Internet Engineering Task Force (IETF)
Request for Comments: 7877
Category: Standards Track
ISSN: 2070-1721

K. Cartwright
V. Bhatia
TNS
S. Ali
NeuStar
D. Schwartz
XConnect

August 2016

Session Peering Provisioning Framework (SPPF)

Abstract

This document specifies the data model and the overall structure for a framework to provision Session Establishment Data (SED) into Session Data Registries and SIP Service Provider (SSP) data stores. The framework is called the "Session Peering Provisioning Framework" (SPPF). The provisioned data is typically used by network elements for session establishment.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7877.

Table of Contents

1. Introduction 3
2. Terminology 6
3. Framework High-Level Design 7
3.1. Framework Data Model 7
3.2. Time Value 10
3.3. Extensibility 10
4. Substrate Protocol Requirements 11
4.1. Mandatory Substrate 11
4.2. Connection Oriented 11
4.3. Request and Response Model 11
4.4. Connection Lifetime 11
4.5. Authentication 12
4.6. Authorization 12
4.7. Confidentiality and Integrity 12
4.8. Near Real Time 12
4.9. Request and Response Sizes 12
4.10. Request and Response Correlation 13
4.11. Request Acknowledgement 13
5. Base Framework Data Structures and Response Codes 13
5.1. Basic Object Type and Organization Identifiers 13
5.2. Various Object Key Types 14
5.2.1. Generic Object Key Type 14
5.2.2. Derived Object Key Types 15
5.3. Response Message Types 16
6. Framework Data Model Objects 18
6.1. Destination Group 18
6.2. Public Identifier 19
6.3. SED Group 25
6.4. SED Record 29
6.5. SED Group Offer 33
6.6. Egress Route 35

7. Framework Operations 36
7.1. Add Operation 37
7.2. Delete Operation 37
7.3. Get Operations 38
7.4. Accept Operations 38
7.5. Reject Operations 39
7.6. Get Server Details Operation 39
8. XML Considerations 40
8.1. Namespaces 40
8.2. Versioning and Character Encoding 40
9. Security Considerations 41
9.1. Confidentiality and Authentication 41
9.2. Authorization 41
9.3. Denial of Service 41
9.3.1. DoS Issues Inherited from the Substrate Mechanism . . 42
9.3.2. DoS Issues Specific to SPPF 42
9.4. Information Disclosure 43
9.5. Non-repudiation 43
9.6. Replay Attacks 43
9.7. Compromised or Malicious Intermediary 44
10. Internationalization Considerations 44
11. IANA Considerations 44
11.1. URN Assignments 44
11.2. Organization Identifier Namespace Registry 45
12. Formal Specification 45
13. References 54
13.1. Normative References 54
13.2. Informative References 55
Acknowledgements 57
Authors' Addresses 57

1. Introduction

Service Providers (SPs) and enterprises use routing databases known as Registries to make session routing decisions for Voice over IP, SMS, and Multimedia Messaging Service (MMS) traffic exchanges. This document is narrowly focused on the provisioning framework for these Registries. This framework prescribes a way for an entity to provision session-related data into a Session Peering Provisioning Protocol (SPPP) Registry (or "Registry"). The data being provisioned can be optionally shared with other participating peering entities. The requirements and use cases driving this framework have been documented in [RFC6461].

Three types of provisioning flows have been described in the use case document: client to Registry, Registry to local data repository, and Registry to Registry. This document addresses client-to-Registry flow enabling the ability to provision Session Establishment Data

(SED). The framework that supports the flow of messages to facilitate client-to-Registry provisioning is referred to as the "Session Peering Provisioning Framework" (SPPF).

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

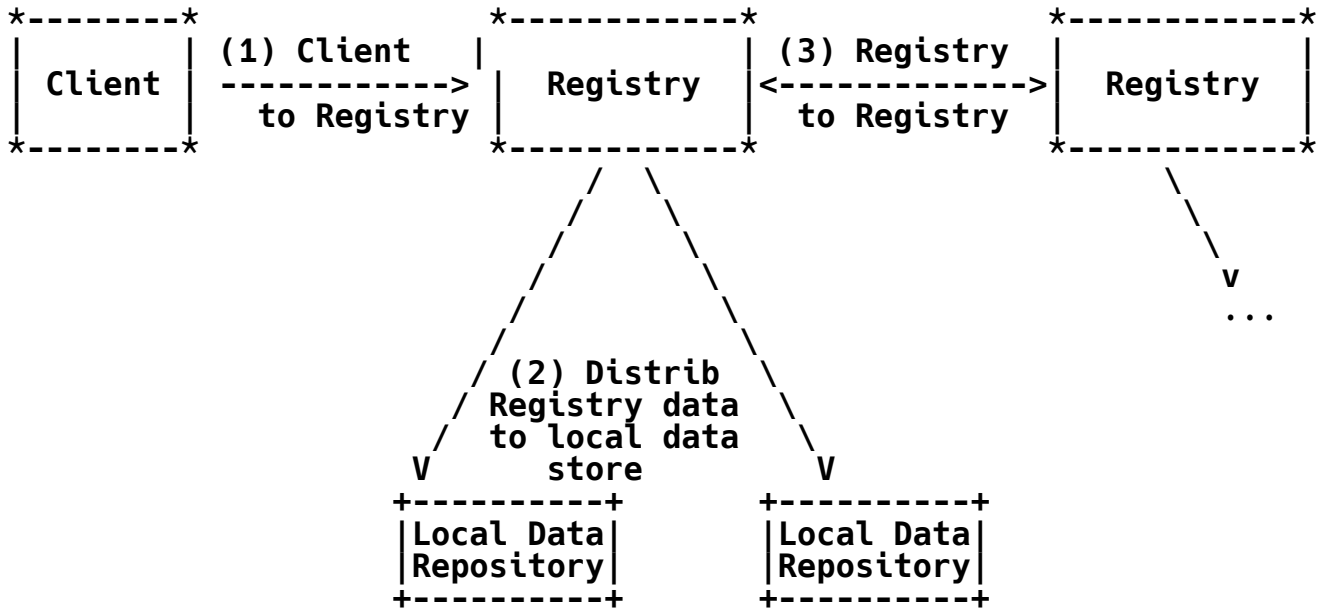

Figure 1: Three Registry Provisioning Flows

A "terminating" SSP provisions SED into the Registry to be selectively shared with other peer SSPs.

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

A Registry may distribute the provisioned data into local data repositories or may additionally offer a central query-resolution service (not shown in the above figure) for query purposes.

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

1. A resolution system returns a Lookup Function (LUF) that identifies the target domain to assist in call routing (as described in Section 4.3.3 of [RFC5486]). In this case, the querying entity may use other means to perform the Location Routing Function (LRF), which in turn helps determine the actual location of the Signaling Function in that domain.

2. A resolution system returns an LRF that comprises the location (address) of the Signaling Function in the target domain (as described in [RFC5486]).

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

In this document, an XML Schema is used to describe the building blocks of the SPPF and to express the data types, semantic relationships between the various data types, and various constraints as a binding construct. However, a "protocol" specification is free to choose any data representation format as long as it meets the requirements laid out in the SPPF XML Schema Definition (XSD). As an example, XML and JSON are two widely used data representation formats.

This document is organized as follows:

- Section 2 provides the terminology
- Section 3 provides an overview of SPPF, including functional entities and a data model
- Section 4 specifies requirements for SPPF substrate protocols
- Section 5 describes the base framework data structures, the generic response types that MUST be supported by a conforming substrate "protocol" specification, and the basic object type from which most first-class objects extend
- Section 6 provides a detailed description of the data model object specifications

- Section 7 describes the operations that are supported by the data model
- Section 8 defines XML considerations XML parsers must meet to conform to this specification
- Sections 9 - 11 discuss security, internationalization, and IANA considerations, respectively
- Section 12 normatively defines the SPPF using its XSD.

2. Terminology

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

This document reuses terms from [RFC3261], [RFC5486], use cases and requirements documented in [RFC6461], and the ENUM Validation Architecture [RFC4725].

This document defines the following additional terms:

SPPF: Session Peering Provisioning Framework, which is the framework used by a substrate protocol to provision data into a Registry (see arrow labeled "1" in Figure 1 of [RFC6461]). It is the primary scope of this document.

Client: In the context of SPPF, this is an application that initiates a provisioning request. It is sometimes referred to as a "Registry client".

Server: In the context of SPPF, this is an application that receives a provisioning request and responds accordingly.

Registry: The Registry operates a master database of SED for one or more Registrants.

Registrant: The definition of a Registrant is based on [RFC4725]. It is the end user, person, or organization that is the "holder" of the SED being provisioned into the Registry by a Registrar. For example, in [RFC6461], a Registrant is pictured as an SP in Figure 2.

Within the confines of a Registry, a Registrant is uniquely identified by the "rant" element.

Registrar: The definition of a Registrar is based on [RFC4725]. It is an entity that performs provisioning operations on behalf of a Registrant by interacting with the Registry via SPPF operations. In other words, the Registrar is the SPPF client. The Registrar and Registrant roles are logically separate to allow, but not require, a single Registrar to perform provisioning operations on behalf of more than one Registrant.

Peering Organization: A peering organization is an entity to which a Registrant's SED Groups are made visible using the operations of SPPF.

3. Framework High-Level Design

This section introduces the structure of the data model and provides the information framework for the SPPF. The data model is defined along with all the objects manipulated by a conforming substrate protocol and their relationships.

3.1. Framework Data Model

The data model illustrated and described in Figure 2 defines the logical objects and the relationships between these objects supported by SPPF. SPPF defines protocol operations through which an SPPF client populates a Registry with these logical objects. SPPF clients belonging to different Registrars may provision data into the Registry using a conforming substrate protocol that implements these operations

The logical structure presented below is consistent with the terminology and requirements defined in [RFC6461].

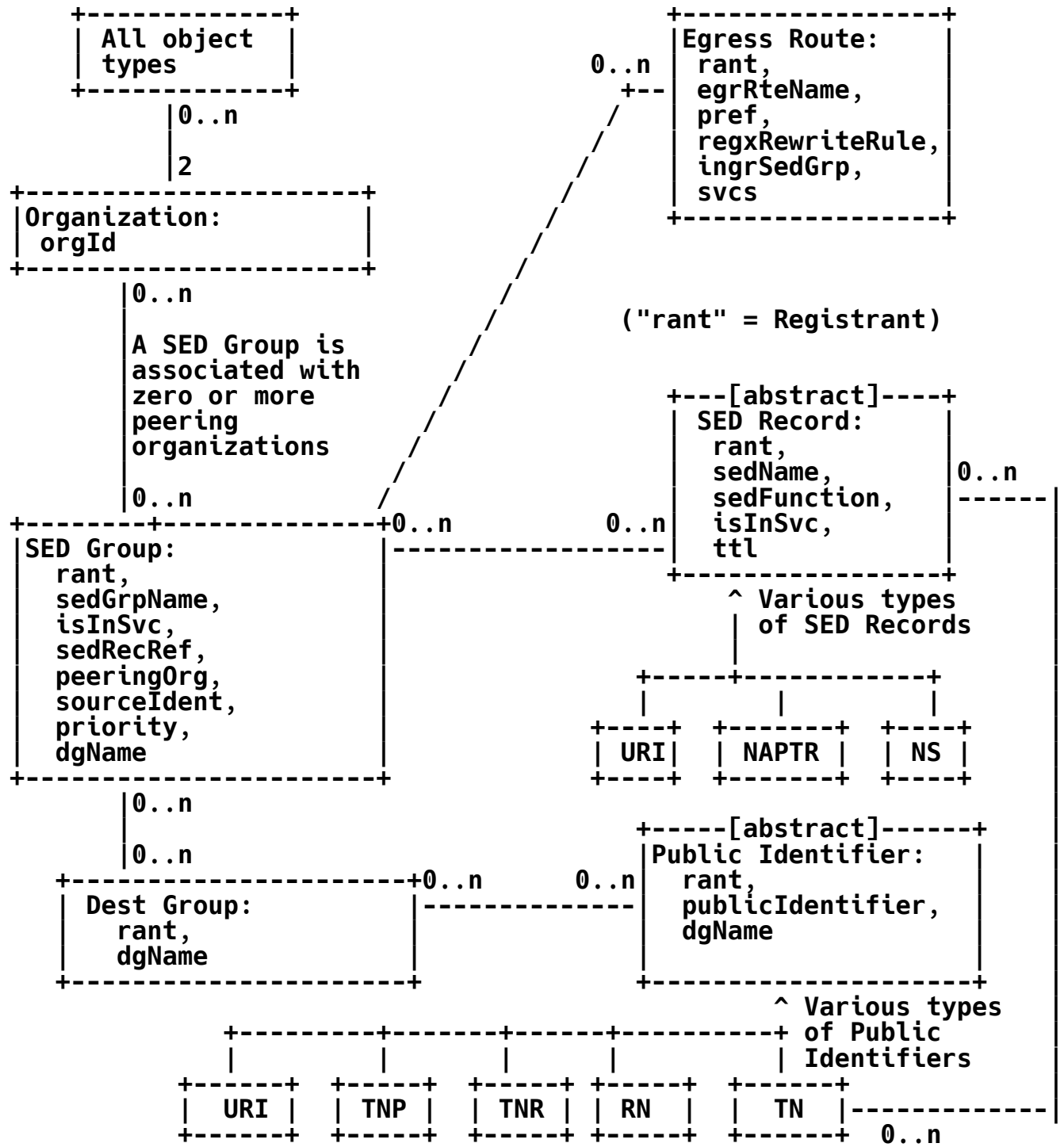

Figure 2: Framework Data Model

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

- Public Identifier:
  From a broad perspective, a Public Identifier is a well-known attribute that is used as the key to perform resolution lookups. Within the context of SPPF, a Public Identifier object can be a Telephone Number (TN), a range of TNs, a Public Switched Telephone Network (PSTN) Routing Number (RN), a TN prefix, or a URI.

  An SPPF Public Identifier may be a member of zero or more Destination Groups to create logical groupings of Public Identifiers that share a common set of SED (e.g., routes).

  A TN Public Identifier may optionally be associated with zero or more individual SED Records. This ability for a Public Identifier to be directly associated with a SED Record, as opposed to forcing membership in one or more Destination Groups, supports use cases where the SED Record contains data specifically tailored to an individual TN Public Identifier.

- Destination Group:
  A named logical grouping of zero or more Public Identifiers that can be associated with one or more SED Groups for the purpose of facilitating the management of their common SED.

- SED Group:
  A SED Group contains a set of SED Record references, a set of Destination Group references, and a set of peering organization identifiers. This is used to establish a three-part relationship between a set of Public Identifiers, the SED shared across these Public Identifiers, and the list of peering organizations whose query responses from the resolution system may include the SED contained in a given SED Group. In addition, the sourceIdent element within a SED Group, in concert with the set of peering organization identifiers, enables fine-grained source-based routing. For further details about the SED Group and source-based routing, refer to the definitions and descriptions in Section 6.1.

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

  To support the use cases defined in [RFC6461], the SPPF defines three types of SED Records: URIType, NAPTRType, and NSType. These SED Records extend the abstract type SedRecType and inherit the

common attribute "priority" that is meant for setting precedence across the SED Records defined within a SED Group in a protocol-agnostic fashion.

- Egress Route:
  In a high-availability environment, the originating SSP likely has more than one egress path to the ingress SBE of the target SSP. The Egress Route allows the originating SSP to choose a specific egress SBE to be associated with the target ingress SBE. The "svcs" element specifies ENUM services (e.g., E2U+pstn:sip+sip) that are used to identify the SED Records associated with the SED Group that will be modified by the originating SSP.

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

3.2. Time Value

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

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

3.3. Extensibility

The framework contains various points of extensibility in the form of the "ext" elements. Extensions used beyond the scope of private SPPF installations need to be documented in an RFC, and the first such extension is expected to define an IANA registry, holding a list of documented extensions.

4. Substrate Protocol Requirements

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

4.1. Mandatory Substrate

None of the existing transport protocols carried directly over IP, appearing as "Protocol" in the IPv4 headers or "Next Header" in the IPv6 headers, meet the requirements listed in this section to carry SPPF.

Therefore, one choice to carry SPPF has been provided in "Session Peering Provisioning (SPP) Protocol over SOAP" [RFC7878], using SOAP as the substrate. To encourage interoperability, the SPPF server MUST provide support for this protocol. With time, it is possible that other choices may surface that comply with the requirements discussed above.

4.2. Connection Oriented

The SPPF follows a model where a client establishes a connection to a server in order to further exchange SPPF messages over such a point-to-point connection. Therefore, a substrate protocol for SPPF will be connection oriented.

4.3. Request and Response Model

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

Therefore, a substrate protocol for SPPF will follow the request-response model by ensuring a response is sent to the request initiator.

4.4. Connection Lifetime

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

Therefore, a protocol suitable for SPPF SHOULD be able to support both short-lived and long-lived connections.

4.5. Authentication

All SPPF objects are associated with a Registrant identifier. An SPPF client provisions SPPF objects on behalf of Registrants. An authenticated SPP client is a Registrar. Therefore, the SPPF substrate protocol MUST provide means for an SPPF server to authenticate an SPPF client.

4.6. Authorization

After successful authentication of the SPPF client as a Registrar, the Registry performs authorization checks to determine if the Registrar is authorized to act on behalf of the Registrant whose identifier is included in the SPPF request. Refer to Section 9 for further guidance.

4.7. Confidentiality and Integrity

SPPF objects that the Registry manages can be private in nature. Therefore, the substrate protocol MUST provide means for data integrity protection.

If the data is compromised in-flight between the SPPF client and Registry, it will seriously affect the stability and integrity of the system. Therefore, the substrate protocol MUST provide means for data integrity protection.

4.8. Near Real Time

Many use cases require responses in near real time from the server (in the range of a few multiples of round-trip time between the server and client). Therefore, a Data for Reachability of Inter-/Intra-NetworK SIP (DRINKS) substrate protocol MUST support near real-time responses to requests submitted by the client.

4.9. Request and Response Sizes

Use of SPPF may involve simple updates that may consist of a small number of bytes, such as the update of a single Public Identifier. Other provisioning operations may constitute a large dataset, as in adding millions of records to a Registry. As a result, a suitable substrate protocol for SPPF SHOULD accommodate datasets of various sizes.

4.10. Request and Response Correlation

A substrate protocol suitable for SPPF MUST allow responses to be correlated with requests.

4.11. Request Acknowledgement

Data transported in the SPPF is likely crucial for the operation of the communication network that is being provisioned. An SPPF client responsible for provisioning SED to the Registry has a need to know if the submitted requests have been processed correctly.

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

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

5. Base Framework Data Structures and Response Codes

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

5.1. Basic Object Type and Organization Identifiers

All first-class objects extend the type BasicObjType. It consists of the Registrant organization, the Registrar organization, the date and time of object creation, and the last date and time the object was modified. The Registry MUST store the date and time of the object creation and modification, if applicable, for all Get operations (see Section 7). If the client passed in either date or time values, the Registry MUST ignore it. The Registrar performs the SPPF operations on behalf of the Registrant, the organization that owns the object.

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

The identifiers used for Registrants (rant) and Registrars (rar) are instances of OrgIdType. The OrgIdType is defined as a string and all OrgIdType instances MUST follow the textual convention: "namespace:value" (for example, "iana-en:32473"). Specifically:

Strings used as OrgIdType Namespace identifiers MUST conform to the following syntax in the Augmented Backus-Naur Form (ABNF) [RFC5234].

```
namespace = ALPHA *(ALPHA/DIGIT/"-")
```

See Section 11 for the corresponding IANA registry definition.

5.2. Various Object Key Types

The SPPF data model contains various object relationships. In some cases, these object relationships are established by embedding the unique identity of the related object inside the relating object. Note that an object's unique identity is required to Delete or Get the details of an object. The following subsections normatively define the various object keys in SPPF and the attributes of those keys.

"Name" attributes that are used as components of object key types MUST be compared using the toCasefold() function, as specified in Section 3.13 of [Unicode6.1] (or a newer version of Unicode). This function performs case-insensitive comparisons.

5.2.1. Generic Object Key Type

Most objects in SPPF are uniquely identified by an object key that has the object's name, type, and Registrant's organization ID as attributes. The abstract type called ObjKeyType is where this unique identity is housed. Any concrete representation of the ObjKeyType MUST contain the following:

Object Name: The name of the object.

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

Type: The value that represents the type of SPPF object. This is required as different types of objects in SPPF, that belong to the same Registrant, can have the same name.

The structure of abstract ObjKeyType is as follows:

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

5.2.2. Derived Object Key Types

The SPPF data model contains certain objects that are uniquely identified by attributes, different from or in addition to the attributes in the generic object key described in the previous section. Object keys of this kind are derived from the abstract ObjKeyType and defined in their own abstract key types. Because these object key types are abstract, they MUST be specified in a concrete form in any SPPF-conforming substrate "protocol" specification. These are used in Delete and Get operations and may also be used in Accept and Reject operations.

Following are the derived object keys in an SPPF data model:

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

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

A SED Group Offer object MUST use SedGrpOfferKeyType. Refer to Section 6.5 for a description of the SED Group Offer object.

- PubIdKeyType: This uniquely identifies a Public Identity object. This key type extends from the abstract ObjKeyType. Any concrete definition of PubIdKeyType MUST contain the elements that identify the value and type of Public Identity and also contain the organization ID of the Registrant that is the owner of the Public Identity object. A Public Identity object in SPPF is uniquely identified by the Registrant's organization ID, the value of the Public Identity, and the type of the Public Identity object. Consequently, any concrete representation of the PubIdKeyType MUST contain the following attributes:

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

  * Value: The value of the Public Identity.

  * Type: The type of the Public Identity object.

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

- The structure of abstract PubIdKeyType is as follows:

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

A Public Identity object MUST use attributes of PubIdKeyType for its unique identification. Refer to Section 6 for a description of a Public Identity object.

5.3. Response Message Types

The following table contains the list of response types that MUST be defined for a substrate protocol used to carry SPPF. An SPPF server MUST implement all of the following at minimum.

Response Type	Description
Request succeeded	A given request succeeded.
Request syntax invalid	The syntax of a given request was found to be invalid.
Request too large	The count of entities in the request is larger than the server is willing or able to process.
Version not supported	The server does not support the version of the SPPF protocol specified in the request.
Command invalid	The operation and/or command being requested by the client is invalid and/or not supported by the server.
System temporarily unavailable	The SPPF server is temporarily not available to serve the client request.
Unexpected internal system or server error	The SPPF server encountered an unexpected error that prevented the server from fulfilling the request.
Attribute value invalid	The SPPF server encountered an attribute or property in the request that had an invalid/bad value. Optionally, the specification MAY provide a way to indicate the Attribute Name and the Attribute Value to identify the object that was found to be invalid.
Object does not exist	An object present in the request does not exist on the SPPF server. Optionally, the specification MAY provide a way to indicate the Attribute Name and the Attribute Value that identifies the nonexistent object.
Object status or ownership does not allow for operation	The operation requested on an object present in the request cannot be performed because the object is in a status that does not allow said operation, or the user requesting the operation is not authorized to perform said operation on the object. Optionally, the specification MAY provide a way to indicate the Attribute Name and the Attribute Value that identifies the object.

Table 1: Response Types

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

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

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

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

- Response types "Attribute value invalid" and "Object does not exist" MUST NOT be used interchangeably. Response type "Object does not exist" MUST be returned by an Update/Del/Accept/Reject operation when the data element(s) used to uniquely identify a preexisting object does not exist. If the data elements used to uniquely identify an object are malformed, then response type "Attribute value invalid" MUST be returned.

6. Framework Data Model Objects

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

6.1. Destination Group

A Destination Group represents a logical grouping of Public Identifiers with common SED. The substrate protocol MUST support the ability to Add, Get, and Delete Destination Groups (refer to Section 7 for a generic description of various operations).

A Destination Group object MUST be uniquely identified by attributes as defined in the description of "ObjKeyType" in "Generic Object Key Type" (Section 5.2.1 of this document).

The DestGrpType object structure is defined as follows:

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

The DestGrpType object is composed of the following elements:

- base: All first-class objects extend BasicObjType (see Section 5.1).
- dgName: The character string that contains the name of the Destination Group.
- ext: Point of extensibility described in Section 3.3.

6.2. Public Identifier

A Public Identifier is the search key used for locating the SED. In many cases, a Public Identifier is attributed to the end user who has a retail relationship with the SP or Registrant organization. SPPF supports the notion of the carrier-of-record as defined in [RFC5067]. Therefore, the Registrant under which the Public Identifier is being created can optionally claim to be a carrier-of-record.

SPPF identifies three types of Public Identifiers: TNs, RNs, and URIs. SPPF provides structures to manage a single TN, a contiguous range of TNs, and a TN prefix. The substrate protocol MUST support the ability to Add, Get, and Delete Public Identifiers (refer to Section 7 for a generic description of various operations).

A Public Identity object MUST be uniquely identified by attributes as defined in the description of "PubIdKeyType" in Section 5.2.2.

The abstract XSD type PubIdType is a generalization for the concrete Public Identifier schema types. The PubIdType element "dgName" represents the name of a Destination Group of which a given Public Identifier may be a member. Note that this element may be present multiple times so that a given Public Identifier may be a member of multiple Destination Groups. The PubIdType object structure is defined as follows:

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

A Public Identifier may be a member of zero or more Destination Groups. When a Public Identifier is a member of a Destination Group, it is intended to be associated with SED through the SED Group(s) that is associated with the Destination Group. When a Public Identifier is not member of any Destination Group, it is intended to be associated with SED through the SED Records that are directly associated with the Public Identifier.

A TN is provisioned using the TNType, an extension of PubIdType. Each TNType object is uniquely identified by the combination of its value contained within the <tn> element and its Registrant ID. TNType is defined as follows:

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

<complexType name="CORInfoType">
 <sequence>
   <element name="corClaim" type="boolean" default="true"/>
   <element name="cor" type="boolean" default="false" minOccurs="0"/>
   <element name="corDate" type="dateTime" minOccurs="0"/>
 </sequence>
</complexType>

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

TNType consists of the following attributes:

- tn: Telephone number to be added to the Registry.

- sedRecRef: Optional reference to SED Records that are directly associated with the TN Public Identifier. Following the SPPF data model, the SED Record could be a protocol-agnostic URIType or another type.

- corInfo: corInfo is an optional parameter of type CORInfoType that allows the Registrant organization to set forth a claim to be the carrier-of-record (see [RFC5067]). This is done by setting the value of the <corClaim> element of the CORInfoType object structure to "true". The other two parameters of the CORInfoType, <cor> and <corDate>, are set by the Registry to describe the

outcome of the carrier-of-record claim by the Registrant. In general, inclusion of the <corInfo> parameter is useful if the Registry has the authority information, such as the number portability data, etc., in order to qualify whether the Registrant claim can be satisfied. If the carrier-of-record claim disagrees with the authority data in the Registry, whether or not a TN Add operation fails is a matter of policy and is beyond the scope of this document.

An RN is provisioned using the RNType, an extension of PubIDType. The Registrant organization can add the RN and associate it with the appropriate Destination Group(s) to share the route information. This allows SSPs to use the RN search key to derive the Ingress Routes for session establishment at the runtime resolution process (see [RFC6116]). Each RNType object is uniquely identified by the combination of its value inside the <rn> element and its Registrant ID. RNType is defined as follows:

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

RNType has the following attributes:

- rn: The RN used as the search key.

- corInfo: corInfo is an optional parameter of type CORInfoType that allows the Registrant organization to set forth a claim to be the carrier-of-record (see [RFC5067]).

TNRType structure is used to provision a contiguous range of TNs. The object definition requires a starting TN and an ending TN that together define the span of the TN range, including the starting and ending TN. Use of TNRType is particularly useful when expressing a TN range that does not include all the TNs within a TN block or prefix. The TNRType definition accommodates the open number plan as well such that the TNs that fall in the range between the start and end TN may include TNs with different length variance. Whether the Registry can accommodate the open number plan semantics is a matter of policy and is beyond the scope of this document. Each TNRType object is uniquely identified by the combination of its value that,

in turn, is a combination of the <startTn> and <endTn> elements and its Registrant ID. The TNRType object structure definition is as follows:

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

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

TNRType has the following attributes:

- startTn: The starting TN in the TN range.

- endTn: The last TN in the TN range.

- corInfo: corInfo is an optional parameter of type CORInfoType that allows the Registrant organization to set forth a claim to be the carrier-of-record (see [RFC5067]).

In some cases, it is useful to describe a set of TNs with the help of the first few digits of the TN, also referred to as the TN prefix or a block. A given TN prefix may include TNs with different length variance in support of the open number plan. Once again, whether the Registry supports the open number plan semantics is a matter of policy, and it is beyond the scope of this document. The TNPType data structure is used to provision a TN prefix. Each TNPType object is uniquely identified by the combination of its value in the <tnPrefix> element and its Registrant ID. TNPType is defined as follows:

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

TNPType consists of the following attributes:

- tnPrefix: The TN prefix.

- corInfo: corInfo is an optional parameter of type CORInfoType that allows the Registrant organization to set forth a claim to be the carrier-of-record (see [RFC5067]).

In some cases, a Public Identifier may be a URI, such as an email address. The URIPubIdType object is comprised of the data element necessary to house such Public Identifiers. Each URIPubIdType object is uniquely identified by the combination of its value in the <uri> element and its Registrant ID. URIPubIdType is defined as follows:

```
<complexType name="URIPubIdType">
 <complexContent>
  <extension base="sppfb:PubIdType">
   <sequence>
    <element name="uri" type="anyURI"/>
    <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

URIPubIdType consists of the following attributes:

- uri: The value that acts as the Public Identifier.
- ext: Point of extensibility described in Section 3.3.

6.3. SED Group

SED Group is a grouping of one or more Destination Groups, the common SED Records, and the list of peer organizations with access to the SED Records associated with a given SED Group. It is this indirect linking of Public Identifiers to their SED that significantly improves the scalability and manageability of the peering data. Additions and changes to SED information are reduced to a single operation on a SED Group or SED Record rather than millions of data updates to individual Public Identifier records that individually contain their peering data. The substrate protocol MUST support the ability to Add, Get, and Delete SED Groups (refer to Section 7 for a generic description of various operations).

A SED Group object MUST be uniquely identified by attributes as defined in the description of "ObjKeyType" in "Generic Object Key Type" (Section 5.2.1 of this document).

The SedGrpType object structure is defined as follows:

```
<complexType name="SedGrpType">
 <complexContent>
  <extension base="sppfb:BasicObjType">
   <sequence>
    <element name="sedGrpName" type="sppfb:ObjNameType"/>
    <element name="sedRecRef" type="sppfb:SedRecRefType"
             minOccurs="0" maxOccurs="unbounded"/>
    <element name="dgName" type="sppfb:ObjNameType"
             minOccurs="0" maxOccurs="unbounded"/>
    <element name="peeringOrg" type="sppfb:OrgIdType"
             minOccurs="0" maxOccurs="unbounded"/>
    <element name="sourceIdent" type="sppfb:SourceIdentType"
             minOccurs="0" maxOccurs="unbounded"/>
    <element name="isInSvc" type="boolean"/>
    <element name="priority" type="unsignedShort"/>
    <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>

<complexType name="SedRecRefType">
 <sequence>
  <element name="sedKey" type="sppfb:ObjKeyType"/>
  <element name="priority" type="unsignedShort"/>
  <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
 </sequence>
</complexType>
```

The SedGrpType object is composed of the following elements:

- base: All first-class objects extend BasicObjType (see Section 5.1).

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

- sedRecRef: Set of zero or more objects of type SedRecRefType that house the unique keys of the SED Records (containing the SED) that the SedGrpType object refers to and their relative priority within the context of this SED Group.

- dgName: Set of zero or more names of DestGrpType object instances. Each dgName name, in association with this SED Group's Registrant ID, uniquely identifies a DestGrpType object instance whose associated Public Identifiers are reachable using the SED housed in this SED Group. An intended side effect of this is that a SED Group cannot provide session establishment information for a Destination Group belonging to another Registrant.

- peeringOrg: Set of zero or more peering organization IDs that have accepted an offer to receive this SED Group's information. Note that this identifier "peeringOrg" is an instance of OrgIdType. The set of peering organizations in this list is not directly settable or modifiable using the addSedGrpsRqst operation. This set is instead controlled using the SED Offer and Accept operations.

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

- isInSvc: A boolean element that defines whether this SED Group is in service. The SED contained in a SED Group that is in service is a candidate for inclusion in resolution responses for Public Identities residing in the Destination Group associated with this SED Group. The session establishment information contained in a SED Group that is not in service is not a candidate for inclusion in resolution responses.

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

- ext: Point of extensibility described in Section 3.3.

As described above, the SED Group contains a set of references to SED Record objects. A SED Record object is based on an abstract type: SedRecType. The concrete types that use SedRecType as an extension base are NAPTRType, NSType, and URIType. The definitions of these types are included in "SED Record" (Section 6.4 of this document).

The SedGrpType object provides support for source-based routing via the peeringOrg data element and more granular source-based routing via the source identity element. The source identity element provides the ability to specify zero or more of the following in association with a given SED Group: a regular expression that is

matched against the resolution client IP address, a regular expression that is matched against the root domain name(s), and/or a regular expression that is matched against the calling party URI(s). The result will be that, after identifying the visible SED Groups whose associated Destination Group(s) contains the lookup key being queried and whose peeringOrg list contains the querying organization's organization ID, the resolution server will evaluate the characteristics of the Source URI, Source IP address, and root domain of the lookup key being queried. The resolution server then compares these criteria against the source identity criteria associated with the SED Groups. The SED contained in SED Groups that have source-based routing criteria will only be included in the resolution response if one or more of the criteria matches the source criteria from the resolution request. The source identity data element is of type SourceIdentType, whose structure is defined as follows:

```
<complexType name="SourceIdentType">
 <sequence>
  <element name="sourceIdentRegex" type="sppfb:RegexType"/>
  <element name="sourceIdentScheme"
           type="sppfb:SourceIdentSchemeType"/>
  <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
 </sequence>
</complexType>

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

The SourceIdentType object is composed of the following data elements:

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

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

- ext: Point of extensibility described in Section 3.3.

6.4. SED Record

SED Group represents a combined grouping of SED Records that define SED. However, SED Records need not be created to just serve a single SED Group. SED Records can be created and managed to serve multiple SED Groups. As a result, a change, for example, to the properties of a network node used for multiple routes would necessitate just a single update operation to change the properties of that node. The change would then be reflected in all the SED Groups whose SED Record set contains a reference to that node. The substrate protocol MUST support the ability to Add, Get, and Delete SED Records (refer to Section 7 for a generic description of various operations).

A SED Record object MUST be uniquely identified by attributes as defined in the description of "ObjKeyType" in "Generic Object Key Type" (Section 5.2.1 of this document).

The SedRecType object structure is defined as follows:

```
<complexType name="SedRecType" abstract="true">
 <complexContent>
  <extension base="sppfb:BasicObjType">
   <sequence>
    <element name="sedName" type="sppfb:ObjNameType"/>
    <element name="sedFunction" type="sppfb:SedFunctionType"
             minOccurs="0"/>
    <element name="isInSvc" type="boolean"/>
    <element name="ttl" type="positiveInteger" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>

<simpleType name="SedFunctionType">
 <restriction base="token">
  <enumeration value="routing"/>
  <enumeration value="lookup"/>
 </restriction>
</simpleType>
```

The SedRecType object is composed of the following elements:

- base: All first-class objects extend BasicObjType (see Section 5.1).

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

- sedFunction: As described in [RFC6461], SED falls primarily into one of two categories or functions: LUF and LRF. To remove any ambiguity as to the function a SED Record is intended to provide, this optional element allows the provisioning party to make its intentions explicit.

- isInSvc: A boolean element that defines whether or not this SED Record is in service. The session establishment information contained in a SED Record that is in service is a candidate for inclusion in resolution responses for TNs that are either directly associated to this SED Record or for Public Identities residing in a Destination Group that is associated to a SED Group, which, in turn, has an association to this SED Record.

- ttl: Number of seconds that an addressing server may cache a particular SED Record.

As described above, SED Records are based on abstract type SedRecType. The concrete types that use SedRecType as an extension base are NAPTRType, NSType, and URIType. The definitions of these types are included below. The NAPTRType object is comprised of the data elements necessary for a Naming Authority Pointer (NAPTR) (see [RFC3403]) that contains routing information for a SED Group. The NSType object is comprised of the data elements necessary for a DNS name server that points to another DNS server that contains the desired routing information. The NSType is relevant only when the resolution protocol is ENUM (see [RFC6116]). The URIType object is comprised of the data elements necessary to house a URI.

The data provisioned in a Registry can be leveraged for many purposes and queried using various protocols including SIP, ENUM, and others. As such, the resolution data represented by the SED Records must be in a form suitable for transport using one of these protocols. In the NAPTRType, for example, if the URI is associated with a Destination Group, the user part of the replacement string <uri> that may require the Public Identifier cannot be preset. As a SIP Redirect, the resolution server will apply <ere> pattern on the input Public Identifier in the query and process the replacement string by substituting any back references in the <uri> to arrive at the final URI that is returned in the SIP Contact header. For an ENUM query, the resolution server will simply return the values of the <ere> and <uri> members of the URI.

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

   <complexType name="NSType">
    <complexContent>
     <extension base="sppfb:SedRecType">
      <sequence>
       <element name="hostName" type="token"/>
       <element name="ipAddr" type="sppfb:IPAddrType"
                minOccurs="0" maxOccurs="unbounded"/>
       <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
      </sequence>
     </extension>
    </complexContent>
   </complexType>

   <complexType name="IPAddrType">
    <sequence>
     <element name="addr" type="sppfb:AddrStringType"/>
     <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
    </sequence>
    <attribute name="type" type="sppfb:IPType" default="IPv4"/>
   </complexType>

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

   <complexType name="URIType">
    <complexContent>
     <extension base="sppfb:SedRecType">
      <sequence>
       <element name="ere" type="token" default="^(.*)$"/>
```

```
      <element name="uri" type="anyURI"/>
      <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
     </sequence>
    </extension>
   </complexContent>
  </complexType>

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

The NAPTRType object is composed of the following elements:

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

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

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

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

- ext: Point of extensibility described in Section 3.3.

The NSType object is composed of the following elements:

- hostName: Root-relative host name of the name server.

- ipAddr: Zero or more objects of type IpAddrType. Each object holds an IP Address and the IP Address type ("IPv4" or "IPv6").

- ext: Point of extensibility described in Section 3.3.

The URIType object is composed of the following elements:

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

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

6.5. SED Group Offer

The list of peer organizations whose resolution responses can include the SED contained in a given SED Group is controlled by the organization to which a SED Group object belongs (its Registrant) and the peer organization that submits resolution requests (a data recipient, also known as a peering organization). The Registrant offers access to a SED Group by submitting a SED Group Offer. The data recipient can then accept or reject that offer. Not until access to a SED Group has been offered and accepted will the data recipient's organization ID be included in the peeringOrg list in a SED Group object, and that SED Group's peering information becomes a candidate for inclusion in the responses to the resolution requests submitted by that data recipient. The substrate protocol MUST support the ability to Add, Get, Delete, Accept, and Reject SED Group Offers (refer to Section 7 for a generic description of various operations).

A SED Group Offer object MUST be uniquely identified by attributes as defined in the description of "SedGrpOfferKeyType" in "Derived Object Key Types" (Section 5.2.2 of this document).

The SedGrpOfferType object structure is defined as follows:

```
<complexType name="SedGrpOfferType">
 <complexContent>
  <extension base="sppfb:BasicObjType">
   <sequence>
    <element name="sedGrpOfferKey" type="sppfb:SedGrpOfferKeyType"/>
    <element name="status" type="sppfb:SedGrpOfferStatusType"/>
    <element name="offerDateTime" type="dateTime"/>
    <element name="acceptDateTime" type="dateTime" minOccurs="0"/>
    <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>

<complexType name="SedGrpOfferKeyType" abstract="true">
 <annotation>
  <documentation>
  -- Generic type that represents the key for a SED Group Offer. Must
     be defined in concrete form in a substrate "protocol"
     specification. --
  </documentation>
 </annotation>
</complexType>

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

The SedGrpOfferType object is composed of the following elements:

- base: All first-class objects extend BasicObjType (see Section 5.1).

- sedGrpOfferKey: The object that identifies the SED that is or has been offered and the organization to which it is or has been offered.

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

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

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

6.6. Egress Route

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

An Egress Route object MUST be uniquely identified by attributes as defined in the description of "ObjKeyType" in "Generic Object Key Type" (Section 5.2.1 of this document).

Assume that the target SSP has offered, as part of its SED, to share one or more Ingress Routes and that the originating SSP has accepted the offer. In order to add the Egress Route to the Registry, the originating SSP uses a valid regular expression to rewrite the Ingress Route in order to include the egress SBE information. Also, more than one Egress Route can be associated with a given Ingress Route in support of fault-tolerant configurations. The supporting SPPF structure provides a way to include route precedence information to help manage traffic to more than one outbound egress SBE.

The substrate protocol MUST support the ability to Add, Get, and Delete Egress Routes (refer to Section 7 for a generic description of various operations). The EgrRteType object structure is defined as follows:

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

The EgrRteType object is composed of the following elements:

- base: All first-class objects extend BasicObjType (see Section 5.1).
- egrRteName: The name of the Egress Route.
- pref: The preference of this Egress Route relative to other Egress Routes that may get selected when responding to a resolution request.
- regxRewriteRule: The regular expression rewrite rule that should be applied to the regular expression of the ingress NAPTR(s) that belongs to the Ingress Route.
- ingrSedGrp: The ingress SED Group that the Egress Route should be used for.
- svcs: ENUM service(s) that is served by an Egress Route. This element is used to identify the ingress NAPTRs associated with the SED Group to which an Egress Route's regxRewriteRule should be applied. If no ENUM service(s) is associated with an Egress Route, then the Egress Route's regxRewriteRule should be applied to all the NAPTRs associated with the SED Group. This field's value must be of the form specified in [RFC6116] (e.g., E2U+pstn:sip+sip). The allowable values are a matter of policy and are not limited by this protocol.
- ext: Point of extensibility described in Section 3.3.

7. Framework Operations

In addition to the operation-specific object types, all operations MAY specify the minor version of the protocol that when used in conjunction with the major version (which can be, for instance, specified in the protocol Namespace) can serve to identify the version of the SPPF protocol that the client is using. If the minor version is not specified, the latest minor version supported by the SPPF server for the given major version will be used. Additionally, operations that may potentially modify persistent protocol objects SHOULD include a transaction ID as well.

7.1. Add Operation

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

Note that this effectively allows modification of a preexisting object.

If the entity that issued the command is not authorized to perform this operation, an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" (Section 5.3 of this document).

7.2. Delete Operation

Any conforming substrate "protocol" specification MUST provide a definition for the operation that deletes one or more SPPF objects from the Registry using the object's key.

If the entity that issued the command is not authorized to perform this operation, an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" (Section 5.3 of this document).

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

- Destination Groups: When a Destination Group is deleted, any cross-references between that destination group and any SED Group must be automatically removed by the SPPF implementation as part of fulfilling the deletion request. Similarly, any cross-references between that Destination Group and any Public Identifier must be removed by the SPPF implementation.

- SED Groups: When a SED Group is deleted, any references between that SED Group and any Destination Group must be automatically removed by the SPPF implementation as part of fulfilling the deletion request. Similarly, any cross-references between that SED Group and any SED Records must be removed by the SPPF

implementation as part of fulfilling the deletion request. Furthermore, SED Group Offers relating to that SED Group must also be deleted.

- SED Records: When a SED Record is deleted, any cross-references between that SED Record and any SED Group must be removed by the SPPF implementation as part of fulfilling the deletion request. Similarly, any reference between that SED Record and any Public Identifier must be removed by the SPPF implementation.

- Public Identifiers: When a Public Identifier is deleted, any cross-references between that Public Identifier and any referenced Destination Group must be removed by the SPPF implementation as part of fulfilling the deletion request. Any references to SED Records associated directly to that Public Identifier must also be deleted by the SPPF implementation.

Deletes MUST be atomic.

7.3. Get Operations

At times, on behalf of the Registrant, the Registrar may need to get information about SPPF objects that were previously provisioned in the Registry. A few examples include logging, auditing, and pre-provisioning dependency checking. This query mechanism is limited to aid provisioning scenarios and should not be confused with query protocols provided as part of the resolution system (e.g., ENUM and SIP).

Any conforming "protocol" specification MUST provide a definition for the operation that queries the details of one or more SPPF objects from the Registry using the object's key. If the entity that issued the command is not authorized to perform this operation, an appropriate error message MUST be returned from among the response messages defined in Section 5.3.

If the response to the Get operation includes an object(s) that extends the BasicObjType, the Registry MUST include the "cDate" and "mDate", if applicable.

7.4. Accept Operations

In SPPF, a SED Group Offer can be accepted or rejected by, or on behalf of, the Registrant to which the SED Group has been offered (refer to Section 6.5 of this document for a description of the SED Group Offer object). The Accept operation is used to accept the SED Group Offers. Any conforming substrate "protocol" specification MUST provide a definition for the operation to accept SED Group Offers by,

or on behalf of, the Registrant, using the SED Group Offer object key.

Not until access to a SED Group has been offered and accepted will the Registrant's organization ID be included in the peeringOrg list in that SED Group object, and that SED Group's peering information becomes a candidate for inclusion in the responses to the resolution requests submitted by that Registrant. A SED Group Offer that is in the "offered" status is accepted by, or on behalf of, the Registrant to which it has been offered. When the SED Group Offer is accepted, the SED Group Offer is moved to the "accepted" status and the data recipient's organization ID is added into the list of peerOrgIds for that SED Group.

If the entity that issued the command is not authorized to perform this operation, an appropriate error message MUST be returned from amongst the response messages defined in "Response Message Types" (Section 5.3 of this document).

7.5. Reject Operations

In SPPF, a SED Group Offer object can be accepted or rejected by, or on behalf of, the Registrant to which the SED Group has been offered (refer to "Framework Data Model Objects", Section 6 of this document, for a description of the SED Group Offer object). Furthermore, that offer may be rejected, regardless of whether or not it has been previously accepted. The Reject operation is used to reject the SED Group Offer. When the SED Group Offer is rejected, that SED Group Offer is deleted, and, if appropriate, the data recipient's organization ID is removed from the list of peeringOrg IDs for that SED Group. Any conforming substrate "protocol" specification MUST provide a definition for the operation to reject SED Group Offers by, or on behalf of, the Registrant, using the SED Group Offer object key.

If the entity that issued the command is not authorized to perform this operation, an appropriate error message MUST be returned from among the response messages defined in "Response Message Types" (Section 5.3 of this document).

7.6. Get Server Details Operation

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

Any conforming substrate "protocol" specification MUST provide a definition for the operation to request such details from the SPPF server. If the entity that issued the command is not authorized to perform this operation, an appropriate error message MUST be returned from among the response messages defined in "Response Message Types" (Section 5.3 of this document).

8. XML Considerations

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

XML is case sensitive. Unless stated otherwise, the character casing of XML specifications in this document MUST be preserved to develop a conforming specification.

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

8.1. Namespaces

All SPPF elements are defined in the Namespaces in the "IANA Considerations" and "Formal Framework Specification" sections of this document.

8.2. Versioning and Character Encoding

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

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

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

Example XML declarations:

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

9. Security Considerations

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

9.1. Confidentiality and Authentication

With respect to confidentiality and authentication, the substrate protocol requirements section of this document contains security properties that the substrate protocol must provide, so that authenticated endpoints can exchange data confidentially and with integrity protection. Refer to Section 4 of this document and [RFC7878] for the specific solutions to authentication and confidentiality.

9.2. Authorization

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

9.3. Denial of Service

In general, guidance on Denial-of-Service (DoS) issues is given in "Internet Denial of Service Considerations" [RFC4732], which also gives a general vocabulary for describing the DoS issue.

SPPF is a high-level client-server protocol that can be implemented on lower-level mechanisms such as remote procedure call and web-service API protocols. As such, it inherits any Denial-of-Service issues inherent to the specific lower-level mechanism used for any implementation of SPPF. SPPF also has its own set of higher-level exposures that are likely to be independent of lower-layer mechanism choices.

9.3.1. DoS Issues Inherited from the Substrate Mechanism

In general, an SPPF implementation is dependent on the selection and implementation of a lower-level substrate protocol and a binding between that protocol and SPPF. The archetypal SPPF implementation uses XML [W3C.REC-xml-20081126] representation in a SOAP [SOAPREF] request/response framework over HTTP [RFC7230], probably also uses Transport Layer Security (TLS) [RFC5246] for on-the-wire data integrity and participant authentication, and might use HTTP Digest authentication [RFC2069].

The typical deployment scenario for SPPF is to have servers in a managed facility; therefore, techniques such as Network Ingress Filtering [RFC2827] are generally applicable. In short, any DoS mechanism affecting a typical HTTP implementation would affect such an SPPF implementation; therefore, the mitigation tools for HTTP in general also apply to SPPF.

SPPF does not directly specify an authentication mechanism; instead, it relies on the lower-level substrate protocol to provide for authentication. In general, authentication is an expensive operation, and one apparent attack vector is to flood an SPPF server with repeated requests for authentication, thereby exhausting its resources. Therefore, SPPF implementations SHOULD be prepared to handle authentication floods, perhaps by noting repeated failed login requests from a given source address and blocking that source address.

9.3.2. DoS Issues Specific to SPPF

The primary defense mechanism against DoS within SPPF is authentication. Implementations MUST tightly control access to the SPPF service, SHOULD implement DoS and other policy control screening, and MAY employ a variety of policy violation reporting and response measures such as automatic blocking of specific users and alerting of operations personnel. In short, the primary SPPF response to DoS-like activity by a user is to block that user or subject their actions to additional review.

SPPF allows a client to submit multiple-element or "batch" requests that may insert or otherwise affect a large amount of data with a single request. In the simplest case, the server progresses sequentially through each element in a batch, completing one before starting the next. Mid-batch failures are handled by stopping the batch and rolling back the data store to its pre-request state. This "stop and roll back" design provides a DoS opportunity. A hostile client could repeatedly issue large batch requests with one or more failing elements, causing the server to repeatedly stop and roll back

large transactions. The suggested response is to monitor clients for such failures and take administrative action (such as blocking the user) when an excessive number of rollbacks is reported.

An additional suggested response is for an implementer to set their maximum allowable XML message size and their maximum allowable batch size at a level that they feel protects their operational instance, given the hardware sizing they have in place and the expected load and size needs that their users expect.

9.4. Information Disclosure

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

9.5. Non-repudiation

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

9.6. Replay Attacks

Anti-replay protection ensures that a given SPPF object replayed at a later time won't affect the integrity of the system. SPPF provides at least one mechanism to fight against replay attacks. Use of the optional client transaction identifier allows the SPPF client to correlate the request message with the response and to be sure that it is not a replay of a server response from earlier exchanges. Use of unique values for the client transaction identifier is highly encouraged to avoid chance matches to a potential replay message.

9.7. Compromised or Malicious Intermediary

The SPPF client or Registrar can be a separate entity acting on behalf of the Registrant in facilitating provisioning transactions to the Registry. Therefore, even though the substrate layer provides end-to-end protection for each specific SPPP connection between client and server, data might be available in clear text before or after it traverses an SPPP connection. Therefore, a man-in-the-middle attack by one of the intermediaries is a possibility that could affect the integrity of the data that belongs to the Registrant and/or expose peering data to unintended actors.

10. Internationalization Considerations

Character encodings to be used for SPPF elements are described in Section 8.2. The use of time elements in the protocol is specified in Section 3.2. Where human-readable messages that are presented to an end user are used in the protocol, those messages SHOULD be tagged according to [RFC5646], and the substrate protocol MUST support a respective mechanism to transmit such tags together with those human-readable messages.

11. IANA Considerations

11.1. URN Assignments

This document uses URNs to describe XML Namespaces and XML Schemas conforming to a Registry mechanism described in [RFC3688].

Two URI assignments have been made:

Registration for the SPPF XML Namespace:
urn:ietf:params:xml:ns:sppf:base:1
Registrant Contact: The IESG
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the XML Schema:

URI: urn:ietf:params:xml:schema:sppf:1
Registrant Contact: IESG
XML: See "Formal Specification" (Section 12 of this document).

11.2. Organization Identifier Namespace Registry

IANA has created and will maintain a registry titled "SPPF OrgIdType Namespaces". The formal syntax is described in Section 5.1.

Assignments consist of the OrgIdType Namespace string and the definition of the associated Namespace. This document makes the following initial assignment for the OrgIdType Namespaces:

OrgIdType Namespace string	Namespace
IANA Enterprise Numbers	iana-en

Future assignments are to be made through the well-known IANA Policy "RFC Required" (see Section 4.1 of [RFC5226]). Such assignments will typically be requested when a new Namespace for identification of SPs is defined.

12. Formal Specification

This section provides the XSD for the SPPF protocol.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:sppfb="urn:ietf:params:xml:ns:sppf:base:1"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:ietf:params:xml:ns:sppf:base:1"
elementFormDefault="qualified" xml:lang="EN">
 <annotation>
  <documentation>
   ---- Generic object key types to be defined by specific
        substrate/architecture.  The types defined here can
        be extended by the specific architecture to
        define the Object Identifiers. ----
  </documentation>
 </annotation>
 <complexType name="ObjKeyType"
  abstract="true">
  <annotation>
   <documentation>
    ---- Generic type that represents the
         key for various objects in SPPF. ----
   </documentation>
  </annotation>
 </complexType>
```

```
    <complexType name="SedGrpOfferKeyType" abstract="true">
     <complexContent>
      <extension base="sppfb:ObjKeyType">
       <annotation>
        <documentation>
        ---- Generic type that represents
             the key for a SED Group Offer. ----
        </documentation>
       </annotation>
      </extension>
     </complexContent>
    </complexType>

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

    <annotation>
     <documentation>
       ---- Object Type Definitions ----
     </documentation>
    </annotation>

    <complexType name="SedGrpType">
     <complexContent>
      <extension base="sppfb:BasicObjType">
       <sequence>
        <element name="sedGrpName" type="sppfb:ObjNameType"/>
        <element name="sedRecRef" type="sppfb:SedRecRefType"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="dgName" type="sppfb:ObjNameType"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="peeringOrg" type="sppfb:OrgIdType"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="sourceIdent" type="sppfb:SourceIdentType"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="isInSvc" type="boolean"/>
        <element name="priority" type="unsignedShort"/>
```

```
      <element name="ext"
      type="sppfb:ExtAnyType" minOccurs="0"/>
     </sequence>
    </extension>
   </complexContent>
  </complexType>
  <complexType name="DestGrpType">
   <complexContent>
    <extension base="sppfb:BasicObjType">
     <sequence>
      <element name="dgName"
      type="sppfb:ObjNameType"/>
     </sequence>
    </extension>
   </complexContent>
  </complexType>
  <complexType name="PubIdType" abstract="true">
   <complexContent>
    <extension base="sppfb:BasicObjType">
     <sequence>
      <element name="dgName" type="sppfb:ObjNameType"
               minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
    </extension>
   </complexContent>
  </complexType>
  <complexType name="TNType">
   <complexContent>
    <extension base="sppfb:PubIdType">
     <sequence>
      <element name="tn" type="sppfb:NumberValType"/>
      <element name="corInfo" type="sppfb:CORInfoType" minOccurs="0"/>
      <element name="sedRecRef" type="sppfb:SedRecRefType"
               minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
    </extension>
   </complexContent>
  </complexType>
  <complexType name="TNRType">
   <complexContent>
    <extension base="sppfb:PubIdType">
     <sequence>
      <element name="range" type="sppfb:NumberRangeType"/>
      <element name="corInfo" type="sppfb:CORInfoType" minOccurs="0"/>
     </sequence>
    </extension>
   </complexContent>
  </complexType>
```

```
    <complexType name="TNPType">
     <complexContent>
      <extension base="sppfb:PubIdType">
       <sequence>
        <element name="tnPrefix" type="sppfb:NumberValType"/>
        <element name="corInfo" type="sppfb:CORInfoType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="RNType">
     <complexContent>
      <extension base="sppfb:PubIdType">
       <sequence>
        <element name="rn" type="sppfb:NumberValType"/>
        <element name="corInfo" type="sppfb:CORInfoType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
     <complexType name="URIPubIdType">
     <complexContent>
      <extension base="sppfb:PubIdType">
       <sequence>
        <element name="uri" type="anyURI"/>
        <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="SedRecType" abstract="true">
     <complexContent>
      <extension base="sppfb:BasicObjType">
       <sequence>
        <element name="sedName" type="sppfb:ObjNameType"/>
        <element name="sedFunction" type="sppfb:SedFunctionType"
                 minOccurs="0"/>
        <element name="isInSvc" type="boolean"/>
        <element name="ttl" type="positiveInteger" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="NAPTRType">
     <complexContent>
      <extension base="sppfb:SedRecType">
       <sequence>
        <element name="order" type="unsignedShort"/>
```

```
        <element name="flags" type="sppfb:FlagsType" minOccurs="0"/>
        <element name="svcs" type="sppfb:SvcType"/>
        <element name="regx" type="sppfb:RegexParamType" minOccurs="0"/>
        <element name="repl" type="sppfb:ReplType" minOccurs="0"/>
        <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="NSType">
     <complexContent>
      <extension base="sppfb:SedRecType">
       <sequence>
        <element name="hostName" type="token"/>
        <element name="ipAddr" type="sppfb:IPAddrType"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="URIType">
     <complexContent>
      <extension base="sppfb:SedRecType">
       <sequence>
        <element name="ere" type="token" default="^(.*)$"/>
        <element name="uri" type="anyURI"/>
        <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="SedGrpOfferType">
     <complexContent>
      <extension base="sppfb:BasicObjType">
       <sequence>
        <element name="sedGrpOfferKey" type="sppfb:SedGrpOfferKeyType"/>
        <element name="status" type="sppfb:SedGrpOfferStatusType"/>
        <element name="offerDateTime" type="dateTime"/>
        <element name="acceptDateTime" type="dateTime" minOccurs="0"/>
        <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
       </sequence>
      </extension>
     </complexContent>
    </complexType>
    <complexType name="EgrRteType">
     <complexContent>
      <extension base="sppfb:BasicObjType">
```

```
      <sequence>
       <element name="egrRteName" type="sppfb:ObjNameType"/>
       <element name="pref" type="unsignedShort"/>
       <element name="regxRewriteRule" type="sppfb:RegexParamType"/>
       <element name="ingrSedGrp" type="sppfb:ObjKeyType"
                minOccurs="0" maxOccurs="unbounded"/>
       <element name="svcs" type="sppfb:SvcType" minOccurs="0"/>
       <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
      </sequence>
     </extension>
    </complexContent>
   </complexType>
   <annotation>
    <documentation>
     ---- Abstract Object and Element Type Definitions ----
    </documentation>
   </annotation>
   <complexType name="BasicObjType" abstract="true">
    <sequence>
     <element name="rant" type="sppfb:OrgIdType"/>
     <element name="rar" type="sppfb:OrgIdType"/>
     <element name="cDate" type="dateTime" minOccurs="0"/>
     <element name="mDate" type="dateTime" minOccurs="0"/>
     <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
    </sequence>
   </complexType>
   <complexType name="RegexParamType">
    <sequence>
     <element name="ere" type="sppfb:RegexType" default="^(.*)$"/>
     <element name="repl" type="sppfb:ReplType"/>
    </sequence>
   </complexType>
   <complexType name="IPAddrType">
    <sequence>
     <element name="addr" type="sppfb:AddrStringType"/>
     <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
    </sequence>
    <attribute name="type" type="sppfb:IPType" default="v4"/>
   </complexType>
   <complexType name="SedRecRefType">
    <sequence>
     <element name="sedKey" type="sppfb:ObjKeyType"/>
     <element name="priority" type="unsignedShort"/>
     <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
    </sequence>
   </complexType>
   <complexType name="SourceIdentType">
    <sequence>
```

```
      <element name="sourceIdentRegex" type="sppfb:RegexType"/>
      <element name="sourceIdentScheme"
               type="sppfb:SourceIdentSchemeType"/>
      <element name="ext" type="sppfb:ExtAnyType" minOccurs="0"/>
     </sequence>
    </complexType>
    <complexType name="CORInfoType">
     <sequence>
      <element name="corClaim" type="boolean" default="true"/>
      <element name="cor" type="boolean" default="false" minOccurs="0"/>
      <element name="corDate" type="dateTime" minOccurs="0"/>
     </sequence>
    </complexType>
    <complexType name="SvcMenuType">
     <sequence>
      <element name="serverStatus" type="sppfb:ServerStatusType"/>
      <element name="majMinVersion" type="token" maxOccurs="unbounded"/>
      <element name="objURI" type="anyURI" maxOccurs="unbounded"/>
      <element name="extURI" type="anyURI"
               minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
    </complexType>
    <complexType name="ExtAnyType">
     <sequence>
      <any namespace="##other" maxOccurs="unbounded"/>
     </sequence>
    </complexType>
    <simpleType name="FlagsType">
     <restriction base="token">
      <length value="1"/>
      <pattern value="[A-Z]|[a-z]|[0-9]"/>
     </restriction>
    </simpleType>
    <simpleType name="SvcType">
     <restriction base="token">
      <minLength value="1"/>
     </restriction>
    </simpleType>
    <simpleType name="RegexType">
     <restriction base="token">
      <minLength value="1"/>
     </restriction>
    </simpleType>
    <simpleType name="ReplType">
     <restriction base="token">
      <minLength value="1"/>
      <maxLength value="255"/>
     </restriction>
```

```
    </simpleType>
    <simpleType name="OrgIdType">
     <restriction base="token"/>
    </simpleType>
    <simpleType name="ObjNameType">
     <restriction base="token">
      <minLength value="3"/>
      <maxLength value="80"/>
     </restriction>
    </simpleType>
    <simpleType name="TransIdType">
     <restriction base="token">
      <minLength value="3"/>
      <maxLength value="120"/>
     </restriction>
    </simpleType>
    <simpleType name="MinorVerType">
     <restriction base="unsignedLong"/>
    </simpleType>
    <simpleType name="AddrStringType">
     <restriction base="token">
      <minLength value="3"/>
      <maxLength value="45"/>
     </restriction>
    </simpleType>
    <simpleType name="IPType">
     <restriction base="token">
      <enumeration value="v4"/>
      <enumeration value="v6"/>
     </restriction>
    </simpleType>
    <simpleType name="SourceIdentSchemeType">
     <restriction base="token">
      <enumeration value="uri"/>
      <enumeration value="ip"/>
      <enumeration value="rootDomain"/>
     </restriction>
    </simpleType>
    <simpleType name="ServerStatusType">
     <restriction base="token">
      <enumeration value="inService"/>
      <enumeration value="outOfService"/>
     </restriction>
    </simpleType>
    <simpleType name="SedGrpOfferStatusType">
     <restriction base="token">
      <enumeration value="offered"/>
      <enumeration value="accepted"/>
```

```
     </restriction>
    </simpleType>
    <simpleType name="NumberValType">
     <restriction base="token">
      <maxLength value="20"/>
      <pattern value="\+?\d\d*"/>
     </restriction>
    </simpleType>
    <simpleType name="NumberTypeEnum">
     <restriction base="token">
      <enumeration value="TN"/>
      <enumeration value="TNPrefix"/>
      <enumeration value="RN"/>
     </restriction>
    </simpleType>
    <simpleType name="SedFunctionType">
     <restriction base="token">
      <enumeration value="routing"/>
      <enumeration value="lookup"/>
     </restriction>
    </simpleType>
    <complexType name="NumberType">
     <sequence>
      <element name="value" type="sppfb:NumberValType"/>
      <element name="type" type="sppfb:NumberTypeEnum"/>
     </sequence>
    </complexType>
    <complexType name="NumberRangeType">
     <sequence>
      <element name="startRange" type="sppfb:NumberValType"/>
      <element name="endRange" type="sppfb:NumberValType"/>
     </sequence>
    </complexType>
   </schema>
```

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <http://www.rfc-editor.org/info/rfc2277>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7878] Cartwright, K., Bhatia, V., Mule, J., and A. Mayrhofer, "Session Peering Provisioning (SPP) Protocol over SOAP", RFC 7878, DOI 10.17487/RFC7878, August 2016, <http://www.rfc-editor.org/info/rfc7878>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

13.2. Informative References

[RFC2069] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E., and L. Stewart, "An Extension to HTTP : Digest Access Authentication", RFC 2069, DOI 10.17487/RFC2069, January 1997, <http://www.rfc-editor.org/info/rfc2069>.

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, DOI 10.17487/RFC2781, February 2000, <http://www.rfc-editor.org/info/rfc2781>.

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <http://www.rfc-editor.org/info/rfc2827>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <http://www.rfc-editor.org/info/rfc3261>.

[RFC3403] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, DOI 10.17487/RFC3403, October 2002, <http://www.rfc-editor.org/info/rfc3403>.

[RFC4725] Mayrhofer, A. and B. Hoeneisen, "ENUM Validation Architecture", RFC 4725, DOI 10.17487/RFC4725, November 2006, <http://www.rfc-editor.org/info/rfc4725>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <http://www.rfc-editor.org/info/rfc4732>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5067] Lind, S. and P. Pfautz, "Infrastructure ENUM Requirements", RFC 5067, DOI 10.17487/RFC5067, November 2007, <http://www.rfc-editor.org/info/rfc5067>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5486] Malas, D., Ed. and D. Meyer, Ed., "Session Peering for Multimedia Interconnect (SPEERMINT) Terminology", RFC 5486, DOI 10.17487/RFC5486, March 2009, <http://www.rfc-editor.org/info/rfc5486>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <http://www.rfc-editor.org/info/rfc5646>.

[RFC6116] Bradner, S., Conroy, L., and K. Fujiwara, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)", RFC 6116, DOI 10.17487/RFC6116, March 2011, <http://www.rfc-editor.org/info/rfc6116>.

[RFC6461] Channabasappa, S., Ed., "Data for Reachability of Inter-/Intra-NetworK SIP (DRINKS) Use Cases and Protocol Requirements", RFC 6461, DOI 10.17487/RFC6461, January 2012, <http://www.rfc-editor.org/info/rfc6461>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[SOAPREF] Gudgin, M., Hadley, M., Moreau, J., and H. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework", W3C REC REC-SOAP12-part1-20030624, June 2003, <http://www.w3.org/TR/soap12-part1/>.

[Unicode6.1] The Unicode Consortium, "The Unicode Standard, Version 6.1.0", (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-02-3), <http://unicode.org/versions/Unicode6.1.0/>.

Acknowledgements

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

Authors' Addresses

Kenneth Cartwright
TNS
1939 Roland Clarke Place
Reston, VA 20191
United States

Email: kcartwright@tnsi.com

Vikas Bhatia
TNS
1939 Roland Clarke Place
Reston, VA 20191
United States

Email: vbhatia@tnsi.com

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

Email: syed.ali@neustar.biz

David Schwartz
XConnect
316 Regents Park Road
London N3 2XJ
United Kingdom

Email: dschwartz@xconnect.net