A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)
draft-ietf-sipping-kpml-07

Status of this Memo

Copyright Notice

Abstract

This document describes a SIP Event Package "kpml" that enables
monitoring of DTMF signals and uses XML documents referred to as Key
Press Markup Language (KPML). The kpml Event Package may be used to
support applications consistent with the principles defined in the
document titled "A Framework for Application Interaction in the

Session Initiation Protocol (SIP)". The event package uses SUBSCRIBE
messages and allows for XML documents that define and describe filter
specifications for capturing key presses (DTMF Tones) entered at a
presentation-free User Interface SIP User Agent (UA). The event
package uses NOTIFY messages and allows for XML documents to report
the captured key presses (DTMF tones), consistent with the filter
specifications, to an Application Server. The scope of this package
is for collecting supplemental key presses or mid-call key presses
(triggers).

Conventions used in this document

RFC2119 [1] provides the interpretations for the key words "MUST",
"MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT",
"RECOMMENDED", "MAY", and "OPTIONAL" found in this document.

The Application Interaction Framework document [19] provides the
interpretations for the terms "User Device", "SIP Application", and
"User Input". This document uses the term "Application" and
"Requesting Application" interchangeably with "SIP Application".

Additionally, the Application Interaction Framework document
discusses User Device Proxies. A common instantiation of a User
Device Proxy is a Public Switched Telephone Network (PSTN) gateway.
Because the normative behavior of a presentation free User Interface
is identical for a presentation free SIP User Agent and a
presentation free User Device Proxy, this document uses "User Device"
for both cases.

Table of Contents

1. Introduction

This document describes a SIP Event Package "kpml" that enables
monitoring of DTMF signals and utilizes XML documents referred to as
Key Press Markup Language (KPML). KPML is a markup [18] that enables
presentation-free User Interfaces as described in the Application
Interaction Framework [19]. The Key Press Stimulus Package is a SIP
Event Notification Package [5] that uses the SUBSCRIBE and NOTIFY
methods of SIP. The subscription filter and notification report
bodies use the Keypad Markup Language, KPML.

The "kpml" event package requires the definition of two new MIME
types, two new URN Sub-Namespaces, and two Schemas for the KPML
Request and the KPML Response. The scope of this package is for
collecting supplemental key presses or mid-call key presses
(triggers). This capability allows an Application Server service
provider to monitor (filter) for a set of DTMF patterns at a SIP User
Agent located either in an end user device or a gateway.

In particular, the "kpml" event package enables "dumb phones" and
"gateways" which receive signals from dumb phones to report user
key-press events. Colloquially, this mechanism provides for "digit
reporting" or "Dual-Tone Multi-Frequency (DTMF) reporting." The
capability eliminates the need for "hair-pinning" (routing media into
and then out of the same device) through a Media Server or
duplicating all the DTMF events, when an Application Server needs to
trigger mid-call service processing on DTMF digit patterns.

A goal of KPML is to fit in an extremely small memory and processing
footprint.

The name of the XML document, KPML, reflects its legacy support role.
The public switched telephony network (PSTN) accomplished signaling
by transporting DTMF tones in the bearer channel (in-band signaling)
from the user terminal to the local exchange.

Voice-over-IP networks transport in-band signals with actual DTMF
waveforms or RFC2833 [13] packets. In RFC2833, the signaling
application inserts RFC2833 named signal packets as well as, or
instead of, generating tones in the media path. The receiving
application receives the signal information in the media stream.

RFC2833 tones are ideal for conveying telephone-events point-to-point
in an RTP stream, as in the context of straightforward sessions like
a 2-party call or simple, centrally mixed conference. However, there
are other environments where additional or alternative requirements
are needed. These other environments include protocol translation
and complex call control.

An interested application could request notifications of every key
press. However, many of the use cases for such signaling show that
many applications are interested in only one or a few keystrokes.
Thus a mechanism is needed for specifying to the user's interface
what stimuli the application requires.

2. Protocol Overview

The "kpml" event package uses explicit subscription notification
requests using the SIP SUBSCRIBE and NOTIFY methods. An Application
that wants to collect digits, creates an application/kpml-request+xml
document with the digit patterns of interest to the Application, and
places this document in its SUBSCRIBE request. SIP SUBSCRIBE
messages are routed to the User Interface using standard SIP request
routing. KPML Subscriptions do not fork, since they are always sent
to a SIP URI that has GRUU [8] properties. The KPML request
contained in the SUBSCRIBE message identifies the target media stream
by referencing the dialog identifiers corresponding to the session
responsible for the media stream. Once a subscription is
established, the User Interface sends application/kpml-response+xml
documents in NOTIFY requests when digits are collected, or timeouts
or errors occur.

A KPML subscription can be persistent or one-shot. Persistent
requests are active until either the subscription terminates, the
Application replaces the request, or the Application deletes the
request by sending a null document on the dialog, or the Application
explicitly deletes the subscription by sending a SUBCRIBE with an
expires value of zero (0).

One-shot requests terminate the subscription upon the receipt of DTMF
values which provide a match. The "persist" KPML element specifies
whether the subscription remains active for the duration specified in
the SUBSCRIBE message or if it automatically terminates upon a
pattern match.

NOTIFY messages can contain XML documents. If the User Interface
matches a digitmap, the NOTIFY message (response) contains an XML
document that indicates the User Input detected and whether the User
Interface suppressed the representation of User Input, such as tones,
or RFC2833, from the media streams. If the User Interface
encountered an error condition, such as a timeout, this will also be
reported.

3. Key Concepts

3.1 Subscription Duration

   KPML recognizes two types of subscriptions: one-shot and persistent.
   Persistent subscriptions have two sub-types: continuous notify and
   single-notify.

   One-shot subscriptions terminate after a pattern match occurs and a
   report is issued in a NOTIFY message. If the User Interface detects
   a key press stimulus that triggers a one-shot KPML event, then the
   User Interface (notifier) MUST set the "Subscription-State" in the
   NOTIFY message to "terminated". At this point the User Interface
   MUST consider the subscription expired.

   Persistent subscriptions remain active at the User Interface, even
   after a match. For continuous notify persistent subscriptions, the
   User Interface will emit a NOTIFY message whenever the User Input
   matches a subscribed pattern. For single-notify persistent
   subscriptions, the user device will emit a NOTIFY message at the
   first match, but will not emit further NOTIFY messages until the
   Application issues a new subscription request on the subscription
   dialog.

      NOTE: The single-notify persistent subscription enables lock step
      (race-free) quarantining of User Input between different digit
      maps.

   The "persist" attribute to the <pattern> tag in the KPML subscription
   body affects the lifetime of the subscription.

   If the "persist" attribute is "one-shot", then once there is a match
   (or no match is possible), the subscription ends after the User
   Interface notifies the Application.

   If the "persist" attribute is "persist" or "single-notify", then the
   subscription ends when the Application explicitly ends it or the User
   Interface terminates the subscription.

   If the User Interface does not support persistent subscriptions, it
   returns a NOTIFY message with the KPML status code set to 531. If
   there are digits in the buffer and the digits match an expression in
   the SUBSCRIBE filter, the User Interface prepares the appropriate
   NOTIFY response message.

   Note the values of the "persist" attribute are case sensitive.

3.2 Timers

   To address the various key press collection scenarios, three timers

are defined. They are the extra, critical, and inter-digit timers.
o The inter-digit timer is the maximum time to wait between digits.
   Note: unlike MGCP [15] or H.248 [16], there is no start timer, as
   that concept does not apply in the KPML context.
o The critical timer is the time to wait for another digit if the
   collected digits can match more than one potential pattern.
o The extra timer is the time to wait for another digit if the
   collected digits can only match one potential pattern, but a
   longer match for this pattern is possible.

The User Interface MAY support an inter-digit timeout value. This is
the amount of time the User Interface will wait for User Input before
returning a timeout error result on a partially matched pattern. The
application can specify the inter-digit timeout as an integer number
of milliseconds by using the "interdigittimer" attribute to the
<pattern> tag. The default is 4000 milliseconds. If the User
Interface does not support the specification of an inter-digit
timeout, the User Interface MUST silently ignore the specification.
If the User Interface supports the specification of an inter-digit
timeout, but not to the granularity specified by the value presented,
the User Interface MUST round up the requested value to the closest
value it can support.

The purpose of the inter-digit timeout is to protect applications
from starting to match a pattern, yet never return a result. This
can occur, for example, if the user accidentally enters a key that
begins to match a pattern. However, since the user accidentally
entered the key, the rest of the pattern never comes. Moreover, when
the user does enter a pattern, since they have already entered a key,
the pattern may not match, or may not match as expected. Likewise,
consider the case where the user thinks they entered a key press, but
the User Interface does not detect the key. This could occur when
collecting ten digits, but the device actually only receives 9. In
this case, the User Interface will wait forever for the tenth key
press, while the user becomes frustrated wondering why the
application is not responding.

The User Interface MAY support a critical-digit timeout value. This
is the amount of time the User Interface will wait for another key
press when it already has a matched <regex> but there is another,
longer <regex> that may also match the pattern. The application can
specify the critical-digit timeout as an integer number of
milliseconds by using the "criticaldigittimer" attribute to the
<pattern> tag. The default is 1000 milliseconds.

The purpose of the critical-digit timeout is to allow the application
to collect longer matches than the shortest presented. This is
unlike MGCP [15], where the shortest match gets returned. For

example, if the application registers for the patterns "0011", "011",
"00", and "0", the critical-digit timeout enables the User Interface
to distinguish between "0", "00", "011", "0011". Without this
feature, the only value that the User Interface can detect is "0".

The User Interface MAY support an extra-digit timeout value. This is
the amount of time the User Interface will wait for another key press
when it already has matched the longest <regex>. The application can
specify the extra-digit timeout as an integer number of milliseconds
by using the "extradigittimer" attribute to the <pattern> tag. The
default is 500 milliseconds. If there is no enterkey specified, then
the User Interface MAY default the exteradigittimer to zero.

The purpose of the extra-digit timeout is to allow the User Interface
to collect the enterkey. Without this feature, the User Interface
would match the pattern, and the enterkey would be buffered and
returned as the next pattern.

3.3 Pattern Matches

During the subscription lifetime, the User Interface may detect a key
press stimulus that triggers a KPML event. In this case, the User
Interface (notifier) MUST return the appropriate KPML document.

The pattern matching logic works as follows. KPML User Interfaces
MUST follow the logic presented in this section so that different
implementations will perform deterministically on the same KPML
document given the same User Input.

A kpml request document contains a <pattern> element with a series of
<regex> tags. Each <regex> element specifies a potential pattern for
the User Interface to match. The Section 5.1 describes the DRegex,
or digit regular expression, language.

The pattern match algorithm matches the longest regular expression.
This is the same mode as H.248.1 [16] and not the mode presented by
MGCP [15]. The pattern match algorithm choice has an impact on
determining when a pattern matches. Consider the following KPML
document.

```
<?xml version="1.0" encoding="UTF-8"?>
<kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
      version="1.0">
  <pattern>
    <regex>0</regex>
    <regex>011</regex>
  </pattern>
</kpml-request>
```

                        Figure 1: Greedy Matching

In Figure 1, if we were to match on the first found pattern, the
string "011" would never match. This happens because the "0" rule
would match first.

While this behavior is what most applications desire, it does come at
a cost. Consider the following KPML document snippet.

```
  <regex>x{7}</regex>
  <regex>x{10}</regex>
```

                        Figure 2: Timeout Matching

Figure 2 shows a typical North American dial plan. From an
application perspective, users expect a seven-digit number to respond
quickly, not waiting the typical inter-digit critical timer (usually
four seconds). Conversely, the User does not want the system to cut
off their ten-digit number at seven digits because they did not enter
the number fast enough.

One approach to this problem is to have an explicit dial string
terminator. Often, it is the pound key (#). Now, consider the
following snippet.

```
<regex>x{7}#</regex>
<regex>x{10}#</regex>
```

                    Figure 3: Timeout Matching with Enter

The problem with the approach in Figure 3 is that the "#" will appear
in the returned dial string. Moreover, one often wants to allow the
user to enter the string without the dial string termination key. In
addition, using explicit matching on the key means one has to double
the number of patterns, e.g., "x{7}", "x{7}#", "x{10}", and "x{10}#".

The approach used in KPML is to have an explicit "Enter Key", as
shown in the following snippet.

```
<pattern enterkey="#">
  <regex>x{7}</regex>
  <regex>x{10}</regex>
</pattern>
```

Figure 4: Timeout Matching with Enter Key

In Figure 4, the enterkey attribute to the <pattern> tag specifies a
string that terminates a pattern. In this situation, if the user
enters seven digits followed by the "#" key, the pattern matches (or
fails) immediately. KPML indicates a terminated nomatch with a KPML
status code 402.
   NOTE: The enterkey is a string. The enterkey can be a sequence
   of key presses, such as "**".

Some patterns look for long duration key presses. For example, some
applications look for long "#" or long "*".

KPML uses the "L" modifier to <regex> characters to indicate long key
presses. The following KPML document looks for a long pound of at
least 3 seconds.

```
<?xml version="1.0" encoding="UTF-8"?>
<kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
      version="1.0">
  <pattern long="3000">
    <regex>L#</regex>
  </pattern>
</kpml-request>
```

The request can specify what constitutes "long" by setting the long
attribute to the <pattern>. This attribute is an integer
representing the number of milliseconds. If the user presses a key
for longer than "long" milliseconds, the Long modifier is true. The
default length of the long attribute is 2500 milliseconds.

User Interfaces MUST distinguish between long and short input when
the KPML document specifies both in a document. However, if there is
not a corresponding long key press pattern in a document, the User
Interface MUST match the key press pattern irrespective of the length
of time the user presses the key.

As an example, in the following snippet in Figure 6, the User
Interface discriminates between a long "*" and a normal "*", but any
length "#" will match the pattern.

```
<pattern>
  <regex tag="short_star">*</regex>
  <regex tag="long_star">L*</regex>
  <regex>#</regex>
</pattern>
```

Figure 6: Long and Short Matching

Some User Interfaces are unable to present long key presses. An
example is an old private branch exchange (PBX) phone set that emits
fixed-length tones when the user presses a key. To address this
issue, the User Interface MAY interpret a succession of presses of a
single key to be equivalent to a long key press of the same key. The
Application indicates it wants this behavior by setting the
"longrepeat" attribute to the <pattern> to "true".

The KPML document specifies if the patterns are to be persistent by
setting the "persist" attribute to the <pattern> tag to "persist" or
"single-notify". Any other value, including "one-shot", indicates
the request is a one-shot subscription. If the User Interface does
not support persistent subscriptions, it returns a KPML document with
the KPML status code set to 531. If there are digits in the buffer
and the digits match an expression in the KPML document, the User
Interface emits the appropriate kpml notification.

Note the values of the "persist" attribute are case sensitive.

Some User Interfaces may support multiple regular expressions in a
given pattern request. In this situation, the application may wish
to know which pattern triggered the event.

KPML provides a "tag" attribute to the <regex> tag. The "tag" is an
opaque string that the User Interface sends back in the notification
report upon a match in the digit map. In the case of multiple
matches, the User Interface MUST chose the longest match in the KPML
document. If multiple matches match the same length, the User
Interface MUST chose the first expression listed in the subscription
KPML document based on KPML document order.

If the User Interface cannot support multiple regular expressions in
a pattern request, the User Interface MUST return a KPML document
with the KPML status code set to 532. If the User Interface cannot
support the number of regular expressions in the pattern request, the
User Interface MUST return a KPML document with the KPML status code

set to 534.

> NOTE: We could mandate a minimum number of regular expressions a
> User Interface must support per subscription request and globally.
> However, such minimums tend to become designed-in, hard-coded
> limits. For guidance, one should be able to easily handle tens of
> expressions per subscription and thousands globally. A good
> implementation should have effectively no limits. That said, to
> counter possible denial of service attacks, implementers of User
> Interfaces should be aware of the 534 and 501 status codes, and
> feel free to use them.

3.4 Digit Suppression

Under basic operation, a KPML User Interface will transmit in-band
tones (RFC2833 [13] or actual tone) in parallel with User Input
reporting.

> NOTE: If KPML did not have this behavior, then a User Interface
> executing KPML could easily break called applications. For
> example, take a personal assistant that uses "*9" for attention.
> If the user presses the "*" key, KPML will hold the digit, looking
> for the "9". What if the user just enters a "*" key, possibly
> because they accessed an IVR system that looks for "*"? In this
> case, the "*" would get held by the User Interface, because it is
> looking for the "*9" pattern. The user would probably press the
> "*" key again, hoping that the called IVR system just did not hear
> the key press. At that point, the User Interface would send both
> "*" entries, as "**" does not match "*9". However, that would not
> have the effect the user intended when they pressed "*".

On the other hand, there are situations where passing through tones
in-band is not desirable. Such situations include call centers that
use in-band tone spills to initiate a transfer.

For those situations, KPML adds a suppression tag, "pre", to the
<regex> tag. There MUST NOT be more than one <pre> tag in any given
<regex> tag.

If there is only a single <pattern> and a single <regex>, suppression
processing is straightforward. The end-point passes User Input until
the stream matches the regular expression <pre>. At that point, the
User Interface will continue collecting User Input, but will suppress
the generation or pass-through of any in-band User Input.

If the User Interface suppressed stimulus, it MUST indicate this by
including the attribute "suppressed" with a value of "true" in the
notification.

Clearly, if the User Interface is processing the KPML document
against buffered User Input, it is too late to suppress the
transmission of the User Input, as the User Interface has long sent
the stimulus. This is a situation where there is a <pre>
specification, but the "suppressed" attribute will not be "true" in
the notification. If there is a <pre> tag that the User Interface
matched and the User Interface is unable to suppress the User Input,
it MUST set the "suppressed" attribute to "false".

A KPML User Interface MAY perform suppression. If it is not capable
of suppression, it ignores the suppression attribute. It MUST set
the "suppressed" attribute to "false". In this case, the pattern to
match is the concatenated pattern of pre+value.

At some point in time, the User Interface will collect enough User
Input to the point it matches a <pre> pattern. The interdigittimer
attribute indicates how long to wait once the user enters stimulus
before reporting a time-out error. If the interdigittimer expires,
the User Interface MUST issue a time-out report, transmit the
suppressed User Input on the media stream, and stop suppression.

Once the User Interface detects a match and it sends a NOTIFY request
to report the User Input, the User Interface MUST stop suppression.
Clearly, if subsequent User Input matches another <pre> expression,
then the User Interface MUST start suppression.

After suppression begins, it may become clear that a match will not
occur. For example, take the expression
<regex><pre>*8</pre>xxx[2-9]xxxxxx</regex>
At the point the User Interface receives "*8", it will stop
forwarding stimulus. Let us say that the next three digits are
"408". If the next digit is a zero or one, the pattern will not
match.

   NOTE: It is critically important for the User Interface to have a
   sensible inter-digit timer. This is because an errant dot (".")
   may suppress digit sending forever.

Applications should be very careful to indicate suppression only when
they are fairly sure the user will enter a digit string that will
match the regular expression. In addition, applications should deal
with situations such as no-match or time-out. This is because the
User Interface will hold digits, which will have obvious User
Interface issues in the case of a failure.

3.5 User Input Buffer Behavior

User Interfaces MUST buffer User Input upon receipt of an

authenticated and accepted subscription. Subsequent KPML documents
apply their patterns against the buffered User Input. Some
applications use modal interfaces where the first few key presses
determine what the following key presses mean. For a novice user,
the application may play a prompt describing what mode the
application is in. However, "power users" often barge through the
prompt.

User Interfaces MUST NOT provide a subscriber with digits that were
detected prior to the authentication and authorization of that
subscriber. Without prohibition, a subscriber might be able to gain
access to calling card or other information that predated the
subscriber's participation in the call. Note that this prohibition
MUST be applied on a per-subscription basis.

KPML provides a <flush> tag in the <pattern> element. The default is
not to flush User Input. Flushing User Input has the effect of
ignoring key presses entered before the installation of the KPML
subscription. To flush User Input, include the tag
<flush>yes</flush>
in the KPML subscription document. Note that this directive affects
only the current subscription dialog/id combination.

Lock step processing of User Input is where the User Interface issues
a notification, the Application processes the notification while the
User Interface buffers additional User Input, the Application
requests more User Input, and only then does the User Interface
notify the Application based on the collected User Input. To direct
the User Interface to operate in lock-step mode, set the <pattern>
attribute persist="single-notify".

The User Interface MUST be able to process <flush>no</flush>. This
directive is effectively a no-op.

Other string values for <flush> may be defined in the future. If the
User Interface receives a string it does not understand, it MUST
treat the string as a no-op.

If the user presses a key that cannot match any pattern within a
<regex> tag, the User Interface MUST discard all buffered key presses
up to and including the current key press from consideration against
the current or future KPML documents on a given dialog. However, as
described above, once there is a match, the User Interface buffers
any key presses the user entered subsequent to the match.

   NOTE: This behavior allows for applications to only receive User
   Input that is of interest to them. For example, a pre-paid
   application only wishes to monitor for a long pound. If the user

enters other stimulus, presumably for other applications, the
pre-paid application does not want notification of that User
Input. This feature is fundamentally different than the behavior
of TDM-based equipment where every application receives every key
press.

To limit reports to only complete matches, set the "nopartial"
attribute to the <pattern> tag to "true". In this case, the User
Interface attempts to match a rolling window over the collected User
input.

KPML subscriptions are independent. Thus it is not possible for the
current document to know if a following document will enable barging
or want User Input flushed. Therefore, the User Interface MUST
buffer all User Input, subject to the forced_flush caveat described
below.

On a given SUBSCRIBE dialog with a given id, the User Interface MUST
buffer all User Input detected between the time of the report and the
receipt of the next document, if any. If the next document indicates
a buffer flush, then the interpreter MUST flush all collected User
Input from consideration from KPML documents received on that dialog
with the given event id. If the next document does not indicate
flushing the buffered User Input, then the interpreter MUST apply the
collected User Input (if possible) against the digit maps presented
by the script's <regex> tags. If there is a match, the interpreter
MUST follow the procedures in Section 5.3. If there is no match, the
interpreter MUST flush all of the collected User Input.

Given the potential for needing an infinite buffer for User Input,
the User Interface MAY discard the oldest User Input from the buffer.
If the User Interface discards digits, when the User Interface issues
a KPML notification, it MUST set the forced_flush attribute of the
<response> tag to "true". For future use, the Application MUST
consider any non-null value, other than "false" that it does not
understand, to be the same as "true".
   NOTE: The requirement to buffer all User Input for the entire
   length of the session is not really onerous under normal
   operation. For example, if one has a gateway with 8,000 sessions,
   and the gateway buffers 50 key presses on each session, the
   requirement is only 400,000 bytes, assuming one byte per key
   press.

Unless there is a suppress indicator in the digit map, it is not
possible to know if the User Input is for local KPML processing or
for other recipients of the media stream. Thus, in the absence of a
suppression indicator, the User Interface transmits the User Input to
the far end in real time, using either RFC2833, generating the

appropriate tones, or both.

3.6 DRegex

3.6.1 Overview

This subsection is informative in nature.

The Digit REGular EXpression (DRegex) syntax is a telephony-oriented
mapping of POSIX Extended Regular Expressions (ERE) [17].

KPML does not use full POSIX ERE for the following reasons.
o KPML will often run on high density or extremely low power and
   memory footprint devices.
o Telephony application convention uses the star symbol ("*") for
   the star key and "x" for any digit 0-9. Requiring the developer
   to escape the star ("*") and expand the "x" ("[0-9]") is error
   prone. This also leads DRegex to use the dot (".") to indicate
   repetition, which was the function of the unadorned star in POSIX
   ERE.
o Implementation experience with MGCP [15] and H.248.1 [16] has been
   that implementers and users have a hard time understanding the
   precedence of the alternation operator ("|"). This is due both to
   an under specification of the operator in those documents and
   conceptual problems for users. Thus the SIPPING Working Group
   concluded that DRegex should not support alternation. That said,
   each KPML <pattern> element may contain multiple regular
   expressions (<regex> elements). Thus it is straightforward to
   have pattern alternatives (use multiple <regex> elements) without
   the problems associated with the alternation operator ("|"). Thus
   DRegex does not support the POSIX alternation operator.
o DRegex includes character classes (characters enclosed in square
   brackets). However, the negation operator inside a character
   class only operates on numbers. That is, a negation class
   implicitly includes A-D, *, and #. Including A-D, *, and # in a
   negation operator is a no-op. Those familiar with POSIX would
   expect negation of the digits 4 and 5, e.g., "[^45]", to include
   all other characters (including A-D, *, and #), while those
   familiar with telephony digit maps would expect negation to
   implicitly exclude non-digit characters. Since the complete
   character set of DRegex is very small, constructing a negation
   class using A-D, *, and # requires the user to specify the
   positive inverse mapping. For example, to specify all key
   presses, including A-D and *, except #, the specification would be
   "[0-9A-D*]" instead of "[^#]".

The following table shows the mapping from DRegex to POSIX ERE.

```
                    +--------+-----------+
                    | DRegex | POSIX ERE |
                    +--------+-----------+
                    | *      | \*        |
                    | .      | *         |
                    | x      | [0-9]     |
                    | [xc]   | [0-9c]    |
                    +--------+-----------+
```

                   Table 1: DRegex to POSIX ERE Mapping

   The first substitution, which replaces a star for an escaped star, is
   because telephony application designers are used to using the star
   for the (very common) star key. Requiring an escape sequence for
   this common pattern would be error prone. In addition, the usage
   found in DRegex is the same as found in MGCP [15] and H.248.1 [16].

   Likewise, the use of the dot instead of star is common usage from
   MGCP and H.248.1, and reusing the star in this context would also be
   confusing and error prone.

   The "x" character is a common indicator of the digits 0 through 9.
   We use it here, continuing the convention. Clearly, for the case
   "[xc]", where c is any character, the substitution is not a blind
   replacement of "[0-9]" for "x", as that would result in "[[0-9]c]",
   which is not a legal POSIX ERE. Rather, the substitution for "[xc]"
   is "[0-9c]".

      NOTE: "x" does not include the characters *, #, nor A through D.

   Users need to take care not to confuse the DRegex syntax with POSIX
   EREs. They are NOT identical. In particular there are many features
   of POSIX EREs that DRegex does not support.

   As an implementation note, if one makes the substitutions described
   in the above table, then a standard POSIX ERE engine can parse the
   digit string. However, the mapping does not work in the reverse
   (POSIX ERE to DRegex) direction. DRegex only implements the
   normative behavior described below.

3.6.2 Operation

   White space is removed before parsing DRegex. This enables sensible
   pretty printing in XML without affecting the meaning of the DRegex
   string.

   The following rules demonstrate the use of DRegex in KPML.

```
+-------------------------------+-------------------------------+
| Entity                        | Matches                       |
+-------------------------------+-------------------------------+
| character                     | digits 0-9, *, #, and A-D (case |
|                               | insensitive)                  |
| *                             | the * character               |
| #                             | the # character               |
| [character selector]          | Any character in selector     |
| [^digit selector]             | Any digit (0-9) not in selector |
| [range1-range2]               | Any character in range from   |
|                               | range1 to range2, inclusive   |
| x                             | Any digit 0-9                 |
| {m}                           | m repetitions of previous     |
|                               | pattern                       |
| {m,}                          | m or more repetitions of      |
|                               | previous pattern              |
| {,n}                          | At most n (including zero)    |
|                               | repetitions of previous pattern |
| {m,n}                         | at least m and at most n      |
|                               | repetitions of previous pattern |
| Lc                            | Match the character c if it is |
|                               | "long"; c is a digit 0-9 and  |
|                               | A-D, #, or *.                 |
+-------------------------------+-------------------------------+
```

```
+-------------+-------------------------------------------+
| Example     | Description                               |
+-------------+-------------------------------------------+
| 1           | Matches the digit 1                       |
| [179]       | Matches 1, 7, or 9                        |
| [2-9]       | Matches 2, 3, 4, 5, 6, 7, 8, 9            |
| [^15]       | Matches 0, 2, 3, 4, 6, 7, 8, 9            |
| [02-46-9A-D] | Matches 0, 2, 3, 4, 6, 7, 8, 9, A, B, C, D |
| x           | Matches 0, 1, 2, 3, 4, 5, 6, 7, 8, 9      |
| *6[179#]    | Matches *61, *67, *69, or *6#             |
| x{10}       | Ten digits (0-9)                          |
| 011x{7,15}  | 011 followed by seven to fifteen digits   |
| L*          | Long star                                 |
+-------------+-------------------------------------------+
```

3.7 Monitoring Direction

SIP identifies dialogs by their dialog identifier. The dialog
identifier is the remote-tag, local-tag, and Call-ID entities defined
in RFC3261 [4].

One method of determining the dialog identifier, particularly for

third-party applications, is the SIP Dialog Package [21].

For most situations, such as a monaural point-to-point call with a
single codec, the stream to monitor is obvious. In such situations
the Application need not specify which stream to monitor.

But there may be ambiguity in specifying only the SIP dialog to
monitor. The dialog may specify multiple SDP streams that could
carry key press events. For example, a dialog may have multiple
audio streams. Wherever possible, the User Interface MAY apply local
policy to disambiguate which stream or streams to monitor. In order
to have an extensible mechanism for identifying streams, the
mechanism for specifying streams is as an element content to the
<stream> tag. The only content defined today is the
<stream>reverse</stream> tag.

By default, the User Interface monitors key presses emanating from
the User Interface. Given a dialog identifier of Call-ID, local-tag,
and remote-tag, the User Interface monitors the key presses
associated with the local-tag.

In the media proxy case, and potentially other cases, there is a need
to monitor the key presses arriving from the remote user agent. The
optional <stream> element to the <request> tag specifies which stream
to monitor. The only legal value is "reverse", which means to
monitor the stream associated with the remote-tag. The User
Interface MUST ignore other values.
   NOTE: The reason this is a tag is so individual stream selection,
   if needed, can be addressed in a backwards-compatible way.
   Further specification of the stream to monitor is the subject of
   future standardization.

3.8 Multiple Simultaneous Subscriptions

An Application MAY register multiple User Input patterns in a single
KPML subscription. If the User Interface supports multiple,
simultaneous KPML subscriptions, the Application installs the
subscriptions either in a new SUBSCRIBE-initiated dialog or on an
existing SUBSCRIBE-initiated dialog with a new event id tag. If the
User Interface does not support multiple, simultaneous KPML
subscriptions, the User Interface MUST respond with an appropriate
KPML status code.

Some User Interfaces may support multiple key press event
notification subscriptions at the same time. In this situation, the
User Interface honors each subscription individually and
independently.

A SIP user agent may request multiple subscriptions on the same
SUBSCRIBE dialog, using the id parameter to the kpml event request.

One or more SIP user agents may request independent subscriptions on
different SIP dialogs, although reusing the same dialog for multiple
subscriptions is NOT RECOMMENDED.

If the User Interface does not support multiple, simultaneous
subscriptions, the User Interface MUST return a KPML document with
the KPML status code set to 533 on the dialog that requested the
second subscription. The User Interface MUST NOT modify the state of
the first subscription on account of the second subscription attempt.

4. Event Package Formal Definition

4.1 Event Package Name

This document defines a SIP Event Package as defined in RFC 3265 [5].
The event-package token name for this package is:

    "kpml"

4.2 Event Package Parameters

This package does not define any event package parameters.

4.3 SUBSCRIBE Bodies

Applications using this event package include an
application/kpml-request+xml body in SUBSCRIBE requests to indicate
which digit patterns they are interested in. The syntax of this body
type is formally described in Section 5.2

4.4 Subscription Duration

The subscription lifetime should be longer than the expected call
time. Subscriptions to this event package MAY range from minutes to
weeks. Subscriptions in hours or days are more typical and are
RECOMMENDED. The default subscription duration for this event
package is 7200 seconds.
   Subscribers MUST be able to handle the User Interface returning an
   Expires value smaller than the requested value. Per RFC3265 [5],
   the subscription duration is the value returned by the Notifier in
   the 200 OK Expires header.

4.5 NOTIFY Bodies

NOTIFY requests can contain application/kpml-response+xml (KPML
Response) bodies. The syntax of this body type is formally described
in Section 5.3. NOTIFY requests in immediate response to a SUBSCRIBE
request MUST NOT contain a body unless notifying the subscriber of an
error condition or previously buffered digits.

Notifiers MAY send notifications with any format acceptable to the
subscriber (based on the subscriber inclusion of these formats in an
Accept header). A future extension MAY define other NOTIFY bodies.
If no "Accept" header is present in the SUBSCRIBE, the body type
defined in this document MUST be assumed.

4.6 Subscriber generation of SUBSCRIBE requests

A kpml request document contains a <pattern> element with a series of
<regex> tags. Each <regex> element specifies a potential pattern for
the User Interface to match. The Section 5.1 describes the DRegex,
or digit regular expression, language.

KPML specifies key press event notification filters. The MIME type
for KPML requests is application/kpml-request+xml.

The KPML request document MUST be well formed and SHOULD be valid.
KPML documents MUST conform to XML 1.0 [18] and MUST use UTF-8
encoding.

Because of the potentially sensitive nature of the information
reported by KPML, subscribers SHOULD use sips: and MAY use S/MIME on
the content.

Subscribers MUST be prepared for the notifier to insist on
authentication of the subscription request. Subscribers MUST be
prepared for the notifier to insist on using a secure communication
channel.

4.7 Notifier processing of SUBSCRIBE requests

The user information transported by KPML is potentially sensitive.
For example, it could include calling card or credit card numbers.
Thus the User Interface (notifier) MUST authenticate the requesting
party in some way before accepting the subscription.

User Interfaces MUST implement SIP Digest authentication as required
by RFC3261 [4] and MUST implement the sips: scheme and TLS.

Upon authenticating the requesting party, the User Interface

determines if the requesting party has authorization to monitor the
user's key presses. Determining authorization policies and
procedures is beyond the scope of this specification.

The User Interface MUST return a Contact URI that has GRUU [8]
properties in the Contact header of a SIP INVITE, 1xx, or 2xx
response.

After authorizing the request, the User Interface checks to see if
the request is to terminate a subscription. If the request will
terminate the subscription, the User Interface does the appropriate
processing, including the procedures described in Section 5.2.

If the request has no KPML body, then any KPML document running on
that dialog, and addressed by the event id, if present, immediately
terminates. This is a mechanism for unloading a KPML document while
keeping the SUBSCRIBE-initiated dialog active. This can be important
for secure sessions that have high costs for session establishment.
The User Interface follows the procedures described in Section 5.2.

If the dialog referenced by the kpml subscription does not exist, the
User Interface follows the procedures in Section 5.3. Note the User
Interface MUST issue a 200 OK to the SUBSCRIBE request before issuing
the NOTIFY, as the SUBSCRIBE itself is well formed.

If the request has a KPML body, the User Interface parses the KPML
document. The User Interface SHOULD validate the XML document
against the schema presented in Section 5.2. If the document is not
valid, the User Interface rejects the SUBSCRIBE request with an
appropriate error response and terminates the subscription. If there
is a loaded KPML document on the subscription, the User Interface
unloads the document.

In addition, if there is a loaded KPML document on the subscription,
the end device unloads the document.

Following the semantics of SUBSCRIBE, if the User Interface receives
a resubscription, the User Interface MUST terminate the existing KPML
request and replace it with the new request.

It is possible for the INVITE usage of the dialog to terminate during
key press collection. The cases enumerated here are explicit
subscription termination, automatic subscription termination, and
underlying (INVITE-initiated) dialog termination.

If a SUBSCRIBE request has an expires of zero (explicit SUBSCRIBE
termination), includes a KPML document, and there is buffered User
Input, then the User Interface attempts to process the buffered

digits against the document. If there is a match, the User Interface
MUST generate the appropriate KPML report with the KPML status code
of 200. The SIP NOTIFY body terminates the subscription by setting
the subscription state to "terminated" and a reason of "timeout".

If the SUBSCRIBE request has an expires of zero and no KPML body or
the expires timer on the SUBSCRIBE-initiated dialog fires at the User
Interface (notifier), then the User Interface MUST issue a KPML
report with the KPML status code 487, Subscription Expired. The
report also includes the User Input collected up to the time the
expires timer expired or when the subscription with expires equal to
zero was processed. This could be the null string.

Per the mechanisms of RFC3265 [5], the User Interface MUST terminate
the SIP SUBSCRIBE dialog. The User Interface does this via the SIP
NOTIFY body transporting the final report described in the preceding
paragraph. In particular, the subscription state will be
"terminated" and a reason of "timeout".

Terminating the subscription when a dialog terminates ensures
reauthorization (if necessary) for attaching to subsequent
subscriptions.

If a SUBSCRIBE request references a dialog that is not present at the
User Interface, the User Interface MUST generate a KPML report with
the KPML status code 481, Dialog Not Found. The User Interface
terminates the subscription by setting the subscription state to
"terminated".

If the KPML document is not valid, the User Interface generates a
KPML report with the KPML status code 501, Bad Document. The User
Interface terminates the subscription by setting the subscription
state to "terminated".

If the document is valid but the User Interface does not support a
namespace in the document, the User Interface MUST respond with a
KPML status code 502, Namespace Not Supported.

4.8 Notifier generation of NOTIFY requests

Immediately after a subscription is accepted, the Notifier MUST send
a NOTIFY with the current location information as appropriate based
on the identity of the subscriber. This allows the Subscriber to
resynchronize its state.

The User Interface (notifier in SUBSCRIBE/NOTIFY parlance) generates
NOTIFY requests based on the requirements of RFC3265 [5].
Specifically, if a SUBSCRIBE request is valid and authorized, it will

result in an immediate NOTIFY.

The KPML payload distinguishes between an initial NOTIFY and a NOTIFY informing of key presses. If there is no User Input buffered at the time of the SUBSCRIBE (see below) or the buffered User Input does not match the new KPML document, then the immediate NOTIFY MUST NOT contain a KPML body. If User Interface has User Input buffered that result in a match using the new KPML document, then the NOTIFY MUST return the appropriate KPML document.

The NOTIFY in response to a SUBSCRIBE request has no KPML if there are no matching buffered digits. An example of this is in Figure 10.

If there are buffered digits in the SUBSCRIBE request that match a pattern, then the NOTIFY message in response to the SUBSCRIBE request MUST include the appropriate KPML document.

```
NOTIFY sip:application@example.com SIP/2.0
Via: SIP/2.0/UDP proxy.example.com
Max-Forwards: 70
To: <sip:application@example.com>
From: <sip:endpoint@example.net>
Call-Id: 439hu409h4h09903fj0ioij
Subscription-State: active; expires=7200
CSeq: 49851 NOTIFY
Event: kpml
```

Figure 10: Immediate NOTIFY Example

All subscriptions MUST be authenticated, particularly those that match on buffered input.

KPML specifies the key press notification report format. The MIME type for KPML reports is application/kpml-response+xml. The default MIME type for the kpml event package is application/kpml-response+xml.

If the requestor is not using a secure transport protocol such as TLS for every hop (e.g., by using a sips: URI), the User Interface SHOULD use S/MIME to protect the user information in responses.

When the user enters key press(es) that match a <regex> tag, the User Interface will issue a report.

After reporting, the interpreter terminates the KPML session unless the subscription has a persistence indicator. If the subscription does not have a persistence indicator, the User Interface MUST set the state of the subscription to "terminated" in the NOTIFY report.

If the subscription does not have a persistence indicator, to collect
more digits the requestor must issue a new request.

   NOTE: This highlights the "one shot" nature of KPML, reflecting
   the balance of features and ease of implementing an interpreter.

KPML reports have two mandatory attributes, code and text. These
attributes describe the state of the KPML interpreter on the User
Interface. Note the KPML status code is not necessarily related to
the SIP result code. An important example of this is where a legal
SIP subscription request gets a normal SIP 200 OK followed by a
NOTIFY, but there is something wrong with the KPML request. In this
case, the NOTIFY would include the KPML status code in the KPML
report. Note that from a SIP perspective, the SUBSCRIBE and NOTIFY
were successful. Also, if the KPML failure is not recoverable, the
User Interface will most likely set the Subscription-Sate to
"terminated". This lets the SIP machinery know the subscription is
no longer active.

If a pattern matches, the User Interface will emit a KPML report.
Since this is a success report, the code is "200" and the text is
"OK".

The KPML report includes the actual digits matched in the digit
attribute. The digit string uses the conventional characters '*' and
'#' for star and octothorpe respectively. The KPML report also
includes the tag attribute if the regex that matched the digits had a
tag attribute.

If the subscription requested digit suppression and the User
Interface suppressed digits, the suppressed attribute indicates
"true". The default value of suppressed is "false".

   NOTE: KPML does not include a timestamp. There are a number of
   reasons for this. First, what timestamp would in include? Would
   it be the time of the first detected key press? The time the
   interpreter collected the entire string? A range? Second, if the
   RTP timestamp is a datum of interest, why not simply get RTP in
   the first place? That all said, if it is really compelling to
   have the timestamp in the response, it could be an attribute to
   the <response> tag.

Note that if the monitored (INVITE-initiated) dialog terminates, the
Notifier still MUST explicitly terminate the KPML subscriptions
monitoring that dialog.

4.9 **Subscriber processing of NOTIFY requests**

   If there is no KPML body, it means the SUBSCRIBE was successful.
   This establishes the dialog if there is no buffered User Input to
   report.

   If there is a KPML document, and the KPML status code is 200, then a
   match occurred.

   If there is a KPML document, and the KPML status code is between 400
   and 499, then an error occurred with User Input collection. The most
   likely cause is a timeout condition.

   If there is a KPML document, and the KPML status code is between 500
   and 599, then an error occurred with the subscription. See Section 6
   for more on the meaning of KPML status codes.

   The subscriber MUST be mindful of the subscription state. The User
   Interface may terminate the subscription at any time.

4.10 **Handling of Forked Requests**

   Forked requests are NOT ALLOWED for this event type. Subscriptions
   to this event package MUST only be sent to SIP URIs which have GRUU
   properties.

4.11 **Rate of notifications**

   The User Interface MUST NOT generate messages faster than 25 messages
   per second, or one message every 40 milliseconds. This is the
   minimum time period for MF digit spills. Even 30-millisecond DTMF,
   as one sometimes finds in Japan, has a 20-millisecond off time,
   resulting in a 50-millisecond interdigit time. This document
   strongly RECOMMENDS AGAINST using KPML for digit-by-digit messaging,
   such as would be the case if the only <regex> is "x".

   The sustained rate of notification shall be no more than 100 Notifies
   per minute.

   The User Interface MUST reliably deliver notifications. Because
   there is no meaningful metric for throttling requests, the User
   Interface SHOULD send NOTIFY messages over a congestion-controlled
   transport, such as TCP.
      Note that all SIP implementations are already required to
      implement SIP over TCP.

[4.12](#) State Agents and Lists

KPML requests are sent to a specific SIP URI that has GRUU properties and attempt to monitor a specific stream that corresponds with a specific target dialog. Consequently, implementers MUST NOT define state agents for this event package nor allow subscriptions for this event package to resource lists using the event list extension [[22](#)].

[4.13](#) Behavior of a Proxy Server

There are no additional requirements on a SIP Proxy, other than to transparently forward the SUBSCRIBE and NOTIFY methods as required in SIP.

[5](#). Formal Syntax

[5.1](#) DRegex

The following definition follows [RFC2234](#) [[2](#)]. The definition of DIGIT is from [RFC2234](#), namely the characters "0" through "9". Note the DRegexCharacter is not a HEXDIG from [RFC2234](#). In particular, DRegexCharacter neither includes "E" nor "F". Note that DRegexCharacter is case insensitive.

```
DRegex          = 1*( DRegexPosition [ RepeatCount ] )
DRegexPosition  = DRegexSymbol / DRegexSet
DRegexSymbol    = [ "L" ] DRegexCharacter
DRegexSet       = "[" 1*DRegexSetList "]"
DRegexSetList   = DRegexCharacter [ "-" DRegexCharacter ]
DRegexCharacter = DIGIT / "A" / "B" / "C" / "D" / "*" / "#" /
                                   / "a" / "b" / "c" / "d"
RepeatCount     = "." / "{" RepeatRange "}"
RepeatRange     = Count / ( Count "," Count ) /
                          ( Count "," ) / ( "," Count )
Count           = 1*DIGIT
```

Note that future extensions to this document may introduce other characters for DRegexCharacter, in the scheme of H.248.1 [[16](#)] or possibly as named strings or XML namespaces.

[5.2](#) KPML Request

The following syntax for KPML requests uses the XML Schema [[9](#)].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:kpml-request"
 xmlns="urn:ietf:params:xml:ns:kpml-request"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
       <xs:element name="kpml-request">
         <xs:annotation>
           <xs:documentation>IETF Keypad Markup Language Request
           </xs:documentation>
         </xs:annotation>
         <xs:complexType>
           <xs:sequence>
             <xs:element name="stream" minOccurs="0">
               <xs:complexType>
                 <xs:choice>
                   <xs:element name="reverse" minOccurs="0"/>
                   <xs:any namespace="##other"/>
                 </xs:choice>
               </xs:complexType>
             </xs:element>
             <xs:element name="pattern">
               <xs:complexType>
                 <xs:sequence>
                   <xs:element name="flush" minOccurs="0">
                     <xs:annotation>
                       <xs:documentation>
                         Default is to not flush buffer
                       </xs:documentation>
                     </xs:annotation>
                     <xs:complexType>
                       <xs:simpleContent>
                         <xs:extension base="xs:string"/>
                       </xs:simpleContent>
                     </xs:complexType>
                   </xs:element>
                   <xs:element name="regex" maxOccurs="unbounded">
                     <xs:annotation>
                       <xs:documentation>
                         Key press notation is a string to allow
                         for future extension of non-16 digit
                         keypads or named keys
                       </xs:documentation>
                     </xs:annotation>
                     <xs:complexType mixed="true">
                       <xs:choice>
                         <xs:element name="pre" minOccurs="0">
                           <xs:complexType>
                             <xs:simpleContent>
                               <xs:extension base="xs:string"/>
                             </xs:simpleContent>
                           </xs:complexType>
```

```
              </xs:element>
              <xs:any namespace="##other"/>
            </xs:choice>
            <xs:attribute name="tag" type="xs:string"
                          use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="persist" use="optional">
        <xs:annotation>
          <xs:documentation>Default is "one-shot"
          </xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="one-shot"/>
            <xs:enumeration value="persist"/>
            <xs:enumeration value="single-notify"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="interdigittimer"
                    type="xs:integer"
                    use="optional">
        <xs:annotation>
          <xs:documentation>Default is 4000 (ms)
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="criticaldigittimer"
                    type="xs:integer"
                    use="optional">
        <xs:annotation>
          <xs:documentation>Default is 1000 (ms)
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="extradigittimer"
                    type="xs:integer"
                    use="optional">
        <xs:annotation>
          <xs:documentation>Default is 500 (ms)
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="long" type="xs:integer"
                    use="optional"/>
      <xs:attribute name="longrepeat" type="xs:boolean"
```

```
                                   use="optional"/>
                 <xs:attribute name="nopartial" type="xs:boolean"
                               use="optional">
                   <xs:annotation>
                     <xs:documentation>Default is false
                     </xs:documentation>
                   </xs:annotation>
                 </xs:attribute>
                 <xs:attribute name="enterkey" type="xs:string"
                               use="optional">
                   <xs:annotation>
                     <xs:documentation>No default enterkey
                     </xs:documentation>
                   </xs:annotation>
                 </xs:attribute>
               </xs:complexType>
             </xs:element>
           </xs:sequence>
           <xs:attribute name="version" type="xs:string"
                         use="required"/>
         </xs:complexType>
       </xs:element>
     </xs:schema>
```

                Figure 12: XML Schema for KPML Requests

5.3 **KPML Response**

   The following syntax for KPML responses uses the XML Schema [9].

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema targetNamespace="urn:ietf:params:xml:ns:kpml-response"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="urn:ietf:params:xml:ns:kpml-response"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
     <xs:element name="kpml-response">
       <xs:annotation>
         <xs:documentation>IETF Keypad Markup Language Response
         </xs:documentation>
       </xs:annotation>
       <xs:complexType>
         <xs:attribute name="version" type="xs:string"
                       use="required"/>
         <xs:attribute name="code" type="xs:string"
                       use="required"/>
         <xs:attribute name="text" type="xs:string"
```

```
                        use="required"/>
          <xs:attribute name="suppressed" type="xs:boolean"
                        use="optional"/>
          <xs:attribute name="forced_flush" type="xs:string"
                        use="optional">
            <xs:annotation>
              <xs:documentation>
                String for future use for e.g., number of digits lost.
              </xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="digits" type="xs:string"
                        use="optional"/>
          <xs:attribute name="tag" type="xs:string" use="optional">
            <xs:annotation>
              <xs:documentation>Matches tag from regex in request
              </xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:schema>
```

6. Enumeration of KPML Status Codes

KPML status codes broadly follow their SIP counterparts. Codes that
start with a 2 indicate success. Codes that start with a 4 indicate
failure. Codes that start with a 5 indicate a server failure,
usually a failure to interpret the document or to support a requested
feature.

KPML clients MUST be able to handle arbitrary status codes by
examining the first digit only.

Any text can be in a KPML report document. KPML clients MUST NOT
interpret the text field.

```
+------+----------------------------------------------------+
| Code | Text                                               |
+------+----------------------------------------------------+
| 200  | Success                                            |
| 402  | User Terminated Without Match                      |
| 423  | Timer Expired                                      |
| 481  | Dialog Not Found                                   |
| 487  | Subscription Expired                               |
| 501  | Bad Document                                       |
| 502  | Namespace Not Supported                            |
| 531  | Persistent Subscriptions Not Supported             |
| 532  | Multiple Regular Expressions Not Supported         |
| 533  | Multiple Subscriptions on a Dialog Not Supported   |
| 534  | Too Many Regular Expressions                       |
+------+----------------------------------------------------+
```

Table 4: KPML Status Codes

7. IANA Considerations

This document registers a new SIP Event Package, two new MIME types,
and two new XML namespaces.

7.1 SIP Event Package Registration

Package name: kpml
Type: package
Contact: Eric Burger, <e.burger@ieee.org>
Change Controller: SIPPING Working Group delegated from the IESG
Published Specification: RFCXXXX

7.2 MIME Media Type application/kpml-request+xml

MIME media type name: application
MIME subtype name: kpml-request+xml
Required parameters: none
Optional parameters: Same as charset parameter application/xml as
    specified in XML Media Types [3]
Encoding considerations: See RFC3023 [3].
Security considerations: See Section 10 of RFC3023 [3] and Section 8
    of RFCXXXX
Interoperability considerations: See RFC2023 [3] and RFCXXXX
Published specification: RFCXXXX
Applications which use this media type: Session-oriented applications
    that have primitive User Interfaces.

      Change controller: SIPPING Working Group delegated from the IESG
      Personal and email address for further information: Eric Burger
         <mailto:e.burger@ieee.org>
      Intended usage: COMMON

7.3 MIME Media Type application/kpml-response+xml

      MIME media type name: application
      MIME subtype name: kpml-resposne+xml
      Required parameters: none
      Optional parameters: Same as charset parameter application/xml as
         specified in XML Media Types [3]
      Encoding considerations: See RFC3023 [3].
      Security considerations: See Section 10 of RFC3023 [3] and Section 8
         of RFCXXXX
      Interoperability considerations: See RFC2023 [3] and RFCXXXX
      Published specification: RFCXXXX
      Applications which use this media type: Session-oriented applications
         that have primitive User Interfaces.
      Change controller: SIPPING Working Group delegated from the IESG
      Personal and email address for further information: Eric Burger
         <mailto:e.burger@ieee.org>
      Intended usage: COMMON

7.4 URN Sub-Namespace Registration for urn:ietf:xml:ns:kpml-request

      URI: urn:ietf:params:xml:ns:kpml-request

      Registrant Contact: IETF, SIPPING Work Group <sipping@ietf.org>, Eric
      Burger <e.burger@ieee.org>.

      XML:

      <?xml version="1.0"?>
      <!DOCTYPE html PUBLIC "-//W3C/DTD XHTML Basic 1.0//EN"
               "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
      <html xmlns="http://www.w3.org/1999/xhtml">
        <head>
          <meta http-equiv="content-type"
                content="text/html;charset=iso-8859-1"/>
          <title>Key Press Markup Language Request</title>
        </head>
        <body>
          <h1>Namespace for Key Press Markup Language Request</h1>
          <h2>urn:ietf:params:xml:ns:kpml-request</h2>
          <p>
      RFCXXXX.
          </p>

```
      </body>
    </html>
```

7.5 URN Sub-Namespace Registration for urn:ietf:xml:ns:kpml-response

   URI: urn:ietf:params:xml:ns:kpml-response

   Registrant Contact: IETF, SIPPING Work Group <sipping@ietf.org>, Eric
   Burger <e.burger@ieee.org>.

   XML:

```
   <?xml version="1.0"?>
   <!DOCTYPE html PUBLIC "-//W3C/DTD XHTML Basic 1.0//EN"
             "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
   <html xmlns="http://www.w3.org/1999/xhtml">
     <head>
       <meta http-equiv="content-type"
             content="text/html;charset=iso-8859-1"/>
       <title>Key Press Markup Language Response</title>
     </head>
     <body>
       <h1>Namespace for Key Press Markup Language Response</h1>
       <h2>urn:ietf:params:xml:ns:kpml-response</h2>
       <p>
 <a href="ftp://ftp.rfc-editor.org/in-notes/rfcXXXX.txt">RFCXXXX</a>.
       </p>
     </body>
   </html>
```

7.6 KPML Request Schema Registration

   Per RFC3688 [7], please register the XML Schema for KPML as
   referenced in Section 5.2 of RFCXXXX.

   URI: Please assign.

   Registrant Contact: IETF, SIPPING Work Group <sipping@ietf.org>, Eric
   Burger <e.burger@ieee.org>.

7.7 KPML Response Schema Registration

   Per RFC3688 [7], please register the XML Schema for KPML as
   referenced in Section 5.3 of RFCXXXX.

   URI: Please assign.

Registrant Contact: IETF, SIPPING Work Group <sipping@ietf.org>, Eric
Burger <e.burger@ieee.org>.

8. Security Considerations

The user information transported by KPML is potentially sensitive.
For example, it could include calling card or credit card numbers.
This potentially private information could be provided accidentally
if the notifier does not properly authenticate or authorize a
subscription. Similarly private information (such as a credit card
number or calling card number) could be revealed to an otherwise
legitimate subscriber (one operating an IVR) if digits buffered
earlier in the session are provided unintentionally to the new
subscriber.

Likewise, an eavesdropper could view KPML digit information if it is
not encrypted, or an attacker could inject fraudulent notifications
unless the messages or the SIP path over which they travel are
integrity protected.

Therefore, User Interfaces MUST NOT downgrade their own security
policy. That is, if a User Interface policy is to restrict
notifications to authenticated and authorized subscribers over secure
communications, then the User Interface must not accept an
unauthenticated, unauthorized subscription over an insecure
communication channel.

As an XML markup, all of the security considerations of RFC3023 [3]
and RFC3406 [6] must be met. Pay particular attention to the
robustness requirements of parsing XML.

Key press information is potentially sensitive. For example, it can
represent credit card, calling card, or other personal information.
Hijacking sessions allow unauthorized entities access to this
sensitive information. Therefore, signaling SHOULD be secure, e.g.,
use of TLS and sips: SHOULD be used. Moreover, the information
itself is sensitive; therefore the use of S/MIME or other appropriate
mechanism SHOULD be used.

Subscriptions MUST be authenticated in some manner. As required by
the core SIP [4] specification, all SIP implementations MUST support
digest authentication. In addition, User Interfaces MUST implement
support for the sips: scheme and SIP over TLS. Subscribers MUST
expect the User Interface to demand the use of an authentication
scheme. If the local policy of a User Interface is to use
authentication or secure communication channels, the User Interface
MUST reject subscription requests that do not meet that policy.

User Interfaces MUST begin buffering User Input upon receipt of an
authenticated and accepted subscription. This buffering is done on a
per subscription basis.

9. Examples

This section is informative in nature. If there is a discrepancy
between this section and the normative sections above, the normative
sections take precedence.

9.1 Monitoring for Octothorpe

A common need for pre-paid and personal assistant applications is to
monitor a conversation for a signal indicating a change in user focus
from the party they called through the application to the application
itself. For example, if you call a party using a pre-paid calling
card and the party you call redirects you to voice mail, digits you
press are for the voice mail system. However, many applications have
a special key sequence, such as the octothorpe (#, or pound sign) or
*9 that terminate the called party session and shift the user's focus
to the application.

Figure 16 shows the KPML for long octothorpe.

```
<?xml version="1.0">
<kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
      version="1.0">
  <pattern>
    <regex>L#</regex>
  </pattern>
</kpml-request>
```

Figure 16: Long Octothorpe Example

The regex value L indicates the following digit needs to be a
long-duration key press.

9.2 Dial String Collection

In this example, the User Interface collects a dial string. The
application uses KPML to quickly determine when the user enters a
target number. In addition, KPML indicates what type of number the
user entered.

```
   <?xml version="1.0">
   <kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
         version="1.0">
     <pattern>
       <regex tag="local-operator">0</regex>
       <regex tag="ld-operator"/>00</regex>
       <regex tag="vpn">7[x][x][x]</regex>
       <regex tag="local-number7">9xxxxxxx</regex>
       <regex tag="RI-number">9401xxxxxxx</regex>
       <regex tag="local-number10">9xxxxxxxxxx</regex>
       <regex tag="ddd">91xxxxxxxxxx</regex>
       <regex tag="iddd">011x.</regex>
     </pattern>
   </kpml-request>
```

               Figure 17: Dial String KPML Example Code

   Note the use of the "tag" attribute to indicate which regex matched
   the dialed string. The interesting case here is if the user entered
   "94015551212". This string matches both the "9401xxxxxxx" and
   "9xxxxxxxxxx" regular expressions. Both expressions are the same
   length. Thus the KPML interpreter will pick the "9401xxxxxxx"
   string, as it occurs first in document order. Figure 18 shows the
   response.

```
   <?xml version="1.0"?>
   <kpml-response xmlns="urn:ietf:params:xml:ns:kpml-resposne"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "urn:ietf:params:xml:ns:kpml-response kpml-response.xsd"
         version="1.0"
         code="200" text="OK"
         digits="94015551212" tag="RI-number"/>
```

                 Figure 18: Dial String KPML Response

10. Call Flow Examples

10.1 Supplemental Digits

   This section gives a non-normative example of an application that
   collects supplemental digits. Supplemental digit collection is where
   the network requests additional digits after the caller enters the
   destination address. A typical supplemental dial string is four

   digits in length.

```
   Ingress Gateway       Application Server        Egress Gateway
           |                     |                     |
           |                     |                     |
           |                     |                     |
           |(1) INVITE           |                     |
           |-------------------------------------------->|
           |                     |                     |
           |                     |                     |
           |(2) 200 OK           |                     |
           |<--------------------------------------------|
           |                     |                     |
           |                     |                     |
           |(3) ACK              |                     |
           |-------------------------------------------->|
           |                     |                     |
           |                     |                     |
           |(4) SUBSCRIBE (one-shot)                   |
           |<--------------------|                     |
           |                     |                     |
           |                     |                     |
           |(5) 200 OK           |                     |
           |-------------------->|                     |
           |                     |                     |
           |                     |                     |
           |(6) NOTIFY           |                     |
           |-------------------->|                     |
           |                     |                     |
           |                     |                     |
           |(7) 200 OK           |                     |
           |<--------------------|                     |
           |                     |                     |
           |                     |                     |
           |(8)                  |                     |
           |.....................|                     |
           |                     |                     |
           |                     |                     |
           |(9) NOTIFY (digits)  |                     |
           |-------------------->|                     |
           |                     |                     |
           |                     |                     |
           |(10) 200 OK          |                     |
           |<--------------------|                     |
           |                     |                     |
           |                     |                     |
           |                     |                     |
           |                     |                     |
```
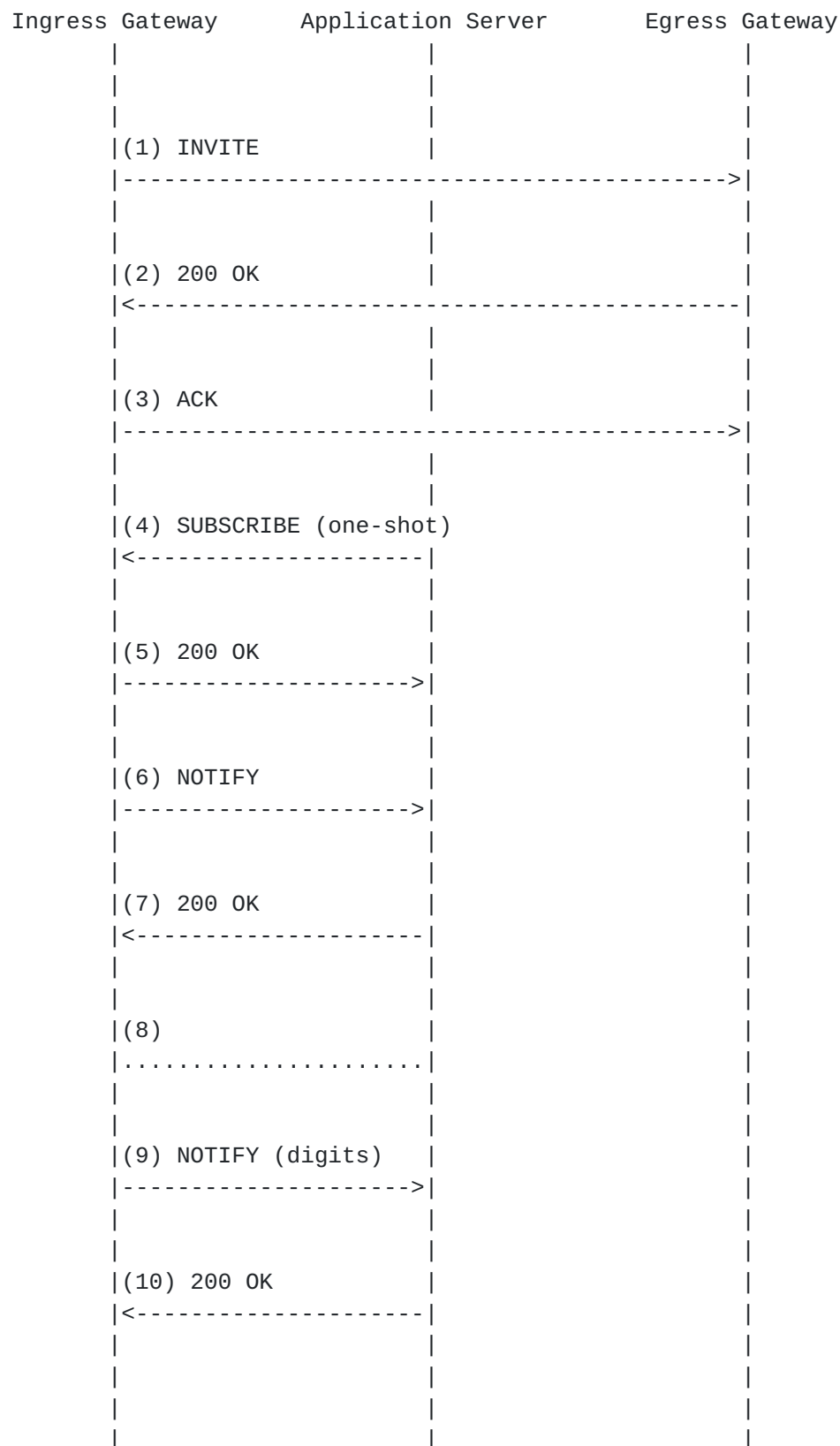
                   Figure 19: Supplemental Digits Call Flow

   In messages (1-3), the ingress gateway establishes a dialog with an
   egress gateway. The application learns the dialog ID through
   out-of-band mechanisms, such as the Dialog Package or being
   co-resident with the egress gateway. Part of the ACK message is
   below, to illustrate the dialog identifiers.

   ACK sip:gw@subA.example.com SIP/2.0
   Via: ...
   Max-Forwards: ...
   Route: ...
   From: <sip:phn@example.com>;tag=jfh21
   To: <sip:gw@subA.example.com>;tag=onjwe2
   Call-ID: 12345592@subA.example.com
   ...

   In message (4), the application requests the gateway collect a string
   of four key presses.

   SUBSCRIBE sip:gw@subA.example.com SIP/2.0
   Via: SIP/2.0/TCP client.subB.example.com;branch=q4i9ufr4ui3
   From: <sip:ap@subB.example.com>;tag=567890
   To: <sip:gw@subA.example.com>
   Call-ID: 12345601@subA.example.com
   CSeq: 1 SUBSCRIBE
   Contact: <sip:ap@client.subB.example.com>
   Max-Forwards: 70
   Event: kpml ;remote-tag="<sip:phn@example.com;tag=jfh21>"
               ;local-tag="sip:gw@subA.example.com;tag=onjwe2"
               ;call-id="12345592@subA.example.com"
   Expires: 7200
   Accept: application/kpml-response+xml
   Content-Type: application/kpml-request+xml
   Content-Length: 292

   <?xml version="1.0" encoding="UTF-8"?>
   <kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation=
         "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
       version="1.0">
     <pattern persist="one-shot">
       <regex>xxxx</regex>
     </pattern>
   </kpml-request>

   Message (5) is the acknowledgement of the subscription request.

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP subB.example.com;branch=q4i9ufr4ui3;
     received=192.168.125.12
From: <sip:ap@subB.example.com>;tag=567890
To: <sip:gw@subA.example.com>;tag=1234567
Call-ID: 12345601@subA.example.com
CSeq: 1 SUBSCRIBE
Contact: <sip:gw27@subA.example.com>
Expires: 3600
Event: kpml
```

Message (6) is the immediate notification of the subscription.

```
NOTIFY sip:ap@client.subB.example.com SIP/2.0
Via: SIP/2.0/UDP subA.example.com;branch=gw27id4993
To: <sip:ap@subB.example.com>;tag=567890
From: <sip:gw@subA.example.com>;tag=1234567
Call-ID: 12345601@subA.example.com
CSeq: 1000 NOTIFY
Contact: <sip:gw27@subA.example.com>
Event: kpml
Subscription-State: active;expires=3599
Max-Forwards: 70
Content-Length: 0
```

Message (7) is the acknowledgment of the notification message.

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP subA.example.com;branch=gw27id4993
To: <sip:ap@subB.example.com>;tag=567890
From: <sip:gw@subA.example.com>;tag=1234567
Call-ID: 12345601@subA.example.com
CSeq: 1000 NOTIFY
```

Some time elapses (8).

The user enters the input. The device provides the notification of
the collected digits in message (9). Since this was a one-shot
subscription, note the Subscription-State is "terminated".

```
NOTIFY sip:ap@client.subB.example.com SIP/2.0
Via: SIP/2.0/UDP subA.example.com;branch=gw27id4993
To: <sip:ap@subB.example.com>;tag=567890
From: <sip:gw@subA.example.com>;tag=1234567
Call-ID: 12345601@subA.example.com
CSeq: 1001 NOTIFY
Contact: <sip:gw27@subA.example.com>
Event: kpml
Subscription-State: terminated
Max-Forwards: 70
Content-Type: application/kpml-response+xml
Content-Length: 258

<?xml version="1.0" encoding="UTF-8"?>
<kpml-response xmlns="urn:ietf:params:xml:ns:kpml-response"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-response kpml-response.xsd"
      version="1.0"
      code="200" text="OK"
      digits="4336"/>
```

Message (10) is the acknowledgement of the notification.

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP subA.example.com;branch=gw27id4993
To: <sip:ap@subB.example.com>;tag=567890
From: <sip:gw@subA.example.com>;tag=1234567
Call-ID: 12345601@subA.example.com
CSeq: 1001 NOTIFY
```

10.2 Multiple Applications

This section gives a non-normative example of multiple applications.
One application collects a destination number to call. That
application then waits for a "long pound." During the call, the call
goes to a personal assistant application, which interacts with the
user. In addition, the personal assistant application looks for a
"short pound."

For clarity, we do not show the INVITE dialogs.

```
Gateway              Card Application        Personal Assistant
   |                      |                       |
   |                      |                       |
   |                      |                       |
   |(1) SUBSCRIBE (persistent)                    |
```

```
        |<-------------------|                    |
        |                    |                    |
        |                    |                    |
        |(2) 200 OK          |                    |
        |------------------->|                    |
        |                    |                    |
        |                    |                    |
        |(3) NOTIFY          |                    |
        |------------------->|                    |
        |                    |                    |
        |                    |                    |
        |(4) 200 OK          |                    |
        |<-------------------|                    |
        |                    |                    |
        |                    |                    |
        |(5)                 |                    |
        |....................|                    |
        |                    |                    |
        |                    |                    |
        |(6) NOTIFY (tag=card)|                   |
        |------------------->|                    |
        |                    |                    |
        |                    |                    |
        |(7) 200 OK          |                    |
        |<-------------------|                    |
        |                    |                    |
        |                    |                    |
        |(8)                 |                    |
        |....................|                    |
        |                    |                    |
        |                    |                    |
        |(9) NOTIFY (tag=number)                  |
        |------------------->|                    |
        |                    |                    |
        |                    |                    |
        |(10) 200 OK         |                    |
        |<-------------------|                    |
        |                    |                    |
        |                    |                    |
        |(11) SUBSCRIBE      |                    |
        |<----------------------------------------|
        |                    |                    |
        |                    |                    |
        |(12) 200 OK         |                    |
        |---------------------------------------->|
        |                    |                    |
        |                    |                    |
        |(13) NOTIFY         |                    |
```

```
     |----------------------------------------->|
     |                     |                    |
     |                     |                    |
     |(14) 200 OK          |                    |
     |<-----------------------------------------|
     |                     |                    |
     |                     |                    |
     |(15)                 |                    |
     |..........................................|
     |                     |                    |
     |                     |                    |
     |(16) NOTIFY (tag=number)                  |
     |----------------------------------------->|
     |                     |                    |
     |                     |                    |
     |(17) 200 OK          |                    |
     |<-----------------------------------------|
     |                     |                    |
     |                     |                    |
     |(18)                 |                    |
     |..........................................|
     |                     |                    |
     |                     |                    |
     |(19) NOTIFY (tag=#)  |                    |
     |----------------------------------------->|
     |                     |                    |
     |                     |                    |
     |(20) 200 OK          |                    |
     |<-----------------------------------------|
     |                     |                    |
     |                     |                    |
     |(21)                 |                    |
     |..........................................|
     |                     |                    |
     |                     |                    |
     |(22) NOTIFY (tag=number)                  |
     |----------------------------------------->|
     |                     |                    |
     |                     |                    |
     |(23) 200 OK          |                    |
     |<-----------------------------------------|
     |                     |                    |
     |                     |                    |
     |(24)                 |                    |
     |..........................................|
     |                     |                    |
     |                     |                    |
     |(25) NOTIFY (L#)     |                    |
```

```
        |-------------------->|                         |
        |                     |                         |
        |                     |                         |
        |(26) 200 OK          |                         |
        |<--------------------|                         |
        |                     |                         |
        |                     |                         |
        |                     |                         |
        |                     |                         |
```
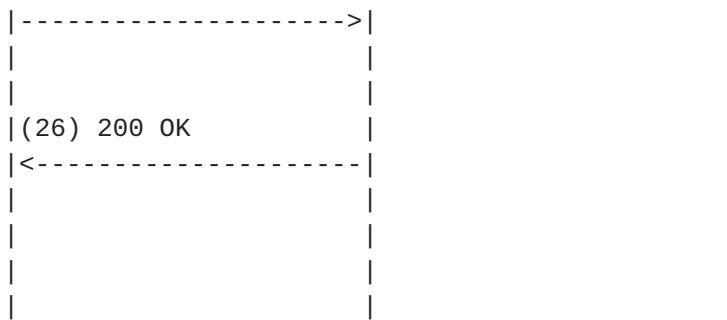
                Figure 27: Multiple Application Call Flow

     Message (1) is the subscription request for the card number.

     SUBSCRIBE sip:gw@subA.example.com SIP/2.0
     Via: SIP/2.0/TCP client.subB.example.com;branch=3qo3j0ouq
     From: <sip:ap@subB.example.com>;tag=978675
     To: <sip:gw@subA.example.com>
     Call-ID: 12345601@subA.example.com
     CSeq: 20 SUBSCRIBE
     Contact: <sip:ap@client.subB.example.com>
     Max-Forwards: 70
     Event: kpml ;remote-tag="<sip:phn@example.com;tag=jfi23>"
                 ;local-tag="sip:gw@subA.example.com;tag=oi43jfq"
                 ;call-id="12345598@subA.example.com"
     Expires: 7200
     Accept: application/kpml-response+xml
     Content-Type: application/kpml-request+xml
     Content-Length: 339

     <?xml version="1.0" encoding="UTF-8"?>
     <kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation=
             "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
           version="1.0">
       <pattern persist="persist">
         <regex tag="card">x{16}</regex>
         <regex tag="number">x{10}</regex>
       </pattern>
     </kpml-request>

     Messages 2-4 are not shown for brevity. Message (6) is the
     notification of the card number.
```

```
NOTIFY sip:ap@client.subB.example.com SIP/2.0
Via: SIP/2.0/UDP subA.example.com;branch=3qo3j0ouq
To: <sip:ap@subB.example.com>;tag=978675
From: <sip:gw@subA.example.com>;tag=9783453
Call-ID: 12345601@subA.example.com
CSeq: 3001 NOTIFY
Contact: <sip:gw27@subA.example.com>
Event: kpml
Subscription-State: active;expires=3442
Max-Forwards: 70
Content-Type: application/kpml-response+xml
Content-Length: 271

<?xml version="1.0" encoding="UTF-8"?>
<kpml-response xmlns="urn:ietf:params:xml:ns:kpml-response"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-response kpml-response.xsd"
      version="1.0"
      code="200" text="OK"
      digits="9999888877776666"/>
```

Message (7) is the acknowledgement of the notification. Time goes by
in (8). Message (9) is the notification of the dialed number.

```
NOTIFY sip:ap@client.subB.example.com SIP/2.0
Via: SIP/2.0/UDP subA.example.com;branch=3qo3j0ouq
To: <sip:ap@subB.example.com>;tag=978675
From: <sip:gw@subA.example.com>;tag=9783453
Call-ID: 12345601@subA.example.com
CSeq: 3001 NOTIFY
Contact: <sip:gw27@subA.example.com>
Event: kpml
Subscription-State: active;expires=3542
Max-Forwards: 70
Content-Type: application/kpml-response+xml
Content-Length: 278

<?xml version="1.0" encoding="UTF-8"?>
<kpml-response xmlns="urn:ietf:params:xml:ns:kpml-response"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-response kpml-response.xsd"
      version="1.0"
      code="200" text="OK"
      digits="2225551212" tag="number"/>
```

Message (11) is the request for long-pound monitoring.

```
SUBSCRIBE sip:gw@subA.example.com SIP/2.0
Via: SIP/2.0/TCP client.subB.example.com;branch=3qo3j0ouq
From: <sip:ap@subB.example.com>;tag=978675
To: <sip:gw@subA.example.com>
Call-ID: 12345601@subA.example.com
CSeq: 21 SUBSCRIBE
Contact: <sip:ap@client.subB.example.com>
Max-Forwards: 70
Event: kpml ;remote-tag="<sip:phn@example.com;tag=jfi23>"
            ;local-tag="sip:gw@subA.example.com;tag=oi43jfq"
            ;call-id="12345598@subA.example.com"
Expires: 7200
Accept: application/kpml-response+xml
Content-Type: application/kpml-request+xml
Content-Length: 295

<?xml version="1.0" encoding="UTF-8"?>
<kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
      version="1.0">
  <pattern persist="single-notify">
    <regex>L#</regex>
  </pattern>
</kpml-request>
```

Message (13) is the request from the personal assistant application
for number and pound sign monitoring.

```
SUBSCRIBE sip:gw@subA.example.com SIP/2.0
Via: SIP/2.0/TCP pahost.example.com;branch=xzvsadf
From: <sip:pa@example.com>;tag=4rgj0f
To: <sip:gw@subA.example.com>
Call-ID: 93845@pahost.example.com
CSeq: 21 SUBSCRIBE
Contact: <sip:pa12@pahost.example.com>
Max-Forwards: 70
Event: kpml ;remote-tag="<sip:phn@example.com;tag=jfi23>"
             ;local-tag="sip:gw@subA.example.com;tag=oi43jfq"
             ;call-id="12345598@subA.example.com"
Expires: 7200
Accept: application/kpml-response+xml
Content-Type: application/kpml-request+xml
Content-Length: 332

<?xml version="1.0" encoding="UTF-8"?>
<kpml-request xmlns="urn:ietf:params:xml:ns:kpml-request"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-request kpml-request.xsd"
      version="1.0">
  <pattern persist="persist">
    <regex tag="number">x{10}</regex>
    <regex tag="#">#</regex>
  </pattern>
</kpml-request>
```

Message (18) is the notification of the number collected.

```
NOTIFY sip:pa@example.com SIP/2.0
Via: SIP/2.0/UDP subA.example.com;branch=xzvsadf
To: <sip:pa@example.com>;tag=4rgj0f
From: <sip:gw@subA.example.com>;tag=9788823
Call-ID: 93845@pahost.example.com
CSeq: 3021 NOTIFY
Contact: <sip:gw27@subA.example.com>
Event: kpml
Subscription-State: active;expires=3540
Max-Forwards: 70
Content-Type: application/kpml-response+xml
Content-Length: 278

<?xml version="1.0" encoding="UTF-8"?>
<kpml-response xmlns="urn:ietf:params:xml:ns:kpml-response"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
        "urn:ietf:params:xml:ns:kpml-response kpml-response.xsd"
```

```
        version="1.0"
        code="200" text="OK" digits="3335551212" tag="number"/>
```

Message (21) is the notification of pound sign detected.

```
NOTIFY sip:pa@example.com SIP/2.0
Via: SIP/2.0/UDP subA.example.com;branch=xzvsadf
To: <sip:pa@example.com>;tag=4rgj0f
From: <sip:gw@subA.example.com>;tag=9788823
Call-ID: 93845@pahost.example.com
CSeq: 3022 NOTIFY
Contact: <sip:gw27@subA.example.com>
Event: kpml
Subscription-State: active;expires=3540
Max-Forwards: 70
Content-Type: application/kpml-response+xml
Content-Length: 264

<?xml version="1.0" encoding="UTF-8"?>
<kpml-response xmlns="urn:ietf:params:xml:ns:kpml-response"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=
          "urn:ietf:params:xml:ns:kpml-response kpml-response.xsd"
        version="1.0"
        code="200" text="OK"
        digits="#" tag="#"/>
```

Message (27) is the notification of long pound to the card application.

```
NOTIFY sip:ap@client.subB.example.com SIP/2.0
Via: SIP/2.0/UDP subA.example.com;branch=3qo3j0ouq
To: <sip:ap@subB.example.com>;tag=978675
From: <sip:gw@subA.example.com>;tag=9783453
Call-ID: 12345601@subA.example.com
CSeq: 3037 NOTIFY
Contact: <sip:gw27@subA.example.com>
Event: kpml
Subscription-State: active;expires=3216
Max-Forwards: 70
Content-Type: application/kpml-response+xml
Content-Length: 256

<?xml version="1.0" encoding="UTF-8"?>
<kpml-response xmlns="urn:ietf:params:xml:ns:kpml-response"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation=
       "urn:ietf:params:xml:ns:kpml-response kpml-response.xsd"
     version="1.0"
     code="200" text="OK"
     digits="#"/>
```

11. References

11.1 Normative References

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

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

[3]   Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC
      3023, January 2001.

[4]   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.

[5]   Roach, A., "Session Initiation Protocol (SIP)-Specific Event
      Notification", RFC 3265, June 2002.

[6]   Daigle, L., van Gulik, D., Iannella, R. and P. Faltstrom,
      "Uniform Resource Names (URN) Namespace Definition Mechanisms",
      BCP 66, RFC 3406, October 2002.

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

        2004.

   [8]   Rosenberg, J., "Obtaining and Using Globally Routable User Agent
         (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)",
         draft-ietf-sip-gruu-01 (work in progress), February 2004.

   [9]   Thompson, H., Beech, D., Maloney, M. and N. Mendelsohn, "XML
         Schema Part 1: Structures", W3C REC REC-xmlschema-1-20010502,
         May 2001.

11.2 Informative References

   [10] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson,
         "RTP: A Transport Protocol for Real-Time Applications", RFC
         1889, January 1996.

   [11] Handley, M. and V. Jacobson, "SDP: Session Description
         Protocol", RFC 2327, April 1998.

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

   [13] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits,
         Telephony Tones and Telephony Signals", RFC 2833, May 2000.

   [14] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with
         Session Description Protocol (SDP)", RFC 3264, June 2002.

   [15] Andreasen, F. and B. Foster, "Media Gateway Control Protocol
         (MGCP) Version 1.0", RFC 3435, January 2003.

   [16] Groves, C., Pantaleo, M., Anderson, T. and T. Taylor, "Gateway
         Control Protocol Version 1", RFC 3525, June 2003.

   [17] Institute of Electrical and Electronics Engineers, "Information
         Technology - Portable Operating System Interface (POSIX) - Part
         1: Base Definitions, Chapter 9", IEEE Standard 1003.1, June
         2001.

   [18] Bray, T., Paoli, J., Sperberg-McQueen, C. and E. Maler,
         "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C
         REC REC-xml-20001006, October 2000.

   [19] Rosenberg, J., "A Framework for Application Interaction in the
         Session Initiation Protocol (SIP)",
         draft-ietf-sipping-app-interaction-framework-01 (work in
         progress), February 2004.

   [20]   Burger, E., Van Dyke, J. and A. Spitzer, "Media Server Control
          Markup Language (MSCML) and Protocol", draft-vandyke-mscml-04
          (work in progress), March 2004.

   [21]   Rosenberg, J. and H. Schulzrinne, "An INVITE Inititiated Dialog
          Event Package for the Session Initiation Protocol (SIP",
          draft-ietf-sipping-dialog-package-02 (work in progress), June
          2003.

   [22]   Roach, A., Rosenberg, J. and B. Campbell, "A Session Initiation
          Protocol (SIP) Event Notification Extension for Resource
          Lists", draft-ietf-simple-event-list-05 (work in progress),
          August 2004.

Authors' Addresses

   Eric Burger
   Brooktrout Technology, Inc.
   18 Keewaydin Dr.
   Salem, NH 03079
   USA

   EMail: eburger@brooktrout.com

   Martin Dolly
   AT&T Labs

   EMail: mdolly@att.com

Appendix A. Contributors

   Ophir Frieder of the Illinois Institute of Technology collaborated on
   the development of the buffer algorithm.

   Jeff Van Dyke worked enough hours and wrote enough text to be
   considered an author under the old rules.

   Robert Fairlie-Cuninghame, Cullen Jennings, Jonathan Rosenberg, and
   we were the members of the Application Stimulus Signaling Design
   Team. All members of the team contributed to this work. In
   addition, Jonathan Rosenberg postulated DML in his "A Framework for
   Stimulus Signaling in SIP Using Markup" draft.

   This version of KPML has significant influence from MSCML [20], the
   SnowShore Media Server Control Markup Language. Jeff Van Dyke and
   Andy Spitzer were the primary contributors to that effort.

Rohan Mahy did a significant reorganization of the content, as well as providing considerable moral support in the production of this document.

That said, any errors, misinterpretation, or fouls in this document are our own.

Appendix B. Acknowledgements

Hal Purdy and Eric Cheung of AT&T Laboratories helped immensely through many conversations and challenges.

Steve Fisher of AT&T Laboratories suggested the digit suppression syntax and provided excellent review of the document.

Terence Lobo of SnowShore Networks made it all work.

Jerry Kamitses, Swati Dhuleshia, Shaun Bharrat, Sunil Menon, and Bryan Hill helped with clarifying the buffer behavior and DRegex syntax.

Silvano Brewster and Bill Fenner of AT&T Laboratories, and Joe Zebarth of Nortel helped considerably with making the text clear and DRegex tight.

Bert Culpepper and Allison Mankin gave an early version of this document a good scouring.

Scott Hollenbeck provided XML and MIME review. Tim Bray pointed out the general issue of UTF-8 versus UTF-16 with XML.

Intellectual Property Statement

Acknowledgment