DRINKS
Internet-Draft
Intended status: Standards Track
Expires: August 2, 2012

K. Cartwright
V. Bhatia
TNS

January 30, 2012

Session Peering Provisioning (SPP) Protocol over SOAP
draft-ietf-drinks-spp-protocol-over-soap-00

Abstract

The Session Peering Provisioning Framework (SPPF) is an XML framework 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 protocol for SPPF 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 SPPF XML structures over SOAP and HTTP(s).

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 August 2, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 4
2. Terminology 5
3. SOAP Features and Protocol Layering 6
4. HTTP(s) Features and SPPF 9
5. Authentication and Session Management 10
6. SPP Protocol SOAP Data Structures 11
6.1. Concrete Object Key Types 11
6.1.1. Generic Object Key 11
6.1.2. Public Identity Object Key 12
6.1.3. Route Group Offer Key 13
6.2. Operation Request and Response Structures 13
6.2.1. Add Operation Structure 14
6.2.2. Delete Operation Structure 17
6.2.3. Accept Operation Structure 20
6.2.4. Reject Operation Structure 23
6.2.5. Batch Operation Structure 26
6.2.6. Get Operation Structure 29
6.2.7. Get Route Group Offers Operation Structure 31
6.2.8. Generic Query Response 32
6.2.9. Get Server Details Operation Structure 33
6.3. Response Codes and Messages 35
7. Protocol Operations 37
8. SPPF SOAP WSDL Definition 38
9. SPP Protocol over SOAP Examples 49
9.1. Add Destination Group 49
9.2. Add Route Records 51
9.3. Add Route Records -- URIType 52
9.4. Add Route Group 53
9.5. Add Public Identity -- Successful COR claim 55
9.6. Add LRN 57
9.7. Add TN Range 58
9.8. Add TN Prefix 59
9.9. Enable Peering -- Route Group Offer 60
9.10. Enable Peering -- Route Group Offer Accept 62
9.11. Add Egress Route 63
9.12. Remove Peering -- Route Group Offer Reject 65
9.13. Get Destination Group 66

9.14. Get Public Identity 68
9.15. Get Route Group Request 69
9.16. Get Route Group Offers Request 71
9.17. Get Egress Route 73
9.18. Delete Destination Group 74
9.19. Delete Public Identity 75
9.20. Delete Route Group Request 77
9.21. Delete Route Group Offers Request 78
9.22. Delete Egress Route 79
9.23. Batch Request 80
10. Security Considerations 83
10.1. Integrity, Privacy, and Authentication 83
10.2. Vulnerabilities 83
10.3. Deployment Environment Specifics 83
11. IANA Considerations 84
12. Acknowledgements 85
13. References 86
13.1. Normative References 86
13.2. Informative References 86
Authors' Addresses 87

1. Introduction

SPPF, defined in [I-D.draft-ietf-drinks-spp-framework], 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 SPPF 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 SPPF is likely to reside in a separate organization's network, resulting the SPPF provisioning transactions traversing the Internet as they are propagated from the SPPF client to the SPPF 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 SPPF XML structures over SOAP and HTTP(s).

The specification in this document for transporting SPPF 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 SPP Protocol 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 SPP Protocol over SOAP are limited. Most SOAP features are not necessary for SPPF. SPP Protocol over SOAP 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 SPPF message itself, whose structure is defined by the combination of one of the WSDL operations defined in this document and the SPPF XML data structures defined in this document and the SPPF document. SPPF does not rely on any data elements in the SOAP header. All relevant data elements are defined in the SPPF XML schema described in [I-D.draft-ietf-drinks-spp-framework] and the SPPF 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 SPPF SOAP messages is defined later in this document, which imports by reference the XML data types contained in the SPPF schema. The IANA registry where the SPPF 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.

```
                                +-------------+
                            (1) |  Transport  |Example:
                                |   Protocol  |  TCP, TLS, BEEP, etc.
                                +-------------+
                                       |
                                       V
                                +-------------+
                            (2) |   Message   |Example:
                                |   Envelope  | HTTP, SOAP, None, etc.
                                +-------------+
                                       |
                                       V
                               +--------------+
                        +------|    SOAP      |-----+
                        |   (3) |  Operation   |     |
                Contains |       +--------------+     | Contains
                        |          Example:          |
                        V      submitAddRqst         V
                  +--------------+           +-------------+
                  |SOAP Request  |           |SOAP Response|
         Example:|  Message     | (4)       |   Message   | Example:
       spppAdd   | (Operation   |           | (Operation  | spppAdd
      RequestMsg |   Input)     |           |  Output)    | ResponseMsg
                  +--------------+           +-------------+
                           |                       |
                  Contains |                       | Contains
                           |                       |
                           V                       V
                 +---------------+         +---------------+
          Example:|    Wrapped    | (5)     |  Wrapped      | Example:
         spppAdd  |Request Object |         |Response Object| spppAdd
         Request  +---------------+         +---------------+ Response
                           |                       |
                  Contains |                       | Contains
                           |                       |
                           V                       V
                   +-------------+          +---------------+
                   |  SPPF       |          |   SPPF        |
                   |XML Types    |    (6)   | XML Types     |
                   +-------------+          +---------------+
```

Figure 1: Layering and Technical Structure of the SPP Protocol over SOAP Messages

The operations supported by SPP Protocol over SOAP 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 SPPF XML data types that comprise the inputs and the outputs, respectively, of the SOAP operation.

SOAP faults are not used by the SPP Protocol over SOAP. All 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.

SPPF is a request/reply framework that allows a client application to submit provisioning data and query requests to a server. The SPPF 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 framework document.

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

1. Layer 1: The transport protocol layer represents the communication mechanism between the client and server. SPPF 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 SPPF data model objects that are being acted on (created, modified, queried).

4. HTTP(s) Features and SPPF

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 SPP Protocol 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 SPPF HTTP connections.

HTTP 1.1 [RFC2616] or higher SHOULD be used.

5. Authentication and Session Management

To achieve integrity and privacy, conforming SPP Protocol 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 SPPF 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. SPP Protocol SOAP Data Structures

SPP Protocol 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 this 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 operations in SPPF require an object key that uniquely identifies the object(s) on which a given operation needs to be performed. SPPF defines the XML structure of the any such object key in an abstact manner and delegates the concrete represenation to any conforming transport protocol. The following sub-sections define the various types of concrete object key types used in various operations in SPP Protocol over SOAP:

6.1.1. Generic Object Key

Most objects in SPP Protocol over SOAP are unqiuely identified by the attributes in the concrete ObjKeyType. The XML representation of ObjKeyType is as below:

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

The ObjKeyType has the data elements as described below:

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

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

- o type: The enumeration vaue that represents the type of SPPF 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="sppfb:PubIdKeyType">
   <sequence>
    <element name="rant" type="sppfb:OrgIdType"/>
    <element name="dgName" type="sppfb:ObjNameType" minOccurs="0"/>
    <choice>
     <element name="number"
     type="sppfb:NumberType"/>
     <element name="range"
      type="sppfb:NumberRangeType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

The PubIdKeyType has the data elements as described below:

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

- o 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.

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

- o range: An element of type NumberRangeType (refer framework 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="sppfb:RteGrpOfferKeyType">
   <sequence>
    <element name="rteGrpKey" type="sppfs:ObjKeyType"/>
    <element name="offeredTo" type="sppfb:OrgIdType"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

The RteGrpOfferKeyType has the data elements as described below:

- o rteGrpKey: Identifies the Route Group that was offered.

- o 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 SPPF client interacts with an SPPF 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 SPPF client can submit to an SPPF server and the semantics of those operations are defined in the "Framework Operations" section of the framework document. The following sub-sections describe the XML data structures that are used for each of

those types of operations for a SPP Protocol over SOAP 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 SPP Protocol over SOAP Add request is wrapped within the <spppAddRequest> element while an SPPF Add response is wrapped within an <spppAddResponse> element. The following sub-sections describe the spppAddRequest and spppAddResponse elements. Refer the "SPPF SOAP Examples" section of this document for an example of Add operation on each type of SPPF object.

6.2.1.1. Add Request

An SPP Protocol over SOAP Add request definition is contained within the generic <spppAddRequest> element.

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

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

- o minorVer: Zero or one minor version identifier, indicating the minor version of the SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF server can be retrieved by the client using the SPPF server menu operation described later in the document.

- o obj: One or more elements of abstract type BasicObjType (defined in the framework document). Each element contains all the attributes of an SPPF object that that the client is requesting the SPPF server to add. Refer the "Framework Data Model Objects" section of the framework document for the XML structure of all concrete types, for various SPPF objects, that extend from abstract BasicObjType and hence are eligible to be passed into this element. The elements are processed by the SPPF 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 SPPF 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 SPPF 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 SPP Protocol over SOAP add response object is contained within the generic <spppAddResponse> element. This response structure is used for all types of SPPF objects that are provisioned by the SPPF client.

```
<element name="spppAddResponse">
  <complexType>
    <sequence>
      <element name="clientTransId" type="sppfb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="sppfb:TransIdType"/>
      <element name="overallResult" type="sppfb:ResultCodeType"/>
      <element name="dtlResult" type="sppfs: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="sppfs:ResultCodeType">
    <sequence>
     <element name="obj" type="sppfb:BasicObjType"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

An <spppAddResponse> contains the elements necessary for the SPPF 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 SPPF Add response are described as follows:

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

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

for a given SPPF server.

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

- o dtlResult: An optional response code, response message, and BasicObjType (as defined in the framework 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 SPP Protocol over SOAP Del request is wrapped within the <spppDelRequest> element while a SPPF Del response is wrapped within the generic <spppDelResponse> element. The following sub-sections describe the spppDelRequest and spppDelResponse elements. Refer the "SPPF SOAP Examples" section of this document for an example of Delete operation on each type of SPPF object.

6.2.2.1. Delete Request

An SPP Protocol over SOAP Del request definition is contained within the generic <spppDelRequest> element.

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

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

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

- o minorVer: Zero or one minor version identifier, indicating the minor version of the SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF server can be retrieved by the client using the SPPF server menu operation described later in the document.

- o objKey: One or more elements of abstract type ObjKeyType (as defined in the framework 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 SPPF objects, which are eligible to be passed into this element. The elements are processed by the SPPF 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 SPPF 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 SPPF 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 SPP Protocol over SOAP delete response object is contained within the generic <sppDeleteResponse> element. This response structure is used for a delete request on all types of SPPF objects that are provisioned by the SPPF client.

```
<element name="spppDelResponse">
  <complexType>
    <sequence>
      <element name="clientTransId" type="sppfb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="sppfb:TransIdType"/>
      <element name="overallResult" type="sppfb:ResultCodeType"/>
      <element name="dtlResult" type="sppfs: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="sppfs:ResultCodeType">
    <sequence>
     <element name="objKey" type="sppfb:ObjKeyType"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

An <spppDelResponse> contains the elements necessary for the SPPF 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 SPPF Delete response are described as follows:

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

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

for a given SPPF server.

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

- o dtlResult: An optional response code, response message, and ObjKeyType (as defined in the framework 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 SPPF, 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 "Framework Data Model Objects" section of the framework 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 SPPF Accept operation is wrapped within the <spppAcceptRequest> element while an SPPF Accept response is wrapped within the generic <spppAcceptResponse> element. The following sub-sections describe the spppAcceptRequest and spppAcceptResponse elements. Refer the "SPPF 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 SPP Protocol over SOAP Accept request definition is contained within the generic <sppAcceptRequest> element.

```
<element name="spppAcceptRequest">
 <complexType>
  <sequence>
   <element name="clientTransId"
   type="sppfb:TransIdType" minOccurs="0"/>
   <element name="minorVer"
   type="sppfb:MinorVerType" minOccurs="0"/>
   <element name="rteGrpOfferKey"
   type="sppfs: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 SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF server can be retrieved by the client using the SPPF 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 SPPF 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 SPPF 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 SPPF 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 SPP Protocol over SOAP 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="sppfb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="sppfb:TransIdType"/>
      <element name="overallResult" type="sppfb:ResultCodeType"/>
      <element name="dtlResult"
      type="sppfs: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="sppfs:ResultCodeType">
    <sequence>
     <element name="rteGrpOfferKey" type="sppfs:RteGrpOfferKeyType"/>
    </sequence>
   </extension>
  </complexContent>
</complexType>
```

An <spppAcceptResponse> contains the elements necessary for the SPPF 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 SPPF Accept response are described as follows:

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

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

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

- o 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 SPPF, Route Group Offer can be accepted or rejected by, or on behalf of, the registrant to whom the Route Group has been offered (refer "Framework 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 SPPF Reject operation is wrapped within the <spppRejectRequest> element while an SPPF Reject response is wrapped within the generic <spppRejecResponse> element. The following sub-sections describe the spppRejectRequest and spppRejecResponse elements. Refer the "SPPF SOAP Examples" section of this document for an example of Reject operation on a Route Group Offer.

6.2.4.1. Reject Request

An SPP Protocol over SOAP Reject request definition is contained within the generic <spppRejectRequest> element.

```
<element name="spppRejectRequest">
 <complexType>
  <sequence>
   <element name="clientTransId"
   type="sppfb:TransIdType" minOccurs="0"/>
   <element name="minorVer"
   type="sppfb:MinorVerType" minOccurs="0"/>
   <element name="rteGrpOfferKey"
   type="sppfs: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 SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF server can be retrieved by the client using the SPPF 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 SPPF 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 SPPF 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 SPPF 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 SPP Protocol over SOAP 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="sppfb:TransIdType"
       minOccurs="0"/>
      <element name="serverTransId" type="sppfb:TransIdType"/>
      <element name="overallResult" type="sppfb:ResultCodeType"/>
      <element name="dtlResult"
      type="sppfs: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="sppfs:ResultCodeType">
    <sequence>
     <element name="rteGrpOfferKey" type="sppfs:RteGrpOfferKeyType"/>
    </sequence>
   </extension>
  </complexContent>
</complexType>
```

An <spppRejectResponse> contains the elements necessary for the SPPF

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 SPPF 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 SPPF client passed into the SPPF update request. When included in the request, the SPPF 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 SPPF 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 SPP Protocol over SOAP Batch request XML structure allows the SPPF client to send any of of Add, Del, Accept or Reject operations together in one single request. This gives an SPPF 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 SPPF Batch response is wrapped within the <spppBatchResponse> element. This following sub-sections describe the spppBatchRequest and spppBatchResponse elements. Refer the "SPPF SOAP Examples" section of this document for an example of a batch operation.

6.2.5.1. Batch Request Structure

An SPP Protocol over SOAP Batch request definition is contained within the generic <spppBatchRequest> element.

```
<element name="spppBatchRequest">
<complexType>
 <sequence>
  <element name="clientTransId"
  type="sppfb:TransIdType" minOccurs="0"/>
  <element name="minorVer"
  type="sppfb:MinorVerType" minOccurs="0"/>
   <choice minOccurs="1" maxOccurs="unbounded">
    <element name="addObj" type="sppfb:BasicObjType"/>
    <element name="delObj" type="sppfb:ObjKeyType"/>
    <element name="acceptRteGrpOffer"
    type="sppfs:RteGrpOfferKeyType"/>
    <element name="rejectRteGrpOffer"
    type="sppfs: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 SPPF client, identifies this request. This value can be used at the discretion of the SPPF client to track, log or correlate requests and their responses. SPPF server MUST echo back this value to the client in the corresponding response to the incoming request. SPPF server will not check this value for uniqueness.

- minorVer: Zero or one minor version identifier, indicating the minor version of the SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF server can be retrieved by the client using the SPPF 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 .

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

- o 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 SPPF 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 SPPF 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 SPP Protocol over SOAP 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 SPPF operations.

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

An <spppBatchResponse> contains the elements necessary for an SPPF 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 SPPF 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 SPPF client passed into the SPPF update request. When included in the request, the SPPF 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 SPPF 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 SPPF Get operation is contained within

the generic <spppGetRequest> element while an SPPF 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 SPPF object

6.2.6.1. Get Request

```
<element name="spppGetRequest">
 <complexType>
  <sequence>
   <element name="minorVer"
   type="sppfb:MinorVerType" minOccurs="0"/>
   <element name="objKey"
   type="sppfb: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 SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF server can be retrieved by the client using the SPPF server menu operation described later in the document.

- objKey: One or more elements of abstract type ObjKeyType (as defined in the framework 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 SPPF 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="sppfb:MinorVerType"
  minOccurs="0"/>
 <element name="offeredBy" type="sppfb:OrgIdType"
 minOccurs="0" maxOccurs="unbounded"/>
 <element name="offeredTo" type="sppfb:OrgIdType"
 minOccurs="0" maxOccurs="unbounded"/>
 <element name="status" type="sppfb:RteGrpOfferStatusType"
  minOccurs="0"/>
 <element name="rteGrpOfferKey" type="sppfs:RteGrpOfferKeyType"
 minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
</element>
```

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

- o minorVer: Zero or one minor version identifier, indicating the minor version of the SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF server can be retrieved by the client using the SPPF server menu operation described later in the document.

- o 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.

- o 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.

- o 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.

- o 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 SPP Protocol over SOAP query response object is contained within the generic <spppGetResponse> element.

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

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

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

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

- o 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 SPPF server, like the SPPF 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 SPPF 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="sppfb:MinorVerType" minOccurs="0"/>
  </sequence>
 </complexType>
</element>
```

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

o minorVer: Zero or one minor version identifier, indicating the minor version of the SPPF 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 SPPF 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 SPPF server for the given major version. The versions supported by a given SPPF 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 SPP Protocol over SOAP server details response structure is contained within the generic <spppServerStatusResponse> element.

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

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

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

- o svcMenu: Exactly one element of type SvcMenuType which in turn contains the elements to return the server status and major/minor version of the SPPF supported by the SPPF server (refer framework 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 framework document.

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

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

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

- o 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 SPPF responses, or object level response codes that may only be returned in the "dtlResult" element of the SPPF 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 framework document.

7. Protocol Operations

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

8. SPPF SOAP WSDL Definition

The SPPF 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 SPPF protocol. There is a single WSDL structure for each type of SPPF 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:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
xmlns:sppfs="urn:ietf:params:xml:ns:sppf:soap:1"
targetNamespace="urn:ietf:params:xml:ns:sppf:soap:1">
 <wsdl:types>
  <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sppfs="urn:ietf:params:xml:ns:sppf:soap:1"
  targetNamespace="urn:ietf:params:xml:ns:sppf:soap:1">
   <annotation>
    <documentation>
                                          ---- Import base schema ----
                                  </documentation>
   </annotation>
   <import namespace="urn:ietf:params:xml:ns:sppf:base:1"
   schemaLocation="sppfbase.xsd"/>
   <annotation>
    <documentation>
                                          ---- Key type(s) extended
                                          from base schema. ----
                                  </documentation>
   </annotation>
   <complexType name="ObjKeyType">
    <complexContent>
```

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

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

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

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

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

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

```
        </sequence>
       </complexType>
      </element>
      <element name="spppServerStatusResponse">
       <complexType>
        <sequence>
         <element name="overallResult"
         type="sppfs:ResultCodeType"/>
         <element name="svcMenu"
         type="sppfb: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="sppfs:ResultCodeType">
         <sequence>
          <element name="obj" type="sppfb:BasicObjType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
      <complexType name="ObjKeyResultCodeType">
       <complexContent>
        <extension base="sppfs:ResultCodeType">
         <sequence>
          <element name="objKey" type="sppfb:ObjKeyType"/>
         </sequence>
        </extension>
       </complexContent>
      </complexType>
         <complexType name="RteGrpOfferKeyResultCodeType">
       <complexContent>
        <extension base="sppfs:ResultCodeType">
         <sequence>
          <element name="rteGrpOfferKey"
```

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

```
    </wsdl:message>
    <wsdl:message name="spppServerStatusResponseMsg">
     <wsdl:part name="rspns" element="sppfs:spppServerStatusResponse"/>
    </wsdl:message>
    <wsdl:portType name="spppPortType">
     <wsdl:operation name="submitAddRqst">
      <wsdl:input message="sppfs:spppAddRequestMsg"/>
      <wsdl:output message="sppfs:spppAddResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitDelRqst">
      <wsdl:input message="sppfs:spppDelRequestMsg"/>
      <wsdl:output message="sppfs:spppDelResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitAcceptRqst">
      <wsdl:input message="sppfs:spppAcceptRequestMsg"/>
      <wsdl:output message="sppfs:spppAcceptResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitRejectRqst">
      <wsdl:input message="sppfs:spppRejectRequestMsg"/>
      <wsdl:output message="sppfs:spppRejectResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitBatchRqst">
      <wsdl:input message="sppfs:spppBatchRequestMsg"/>
      <wsdl:output message="sppfs:spppBatchResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitGetRqst">
      <wsdl:input message="sppfs:spppGetRequestMsg"/>
      <wsdl:output message="sppfs:spppGetResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitGetRteGrpOffersRqst">
      <wsdl:input message="sppfs:spppGetRteGrpOffersRequestMsg"/>
      <wsdl:output message="sppfs:spppGetResponseMsg"/>
     </wsdl:operation>
     <wsdl:operation name="submitServerStatusRqst">
      <wsdl:input message="sppfs:spppServerStatusRequestMsg"/>
      <wsdl:output message="sppfs:spppServerStatusResponseMsg"/>
     </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="spppSoapBinding" type="sppfs: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="sppfs:spppSoapBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
     </wsdl:port>
    </wsdl:service>
   </wsdl:definitions>
```

Figure 2: WSDL

9. SPP Protocol over SOAP Examples

This section shows XML message exchange between two SIP Service Providers (SSP) and a registry. The messages in this section are valid XML instances that conform to the SPP Protocol over SOAP schema version within this document. This section relies on the XML data structures defined in the base SPPF specification [I-D.draft-ietf-drinks-spp-framework]. 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 SPPF 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 SPPF to associate route preferences for the destination ingress routes and exercise greater control on outbound traffic to the peer's ingress SBEs.

```
---------------+                     +------------------
               |                     |
           +------+              +------+
           | sbe1 |              | sbe2 |
           +------+              +------+
 SSP1          |                     |           SSP2
           +------+              +------+
           | sbe3 |              | sbe4 |
           +------+              +------+
iana-en:111    |                     |     iana-en:222
---------------+                     +------------------
        |                                     |
        |                                     |
        | SPPF   +------------------+   SPPF  |
        +------->|     Registry     |<--------+
                 +------------------+
```

9.1. Add Destination Group

SSP2 adds a destination group to the registry for use later. The SSP2 SPPF 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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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 SPPF 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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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 SPPF 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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
      xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
       xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
        xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppf: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:sppf: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:sppf:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppf: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 SPPF client as a batch in one single SPP Protocol over SOAP 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:sppf:soap:1"
xmlns:urn1="urn:ietf:params:xml:ns:sppf: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:sppf:base:1"
                xmlns:ns3="urn:ietf:params:xml:ns:sppf:soap:1">
         <serverTransId>tx_12354</serverTransId>
         <overallResult>
            <code>1000</code>
            <msg>Request Succeeded.</msg>
         </overallResult>
      </ns3:spppBatchResponse>
   </S:Body>
</S:Envelope>
```

10. Security Considerations

SPPF 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]. SPPF data may include sensitive information, routing data, lists of resolvable addresses, etc. So when used in a production setting and across non-secure networks, SPPF should only be used over communications channels that provide strong encryption for data privacy.

10.1. Integrity, Privacy, and Authentication

The SPPF 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 SPPF 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 SPP Protocol over SOAP implementations MUST support HTTPS.

10.2. Vulnerabilities

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

It is important that SPPF implementations implement an authorization model that considers the source of each SPPF 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 SPP Protocol 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 SPPF 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:sppf: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 alphabetical order: Alexander Mayrhofer, David Schwartz, Deborah A Guyton, Jean-Francois Mule Kenneth Cartwright, Lisa Dusseault, Manjul Maharishi, Mickael Marrache, Otmar Lendl, Peter Saint-Andre, Richard Shockey, Samuel Melloul, Sumanth Channabasappa, Syed Ali, and Vikas Bhatia .

13. References

13.1. Normative References

[I-D.draft-ietf-drinks-spp-framework]
Mule, J-F., Cartwright, K., Ali, S., Mayrhofer, A., and V. Bhatia, "Session Peering Provisioning Framework", draft-ietf-drinks-spp-framework-00 (work in progress), January 2012.

[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