SIMPLE
Internet-Draft
Expires: August 22, 2005

J. Rosenberg
Cisco Systems

February 21, 2005

# A Data Model for Presence
## draft-ietf-simple-presence-data-model-02

## Status of this Memo



This Internet-Draft will expire on August 22, 2005.

## Copyright Notice



## Abstract

This document defines the underlying presence data model and used by Session Initiation Protocol (SIP) for Instant Messaging Leveraging Presence Extensions (SIMPLE) presence agents. The data model provides guidance on how to map various communications systems into presence documents in a consistent fashion.

Table of Contents

## 1. Introduction

Presence conveys the ability and willingness of a user to communicate across a set of devices. RFC 2778 [1] defines a model and terminology for describing systems that provide presence information. RFC 3863 [2] defines an XML document format for representing presence information. In these specifications, presence information is modeled as a series of tuples, each of which contains a status, communications address, and other markup. However, neither specification gives guidance on exactly what a tuple is meant to model, and how to map real world communications systems (and in particular, those built around the Session Initiation Protocol (SIP) [3]) into a presence document.

In particular, several important concepts are not clearly modeled or well delineated by RFC 2778. These are:

Service: A communications service, such as instant messaging or telephony, is a system for interaction between users that provides certain modalities or content.

Device: A communications device is a physical component that a user interacts with in order to make or receive communications. Examples are a phone, PDA or PC.

Person: A person is the end user, and for the purposes of presence, is characterized by states, such as "busy" or "sad" which impact their ability and willingness to communicate.

This specification defines these concepts more fully by means of a presence data model, and concretely defines how to take real world systems and map them into presence documents using that model.

## 2. Definitions

This document makes use of many new terms, which are defined here.

Device: A device models the physical environment in which services manifest themselves for users. Devices have characteristics which are useful in allowing a user to make a choice about which communications service to use.

Service: A service models a form of communications that can be used to interact with the user.

Person: A person models the human user and their states that are relevant to presence systems.





Instance: A single description of a particular service, a particular device or a person. There may be multiple instances for a particular service or device, or multiple person instances in a document, in cases where there is ambiguity that is best resolved by the watcher.

Presentity: A presentity combines devices, services and person information for a complete picture of a user's presence status on the network.

Presentity URI: A URI that acts as a unique identifier for a presentity, and provides a handle for obtaining presence information about that presentity.

Data Component: One of the device, service, or person parts of a presence document.

Status: Dynamic information about a service, person or device.

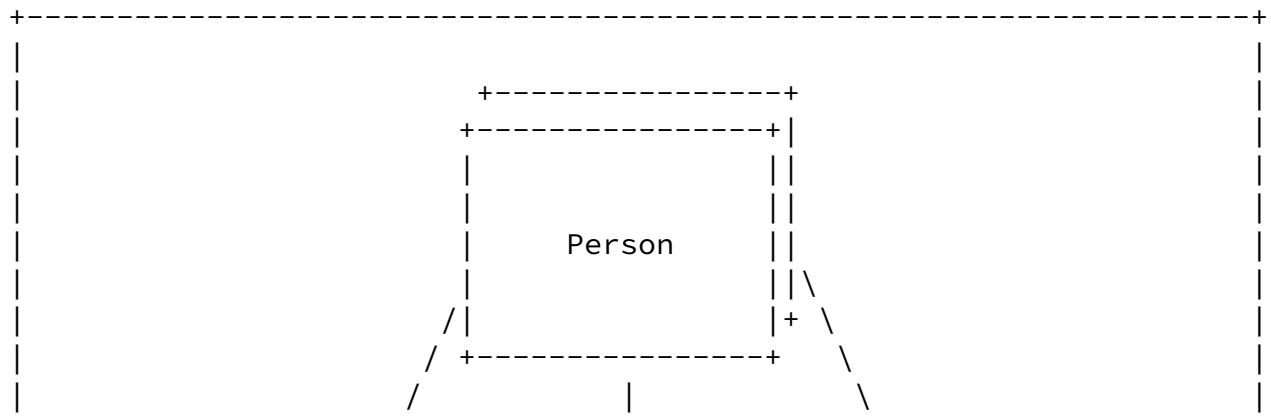
Characteristics: Static information about a service, person or device. Useful in providing context that identifies the service or device as different from another service or device.

A status or characteristic. It represents a single piece of presence information.

Presence Attribute: A synonym for attribute.

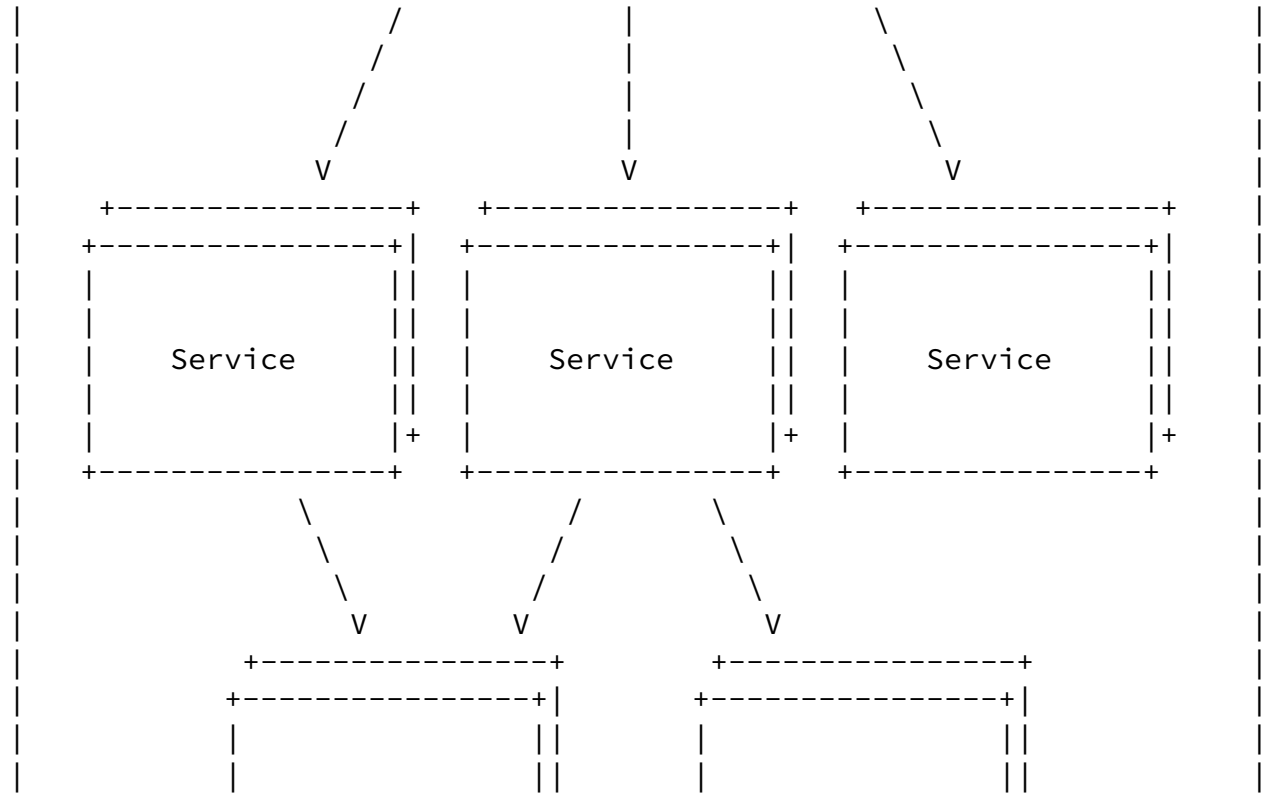
Composition: The act of combining a set of presence and event data about a presentity into a coherent picture of the state of that presentity.

## 3. The Model







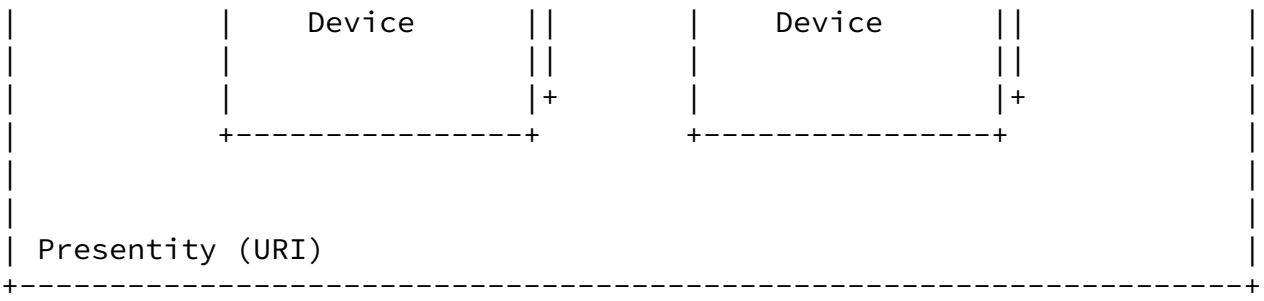



Figure 1

The data model for presence is shown in Figure 1. The model seeks to describe the presentity. There are four components in the model. They are the presentity URI, the person, the service, and the device. The latter three data components contain information (called attributes) that provide a description about the service, person, or device. It is central to this model that each attribute is affilitated with the service, person, or device because they describe that service, presentity or device. This is in contrast to a model whereby the attributes are associated with the service, presentity, or device because they were reported by that service, presentity, or device. As an example, if a cell phone reports that a user is in a





meeting, this would be done by including an attribute as part of the person information, indicating a status of "in-a-meeting". The presence information may also include information on the cell phone as a device. However, even though it is that device which is reporting that the user is in a meeting, the busy indicator is not associated with the device, it is associated with the user and thus placed in the person component.

### 3.1 Presentity URI

The identifier for the presentity is a URI. For each unique presentity in the network, there is one or more presentity URIs. A presentity may have multiple URI because they are identified by both a pres URI [4] and a protocol specific URI, such as a SIP URI [3] or

an XMPP URI [5]. Or, it can be because a user has several aliases in a domain, all of which are equivalent identifiers for the presentity.

When a document is constructed, the presentity URI is ideally set to the identifier used to request the document in the first place. For example, if a document was requested through a SIP SUBSCRIBE request, the presentity URI would match the Request URI of the SUBSCRIBE request. This follows the principle of least surprise, since the entity requesting the document may not be aware of the other identifiers for the presentity.

Independent of its scheme, the presentity URI is independent of any of the services or devices that the presentity possesses. However, the URI is not just a name - it represents a resource that can be subscribed to, in order to find out the status of the user. When the URI is a SIP URI, it will often be the address-of-record for the user, to which SIP calls can be directed. This equivalence is not mandated by this specification, but is a recommended configuration for easing the burden of remembering and storing identifiers for users.

## 3.2 Person

The person data component models information about the user whom the presence data is trying to describe. This information consists of characteristics of the user, and their status.

Characteristics of a person are the static information about a user that does not change under normal circumstances. Such information might include physical characteristics, such as age and height. Status information about a presentity represent the dynamic information about a user. These typically are things the *user* is doing, places the *user* is at, feelings the *user* has, and so on. Examples of typical person status are "in a meeting", "on the phone",





"out to lunch", "happy" and "writing Internet Drafts". The line between static status information and dynamic status information is fuzzy, and it is not important that a line be drawn. The model does not differentiate in a semantically meaningful way between these two types of attributes.

In the model, there can only be one person component per presentity.

In other words, the person component models a single human being, and includes characteristics and status that are related to the communication states for a single human being. Of course, the system has no way to verify that the human described by the person component is actually a single human being, as opposed to a group of users, or even a dog for that matter. As the saying goes, "on the Internet, no one knows you are a dog", and the same is true here. The person component is a facade for a single person; anything that can be made to look like a single person can be modeled with that facade.

As an example, consider the task of using a presence document to describe a customer support help desk. The person component can be considered to be "busy" if none of the support staff are available, and "at lunch" if the help desk department has a group lunch together. The watcher that receives the document will consider the help desk to be a single person; nothing in the document (except perhaps the note element, should its value be "help desk" or something similar) conveys information that would indicate that the person in question is actually a help desk.

However, there can be multiple instances of the person component. This occurs in cases where the state of the person component is ambiguous, as discussed in Section 3.5.

## 3.3 Service

Each presentity has access to a number of services. Each of these represent a point of reachability for communications that can be used to interact with the user. Examples of services are telephony (that is, traditional circuit-based telephone service), push-to-talk, instant messaging, Short Message Service (SMS), and Multimedia Message Service (MMS).

It is difficult to give a precise definition for service. One reasonable approach is to model each software or hardware agent in the system as a service. If a user starts a softphone application on their PC, then that represents a service. If a user has a videophone device, then that represents another service. This is effectively a physical view of services. This definition, however, starts to fall apart when a service is spread across multiple software agents or devices. For example, a SIP URI representing an address-of-record

can be routed to a softphone or a videophone, or both. In that case, one might attempt instead to define a service based on its address on the network. This definition also falls apart when modeling devices or applications that receive calls and dispatch them to different "helpers" based on potentially complex logic. For example, a cellular telephone might house multiple SIP applications, each of which can "register" different handlers based on the method or even body type of the request. Each of those applications or handlers can rightfully be considered a service, but it doesn't have an address on the network distinct from the others.

Because of this inherent difficulty in precisely defining a service, the data model doesn't try to constrain what can be considered a service. Rather, anything can be considered a service so long as it exhibits a set of key properties defined by this model. In particular, Each service is associated with characteristics which identify the nature and capabilities of that service, with reach information that indicates how to connect to the service, with status information representing the state of that service, and relative information that describes the ways in which that service relates to others associated with the presentity.

As a consequence, in this model, services are not explicitly enumerated. There is no central registry by which one goes finds the identifiers for each service. Consequently, each service does not have a single "service" attribute that with values of "ptt" or "telephony". That doesnt mean that these consolidated monikers aren't useful; indeed, they represent an essential summary of what the service is. Such summarization is useful in creating icons that allow a user to choose one service over another. A watcher is free to create such summarization information from any of the information associated with a service. The reach information often provides valuable information for creating such a summarization. Oftentimes, the scheme of the URI is synonomous with the view of what a service is. An "sms" URI [6] clearly indicates SMS, for example. For some URIs, there may be many services available, for example, SIP or tel [7], in which case the scheme is less meaningful to create a summarization. The reach information could also indicate that certain application software has to be invoked (such as a videogame), in which case that aspect of the reach information would be useful for generating an iconic representation of the game.

### 3.3.1 Characteristics

Each service is adorned with characteristics that describe the nature and capabilities of the service that will be experienced when a watcher invokes that URI. The nature of a service is a set of properties that are unchanging across communication sessions

established to that service. The nature of a service tends to be descriptive. Examples of the nature of a service are that it represents an interactive voice response or voicemail server, that it is an auomata, or that it is a telephony service used for the purposes of work. Capabilities, on the other hand, represent properties that might be exhibited, and whether or not they are exhibited depends on negotiation and other dynamic functions that take place during session establishment. Examples of such capabilities are the type of media that might be used, the directionality of communications that are permitted, the SIP extensions supported, and so on. Capabilities can be very complex; RFC 2533, for example [8] describes a model for representing capabilities through N-ary boolean functions. It is difficult to differentiate a capability with one modality (this service only does voice) from a characteristic that represents the nature of a service. However, it is not important to do so.

Characteristics are important when multiple services are indicated. That is because the purpose of listing multiple services in a presence document is to give the watcher a *choice*. That is, the presentity is explicitly offering the watcher an opportunity to contact them using a multiplicity of different services. To help the watcher make a decision, the presence document includes characteristics of each service which help differentiate the services from each other, and give the watcher the context in which to make a choice.

Because their purpose is primarily to facilitate choice, capabilities do not impose a requirement on the the way in which a user reaches that service. For example, if a presence document includes two services, and one supports audio only, while the other supports only video, this does not mean that, when contacting the first service, a user has to offer only an audio stream, or when contacting the second service, a user has to offer only a video stream. A user can use local policy at its discretion in determining what capabilities or communications modalities are offered when they choose to connect with a service. It is not necessary for a watcher to add SIP caller preferences [9] in order to request routing of the request to a service with the characteristics described in the document.

If, in order to reach a service, the user agent must generate a request that exhibits a particular capability or contains a specific header, then this is indicated separately in the reach information,

described below.

One important characteristic of each service is the list of devices on which that service executes. Each device is identified uniquely by a device ID. As such, the service characteristics can include a





list of device IDs. A presence document might also contain a information on each device, but this is a separate part of the document. Indeed, the information on each device might not even be present in the document. In that case, the device IDs listed for each service are nothing more than correlation identifiers, useful for determining when two services run on the same device. The benefit of this model is that information on the devices can be filtered out of a presence document, yet the service information, which includes the device IDs, remains useful and meaningful.

It is perfectly valid for a presence document to contain just a single service. This is permitted even if the presentity actually has multiple services at their disposal. The lack of multiple services in the document merely means that the presentity is not offering a choice to the watcher. In such a case, the service characteristics are less important, but may be helpful in allowing a watcher to decide if they wish to communicate at all.

### 3.3.2 Reach Information

The reach information for a service are the instructions for the recipient of a document on how to correctly contact that service.

When a service is accessible over a communications network, reach information includes a URI that can be "hit" in order to access the service. This URI is called the service URI. However, some services are not accessible over a communications network (such as in-person communications or a written letter), and as such, may not utilize a URI.

Even for services reachable over a communications network, the URI alone may not be sufficient. For example, several applications may be running within a cellular telephone, both of which are reachable through the users SIP address-of-record. However, one application is launched when the INVITE request contains a body of a particular type, and the other is launched for other body types. As another

example, a service may provide complex application logic which operates correctly only when contacted from matching application software. In such a case, even though the communications between instances utilizes a standard protocol (such as SIP), the user experience will not be correct unless the applications are matched.

When the URI is not sufficient, additional attributes of the service can be present that define the instructions on how the service is to be reached. These attributes must be understood in order for the service to be utilized. If a watcher receives a presence document containing reach information it does not understand, it should discard the service information.





The reach information is an important part of the service. When the watcher makes a decision about which service of the presentity they wish to access, the watcher utilizes the reach information for that service. For this reason, each service has to have a unique set of reach information. If this was not the case, the user would have no way to choose between the services. This means that the reach information represents a unique identifier for the service. However, a presence document can contain multiple instances of a particular service, each of which contains the same reach information, but differs in its instance identifier. Multiple instances of a service exist in a document when the state of the service is ambiguous, as discussed in Section 3.5.

Because the reach information serves as an idenfifier for a service, it also serves as a way to figure out whether something is one service or two. Something cannot be a service unless there is a way to reach it separately from another service. As an example, consider a softphone application that can do audio and video. It is not possible to describe this softphone as two services - one capable of just audio, and one capable of just video. Thats because there is no way to reach the video-only service, for example; sending a SIP INVITE with just a video stream doesn't suffice, since one can always add the audio stream later and it will work. Video and audio, in this case, represent capabilities for a single service.

The reach information represents a weak form of contract; the presentity tells the watcher that, if the watcher utilizes the reach information included in the presence document, the watcher might be connected to a service described by the characteristics included in

the presence document. It is important to stress that this is not a guarantee in any way. It cannot be a guarantee for two reasons. Firstly, the service in the document might actually be modelling a number of actual services used by the user, and it may not be possible to connect the watcher to a service with all of the characteristics described in the presence document. Secondly, the preferences of the presentity always take precedence. The caller might ask to be connected to the video service, but it is permissible to connect them to a different service if that is the wish of the presentity.

This loose contract also provides some guidance on the type of URI that are most ideally suited for the service URI. A URN [10] can be used as the service URI. However, since a URI could be resolved to potentially any number of different URI, the characteristics, status, and relative information need to be sensible for all of the URI which can be resolved from the URN. As the URN becomes increasingly "vague" in terms of the service it identifies, the amount of presence attributes that can be included become increasingly small.





The tel URI [7] shares similar properties with a URN, and the same considerations apply. If, for example, the telephone number exists in enum [11] and multiple enum services are defined, including voice and messaging, it is likely that very little characteristic information can be included in that service. If, however, a tel URI with the enum dip indicator is present [12], or there is no enum record for that number, it means that the number corresponds to a telephone on the PSTN, and more can be said about its characteristics, status, and relative priority.

It is important to point out that there can be a many to one mapping of reach information to a service. That is, a particular service can be reachable by potentially an infinite variety of reach information. This is true even if the reach information is just the service URI; it is permissible for multiple service URI to reach the same service. Within any particular document there will be a single service URI. However, it is allowed and even valuable to provide different service URIs to different watchers, or to change the service URI provided to a particular watcher over time. Doing so affords many benefits, in fact. It can allow the recipient of a communications attempt to determine the context for that attempt - that the attempt was made as a result of trying to reach a particular service in a particular

presence document. This can be used as a technique for preventing communications spam, for example [13].

In an ideal system, the URI alone would represent sufficient reach information for each service. A URI is supposed to provide sufficient context for reaching the resource associated with the URI, and thus in theory there is no need for additional context. However, sometimes, additional information is needed. Since the reach information has to be understood in order for the service to be utilized, reach information beyond the URI should be defined and used sparingly.

### 3.3.3 Relative Information

Each service is also associated with a priority, which represents the preference that the user has for usage of one service over another. This does not mean that, when a watcher wishes to communicate with the presentity, that they should always use the service with the highest priority. If that were the case, there would be no point in including multiple services in the presence document. Rather, the priority says, "If you, the watcher, cannot decide which of these to use, or if it is not important to you, this is the order in which I would like you to contact me. However, I am giving you a choice." The priorities are relative to each other, and have no meaning as absolute numbers. If there are two services, and they have priorities of 1 and .5 respectively, this is identical to giving them





priorities of .2 and .1 respectively.

### 3.3.4 Status

Each service also has a status. Status represents dynamic information about the availability of communications using that service. This is in constrast to characteristics, which describe fairly static properties of the various services. The simplest form of status is the basic status, which is a binary indicator of availability for communications using that service. It can have values of either closed or open. Closed means that communication to the service will, in all likelihood, fail, or will not reach the intended party, or will not result in communications as described by the characteristics of the service. As an example, if a call is forwarded to voicemail if the user is busy or unavailable, the

service is marked as closed. Similarly, a presentity may include a hotel phone number as a service URI. After check-out, the phone number will still ring, but reach the chambermaid or the next guest. Thus, it would be declared "closed" by that presentity. As another example, if a user has a SIP URI as their service URI, pointing to a SIP softphone application, and the PC shuts down, calls to that SIP URI will return a 480 response code. This service would also be declared "closed". Open implies the opposite - the communications to this service will likely succeed and reach the desired target.

It is also possible to have status information that is dependent on the characteristics of the communications session that eventually get set up. For example, a status attribute can be defined that indicates that a softphone service is available if instant messaging is used, but unavailable if audio is used.

Other status information might indicate more details on why the service is available or unavailable. For example, a telephony service might have additional status to indicate that the user is on the phone, or that the user is handling 3 calls for that service.

Services inherently have a lot of dynamic state associated with them. For example, consider a wireless telephony service (i.e., a cell phone). There are many dynamic statuses of this service - whether or not the phone is registered, whether or not its roaming, which provider it has roamed into, its signal strength, how many calls it has, what the state of those calls are, how long the user has been in a call, and so on. As another example, consider an IM service. The statuses in this service include whether or not the user is registered, how long they have been registered, whether they have an IM conversation in progress, how many IM conversations are in progress, whether the user is typing, to whom they are typing, and so on.





However, not all of this dynamic state is appropriate to include within a service data component of a presence document. Information is included only when it has a bearing on helping the watcher decide whether or not to initiate communications with that service, or helping them decide when to initiate it, if not now. As an example, whether a cell phone has roamed or not does not pass this litmus test. Knowing this is not likely to have an impact on a decision to use this service.

## 3.4 Device

Devices model the physical operating environment in which services execute. Examples of devices include cell phones, PCs, laptops, PDAs, consumer telephones, enterprise PBX extensions, and operator dispatch consoles.

The mapping of services to devices are many to many. A single service can execute in multiple devices. Consider a SIP telephony service. Two SIP phones can register against a single address-of-record for this service. As a result, the SIP service is associated with two devices. Similarly, a single device can support a multiplicity of services. A cell phone can support a SIP telephony service, an SMS service, and an MMS service. Similarly, a PC can support a SIP telephony service and a SIP videophone service.

Devices are identified with a device ID. A device ID is a URI that is a globally and temporally unique identifier for the device. In particular, a device ID is a URN. The URN has to be unique across all other devices for a particular presentity. However, it is also highly desirable that it be persistent across time, globally unique, and computable in a fashion so that different systems are likely to refer to the device using the same ID. With these properties, differing sources of presence information based on device status can be combined together.

Unfortunately, due to the variety of different devices in existence, it is difficult for a single URN scheme to be used that have these properties. It is anticipated that multiple schemes will be defined, with different ones appropriate for different types of devices. For cellular telephones, the Electronic Serial Number (ESN), for example, is a good identifier. For IP devices, the MAC address is another good one. Unfortunately, neither of these are associated with URN schemes at this time. In the interim, the UUID URN [14] can be used. It has the properties of being globally and temporally unique, but because of its random component, it is not useful as a correlation ID with other presence sources on a network.

Though each device is identified by a unique device ID, there can be





multiple instances of a particular device represented in a document.

Each one will share the same device ID, but differ in its instance identifier. Multiple instances of a device exist in a document when the state of the device is ambiguous, as discussed in Section 3.5.

Though this document does not mandate a particular implementation approach, the device ID is most useful when all of the services on the device have a way to obtain the device ID, and get the same value for it. This would argue for its placement as an operating system feature, and operating system developers interested in implementing this specification are encouraged to provide APIs that allow applications to obtain the device ID. Absent such APIs, applications which report presence information about their devices will have to generate their own device IDs. This leads to the possibility that the applications may choose different device IDs, using different algorithms or data. In the worst case, these may mean that two services which run on the same device, do not appear to.

Like services and person data components, device data components have static characteristics and dynamic status. Characteristics of a device include its physical dimensions and capabilities - the size of its display, the speed of its CPU, and the amount of memory. Status information includes dynamic information about the device. This includes whether or not the device is powered on or off, the amount of battery power that remains in the device, the geographic location of the device, and so on.

The characteristics and status information reported about a device are for the purposes of choice - to allow the user to choose the service based on knowledge of what the device is. The device characteristics and status cannot, in any reliable way, be used to extract information about the nature of the service that will be received on the device. For example, if the device characteristics include the speed of the CPU, and the speed is sufficient to support high quality video compression, this cannot be interpreted to mean that video quality would be good for a video service on that device. Other constraints on the system may reduce the amount of CPU available to that service. If there is a desire to indicate that higher quality video is available on a device, that should be done by including service characteristics that say just that. The speed of the CPU might be useful in helping the watcher differentiate between a device that is a PC and one that is a cell phone, in the case where the watcher wishes to call the user's cell phone.

Similarly, if there is dynamic device status (such as whether the device is on or off), and this state impacts the state of the service, this is represented by adjusting the state of the service. Unless a consumer of a presence document has apriori knowledge

indicating otherwise (note that presence agents often do), the state of a device has no bearing on the state of the service.

Just like services, there is no enumeration of device types - PCs, PDAs, cell phones, etc. Rather, the device is defined by its characteristics, from which a watcher can extrapolate whether the device is a PDA, cell phone, or what have you.

It is important to point out that the device is a *model* if the underlying physical systems in which services execute. There is nothing that says that this model cannot be used to talk about systems where services run in virtualized systems, rather than real ones. For example, if a PC is executing a virtual machine, and running services within that virtual machine, it is perfectly acceptable to use this model to talk about that PC as being composed of two separate devices.

## 3.5 Modeling Ambiguity

Ambiguity is a reality of an presence system, and it is explicitly modeled by this specification. Ambiguity exists when there are multiple pieces of information about a person, a particular device, or a particular service. This ambiguity naturally arises when multiple elements publish information about the person, a particular service, or a particular device. In some cases, a compositor can resolve the ambiguity in an automated way, and combine together the data about the person, device or service into a single coherent description. In other cases, it cannot, perhaps because the compositor lacks the ability to do so.

However, in many cases, the resolution of this ambiguity is best left to the watcher that consumes the document. This consumer could be an application with more information than the compositor, and thus be able to do a better job of resolving the ambiguity. Or, it may be presented to the human user, and the human can often resolve the ambiguity. Unsurprisingly, a human can often do this far better than an automata can.

To model ambiguity, the model allows each service, each device, or the person component to contain multiple instances. Each instance has a unique identifier, called the instance identifier This identifier is unique across all other identifiers for any service, device, or person. That is, its uniqueness is scoped within all of the services, devices and person elements for a particular presentity. The identifier ideally persists over time, since it serves as a valuable handle for setting composition and authorization

policies. Even if there is a single instance for a particular device, service or person, the instance has an instance identifier.





When multiple instances exist in a document, it is important that some of the attributes of the device, service or person help the recipient resolve the ambiguity. For humans, the note field and timestamp serve as valuable tools. For an automata, nearly any attribute of the device, service or person can be used to resolve the ambiguity. The timestamp in particular is very useful for both humans and automata. As described in RFC 3863 [2], the timestamp provides the time of most recent change for the tuple. This specification defines the timestamp for person and device components as well, with the same meaning. Absent other information, the person, device, or service that most recently changed can be used as the more reliable source of data. However, such a resolution algorithm is not normatively required in any way.

## 3.6 Presence Document Properties

The overall presence document has several important properties that are essential to this model.

Firstly, a presence document has a concrete meaning independent of how it is transported, or where it is found. The semantics of a document are the same regardless of whether a document is published by a presence user agent to its compositor, or whether it is distributed from a presence agent to watchers. There are no required or implied behaviors for a recipient of a document. Rather, there are well defined semantics for the document itself, and a recipient of a document can take whatever actions it chooses based on those semantics.

A corollary of this property is that presence systems are infinitely composeable. A presence user agent can publish a document to its presence server. That presence server can compose it with other documents, and place the result in a notification to a watcher. That watcher can actually be another presence agent, combining that document with others it has received, and placing those results in yet another notify.

Yet another corollary of this property is that implied behaviors in reaction to the document cannot ever be assumed. For example, just

because a service indicates that it supports audio does not mean that a watcher will offer audio in a communications attempt to that service. If doing so is necessary to reach the service, this must be indicated explicitly through reach information.

It is also important to understand that the role of the presence document is to help a user make a choice amongst a set of services, and furthermore, to know ahead of time with as much certainty as possible whether a communications attempt will succeed or fail.





Success is a combination of many factors - does the watcher understand the service URI? Can it act on all of the reach information? Does it support a subset of the capabilities associated with the service? Does the person information indicate that the user is likely to answer? All of these checks should ideally be made before attempting communication.

Because the presence document serves to help a user to choose and establish communications, the presentity URI - as the index to that document - represents a form of "one-number" communications. Starting from this URI, all of the communications modalities and their URI for a user can be discovered, and then used to invoke a particular communications service. Rather than having to give out a separate phone number, email address, IM address, VoIP address, and so on, the presentity URI can be provided, and all of the others can be learned from there.

## 4. Motivation for the Model

Presence is defined in [15] as the ability, willingness or desire to communicate across a set of devices. The core of this definition is the conveyance of information about the ability, willingness or desire for communications. Thus, the presence data model needs to be tailored around conveying information that achieves this goal.

The person data component is targeted at conveying willingness and desire for communications. It is used to represent information about the user themselves that affects willingness and desire to communicate. Whether or not I am in a meeting, whether or not I am on the phone - each of these says something about my willingness to communicate, and thus makes sense for inclusion in a presence document.

The service component of the data model aims to convey information on the ability to communicate. The ability to communicate is defined by the services by which a user is reachable. Thus, including them is essential.

How do devices fit in? For many users, devices represent the ability to communicate, not services. Frequently, users make statements like, "call me on my cell phone" or, "I'm at my desk". These are statements for preference for communications using a specific device, as opposed to a service. Thus, it is our expectation that users will want to represent devices as part of the presence data.

Furthermore, the concept of device adds the ability to correlate services together. The device models the underlying platform that supports all of the services on the phone. Its state therefore





impacts all services. For example, if a presence server can determine that a cell phone is off, this says something about the services that run on that device - they are all not available. Thus, if services include indicators about the devices on which they run, device state can be obtained and thus used to compute the state of the services on the device.

The data model tries hard to separate device, service, and person as different concepts. Part of this differentiation is that many attributes will be applicable to some of these, but not others. For example, geographic location is a meaningful attribute of the person (the user has a location) and of a device (the device has a location), but not of a service (services don't inherently have locations). Based on this, geographic location information should only appear as part of device or person, never service. Furthermore, it is possible and meaningful for location information to be conveyed for both device and person, and for these locations to be different. The fact that the presence system might try to determine the location of the person by extrapolation from the location of one of the devices is irrelevant from a data modeling perspective. Person location and device location are not the same thing.

## 5. Encoding

Information represented according to the data model described above

needs to be mapped into an on-the-wire format for transport and storage. The Presence Information Document Format [2] is used for representation of presence data.

The <presence> element contains the presence information for the presentity. The "entity" attribute of this element contains the presentity URI.

The existing <tuple> element in the PIDF document is used to represent the service. This is consistent with the original intent of RFC 2778 and RFC 3863, and achieves backwards compatibility with implementations developed before the model described here was complete. The <contact> element in the <tuple> element is used to encode the service URI. Presence attributes representing dynamic status appear as children to the <status> element, and attributes representing static characteristics appear directly as children of <tuple>. It is not critical that a clean separation between dynamic and static information be made. It is only important that each presence attribute be specified to appear in either <status> or <tuple>.

The "id" attribute of the <tuple> element conveys the service instance. Each <tuple> element with the same <contact> URI





represents a different instance of a particular service.

This specification introduces the <person> element, which can appear as a child to <presence>. There can be zero or more instances of this element per document. Each one has a mandatory "id" attribute which contains the instance identifier for the person. Each <person> element contains a <status> element, which contains any number of elements containing dynamic status information. This is followed by any number of elements that indicate characteristic information. This is followed by an optional <note> and optional <timestamp>.

RFC 3863 defines a <note> element which can be present as a child to <presence>. As it relates to the model defined here, this note element, if present in a document, applies to all person instances which do not have their own <note> element. In other words, if a <person> element has its own <note>, that is the <note> for that <person> element. If a <person> element does not have its own <note> element, the <note> element that is the direct child of <presence> is

the <note> for that <person>. If there is no <note> element underneath the <person> element, and no <note> element that is a direct child of <presence>, then that <person> element has no <note>.

This specification also introduces the <device> element, which can appear as a child to <presence>. There can be zero or more instances of this element per document. Each one has a mandatory "id" attribute which contains the instance identifier for the device. Like <person>, <device> contains the <status> element, which contains any number of elements containing dynamic status information, followed by any number of elements containing characteristic information. This is followed by <device-id&gt, which contains the URN for the device ID for this device. This is followed by an optional <note> and optional <timestamp>.

A client that receives a PIDF document containing the <device> and <person> elements will ignore them. Furthermore, since the semantics of service as defined here are aligned with the meaning of a tuple as defined in RFC 2778 and RFC 3863, documents incorporating the concepts defined in this model are compliant with older implementations.

It's important to note that the mapping of the presence data model into a PIDF document is merely an exercise in syntax.

## 5.1 XML Schema

The XML schemas are broken into a common schema, called common.xsd, which contains common type definitions. This schema is included in the definitions of the person and device elements.





### 5.1.1 Common

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:import namespace="http://www.w3.org/XML/1998/namespace"
  schemaLocation="http://www.w3.org/2001/xml.xsd"/>
 <xs:simpleType name="Timestamp_t">
  <xs:annotation>
```

```
      <xs:documentation>Timestamp type</xs:documentation>
     </xs:annotation>
     <xs:restriction base="xs:dateTime"/>
    </xs:simpleType>
    <xs:complexType name="Note_t">
     <xs:annotation>
      <xs:documentation>Note type</xs:documentation>
     </xs:annotation>
     <xs:simpleContent>
      <xs:extension base="xs:string">
       <xs:attribute ref="xml:lang"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:schema>
```

### 5.1.2 Person

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema targetNamespace="urn:ietf:params:xml:ns:pidf:person"
    xmlns="urn:ietf:params:xml:ns:pidf:person"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="common.xsd"/>
    <xs:complexType name="personCharacteristicAbstractType" abstract="true">
     <xs:annotation>
      <xs:documentation>Abstract type for characteristics for
      person</xs:documentation>
     </xs:annotation>
    </xs:complexType>
    <xs:complexType name="personStatusAbstractType" abstract="true">
     <xs:annotation>
      <xs:documentation>Abstract type for
      status for person</xs:documentation>
     </xs:annotation>
```





```
    </xs:complexType>
    <xs:element name="personCharacteristic"
     type="personCharacteristicAbstractType">
     <xs:annotation>
```

```
      <xs:documentation>Person characteristic
       element (abstract)</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="personStatus" type="personStatusAbstractType">
     <xs:annotation>
      <xs:documentation>Person status element (abstract)</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="person">
     <xs:annotation>
      <xs:documentation>Contains information about
       the human user</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="status">
        <xs:complexType>
         <xs:sequence>
          <xs:element ref="personStatus" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element ref="personCharacteristic"
        minOccurs="0" maxOccurs="unbounded"/>
       <xs:element name="note" type="Note_t" minOccurs="0"/>
       <xs:element name="timestamp" type="Timestamp_t" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:schema>
```

### 5.1.3 Device

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema targetNamespace="urn:ietf:params:xml:ns:pidf:device"
    xmlns="urn:ietf:params:xml:ns:pidf:device"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="common.xsd"/>
```

```
    <xs:complexType name="deviceCharacteristicAbstractType" abstract="true">
     <xs:annotation>
      <xs:documentation>Abstract type for
       characteristics for device</xs:documentation>
     </xs:annotation>
    </xs:complexType>
    <xs:complexType name="deviceStatusAbstractType" abstract="true">
     <xs:annotation>
      <xs:documentation>Abstract type for status for device</xs:documentation>
     </xs:annotation>
    </xs:complexType>
    <xs:element name="deviceCharacteristic"
     type="deviceCharacteristicAbstractType">
     <xs:annotation>
      <xs:documentation>Device characteristic
       element (abstract)</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="deviceID" type="xs:anyURI">
     <xs:annotation>
      <xs:documentation>Device ID, a URN</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="deviceStatus" type="deviceStatusAbstractType">
     <xs:annotation>
      <xs:documentation>Device status element (abstract)</xs:documentation>
     </xs:annotation>
    </xs:element>
    <xs:element name="device">
     <xs:annotation>
      <xs:documentation>Contains information
       about the device</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:sequence>
       <xs:element name="status">
        <xs:complexType>
         <xs:sequence>
          <xs:element ref="deviceStatus" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element ref="deviceCharacteristic"
        minOccurs="0" maxOccurs="unbounded"/>
       <xs:element ref="deviceID"/>
       <xs:element name="note" type="Note_t" minOccurs="0"/>
       <xs:element name="timestamp" type="Timestamp_t" minOccurs="0"/>
```

```
      </xs:sequence>
```





```
      <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
   </xs:element>
  </xs:schema>
```

## 6. Extending the Presence Model

When new presence attributes are added, any such extension has to consider the following questions:

1. Is the new attribute applicable to person, service or device data components? If it is applicable to more than one, what is its meaning in each context? An extension should strive to have each attribute concisely defined for each area of applicability, so that a source can clearly determine to which type of data component it should be applied.

2. Is this new attribute a dynamic status, or a static characteristic? Characteristics are information that describe information about devices, services or a person that help provide context for a consumer of the document to make a decision about whether communications is desired in one place or another. They are therefore descriptive in nature.

## 7. Example Presence Documents

In this section, we give examples of different physical systems, present the model of that system using the concepts described here, and then show the resulting presence document. These examples make use of presence attributes defined in [17] and [18].

### 7.1 Basic IM Client

In this scenario, a provider is offering a service very similar to the instant messaging services offered today by the public providers like AOL, Yahoo, and MSN. In this service, each user has a "screen name" that identifies them in the service. A single client,

generally a PC application, connects to the service at a time. When the client connects, this fact is made available to other watchers of that user in the system. The user has the ability to set a textual note that describes what they are doing, and this note is seen by the watchers in the system. The user can set one of several status messages - such as busy, in a meeting, etc., which are pre-defined notes that the system understands. If a user does not type anything on their keyboard for some time, their status changes to idle on the





screens of the various watchers of the system. The system also indicates the amount of time that the user has been idle.

Whenever a user is connected to the system, they are capable of receiving instant messages. A user can set their status to "invisible", which means that they appear as offline to other users. However, if an IM is sent to them, it will still be delivered.

This system is modeled by representing each presentity in the system with three data components - a person component, a service component, and a device component. The person component describes the state of the user, including the note and the pre-defined status messages. These represent information about the human user, so they are included in the person component. The service tuple represents the IM service. No characteristics are included. The service URI published by the client is set to the client's Globally Routable User Agent URI (GRUU) [16]. The device component is used to model the PC. The device component includes the <user-input> element [17], since the idleness refers to usage of the device, not the service.

The document published by the client would look like this:

```
    <?xml version="1.0" encoding="UTF-8"?>
        <presence xmlns="urn:ietf:params:xml:ns:pidf"
            entity="sip:someone@example.com">
          <tuple id="sg89ae">
            <status>
              <basic>open</basic>
            </status>
            <device-id>mac:8asd7d7d70</device-id>
            <servcaps>
              <methods>
                <method>MESSAGE</method>
                <method>OPTIONS</method>
              </methods>
            </servcaps>
            <contact>sip:gruu-someone-1@example.com</contact>
          </tuple>
          <person>
            <status>
              <activities>
               <activity>on-the-phone</activity>
              </activities>
            </status>
          </person>
          <device device-id="mac:8asd7d7d70">
            <status>
              <idle/>
```

```
          </status>
        </device>
      </presence>
```

It is worth commenting further on the value of having a separate device element just to convey the idle indicator. As described above, the idle indication of interest is really an indicator that the device is idle. By making that explicit, the idle indicator can be used by the presence server to affect the state of other services running on the same device. For example, let say there is a voip application running on the same device. This application reports its presence information using the example below. Since it reports that it runs on the same device, the presence server can use the status of the service to further refine the idle indicator of the device. Specifically, if the user is using their voip application, the presence server knows that the device is in use, even if the IM application reports that the device is idle. Typically, idleness is determined by lack of keyboard or mouse input, neither of which might be used during a voip call.

In a more simplistic case, reporting the idle indicator as part of





the device status allows that indicator to be used for other services on the same device. Taking, again, the example of the voip application on the same device, if the voip application does not report any device information, and a watcher is not provided information on the IM service, the presence document sent to the watcher can include the device status. Because of the usage of the device IDs and the device information, the presence server can correlate the device status as reported by the IM application with the voip service, and use them together.

## 7.2 VoIP Application

In this example, consider a SIP network. The user has a SIP AOR of sip:user@example.com. The user has a single SIP PC client that they run on their office machine. This is a simple SIP softphone, supporting audio only.

The PC client publishes a presence document that has a single tuple, representing the service. It does not include any presentity or

device elements, although it does include a device-id as part of its service characteristics.

The document published by the client would look like this:

```
 <?xml version="1.0" encoding="UTF-8"?>
     <presence xmlns="urn:ietf:params:xml:ns:pidf"
         entity="sip:user@example.com">
       <tuple id="sg89ae">
         <status>
           <basic>open</basic>
         </status>
         <device-id>urn:mac:8asd7d7d70</device-id>
         <servcaps>
           <methods>
             <method>INVITE</method>
             <method>OPTIONS</method>
             <method>BYE</method>
             <method>ACK</method>
             <method>CANCEL</method>
           </methods>
           <audio>true</audio>
         </servcaps>
         <contact>sip:gruu2@example.com</contact>
       </tuple>
     </presence>
```





## 7.3 Cellphone

In this example, the user has a cellphone. This cellphone has an SMS client and a SIP push-to-talk client running. The phone also has a switch that allows the user to select "silent mode". This information is also used by the phone as an indicator of business. As it turns out, the SMS and PTT applications on the phone are totally separate, and each publishes its own information. Indeed, both happen to publish information about the silent mode switch.

The SMS application models itself with three components - a service component, representing the actual SMS service, a device component,

modeling the phone, and a presentity component, modeling the user. Similarly, the PTT app has three components, representing the PTT service, device, and presentity.

If the user is in a PTT call, the PTT application might generate a document that looks like this. Note the inclusion of the busy element as part of the presentity state, which was set based on the silent switch:





```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:someone@example.com">
      <tuple id="sg89ae">
        <status>
```

```
            <basic>closed</basic>
          </status>
          <device-id>urn:esn:600b40c7</device-id>
          <servcaps>
            <methods>
              <method>INVITE</method>
              <method>OPTIONS</method>
              <method>BYE</method>
              <method>ACK</method>
              <method>CANCEL</method>
            </methods>
            <audio>true</audio>
            <duplex>half</duplex>
          </servcaps>
          <contact>sip:gruu-aa@example.com</contact>
        </tuple>
        <person>
          <status>
            <activities>
             <activity>on-the-phone</activity>
             <activity>busy</activity>
            </activities>
          </status>
        </person>
        <device device-id="urn:esn:600b40c7">
          <servcaps>
            <mobility>mobile</mobility>
          </servcaps>
        </device>
      </presence>
```

The SMS application would publish a document that looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:someone@example.com">
      <tuple id="sg89ae">
        <status>
          <basic>open</basic>
        </status>
        <device-id>urn:esn:600b40c7</device-id>
        <contact>sms:1234567</contact>
      </tuple>
      <person>
        <status>
          <activities>
           <activity>busy</activity>
           <activity>on-the-phone</activity>
          </activities>
        </status>
      </person>
      <device device-id="urn:esn:600b40c7"/>
    </presence>
```

The presence server now has two presence documents for a single user. It has conflicting information for the person and device components. It knows it has two services, both of which run on the same device. It merges the two devices into one, and unions the information it has for them. It keeps the services as separate, but changes the PTT URI from the GRUU to the AOR. It merges the person information together, unioning that as well. The resulting raw presence document, after composition, would look like:

```
<?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
        entity="sip:someone@example.com">
      <tuple id="sg89ae">
        <status>
          <basic>open</basic>
        </status>
        <device-id>urn:esn:600b40c7</device-id>
        <contact>sms:1234567</contact>
      </tuple>
      <tuple id="sg89ae">
        <status>
          <basic>closed</basic>
        </status>
        <device-id>urn:esn:600b40c7</device-id>
```

```
            <servcaps>
              <methods>
                <method>INVITE</method>
                <method>OPTIONS</method>
                <method>BYE</method>
                <method>ACK</method>
                <method>CANCEL</method>
              </methods>
              <audio>true</audio>
              <duplex>half</duplex>
            </servcaps>
            <contact>sip:someone@example.com</contact>
          </tuple>
          <person>
            <status>
              <activities>
               <activity>busy</activity>
              </activities>
            </status>
          </person>
          <device device-id="urn:esn:600b40c7">
            <servcaps>
              <mobility>mobile</mobility>
            </servcaps>
          </device>
        </presence>
```

## 8. Security Considerations

The presence information described by the model defined here is very sensitive. It is for this reason that privacy filtering plays a key role in the processing of presence data, as described above. Presence systems based on this model need to provide such a privacy capability, and furthermore, need to protect the integrity and confidentiality of the data.

## 9. Acknowledgements

This document is really a distillation of many ideas discussed over a long period of time. These ideas were contributed by many different

participants in the SIMPLE working group. Aki Niemi, Paul Kyzivat, Cullen Jennings, Ben Campbell, Robert Sparks, Dean Willis, Adam Roach, Hisham Khartabil, and Jon Peterson contributed many of the concepts that are described here. Example presence documents came from Robert Sparks' example presence documents specification, and ideas on defining services through characteristics, rather than





enumeration, come from Adam Roach's service features draft. A special thanks to Steve Donovan for discussions on the topics discussed here.

## 10 Informative References

[1] Day, M., Rosenberg, J. and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.

[2] Sugano, H., Fujimoto, S., Klyne, G., Bateman, A., Carr, W. and J. Peterson, "Presence Information Data Format (PIDF)", RFC 3863, August 2004.

[3] 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.

[4] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.

[5] Saint-Andre, P., "A Uniform Resource Identifier (URI) Scheme for the Extensible Messaging and Presence Protocol (XMPP)", draft-saintandre-xmpp-uri-08 (work in progress), December 2004.

[6] Wilde, E. and A. Vaha-Sipila, "URI scheme for GSM Short Message Service", draft-wilde-sms-uri-08 (work in progress), January 2005.

[7] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

[8] Klyne, G., "A Syntax for Describing Media Feature Sets", RFC 2533, March 1999.

[9] Rosenberg, J., Schulzrinne, H. and P. Kyzivat, "Caller

Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[10] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[11] Faltstrom, P. and M. Mealling, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)", RFC 3761, April 2004.

[12] Stastny, R. and L. Conroy, "The ENUM Dip Indicator parameter for the "tel" URI", draft-ietf-iptel-tel-enumdi-00 (work in progress), February 2005.





[13] Rosenberg, J., "The Session Initiation Protocol (SIP) and Spam", draft-ietf-sipping-spam-00 (work in progress), February 2005.

[14] Mealling, M., "A UUID URN Namespace", draft-mealling-uuid-urn-05 (work in progress), January 2005.

[15] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[16] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-02 (work in progress), July 2004.

[17] Schulzrinne, H., Gurbani, V., Kyzivat, P. and J. Rosenberg, "RPID: Rich Presence: Extensions to the Presence Information Data Format (PIDF)", draft-ietf-simple-rpid-04 (work in progress), October 2004.

[18] Lonnfors, M. and K. Kiss, "User Agent Capability Extension to Presence Information Data Format(PIDF)", draft-ietf-simple-prescaps-ext-02 (work in progress), October 2004.

## Author's Address

Jonathan Rosenberg

Cisco Systems
600 Lanidex Plaza
Parsippany, NJ  07054
US

Phone: +1 973 952-5000
EMail: jdrosen@cisco.com
URI:   http://www.jdrosen.net





## Intellectual Property Statement

this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

## Disclaimer of Validity



## Copyright Statement



## Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.