SIPPING
Internet-Draft
Intended status: Standards Track
Expires: May 20, 2008

D. Petrie
SIPez LLC.
S. Channabasappa
CableLabs
S. Ganesan
Motorola
November 17, 2007

A Schema and Guidelines for Defining Session Initiation Protocol User Agent Profile Datasets
draft-petrie-sipping-profile-datasets-05.txt

Abstract

This document defines the requirements and a format for SIP user agent profile data. An overall schema is specified for the definition of profile datasets. The schema also provides for expressing constraints for how multiple sources of profile data are

to be combined. This document provides a guide to considerations, policies and syntax for defining datasets to be included in profile data. It also explores some specific datasets to test the requirements, assumptions and syntax.

Table of Contents

1. Motivation

Today all SIP user agent implementers use proprietary means of expressing and delivering user, device, and local network profile information to the user agent. The SIP User Agent Profile Delivery Framework [I-D.ietf-sipping-config-framework] and the Framework for Session SIP Session Policies [I-D.hilt-sipping-session-policy-framework] specify how SIP user agents locate and retrieve profile data specific to the user, the device, and the local network. It is important for SIP User Agents to be able to obtain and use these multiple sources of profile data in order to support a wide range of applications without undue complexity.

While these frameworks define the mechanisms for transmitting profile data, they do not define a format for the actual profile data. This document defines the requirements, the default/manditory to support content type for [I-D.ietf-sipping-config-framework], a high level schema for, and guide to how these datasets can be defined. The goal is to enable any SIP user agent to obtain profile data and be functional in a new environment independent of the implementation or model of user agent. The nature of having profile data from four potential sources requires the definition of policies on how to apply the data in an interoperable way across implementations which may have widely varying capabilities.

The ultimate objective of the framework described in the SIP User Agent Profile Delivery Framework and this document is to provide a start up experience similar to that of users of an analog telephone. From the point of view of a user, you just plug in an analog telephone and it works (assuming that you have made the right arrangements with your local phone company). There is no end user setup required to make an analog phone work, at least in a basic sense. So the objective here is to be able to take a new SIP user agent out of the box, plug it in or install the software and have it get its profiles without human intervention other than security measures. This is necessary for cost effective deployment of large numbers of user agents. All user agents do not provide telephone capabilities, but the user set up experience goal is applicable to most of the range of user agent capabilities.

2. Introduction

2.1. Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in RFC 2119[RFC2119].

2.2. Profile Data Terminology

property - a named configurable characteristic of a user agent. A given property has a well-defined range of possible values. A given property may be defined to have a range of values, allow for simultaneous use of many values (as in a list of allowed possibilities), or a set of related values that collectively form a single profile information item.

setting - the binding of a specific value or set of values to a given property.

profile - a collection of settings to be applied for a specific user, device, or local network.

device - SIP user agent, either software or hardware appliance. This is a logical concept, as there may be no physical dedicated device or it may be part of an assembly of devices. In this document, the terms "user agent" and "device" are interchangeable.

user profile - the profile that applies to a specific user. This is best illustrated by the "hotelling" use case - a user has an association for some period of time with a particular device. The user profile is that set of profile data the user wants to associate with that device (e.g. ringtones used when someone calls them, the user's shortcuts).

device profile - data profile that applies to a specific device. In the "hotelling" use case, this is the data that is bound to the device itself independent of the user. It relates to specific capabilities of the device and/or preferences of the owner of the device.

local network profile - data that applies to the user agent in the context of the local network. This is best illustrated by roaming applications; a new device appears in the local network (or a device appears in a new network, depending on the point of view). The local network profile includes settings and perhaps policies that allow the user agent to function in the local network (e.g. how to traverse NAT or firewall, bandwidth constraints).

dataset - a collection of properties.

working profile - the set of property values actually set in a SIP User Agent as a result of merging the profiles from all sources; the actual effective profile for the user agent .

merging - the operation of resolving overlapping settings from multiple profiles. Overlap occurs when the same property occurs in multiple profiles (e.g. device, user, application, local network).

2.3. Overview

In this document requirements are specified for containing and expressing profile data for use on SIP user agents. Though much of this can be considered independent of SIP there is one primary requirement that is not well satisfied through more generic profile data mechanisms. SIP User Agent set up requires the agent to merge settings, which may overlap, from potentially four different sources (see [I-D.ietf-sipping-config-framework]); each source must not only be able to provide profile information, but also express policies regarding how the profile settings may be combined with that from other sources.

A schema and syntax is defined in this document to specify properties that may be aggregated to construct profiles. The general design philosophy is that many small datasets provide flexibility to the implementer to support the aggregated set that best matches the capability of the user agent. The actual properties are not defined in this document (see [I-D.ietf-sipping-media-policy-dataset] and [reference: Core SIP Dataset]). However, some examples are explored here to illustrate the proposed mechanisms and to validate the requirements.

This document defines a set of considerations, syntax and policies that must be specified when defining datasets. These are to help authors of dataset specifications to define datasets that will work in the overall schema defined in this document. The actual specification of these datasets is outside the scope of this document.

3. Design Considerations

The following section defines some of the design considerations that were taken into account when defining the schema, syntax and policies for generating and applying profile data. Section 3.2.6 describes need for merging of the four dataset sources provided in [I-D.ietf-sipping-config-framework].

3.1. Use Cases

In the following use case scenarios the device profile is provided by the device owner/manager. The owner/manager may be a service

provider, an enterprise or a user administering the device setup. The user is assumed to be the end user operating the user agent to perform SIP functions such as telephony, IM etc. In the scenarios that the user profile is provided, the user profile contains user specific properties that the end user has set directly or indirectly through an administration process. The local network profiles represent the suggested policy behavior that the local network operator would like user agents to adhere to [I-D.hilt-sipping-session-policy-framework]. From a security perspective, the local network operator cannot trust the user agent to follow the local network profile policy. The local network operator must use a means external to the user agent to enforce these policies. The local network profile is intended to express to the user agent, the policies that the user agent should follow if the user agent wants to function properly in the local network.

3.1.1. Outbound Proxy Setting

First consider the use cases for a simple user agent property: the outbound proxy. It is not likely that the user would want to influence the outbound proxy for SIP signaling. Conceptually an application might wish to use a specific outbound proxy for signaling related to that application. For this use case, assume that the only the device owner/manager or the local network operator are likely to want to set the outbound proxy property. The device profile defines an outbound proxy perhaps so that the device owner/manager can monitor all signaling. The local network operator also defines an outbound proxy because the proxy allows the SIP signaling to get through a NAT or firewall.

It seems there are few possible solutions to this conflict resolution problem:

- The simple solution is to define a policy where the local network profile overrides the device profile. In this approach the local network profile wins.
- A comprehensive solution is to allow the aggregation of the outbound proxies. In this scenario SIP messages would be sent with a pre-populated route set that had two hops. First the outbound proxy set in the local network profile, then the outbound proxy set in the device profile.

The aggregation approach is closest to solving the requirements to the use case above. By aggregating the two outbound proxies, the local network provided outbound proxy allows the signaling to get out of the local network and the device profile provided outbound proxy is able to monitor all SIP signaling from the user agent.

3.1.2. Codec Settings

Use cases for the codec properties are illustrated here as they are likely one of the more complicated sets of properties with respect to merging and constraining across more than one profile. There are reasonable scenarios where requirements can be rationalized that the device, user and local network profiles may each wish to express preferences and constrains of the codec properties. Without getting into details or syntax of the codec properties, it is assumed that codec properties will need to express a codec definition and a preference order. This is the order that these codecs will be put in SDP for codec negotiation purposes.

The following scenarios illustrate some possible combinations of sources of codec properties from the device, user and local network profiles. The scenarios identify rationale for providing codec properties in each of the profiles.

3.1.2.1. Codec Setting Not Set

In the scenario where a device has no profiles or the profiles contain no codec properties, the device will enable a default set and preference order of codecs. The default set and preference order of codecs is a implementer specific choice. In some implementations it is s subset of the codecs supported by the device.

3.1.2.2. Codec Set in Device Profile

Let us assume a scenario where user agents providing telephony capabilities are deployed. The deployment has very simple requirements such that the user agents have fixed locations and are always associated with the same user. This scenario does not need the separation between the user, device and local network profiles. A single profile would suffice. Another scenario having similar requirements is one where the user and local network profiles do not provide any codec related properties. This might be because the user does not care what codecs are used and the local network does not wish to impose any constraints on the codes used in the network. In the following use case, the device profile is the only source of codec properties.

The codecs in the device profile may differ from the set of codecs supported by the device, due to the administrator of the device profile wanting:

- To have a uniform set of codecs used across all device types
- To exclude the use of a specific codec due to performance issues/ concerns

The resulting device profile data further will constrain the list of codecs that get applied. In addition, the administrator may want to list the order of which the codecs are to be applied. In this scenario the device profile data will dictate the ordered list of codecs to be applied. The use agent will ignore codec types included in the profile that the user agent does not support.

3.1.2.3. Set in Device and User Profiles

In the following scenario users are allowed to express a preference over codecs. Users are probably not likely to express specific codes in the form of G.7XX, etc. They are likely to want to express a preference in the form of wideband, normal and low bandwidth. In the following use case, device and user profiles contain codec properties.

The user may prefer a higher quality codec to be used, if available. Thereby the user profile data may provide an ordered list of codecs to be applied. The device profile also specifies a list of codecs and a default preference order.

The merging of the data sources is as follows:

- The ordering of the codecs will be determined from the user profile data, which overrides the codec preference ordering from the device profile data.
- The set of codecs that may be applied, are the codecs listed in the user data constrained by the list of codecs from the device profile data.

The case in which none of the codecs in the resulting merged profile datasets are supported by the device, the profile data constitutes a misconfiguration between device and user profiles. It may not be possible to successfully establish a session in this case. It is suggested that the user agent provide feedback to the user indicating the misconfiguration. The user agent may also attempt to function in the network by ignoring one or more of the profile constraints.

3.1.2.4. Set in Device and Local Profiles

In another scenario the user is not allowed or does not care to express codec preferences. The owner/manager of the device defines the set of codecs which may be used on the device along with a preference ordering of codecs. There is no user profile or the user profile contains no codec properties. The local network wishes to define a policy over codec usage in the network. It is not clear there is a requirement that the local network be able to express a preference order. However the network operator is very likely to want to express a set of codecs that can or should not be used. The

constraints that the local network operator wishes suggest may relate to the goal of controlling bandwidth or conveying what will work over the available WAN connection. In the following use case, device and local network profiles provide codec properties. The local network may limit the type of codecs that can be applied due to resources available.

The merging of the data sources is as follows:

- The set of codecs that may be used is the ordered list of codecs from the device profile data, further constrained by the local network profile data.

The case in which none of the codecs in the resulting merged profile datasets are supported by the device, the profile data constitutes a misconfiguration between local network and device. It may not be possible to successfully establish a session in this case. It is suggested that the user agent provide feedback to the user indicating the misconfiguration. The user agent may also attempt to function in the network by ignoring one or more of the profile constraints.

3.1.2.5. Set in Device, User and Local Profiles

In this scenario everyone has an opinion on the codecs to be used. The device owner/manager wishes to define a set of codes based upon best interoperability of known end points in the environment. The user wishes to express preferences in the codecs (e.g. prefers wideband audio). The local network wishes to constrain the codecs based upon bandwidth (e.g. a wireless network with limited local network bandwidth, a SOHO network with dialup connectivity, a small office with shared 256kbps WAN connectivity). In the following scenario, device, user and local network profiles provide codec properties.

The merging of the data sources is as follows:

- The ordering of the codecs will be determined from the user profile data, which overrides the ordering from the device profile data.
- The set of codecs that may be used are the codecs listed in the device profile data, constrained by the list of codecs from the user profile data and further constrained by the list of codecs from the local network profile data.

The case in which none of the codecs in the resulting merged profile datasets are supported by the device, the profile data constitutes a misconfiguration between device, user and local network profiles. It may not be possible to successfully establish a session in this case. It is suggested that the user agent provide feedback to the user indicating the misconfiguration. The user agent may also attempt to

function in the network by ignoring one or more of the profile constraints.

3.1.2.6. Derived Requirements

1. A device will have a set of codecs supported, that may be offered. The list of codecs supported by a device may differ from the list of codecs in the device profile data. The list of codecs in the device profile data that get applied is the subset of the codecs supported by the device. Codecs listed in profiles that are not supported by the device are ignored.
2. The device profile data will have a default ordered list of codecs, which implies a preference order that may be offered.
3. The user profile data may provide an ordered list of user preferred codecs. The ordering of the codecs in the user profile data will override the ordering of the codecs in the device profile data. The user list of codecs may further constrain the list of codecs to be used.
4. The local network profile data may provide a list of codecs supported. This list will further constrain the list of codecs that may be offered.
5. The application profile data containing codec data will be ignored.
6. The profiles need the ability to express codecs that may be used and codecs that should not be used.

3.1.3. Transport Protocol Setting

This section describes use cases related to the use of the SIP transport protocol settings for a user agent. It is assumed that user agents are configurable to define what transport protocols (e.g. UDP, TCP, TLS) are to be used for the SIP signaling as well as the default order in which to attempt each of the protocol.

3.1.3.1. Setting Not Set

When none of the profiles are available or the profiles do not specify the SIP transport protocol setting, the device's default signaling transport(s) will be used.

3.1.3.2. Set in Device Profile

In the following scenario, the device profile is the only source of profile data. The signaling transports contained in the device profile may differ from the set of signaling transports supported by the device. This may be due to the administrator of the device profile wanting:

- To have a uniform use of signaling transports used across all device types.
- To mandate TLS for security reasons.
- To exclude the use of a specific signaling transport due to performance issues/concerns.
- To indicate the preferred, default order in which to attempt using each of the transport protocols.

This will result in the device profile data further constraining the list of signaling transports that could be used. The highest preference ordered signaling transport from the device profile dataset will be used first.

3.1.3.3. Set in Device and User Profiles

The following scenario extends the prior case described above. SIP transport protocol properties are provided in both the device and user profiles. Consider that SIP user agents, like email agents, may want to provide the user with options to:

- Mandate that secure transport must be used. If secure transport is not possible the user does not want to use the user agent.
- Prefer secure transport. Attempt to use secure transport. If secure transport will not work, use which ever transport protocol will make communication work.

When the user mandates the use of secure signaling transports only, the user wishes to constrain the available signaling transports to TLS. When indicating a preference to secure transport, the use is specifying a preference order for the use of transport protocols where TLS is the highest priority.

Now consider the merging strategy required to accomplish the goals of this use case scenario where the device and user profiles both contain SIP transport protocol properties. The merging of the data sources is as follows:

- The set of signaling transports that are allowed to be used is constrained by the device profile data. This is further constrained by the user profile data.
- The signaling transports attempted will be those from the merged, constrained list in order of highest to lowest priority.

3.1.3.4. Set in Device and Local Profiles

In the following scenario, device and local network profile data is available. The local network may have a limited set of signaling transports that it supports due to NAT or firewall constraints.

The merging of the data sources is as follows:

- o The set of signaling transports that may be used is the ordered list of signaling transports from the device profile data, further constrained by the local network profile data.

The case in which none of the local network data signaling transports are supported by the device profile data constitutes a misconfiguration between local network and device. The device might not be able to successfully establish a session in this case.

3.1.3.5. Derived Requirements

1. A device will have a set of signaling transports that it supports (note: one can be a set), with a default signaling transport.
2. The set of signaling transports supported by a device may differ from the set of signaling transports in the device profile data. The set of signaling transports in the device profile data is an ordered list, that is a subset of the set of signaling transports supported by the device. This may be due to performance issues associated with one of the signaling transport(s).
3. The user profile data may provide a list of preferred signaling transports to be used (e.g., TLS for securing the signaling).
4. The local network profile data provides a list of signaling transports supported, and will constrain the set of signaling transports that could be used.

3.2. Requirement Descriptions

3.2.1. Implementer Extensibility

Implementers must be able to differentiate each implementation. In addition, it does not serve user agent owners and administrators well to require an orchestrated upgrade for all user agent implementations and profile delivery servers before a new capability or feature can be supported with the required profile data. Hence one of the most important requirements is to support the ability of implementers to extend specified standard datasets to include additional related features and flexibility. It MUST be possible to extend a dataset without breaking user agents that support that dataset. This may require that user agents ignore parts of a dataset that it does not implement or extensions that it does support.

3.2.2. Flexible Capabilities

User agents vary quite widely in their capabilities. Some user agents function like traditional telephones. Some user agents support only text messaging. Some user agents support many media types such as video. Some user agents that function like a telephone have a single line, some have large numbers of lines. There is no

such thing as one size fits all. It MUST be possible for an implementer to choose which datasets to support based upon the capabilities that are supported by the user agent. The schema for containing the profile data MUST support a profile that contains only the data sets that a user agent supports. This allows the profile delivery server to create small profiles for specific devices. However a user agent SHOULD ignore properties for capabilities that it does not support. This allows the profile delivery server to be ignorant of the capabilities of the device. The degree to which the profile delivery server has intelligence of the user agent capabilities is an implementation choice.

3.2.3. XML

XML is perhaps not really a requirement, but a solution base upon requirements. However it is hard to ignore the desire to utilize readily available tools to manage and manipulate profile data such as XSLT, XPATH and XCAP. The requirement that should be considered when defining the schema and syntax is that many user agents have limited resources for supporting advanced XML operation. The simplest XML construct possible should be used, that support the required functionality. It is not a requirement that user agents validate the profile XML document. This relieves the requirement that the Relax NG schema defined in this and other datasets documents be enforced on the user agent. The Relax NG schema should not be used to strictly validate profile XML documents. Unknown elements and attributes should be ignored to allow extensions to be supported. Strict enforcement of the Relax NG schema would make it very difficult to deploy new user agents without lock step upgrades of the profile delivery server. Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols [RFC3470] provides useful information in this regard.

3.2.4. Access Control

Many user agents (e.g. appliances and softphones running on PCs) provide user interfaces that permit the user to edit properties that are logically part of user, application, device or local network profiles. Operators and administrators would like to be able to specify what an end user can change in those profiles and what an end user is not allowed to change. There may also be sensitive data the user agent requires to function, but that the operator of the system does not want the end user to see. For some properties the system operator may allow the user a fixed set of choices among the supported set of possible values. It MUST be possible to express whether an end user may change a dataset property. It MUST be possible to express that a property should not be made visible to the end user. It MUST be possible to express allowable values or ranges

that the end user may change a property to. The access control information SHOULD be optional to the dataset. It might be useful if it was possible to express the access control independent of the properties themselves. The access control specification by itself might be useful to express a general policy that the device owner or local network operator wish to impose.

3.2.5. Data Constraints and Range Definition

There is a need for property value types such as free form text, token/enumerations, integers, real numbers, etc. Many of these properties will have constrained values as opposed to the range of all possible values. These constrains may be due to protocol definitions, implementation limitations, and/or the desire (e.g. by the user, device owner, local network operator) to impose policy on the user agent. The ability to express the property constraints is useful from the perspective of access control as described in the above section. It is also useful to parameterize a user interface (e.g. on the user agent itself or on the profile delivery server) which provides a facility to modify profile data. It MUST be possible for the schema to specify property constraints as ranges or discrete sets of possible values. These constrains SHOULD be optional to the dataset. It might be useful if it was possible to express the constraints independent of the properties themselves. The constraints without the property values might be used to specify the capabilities of a particular user agent implementation.

3.2.6. Support of User, Application, Device, Local Network Sources

[I-D.ietf-sipping-config-framework] specifies a mechanism where the user agent retrieves profile data from as many as four different sources. The separation of the user profile facilitates a hotelling capability and the ability to easily re-assign a user to a different device. The separation of the local network profile facilitates properties specific to operating in the local network in a roaming scenario (e.g. outbound proxy or NAT traversal properties). The local network profile may also impose policy as describe in the next section. The device profile facilitates device capability based properties as well as a means for the device owner or manager (e.g. enterprise or service provider) to impose policy.

The multiple potential sources of profile data add some complexity to the user agent that must consolidate these separate profiles into a single working profile. It would be simpler if we could define each property as only allowed in one of the profiles. However it overly constrains the profiles and takes away desired functionality such as hotelling, roaming and shared profile management. It would also be simpler if we could define one rule for all profile datasets and

properties by which we merge the profile (e.g. local network profile overwrites user profile which overwrites device profile for all data). However this too is overly restrictive and eliminates some very useful functionality.

The rules to merge profile datasets needs to be defined for each dataset. In some cases an entire dataset must be considered atomic when merging one profile source with another. In other cases it makes sense to merge profile datasets, aggregating properties from the dataset provided in each of the profiles. It may also be desirable to have the effect of filtering of dataset properties. The desired effect might be for the owner of the device or the local network operator to constrain what values are allowed for properties in the profiles. This may also be the mechanism to facilitate imposing of policy as described in the next section. The operation of resolving overlapping datasets from multiple profiles, regardless of the means or net result, will be referred to as "merging" in this document.

A profile must have the means to constrain the merging algorithm. Due to the differences in the desired outcome for each data setting, the merging algorithm is specific to the setting. When defining a property setting, the merging algorithm must also be defined. A few of the more commonly used merging algorithms are defined in this document. Most settings are likely to use the common set defined in this document. However authors of profile datasets may define new algorithms for merging dataset properties (see Section 4.11 and Section 5.3).

3.2.7. The Ability to Specify Policy

Local network operators would like to impose policy on users and devices operating in their network. There is a need to constrain the operation and require specific behavior in the network. This might be as simple as to get access to the Internet, user agents must use a specified outbound proxy and NAT traversal mechanism. The network might have limited bandwidth such that the operator would like to constrain codecs or media streams to keep the network functional. The local network may provide emergency service behavior or functionality properties that are more specific than those provided by the device or user profile. The examples here focus on constraints to impose policy from the local network. However the facility to impose policy may be equally useful to the user and device profiles.

It MUST be possible to impose policy in any of the profile sources that constrains, overwrites or modifies properties provided in datasets from other sources.

4. Overall Dataset Schema

Notifiers and Subscribers of the event package defined in [I-D.ietf-sipping-config-framework] SHOULD support the content-type: application/uaprofile+xml. The Notifier SHOULD indicate all of the dataset schemas that is supports by listing all of the MIME types for the supported datasets in the SUBSCRIBE request header: Accept. This document defines an Relax NG Schema for that content-type with the namespace: urn:ietf:params:xml:ns:uaprof, for SIP Profile Datasets that provides:

- a base element type "setting" from which all settings in other schema definitions inherit (this allows other definitions to specify the content models for ways of combining settings; it is analogous to a C++ virtual base class).
- Attributes to the "setting" element that define constraints and other properties used to impose policy that apply to the element value. These attributes are inherited by elements that are derived from the abstract settings element.
- A root element for all property sets (the outermost container).

The full text of the schema is in Appendix A; the following describes the usage of the schema in defining properties and combining them to construct the working profile of a User Agent.

4.1. Data Primitives

Each property in a profile data set is defined using XML Schema Datatypes [W3C.REC-xmlschema-2] and Relax NG Schema. A property is modeled by an XML element derived from the "setting" element in the SIP Profile Data Set Schema. The element content is the setting value.

Properties consisting of one single value can be expressed using a single XML element. Properties that may consist of multiple values require the use of container elements. A container element is defined for such a property. This container can contain multiple XML elements, which each defines a possible value for that property (see examples in Section 4.5.2).

When constructing a property set, the creator of a profile may not be able to know all of the capabilities of the User Agent that will receive that property set. The creator of profile constraints or policies should be aware that a user agent may ignore properties that are unsupported or do not apply to its capabilities.

4.2. Use of Namespaces

XML namespaces [W3C.REC-xml-names-19990114] provide a means to uniquely identify the elements and datatypes defined in a data set. It is therefore RECOMMENDED that each data set specifies its own namespace. The namespace URIs SHOULD be URNs [RFC2141], using the namespace identifier 'ietf' defined by [RFC2648] and extended by [I-D.mealling-iana-xmlns-registry]. The core schema defined in this document defines the namespace: "urn:ietf:params:xml:ns:uaprof". Profile datasets that extend this schema SHOULD define a new namespace by appending a ":" and a unique name to the "urn:ietf:params:xml:ns:uaprof" namespace. These namespaces MUST be registered with IANA.

4.3. The 'propertySet' Element

The root element of a property set is "propertySet"; it is the container that is provided to the user agent. The elements contained within a propertySet contain the specific properties which are to be applied to the user agent. The properties may be simple types with a single value, complex types or container elements with a list of properties.

4.4. The Abstract 'setting_container' Element

The "setting_container" element is the abstract element in which a list of properties which allow mutliple values may be contained. Elements derived from the "setting_container" element may contain zero or more elements derived from the "setting" element. The "setting_container" element has an "excludedPolicy" attribute.

4.5. The Abstract 'setting' Element

The setting element is the abstract element from which all profile properties or settings shall inherit.

The setting element has a number of attributes that provide functionalities, which are generally useful for many properties. These attributes are inherited by properties that are derived from the settings element. This enables the re-use of common functionalities and ensures a common syntax for these elements across different data sets. The following functionalities are provided by attributes of the settings element:

- Property Access Control: 'visibility' attribute
- Policies: 'policy' attribute

Additional attributes are defined in the schema that may used in

elements derived from "setting". By default these attributes cannot be set. These attribute must be explicitly added to elements derived from the "setting" element:

- Unidirectional Properties: 'direction' attribute
- Preferences: 'q' attribute

4.5.1. The 'visibility' Attribute

The attribute "visibility" is defined on the "setting" element to specify whether or not the user agent is permitted to display the property value to the user. This is used to hide setting values that the profile administrator may not want the user to see or know. The "visibility" attribute has two possible values:

- visible: specifies that display of the property value is not restricted. This is the default value of the attribute if it is not specified.
- hidden: Specifies that the user agent SHOULD NOT display the property value. Display of the property value may be allowed using special administrative interfaces, but is not appropriate to the ordinary user.

4.5.2. The 'policy' Attributes

The setting element has an optional 'policy' attribute. The policy attribute is used to define the constraining properties of an element. It defines how the element value is used by an endpoint (e.g. whether it can or can not be used in a session). The following values are defined for the 'policy' attribute:

- allow: the value contained in the element is allowed and SHOULD be used in sessions. This is the default value that is used if the 'policy' attribute is omitted.
- disallow: the value contained in the element is forbidden and SHOULD NOT be used in sessions.

The policy attribute can be omitted if the default policy 'allow' applies.

> OPEN ISSUE: The policy attribute may not be needed for elements outside of a settings_container. Further clarification is needed on this. Using the policy attribute only inside containers would further simplify the specification of profile data.

4.5.3. The 'excludedPolicy' Attributes

The "setting_container" element has an optional 'excludedPolicy' attribute. This attribute specifies the default policy for all values that are not in the container. Elements that are present in the container have their own 'policy' attribute, which defines the policy for that element. The following values are defined for the 'excludedPolicy' attribute:

- allow: values not listed in the container are allowed and MAY be used in sessions. This is the default value that is used if the 'excludedPolicy' attribute is omitted.
- disallow: values not listed in the container are forbidden and MUST NOT be used in sessions.

The excludedPolicy attribute can be omitted if the default policy 'allow' applies. The following example shows a policy that allows the media type audio and disallows all other media types in sessions (effectively, this construct requires the use of audio):

```
<media-types excludedPolicy="disallow">
  <media-type policy="allow">audio</media-type>
</media-types>
```

4.5.4. The 'direction' Attribute

Some properties are unidirectional and only apply to messages or data streams transmitted into one direction. For example, a property for media streams can be restricted to outgoing media streams only. Unidirectional properties can be expressed by adding a 'direction' attribute to the respective element.

The 'direction' attribute can have the following values:

- recvonly: the property only applies to incoming messages/streams.
- sendonly: the property only applies to outgoing messages/streams.
- sendrecv: the property applies to messages/streams in both directions. This is the default value that is used if the 'direction' attribute is omitted.

4.5.5. The 'q' Attribute

It should be possible to express a preference for a certain value, if multiple values are allowed within a property. For example, it should be possible to express that the codecs G.711 and G.729 are allowed, but G.711 is preferred. Preferences can be expressed by adding a 'q' attribute to a property element. Elements derived from

the "setting" element for which multiple occurrences and values are allowed SHOULD have a "q" attribute if the order is significant. Typically these elements are contained in an element derived from the "setting_container" element. The 'q' attribute is only meaningful if the 'policy' attribute set to 'allowed'. It must be ignored in all other cases.

An element with a higher 'q' value is preferred over one with a lower 'q' value. 'q' attribute values range from 0 to 1. The default value is 0.5.

4.6. The 'profileUri' Element

The <profileUri> element contains the URI of this profile on the profile server. The value contained in the profileUri element may be different than the URI subscribe to when retrieving this profile. When the user agent retrieves a profile where the profileUri is different than the subscribe to URI, the user agent SHOULD unsubscribe to the current URI and then subscribe to the new URI.

The <profileUri> element is optional and MAY occur only once inside a <propertySet> element. The profileUri element is specific to the local-network, device, user or application profile in which it occurs. It has no meaning outside of the profile in which it occurs and SHOULD NOT be merged.

4.7. The 'profileCredential' Element

The <profileCredential> element contains the digest authentication information that SHOULD be used for authentication for the profile subscription via SIP or profile retrieval via HTTP, HTTPS, etc. The profileCredential element is optional and MAY occur only once inside a propertySet element. The profileCredential element is specific to the local-network, device, user or application profile in which it occurs. It has no meaning outside of the profile in which it occurs and SHOULD NOT be merged. The profileCredential element MUST contain exactly one of each of the elements: realm, authUser and one of either a1Digest or password.

4.7.1. realm Element

The realm element contains the string that defines the realm to which this credential pertains. The value of the realm element is the same as the realm parameter in the [RFC2617] headers: WWW-Authenticate, Authorization and the SIP [RFC3261] headers: Proxy-Authenticate and Proxy-Authorization. If a match of the realm value is found, the user agent uses the values in the authUser and a1Digest elements contained in the profileCredential element. Exactly one realm

element MUST be contained in a profileCredential element. A wildcard of "*" MAY be used as the realm value in which case the user agent MUST calculate the A1 DIGEST for the realm given in the authentication challenge. If the wildcard is given for the realm, the clear text form of the password contained in the password element MUST also be used.

4.7.2. authUser Element

The authUser element contains the string value of the "username" parameter which SHOULD be used in Authorization and Proxy-Authorization request headers when retrying a request that was challenged for authentication. Exactly one authUser element SHOULD be contained in a profileCredential element.

4.7.3. a1Digest Element

The a1Digest element contains a string with the value of the A1 digest of the username, realm and password as defined in [RFC2617]. At most one a1Digest element MUST be contained in a profileCredential element. The a1Digest element MUST NOT exist in a profileCredential element containing a password element. The username and realm used to construct the value of the a1Digest element MUST match the values of the realm and authUser elements contained in the profileCredential element with the a1Digest element.

4.7.4. password Element

The password element contains the clear text password for use with DIGEST Authentication [RFC2617]. At most one password MUST be contained in a profileCredential element. The password element MUST NOT exist in a profileCredential element containing a a1Digest element. The user agent uses this password along with the realm and authUser elements to calculate the A1 digest used for DIGEST Authentication.

4.8. The 'profileContactUri' Element

The <profileContactUri> element contains a contact URI (e.g. a SIP, HTTP URI or email address) under which the issuer of this profile can be reached. The contact element may, for example, contain the address of a support hotline.

The <profileContactUri> element is optional and MAY occur multiple times inside a <propertySet> element. Multiple instances of the profileContactUri element allow multiple URI schemes to be provided for contact information. The user agent MAY use the URI contained profile-contact-info element which has a URI scheme that the user

agent supports and can make work to provide support help for the profile. The user agent MAY provide the URIs to the user to contact the creator of the profile through other communication channels. The profileContactUri element is specific to the local-network, device, user or application profile in which it occurs. It has no meaning outside of the profile in which it occurs and SHOULD NOT be merged.

4.9. The 'profileInfo' Element

The <profileInfo> element provides a short textual description of the property that should be intelligible to the human user. This element may, for example, contain information about the nature of this profile, such as "Access Network Profile". The text in the <profileInfo> element is in particular be helpful when a user needs to decide whether or not to use a newly downloaded profile or when problems with a profile (e.g. a policy conflict) occur. A user agent MAY display this information in these cases.

The <profileInfo> element is optional and MAY occur only once inside a <propertySet> element. The profileInfo element is specific to the local-network, device, user or application profile in which it occurs. It has not meaning outside of the profile in which it occurs and SHOULD NOT be merged.

4.10. Example Profile Dataset

The following XML example shows a SIP Profile Dataset with example extension setting elements: ddd, foo, bar, boo, containerElement and setting container elements: myContainer, myContainer1, myContainer2 and container3.

```
<?xml version="1.0" encoding="UTF-8"?>
<propertySet xmlns="urn:ietf:params:xml:ns:uaprof">
    <profileUri>sip:a1b2c3d4e5f6@example.com</profileUri>
    <profileCredential>
         <realm>example.com</realm>
         <authUser>fred</authUser>
         <a1Digest>b6b577fd12aa7e1df8d60735ef56fc2e</a1Digest>
         <!-- <password>123</password> -->
    </profileCredential>
    <profileContactUri>tel:+16175551212</profileContactUri>
    <profileContactUri>sip:411@example.com</profileContactUri>
    <profileContactUri>
        http:example.com/sipProfile.html
    </profileContactUri>
    <profileInfo>
        This is an example profile from example.com
    </profileInfo>
    <ddd xmlns="blatz" policy="allow">fff</ddd>
    <foo xmlns="blatz" visibility="visible" policy="allow"
         direction="sendrecv" q="0">
    </foo>
    <bar xmlns="blatz" visibility="hidden" policy=""
         direction="sendonly" q="0.1000">
    </bar>
    <myContainer xmlns="blatz" excludedPolicy="disallow">
        <containerElement q="0.1">aaa</containerElement>
        <containerElement>bbb</containerElement>
        <containerElement q="0.8">ccc</containerElement>
    </myContainer>
    <boo xmlns="newns" q="1">ggg</boo>
    <myContainer1 xmlns="blatz" excludedPolicy="allow">
        <myContainer2 xmlns="newns" excludedPolicy="allow">
        </myContainer2>
    </myContainer1>
    <container3 xmlns="ns3">
        <containerElement q="0.1">111</containerElement>
        <containerElement>222</containerElement>
        <containerElement q="0.8">333</containerElement>
    </container3>
</propertySet>
```

4.11. Merging Property Sets

A UA may receive property sets from multiple sources, which need to be merged into a single combined document the UA can work with.

Properties that have a single value (e.g. the maximum bandwidth allowed) require that a common value is determined for this property

during the merging process. The merging rules for determining this value need to be defined individually for each element in the schema definition. Properties that allow multiple values (i.e. property containers) need to be merged by combining the values from the different data sets. The following sections describe common merging algorithms. A data set definition may refer to these algorithms.

4.11.1. Single Numeric Value Merging Algorithm

A general merging rule for elements with numeric values is to select the largest or the smallest value. For example, a merging rule for a <max-bandwidth> element would be to select the smallest value from the values that are in the competing data sets.

4.11.2. Multiple Enumerated Value Merging Algorithm

Multiple values in property containers are merged by combining the values from each of the competing data sets. This is accomplished by copying the elements from each property container into the merged container. Elements with identical values are only copied once. The 'policy' attribute of two elements with the same value is adjusted during the merging process according to Table 1. If an element exists only in one property container, then the default policy of the other container (i.e. the excludedPolicy) is used when accessing Table 1. For example, if an element is disallowed in one data set and the element is not contained in the other data set but the default policy is allowed for that data set, then the values disallowed and allowed are used to access Table 1. Consequently, the element will be disallowed in the merged data set. Finally, the excludedPolicy attributes of the containers are also merged using Table 1. In addition to these merging rules, each schema may define specific merging rules for each property container.

set 1 \ set 2	allow	disallow
allow	allow	disallow
disallow	disallow	disallow

Table 1: merging policies.

The following example illustrates the merging process for two data sets. All elements are merged into one container and the policy attributes are adjusted according to Table 1. The merged container has the default policy disallow, which is determined using Table 1. The entry for PCMA in the merged data set is redundant since it has the default policy.

```
Data set 1:
<codecs excludedPolicy='allow'>
  <codec policy='disallow'>PCMA</codec>
</codecs>

Data set 2:
<codecs excludedPolicy='disallow'>
  <codec policy='allow'>PCMA</codec>
  <codec policy='allow'>G729</codec>
</codecs>

Merged data set:
<codecs excludedPolicy='disallow'>
  <codec policy='disallow'>PCMA</codec>
  <codec policy='allow'>G729</codec>
</codecs>
```

Some constellations of policy attributes result in an illegal merged data set. They constitute a conflict that can not be resolved automatically. For example, two data sets may define two non-overlapping sets of allowed audio codecs and both disallow all other codes. The resulting merged set of codecs would be empty, which is illegal according to the schema definition of the codecs element. If the use of these properties is enforced by both networks, the UA may experience difficulties or may not be able to set up a session at all.

The combined property set MUST again be valid and well-formed according to the schema definitions. A conflict occurs if the combined property set is not a well-formed document after the merging process is completed.

4.11.3. Closest Value First Merging Algorithm

Some properties require that the values from different data sets are ordered based on the origin of the data set during the merging process. Property values that come from a domain close to the user agent take precedence over values that were in a data set delivered by a remote domain. This order can be used, for example, to select the property value from the closest domain. In many cases, this is the local domain of the user agent. For example, the URI of an outbound proxy could be merged this way. This order can also be used to generate an ordered list of property values during the merging process. For example, multiple values for media intermediaries can be ordered so that the closest media intermediary is traversed before the second closest intermediary and so on.

This merging algorithm requires that the source of a data set is

considered.

If property sets are delivered through the configuration framework [I-D.ietf-sipping-config-framework], the value received through a subscription using the "local-network" profile-type takes precedence over values received through other profile-type subscriptions.

OPEN ISSUE: Can we define an order for 'device', 'user', and 'application' profiles?

The session-specific policy mechanism [I-D.hilt-sipping-session-policy-framework] provides an order among policy servers. This order is based on the order, in which a SIP message traverses the network, starting with the closest domain. This order can directly be used to order property values as described above.

4.12. Common Types

The schema also defines a set of common types that are used in defining data sets (e.g. DataIpPort). [Need to document common types.]

5. Defining Data Sets

This section covers several issues that should be take into consideration when specifying new data set schemas. This is intended to be a guide to authors writing specifications defining a new data set schema or extensions to existing ones.

5.1. Namespace

It is RECOMMENDED that a data set defines a new XML namespace [W3C.REC-xml-names-19990114] to scope all of the properties that are defined in the name space.

5.2. Property Definitions

The properties defined in a data set schema may be simple (i.e. having a single value) or they may be complex (i.e. a container with multiple values). Each property in the data set SHOULD inherit from the "setting" element. Complex properties and all of their child elements each should inherit from "setting" as well.

A data set specification should contain a section which defines the meaning of all of the properties contained in the data set. The objective is to define the property such that implementers have a

clear definition and semantics to interpret properties in a consistent way. User agents not only need to use the same profile content, they need to apply the properties in a consistent way to achieve true interoperability.

The following information should be defined for each property in a data set:

- description: describe the meaning and application of the property.
- cardinality: define how many instances of this property element may occur in a data set (e.g. zero, one or many) as well as its relationship to any other properties in this or other data sets.
- default value: define the default value of this property if it is not set. Describe if the default is different if the property is present and not set vs. completely absent from the data set. Define if the default varies in relation to another property.

5.3. Merging Data Sets

User agents may receive data sets from multiple sources. They need to merge these data sets in order to create an overall data set they can work with. Collisions on data sets may occur if multiple sources provide different values for the same properties. These collisions need to be resolved during the merging process.

A data set schema MUST define rules for merging data sets from different sources for each property that is defined. Considerations for merging data sets are discussed in Section 4.11. A data set schema must define if and how these consideration apply and MAY define alternative merging rules for specific settings. A data set schema must also identify combinations of properties that constitute a conflict that can't resolved. It may provide additional guidelines for the behavior of a user agent in these cases.

6. Candidate Data Sets

The following sections name some of the candidate data sets that are or may be defined. These data sets can be aggregated to form profiles appropriate to the capabilities of a user agent implementation.

- SIP Protocol Data Set: the lowest common denominator set of properties common to all SIP user agents of any capability. A schema covering the elements of this data set can be found in [I-D.petrie-sipping-sip-dataset].
- Media Data Set: this data set contains media related policies. A schema covering the elements of this data set can be found in [I-D.ietf-sipping-media-policy-dataset].

- Identity Data Set: AORs and lines (see [I-D.petrie-sipping-identity-dataset])
- HTTP Protocol Data Set: server settings. Proxy for clients.
- NAT Traversal Data Set: settings for STUN, TURN etc.
- SIP Digit Maps Data Set: [I-D.petrie-sipping-voip-features-dataset]
- VoIP Features: [I-D.petrie-sipping-voip-features-dataset]
- Address Book:
- Buddy List:

7. Security Considerations

Security is mostly a delivery problem. The delivery framework SHOULD provide a secure means of delivering the profile data as it may contain sensitive data that would be undesirable if it were stolen or sniffed. Storage of the profile on the profile delivery server and user agent is an implementation problem. The profile delivery server and the user agent SHOULD provide protection that prevents unauthorized access of the profile data. The profile delivery server and the user agent SHOULD enforce the access control policies defined in the profile data sets if present.

[The point of the access control construct on the data set is to provide some security policy on the visibility and ability to change sensitive properties. Does the access control mechanism also create a security problem where the local network can set or hide properties from the user?]

Some transport mechanisms for delivery of the profile data do not provide a secure means of delivery. In addition some user agents may not have the resources to support the secure mechanism used for delivery (e.g. TLS).

8. IANA Considerations

XML name space registration: urn:ietf:params:xml:ns:uaprof

8.1. Content-type registration for 'application/uaprofile+xml'

To: ietf-types@iana.org
Subject: Registration of MIME media type application/uaprofile+xml
MIME media type name: application
MIME subtype name: uaprofile+xml

Required parameters: (none)
Optional parameters: charset Indicates the character encoding of enclosed XML. Default is UTF-8.
Encoding considerations: Uses XML, which can employ 8-bit characters, depending on the character encoding used. See RFC 3023 [RFC 3023], section 3.2.
Security considerations: This content type is designed to carry SIP user agent profile data, which may be considered private information. Appropriate precautions should be adopted to limit disclosure of this information.
Interoperability considerations: This content type provides a common format for exchange of SIP user agent profile information.
Published specification: RFC XXXX (Note to RFC Editor: Please fill in XXXX with the RFC number of this specification)
Applications which use this media type: SIP user agents and profile delivery servers.
Additional information: Magic number(s): File extension(s): Macintosh File Type Code(s):
Person & email address to contact for further information: Daniel Petrie EMail: dan.ietf AT sipez DOT com
Intended usage: LIMITED USE
Author/Change controller: This specification is a proposed work item of the IETF SIPPING working group, with mailing list address: sipping@ietf.edu.
Other information: This media type is a specialization of application/xml [RFC 3023], and many of the considerations described there also apply to application/uaprof+xml.

9. Change History

[[RFC Editor: Please remove this entire section upon publication as an RFC.]]

9.1. Changes from draft-petrie-sipping-profile-datasets-04

Re-reved and activated draft

9.2. Changes from draft-petrie-sipping-profile-datasets-03

Converted the XML schema to use Relax NG and created a valid schema.

Defined XML name space for schema: "urn:ietf:params:xml:ns:uaprof"

Defined mime type: application/uaprofile+xml to be used as default content type for the configuration framework.

Changed names of elements, attributes and other data types which

contained "-" or "_" to use camel case.

Added password element so that credential can contain either A1 digest or clear text password as the clear text password is required by the user agent in some cases (e.g. http GET) or to wildcard the realm.

9.3. Changes from draft-petrie-sipping-profile-datasets-02

Removed "mandatory" policy attribute value to simplify use and profile merging issues.

Session Independent Policy draft was split into separate media dataset draft and policy drafts. Fixed references and information in this draft to reflect the two drafts.

Added concrete properties contained in elements: profileUri, profileCredential, profileContactUri and profileInfo. Prior to this draft, there were only abstract elements defined in this draft. These elements have been added to contain information that is specific to the profile and are independent of the specific profile datasets contained in the profile.

Split references into normative and informative sections.

Numerous editorial changes

9.4. Changes from draft-petrie-sipping-profile-datasets-01

Split out the core SIP Protocol dataset into a separate draft.

Schema changes: created setting_container, added q and direction attributes along with other tweaks to the schema.

Better integration and coordination with [I-D.ietf-sipping-media-policy-dataset]. The media/codec dataset is now completely contained in the policy draft.

9.5. Changes from draft-petrie-sipping-profile-datasets-00

Added use case scenarios for codecs, SIP transport protocol and outbound proxy to better illustrate requirements. Some of the derived requirements are listed with the use cases.

Added settings element attributes "policy" and "visibility" to provide merging constraints and access control capability. Removed the element based merging constraints using the: forbid, set_any, set_all and set_one elements. This greatly simplifies the degree of

XML operations required to perform the request merging.

Defined default merging policy and profile source precedence along with the option for different policies to be describe in specific settings definition documents.

Added example merging with XML profiles from device and user for the SIP transport protocol.

10. References

10.1. Normative References

[I-D.hilt-sipping-session-policy-framework]
Hilt, V., "A Framework for Session Initiation Protocol (SIP) Session Policies", draft-hilt-sipping-session-policy-framework-01 (work in progress), March 2006.

[I-D.ietf-sipping-config-framework]
Channabasappa, S., "A Framework for Session Initiation Protocol User Agent Profile Delivery", draft-ietf-sipping-config-framework-14 (work in progress), November 2007.

[I-D.mealling-iana-xmlns-registry]
Mealling, M., "The IETF XML Registry", draft-mealling-iana-xmlns-registry-05 (work in progress), June 2003.

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

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

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

[W3C.REC-xml-names-19990114]
Hollander, D., Bray, T., and A. Layman, "Namespaces in XML", W3C REC REC-xml-names-19990114, January 1999.

[W3C.REC-xmlschema-1]
Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, May 2001, <http://www.w3.org/TR/xmlschema-1/>.

[W3C.REC-xmlschema-2]
Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", W3C REC-xmlschema-2, May 2001, <http://www.w3.org/TR/xmlschema-2/>.

10.2. Informative References

[I-D.ietf-sipping-media-policy-dataset]
Hilt, V., "A User Agent Profile Data Set for Media Policy", draft-ietf-sipping-media-policy-dataset-04 (work in progress), May 2007.

[I-D.petrie-sipping-identity-dataset]
Petrie, D., "The Session Initiation Protocol User Agent Identity Profile Data Set", draft-petrie-sipping-identity-dataset-01 (work in progress), October 2006.

[I-D.petrie-sipping-sip-dataset]
Petrie, D., "The Core Session Initiation Protocol User Agent Protocol Data Set", draft-petrie-sipping-sip-dataset-02 (work in progress), October 2006.

[I-D.petrie-sipping-voip-features-dataset]
Petrie, D., "The Session Initiation Protocol User Agent VoIP Features Data Set", draft-petrie-sipping-voip-features-dataset-01 (work in progress), October 2006.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

[RFC2648] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[RFC3470] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols", BCP 70, RFC 3470, January 2003.

Appendix A. Relax NG SIP UA Profile Schema

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
 ns="urn:ietf:params:xml:ns:uaprof"
 datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
   <start>
      <element name="propertySet">
          <optional>
             <ref name="ElementProfileUri"/>
          </optional>
          <optional>
             <ref name="ElementProfileCredential"/>
          </optional>
          <zeroOrMore>
             <element name="profileContactUri">
                 <!-- who to contact for help with this profile -->
                 <data type="anyURI"/>
             </element>
          </zeroOrMore>
          <optional>
              <element name="profileInfo">
                  <text/>
              </element>
          </optional>
          <zeroOrMore>
              <choice>
                  <ref name="PropertySetExtension"/>
                  <ref name="ElementGenericSetting"/>
                  <ref name="ElementGenericSettingContainer"/>
              </choice>
          </zeroOrMore>
      </element>
   </start>
   <!-- example setting with all setting attributes -->
   <!-- <define name="ElementFoo">
            <element name="foo">
                <ref name="SettingAttributes"/>
                <text/>
             </element>
     </define>
    -->
   <define name="ElementProfileUri">
       <!-- URI to subscribe to for this profile -->
       <element name="profileUri">
           <ref name="DataSipUri"/>
       </element>
   </define>
   <define name="ElementProfileCredential">
       <!-- credentials for subscribing or getting profile -->
```

```
            <element name="profileCredential">
                <ref name="DataCredential"/>
            </element>
        </define>
        <define name="PropertySetExtension">
            <!-- place to add new settings in other namespaces -->
            <empty/>
        </define>
        <define name="ElementGenericSettingContainer">
            <element>
                <anyName>
                    <except>
                        <nsName ns="urn:ietf:params:xml:ns:uaprof"/>
                        <nsName ns=""/>
                    </except>
                </anyName>
                <ref name="SettingContainerAttributes"/>
                <zeroOrMore>
                    <ref name="AttributeGeneric"/>
                </zeroOrMore>
              <!-- container can have containers or settings not both -->
                <choice>
                    <zeroOrMore>
                        <ref name="ElementGenericSetting"/>
                    </zeroOrMore>
                    <zeroOrMore>
                        <ref name="ElementGenericSettingContainer"/>
                    </zeroOrMore>
                </choice>
            </element>
        </define>
        <define name="ElementGenericSetting">
            <element>
                <anyName>
                    <except>
                        <nsName ns="urn:ietf:params:xml:ns:uaprof"/>
                        <nsName ns=""/>
                    </except>
                </anyName>
                <ref name="SettingAttributes"/>
                <zeroOrMore>
                    <ref name="AttributeGeneric"/>
                </zeroOrMore>
                <zeroOrMore>
                    <choice>
                        <text/>
                        <ref name="ElementGenericSetting"/>
                    </choice>
```

```
            </zeroOrMore>
        </element>
    </define>
    <define name="AttributeGeneric">
        <attribute>
            <anyName>
                <except>
                    <nsName ns="urn:ietf:params:xml:ns:uaprof"/>
                    <nsName ns=""/>
                </except>
            </anyName>
        </attribute>
    </define>
    <define name="DataCredential">
        <element name="realm">
            <text/>
        </element>
        <element name="authUser">
            <text/>
        </element>
        <choice>
            <element name="a1Digest">
                <data type="string">
                    <param name="pattern">[0-9,a-f]{32,32}</param>
                </data>
            </element>
            <element name="password">
                <text/>
            </element>
        </choice>
    </define>
    <define name="DataSipUri">
        <choice>
            <data type="anyURI">
                <param name="pattern">sip:.*</param>
            </data>
            <data type="anyURI">
                <param name="pattern">sips:.*</param>
            </data>
        </choice>
    </define>
    <define name="DataSipNameAddr">
        <choice>
            <data type="anyURI"><!-- need to tighten this up -->
                <param name="pattern">"?.*"?&lt;?sip:.*</param>
            </data>
            <data type="anyURI">
                <param name="pattern">"?.*"?&lt;?sips:.*</param>
```

```
            </data>
        </choice>
    </define>
    <define name="SettingContainerAttributes">
        <optional>
            <attribute name="excludedPolicy">
                <ref name="DataPolicies"/>
            </attribute>
        </optional>
    </define>
    <define name="SettingAttributes">
        <interleave>
           <optional>
               <ref name="AttributePolicy"/>
           </optional>
           <optional>
               <ref name="AttributeVisibility"/>
           </optional>
           <optional>
               <ref name="AttributeDirection"/>
           </optional>
           <optional>
               <ref name="AttributeQ"/>
           </optional>
        </interleave>
    </define>
    <define name="AttributePolicy">
       <attribute name="policy">
           <ref name="DataPolicies"/>
       </attribute>
    </define>
    <define name="DataPolicies">
        <choice>
            <value></value><!-- default of allow -->
            <value>allow</value>
            <value>disallow</value>
        </choice>
    </define>
    <define name="AttributeVisibility">
        <attribute name="visibility">
           <choice>
               <value></value><!-- default of visible -->
               <value>visible</value>
               <value>hidden</value>
           </choice>
        </attribute>
    </define>
    <define name="AttributeDirection">
```

```
          <attribute name="direction">
             <choice>
                 <value></value><!-- default of sendrecv -->
                 <value>sendrecv</value>
                 <value>sendonly</value>
                 <value>recvonly</value>
             </choice>
          </attribute>
      </define>
      <define name="AttributeQ">
          <attribute name="q">
             <data type="float">
                 <!-- default of 0.5 -->
                 <param name="minInclusive">0</param>
                 <param name="maxInclusive">1</param>
             </data>
          </attribute>
      </define>
      <define name="DataIpPort">
          <data type="integer">
              <param name="minInclusive">1</param>
              <param name="maxInclusive">65535</param>
          </data>
      </define>
      <define name="DataIpTransport">
          <choice>
              <value></value><!-- default of UDP -->
              <value>UDP</value>
              <value>TCP</value>
              <value>TLS</value>
              <value>DTLS</value>
              <value>SCTP</value>
          </choice>
      </define>
  </grammar>
```

Appendix B. Acknowledgments

Authors' Addresses

Daniel Petrie
SIPez LLC.
34 Robbins Rd.
Arlington, MA 02476
US

Phone: +1 617 273 4000
Email: dan.ietf AT SIPez DOT com
URI: http://www.SIPez.com/

Sumanth Channabasappa
CableLabs
858 Coal Creek Circle
Louisville, CO 80027
US

Phone:
Email: sumanth@cablelabs.com
URI:

Sam Ganesan
Motorola
80 Central Street
Boxborough, MA 01719
US

Phone:
Email: sam.ganesan@motorola.com
URI:

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).