```

```
     <documentation>
                           ---- Abstract Object and
                           Element Type
                           Definitions ----
                  </documentation>
    </annotation>
    <complexType name="BasicObjType" abstract="true">
     <sequence>
      <element name="rant" type="spppb:OrgIdType"/>
      <element name="rar" type="spppb:OrgIdType"/>
      <element name="cDate" type="dateTime" minOccurs="0"/>
      <element name="mDate" type="dateTime" minOccurs="0"/>
      <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
     </sequence>
    </complexType>
    <complexType name="RegexParamType">
     <sequence>
      <element name="ere" type="spppb:RegexType" default="^(.*)$"/>
      <element name="repl" type="spppb:ReplType"/>
     </sequence>
    </complexType>
    <complexType name="IPAddrType">
     <sequence>
      <element name="addr" type="spppb:AddrStringType"/>
      <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
     </sequence>
     <attribute name="type" type="spppb:IPType" default="v4"/>
    </complexType>
    <complexType name="RteRecRefType">
     <sequence>
      <element name="rrKey" type="spppb:ObjKeyType"/>
      <element name="priority" type="unsignedShort"/>
      <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
     </sequence>
    </complexType>
    <complexType name="SourceIdentType">
     <sequence>
      <element name="sourceIdentLabel" type="token"/>
      <element name="sourceIdentScheme"
      type="spppb:SourceIdentSchemeType"/>
      <element name="ext" type="spppb:ExtAnyType" minOccurs="0"/>
     </sequence>
    </complexType>
    <complexType name="CORInfoType">
     <sequence>
      <element name="corClaim" type="boolean" default="true"/>
      <element name="cor" type="boolean" default="false" minOccurs="0"/>
      <element name="corDate" type="dateTime" minOccurs="0"/>
```

```
     </sequence>
    </complexType>
    <complexType name="SvcMenuType">
     <sequence>
      <element name="serverStatus" type="spppb:ServerStatusType"/>
      <element name="majMinVersion" type="token" 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>
    <simpleType name="FlagsType">
     <restriction base="token">
      <length value="1"/>
      <pattern value="[A-Z]|[a-z]|[0-9]"/>
     </restriction>
    </simpleType>
    <simpleType name="SvcType">
     <restriction base="token">
      <minLength value="1"/>
     </restriction>
    </simpleType>
    <simpleType name="RegexType">
     <restriction base="token">
      <minLength value="1"/>
     </restriction>
    </simpleType>
    <simpleType name="ReplType">
     <restriction base="token">
      <minLength value="1"/>
      <maxLength value="255"/>
     </restriction>
    </simpleType>
    <simpleType name="OrgIdType">
     <restriction base="token"/>
    </simpleType>
    <simpleType name="ObjNameType">
     <restriction base="token">
      <minLength value="3"/>
      <maxLength value="80"/>
     </restriction>
    </simpleType>
    <simpleType name="TransIdType">
```

```
     <restriction base="token">
      <minLength value="3"/>
      <maxLength value="120"/>
     </restriction>
    </simpleType>
    <simpleType name="MinorVerType">
     <restriction base="unsignedLong"/>
    </simpleType>
    <simpleType name="AddrStringType">
     <restriction base="token">
      <minLength value="3"/>
      <maxLength value="45"/>
     </restriction>
    </simpleType>
    <simpleType name="IPType">
     <restriction base="token">
      <enumeration value="v4"/>
      <enumeration value="v6"/>
     </restriction>
    </simpleType>
    <simpleType name="SourceIdentSchemeType">
     <restriction base="token">
      <enumeration value="uri"/>
      <enumeration value="ip"/>
      <enumeration value="rootDomain"/>
     </restriction>
    </simpleType>
    <simpleType name="ServerStatusType">
     <restriction base="token">
      <enumeration value="inService"/>
      <enumeration value="outOfService"/>
     </restriction>
    </simpleType>
    <simpleType name="RteGrpOfferStatusType">
     <restriction base="token">
      <enumeration value="offered"/>
      <enumeration value="accepted"/>
     </restriction>
    </simpleType>
    <simpleType name="NumberValType">
     <restriction base="token">
      <maxLength value="20"/>
      <pattern value="\+?\d\d*"/>
     </restriction>
    </simpleType>
    <simpleType name="NumberTypeEnum">
     <restriction base="token">
      <enumeration value="TN"/>
```

```
      <enumeration value="TNPrefix"/>
      <enumeration value="RN"/>
     </restriction>
    </simpleType>
    <complexType name="NumberType">
     <sequence>
      <element name="value" type="spppb:NumberValType"/>
      <element name="type" type="spppb:NumberTypeEnum"/>
     </sequence>
    </complexType>
    <complexType name="NumberRangeType">
     <sequence>
      <element name="startRange" type="spppb:NumberValType"/>
      <element name="endRange" type="spppb:NumberValType"/>
     </sequence>
    </complexType>
     <simpleType name="ObjKeyTypeEnum">
      <restriction base="token">
        <enumeration value="RteGrp"/>
        <enumeration value="DestGrp"/>
        <enumeration value="RteRec"/>
        <enumeration value="EgrRte"/>
      </restriction>
    </simpleType>
   </schema>
```

12. Acknowledgments

This document is a result of various discussions held in the DRINKS working group and within the DRINKS protocol design team, which is comprised of the following individuals, in alphabetical order: Alexander Mayrhofer, Deborah A Guyton, David Schwartz, Lisa Dusseault, Manjul Maharishi, Mickael Marrache, Otmar Lendl, Richard Shockey, Samuel Melloul, and Sumanth Channabasappa.

13. References

13.1. Normative References

[I-D.ietf-drinks-sppp-over-soap]
Cartwright, K., "SPPP Over SOAP and HTTP", draft-ietf-drinks-sppp-over-soap-06 (work in progress), October 2011.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC5067] Lind, S. and P. Pfautz, "Infrastructure ENUM Requirements", RFC 5067, November 2007.

13.2. Informative References

[I-D.ietf-drinks-usecases-requirements]
Channabasappa, S., "Data for Reachability of Inter/tra-NetworK SIP (DRINKS) Use cases and Protocol Requirements", draft-ietf-drinks-usecases-requirements-06 (work in progress), August 2011.

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC4725] Mayrhofer, A. and B. Hoeneisen, "ENUM Validation

Architecture", RFC 4725, November 2006.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC5486] Malas, D. and D. Meyer, "Session Peering for Multimedia Interconnect (SPEERMINT) Terminology", RFC 5486, March 2009.

[RFC6116] Bradner, S., Conroy, L., and K. Fujiwara, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)", RFC 6116, March 2011.

Authors' Addresses

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

Email: jfm@cablelabs.com

Kenneth Cartwright
TNS
1939 Roland Clarke Place
Reston, VA 20191
USA

Email: kcartwright@tnsi.com

Syed Wasim Ali
NeuStar
46000 Center Oak Plaza
Sterling, VA 20166
USA

Email: syed.ali@neustar.biz

Alexander Mayrhofer
enum.at GmbH
Karlsplatz 1/9
Wien, A-1010
Austria

Email: alexander.mayrhofer@enum.at

Vikas Bhatia
TNS
1939 Roland Clarke Place
Reston, VA 20191
USA

Email: vbhatia@tnsi.com