Keypad Stimulus Protocol (KPML)
draft-ietf-sipping-kpml-02

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 2, 2004.

Copyright Notice

Copyright (C) The Internet Society (2004). All Rights Reserved.

Abstract

The Key Press Stimulus Protocol uses the SIP SUBSCRIBE/NOTIFY mechanism and Keypad Markup Language (KPML) to provide instructions to SIP User Agents for the reporting of user key presses.

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.

In the narrative discussion, the "user device" is a User Agent that
will report stimulus. it could be, for example, a SIP phone, edge media processor, or media gateway. An "application" is a User Agent requesting the user device to report stimulus. The "user" is an entity that stimulates the user device. In English, the user device is a phone, the application is an application server or proxy server, and the user presses keys to generate stimulus.

Table of Contents

1. Introduction ........................................ 4
2. Key Press Stimulus Protocol .......................... 5
2.1 Model ............................................... 5
2.2 Monitoring Leg ...................................... 7
2.3 Operation .......................................... 7
3. Protocol Machinery ................................... 8
3.1 Event Package Name ................................. 9
3.2 Event Package Parameters ........................... 9
3.3 SUBSCRIBE Bodies .................................. 10
3.4 Subscription Duration .............................. 10
3.5 NOTIFY Bodies ...................................... 10
3.6 Notifier Processing of SUBSCRIBE Requests ....... 10
3.7 Notifier Generation of NOTIFY Requests .......... 12
3.7.1 SIP Protocol-Generated ......................... 12
3.7.2 Match ........................................... 12
3.7.3 Inter-Digit Timeout No Match ................... 12
3.7.4 Dialog Terminated ............................... 13
3.7.5 No Call Leg ..................................... 14
3.7.6 Bad Document .................................... 14
3.7.7 One-Shot vs. Persistent Requests ............... 14
3.8 Subscriber Processing of NOTIFY Requests ....... 15
3.8.1 No KPML Body .................................... 15
3.8.2 KPML Body ....................................... 15
3.9 Handling of Forked Requests ....................... 15
3.10 Rate of Notifications ............................... 15
3.11 State Agents ...................................... 16
4. Message Format - KPML .............................. 16
4.1 KPML Request ...................................... 16
4.1.1 Pattern Matching ................................ 18
4.1.2 Digit Suppression ............................... 20
4.1.3 One-Shot and Persistent Triggers .............. 22
4.1.4 Multiple Patterns ................................ 22
4.1.5 Monitoring Direction ............................ 22
4.1.6 Multiple, Simultaneous Subscriptions .......... 23
4.2 KPML Reports ...................................... 23
4.2.1 Pattern Match Reports ......................... 24
4.2.2 KPML No Match Reports ......................... 24
5. DRegex Syntax ...................................... 25
6. Formal Syntax ....................................... 26

Burger & Dolly Expires August 2, 2004 [Page 2]
7. Enumeration of KPML Status Codes .......................... 27
8. IANA Considerations ........................................... 28
8.1 MIME Media Type application/kpml+xml .......................... 28
8.2 URN Sub-Namespace Registration for urn:ietf:xml:ns:kpml .... 29
8.3 KPML Schema Registration ...................................... 29
9. Security Considerations ......................................... 29
10. Examples .......................................................... 30
10.1 Monitoring for Octothorpe ...................................... 30
10.2 Dial String Collection .......................................... 30
10.3 Interactive Digit Collection ................................... 31
11. Call Flow Example .................................................. 32
11.1 INVITE-Initiated Dialog ......................................... 32
11.2 Third-Party Subscription ........................................ 37
11.3 Remote-End Monitoring .......................................... 38
A. Contributors ....................................................... 40
B. Acknowledgements ................................................ 40

Normative References ................................................. 38
Informative References ................................................ 38
Authors’ Addresses .................................................... 39

Intellectual Property and Copyright Statements .................. 41
1. Introduction

This document describes the Key Press Stimulus Protocol. The Key Press Stimulus Protocol exchanges messages using the SUBSCRIBE and NOTIFY methods of SIP [2] with message bodies formed from the Keypad Markup Language, KPML. KPML is a markup [7] that enables "dumb phones" to report user key-press events. Colloquially, this mechanism provides for "digit reporting" or "DTMF reporting."

A goal of KPML is to fit in an extremely small memory and processing footprint. Note KPML has a corresponding lack of functionality. For those applications that require more functionality, please refer to VoiceXML [8] and MSCML [9].

We strongly discourage the use of non-validating XML parsers, as one can expect problems with future versions of KPML.

The name of the markup, KPML, reflects its legacy support role. The public switched telephony network (PSTN) accomplished end-to-end signaling by transporting Dual-Tone, Multi-Frequency (DTMF) tones in the bearer channel. This is in-band signaling.

From the point of view of an application being signaled, what is important is the fact the stimulus occurred, not the tones used to transport the stimulus. For example, an application may ask the caller to press the "1" key. What the application cares about is the key press, not that there were two cosine waves of 697 Hz and 1209 Hz transmitted.

A SIP-signaled [3] network transports end-to-end signaling with RFC2833 [10] 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 gets the signal information, which is what it wanted in the first place.

RFC2833 correlates the time the end user pressed a digit with the user's media. However, out-of-band signaling methods, as are appropriate for user device to application signaling, do not need millisecond accuracy. On the other hand, they do need reliability, which RFC2833 does not provide.

An interested application could request notifications of every key press. However, many of the use cases for such signaling has the application interested in only one or a few keystrokes. Thus we need a mechanism for specifying to the user device what stimulus the application would like notification of.
2. Key Press Stimulus Protocol

2.1 Model

There are two usage models for the protocol. Functionally, they are both equivalent. However, it is useful to understand the use cases.

The first model is that of a SIP User Agent (UA) that directly interacts, on a given dialog, with the end device. Figure 1 shows a two-party SIP dialog. In this scenario, the SIP UA requests the End Point to report on key press events that would normally emanate from End Point port B. This could represent, for example, a toll by-pass scenario where the End Point is an ingress gateway and the SIP UA is an egress gateway.

In this case, the requesting User Agent requests digit notification on the same dialog established for the call, between SIP ports A and X.

```
+-------+        SIP         +-----+
|       A--------------------X     |
|  End  |                    | SIP |
| Point |        RTP         | UA  |
|       B--------------------Y     |
+-------+                    +-----+
```

Figure 1: Endpoint Model

The second model is that of a third-party application that is interested in entered key presses. Figure 2 shows an established two-party SIP dialog between the End Point and the SIP UA. The requesting application addresses the particular media stream by referencing the established dialog identifier referring to the dialog between SIP ports A and X.
The third model is that of a media proxy. A media proxy is a media relay in the terminology of RFC1889 [11]. However, in addition to the RTP forwarding capability of a RFC1889 media relay, the media proxy can also do light media processing, such as tone detection, tone transcoding (tones to RFC2833 [10], and so on.

The Requesting Application uses dialog identifiers to identify the stream to monitor. The default is to monitor the media entering the End Point. For example, if the Requesting Application in Figure 3 uses the dialog represented by SIP ports V-C, then the media coming from SIP UAa RTP port W gets monitored. Likewise, the dialog represented by A-X directs the End Point to monitor the media coming from SIP UAb RTP Port Y.

Figure 2: Third-Party Model
2.2 Monitoring Leg

The default leg to monitor is the leg represented by the local tag of the SIP dialog at the monitoring End Point. A requesting application MAY request monitoring of the leg represented by the remote tag of the SIP dialog at the monitoring End Point.

Not all End Point devices are able to monitor the remote media stream. However, the End Point MUST be able to report on local (End Point-generated) key press events.

If the requesting application wishes to monitor both legs at a given End Point, the application will establish two subscriptions, one for each leg.

Section 4.1.5 describes how to specify to the End Point which leg of the dialog to monitor.

2.3 Operation

The key press stimulus protocol uses explicit subscription requests and notification requests, using the semantics of SUBSCRIBE/NOTIFY [2].

Following the semantics of SUBSCRIBE, if the user device receives a second subscription on the same dialog, the user device MUST terminate the existing KPML request (if any) and replace it with the new request.

An application may register multiple digit patterns in a single KPML
If the user device supports multiple, simultaneous KPML requests, the application registers the separate requests either in a new SUBSCRIBE-initiated dialog or on an existing SUBSCRIBE-initiated dialog with a new event id tag.

If the user device does not support multiple, simultaneous KPML requests, it responds with an error response code. See Section 4.1.6 for more information.

A KPML request can be persistent or one-shot. Persistent requests are active until either the dialog terminates, including normal subscription expiration, the client replaces them, the client deletes them by sending a null document on the dialog, or the client deletes the subscription by sending a SUBSCRIBE with an expires of zero (0).

Standard SUBSCRIBE processing dictates the end point sends a NOTIFY response if it receives a SUBSCRIBE with an expires of zero.

One-shot requests terminate themselves once a match occurs. The "persist" KPML element specifies whether the subscription remains registered for the duration specified in the SUBSCRIBE message or if it automatically terminates after a pattern matches.

KPML requests route to the user device using standard SIP request routing. A KPML request identifies the leg in question in one of two ways. The first method is to send the request on an existing, INVITE-initiated dialog. The second method is to explicitly identify the call leg by its dialog identifiers.

Response messages are KPML documents (messages). If the user device matched a digit map, the response indicates the digits detected and whether the user device suppressed digits. If the user device had an error, such as a timeout, it will indicate that instead.

3. Protocol Machinery


The registration of a digit map is simply setting a digit event notification filter. When the device detects the digits, it sends an event notification to the application.

The following sub-sections are the formal specification of the KPML SIP-specific event notification package.
3.1 Event Package Name

The name for the Key Press Stimulus Protocol package is "kpml".

3.2 Event Package Parameters

The "leg" parameter identifies the call leg being monitored.

If the "leg" parameter is not present, the SUBSCRIBE MUST be on an established INVITE-initiated SIP dialog. In this case, the leg the end device monitors is the call leg associated with the established dialog. If there is no corresponding dialog or call leg, the end device will send a 481 result code in a KPML notification.

NOTE: The SUBSCRIBE may succeed, resulting in a SIP 200 OK. However, the "current state" will be the KPML 481 result, and the subscription state will be "terminated."

SIP identifies call legs by their dialog identifier. The dialog identifier is the remote-tag, local-tag, and Call-ID entities.

To identify a specific dialog, all three of these parameters MUST be present. Usually, the local-tag is the To: entity with the To tag, the remote-tag is the From: entity including tag, and the call-id matches the Call-ID. Although semantically different, the important entities are the To: and From: tags.

Note 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 End Point 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 <reverse/> tag.

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.

The BNF for these parameters is as follows. The definitions of callid, token, EQUAL, SWS, and DQUOTE are from RFC3261 [3].

```plaintext
call-id   = "call-id" EQUAL DQUOTE callid DQUOTE
from-tag = "from-tag" EQUAL token
to-tag   = "to-tag" EQUAL token
```

The call-id parameter is a quoted string. This is because the BNF for word (which is used by callid) allows for characters not allowed
within token. One usually just copies these elements from the Call-Id, to, and from fields of the SIP INVITE.

One can use any method of determining the dialog identifier. One method available, particularly for third-party applications, is the SIP Dialog Package [12].

3.3 SUBSCRIBE Bodies

Key press filtering requests use KPML, as described in Section 4.1. The MIME type for KPML is application/kpml+xml.

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

Subscribers MUST be prepared for the notifier to insist on authentication at a minimum and encryption as a likelihood.

3.4 Subscription Duration

The subscription lifetime should be longer than the expected call time. The default subscription lifetime (Expires value) MUST be 7200 seconds. This two-hour subscription time is entirely arbitrary. Please contact the editor if you have a better suggestion, and why.

Subscribers MUST be able to handle the end device returning an Expires value smaller than the requested value. Per RFC3265 [2], the subscription duration is the value returned by the end device in the 200 OK response Expires entity.

3.5 NOTIFY Bodies

The key press notification uses KPML, as described in Section 4.2. The MIME type for KPML is application/kpml+xml. The default MIME type for the kpml event package is application/kpml+xml.

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

3.6 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 first action of the end device (notifier) SHOULD be to authenticate the requesting party.
End devices MUST support digest authentication at a minimum.

End devices MUST support the sips: scheme and TLS.

Upon authenticating the requesting party, the end device 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.

NOTE: While it would be good to require both authorization and user notification for KPML, some uses, such as lawful intercept pen registers, have very strict authorization requirements yet have a requirement of no user notification. Conversely, pre-paid applications running on a private network may have no authorization requirements and already have implicit user acceptance of key press monitoring. Thus we cannot give any guidelines here.

After authorizing the request (RECOMMENDED), the end device checks to see if the request is to terminate a subscription. If the request will terminate the subscription, the end device does the appropriate processing, including the procedures described in Section 3.7.4.

If the request has no KPML body, than 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, such as TLS. The end device follows the procedures described in Section 3.7.1.

If the SUBSCRIBE request arrived on an INVITE-initiated dialog, and there is no "leg" parameter to the kpml subscription, then the KPML document acts upon the call legs created by the INVITE-initiated dialog.

If the SUBSCRIBE request has a "leg" parameter to the kpml subscription, then the KPML document acts upon the call leg referred to by the "leg" parameter. If appropriate, the end device SHOULD validate the requestor has authorization to monitor a given leg.

If the SUBSCRIBE request has a "leg" parameter to the kpml subscription, but the referenced leg does not exist, the end device follows the procedures in Section 3.7.5. Note the end device MUST issue a 200 OK before issuing the NOTIFY, as the SUBSCRIBE itself is well-formed.

If the request has a KPML body, the end device parses the KPML document. The end device SHOULD validate the XML document against
the schema presented in Section 6. If the document is not valid, the end device performs the procedures described in Section 3.7.6. If there is a loaded KPML document on the dialog (and given event id, if present), the end device unloads the document.

If the KPML document is valid, and the end device is capable of performing the monitoring, the end device performs the filtering specified by the KPML document. See Section 4 for the specification of KPML.

3.7 Notifier Generation of NOTIFY Requests

3.7.1 SIP Protocol-Generated

The end device (notifier in SUBSCRIBE/NOTIFY parlance) generates NOTIFY requests based on the requirements of RFC3265 [2]. Specifically, unless a SUBSCRIBE request is not valid, all SUBSCRIBE requests will result in an immediate NOTIFY.

The KPML payload distinguishes between a NOTIFY that RFC3265 mandates and a NOTIFY informing of key presses. If there are no digits quarantined at the time of the SUBSCRIBE (see Section 4.1 below) or the quarantined digits do not match the new KPML document, then the immediate NOTIFY MUST NOT contain a KPML body. If end device has digits quarantined that result in a digit match using the new KPML document, then the NOTIFY MUST return the appropriate KPML document.

3.7.2 Match

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

3.7.3 Inter-Digit Timeout No Match

Once a user starts to enter digits, it is highly likely they will enter all of the digits of interest within a specific time period. There is a temporal locality of reference for key presses. It is possible for users to accidentally press a key, however. Moreover, users may start pressing a key and then be lost as to what to do next. For applications to handle this situation, KPML allows applications to request notification if the user starts to enter digits but then stops before a digit map matches.

Once the end point detects a key press that matches the first character of a digit map, the end point starts the interdigit timer specified in the <pattern> tag. Every subsequent key press detected restarts the interdigit timer. If the interdigit timer expires, the
end point generates a KPML report with the KPML status code 423, Timer Expired. The report also includes the digits collected up to the time the timer expired. This could be the null string. After sending the NOTIFY, the end point will resume quarantining additional detected digits.

Applications may have different requirements for the interdigit timer. For example, applications targeted to user populations that tend to key in information slowly may require longer interdigit timers. The specification of the interdigit timer is in milliseconds. The default value is 4000, for 4 seconds. A value of zero indicates disabling the interdigit timer. The End Device MUST round up the requested interdigit timer to the nearest time increment it is capable of detecting.

3.7.4 Dialog Terminated

It is possible for a 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 request, and there are quarantined digits, then the end device attempts to process the quarantined digits against the document. If there is a match, the end device generates 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 subscription was on a SUBSCRIBE-initiated dialog, and there are no more active event id’s associated with the dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

If the requesting party issues a SUBSCRIBE with an expires of zero and no KPML body or the expires timer on the SUBSCRIBE-initiated dialog fires at the end device (notifier), then the end device issues a KPML report with the KPML status code 487, Subscription Expired. The report also includes the digits 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. Also, note that the digits in this case cannot match a digit map. If they did, the end device would have generated a KPML match report if they did.

Again, per the mechanisms of RFC3265 [2], the end device will terminate the SIP SUBSCRIBE dialog. The end device 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". If the subscription was on a SUBSCRIBE-initiated dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

3.7.5 No Call Leg

If a SUBSCRIBE request references a dialog that is not present at the endpoint, usually by specifying a dialog identifier through the leg parameter to the kpml event package, the end point generates a KPML report with the KPML status code 481, Dialog Not Found. The end device terminates the subscription by setting the subscription state to "terminated". If the subscription was on a SUBSCRIBE-initiated dialog, and there are no more active event id’s associated with the dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

IMPORTANT: The end device can invoke this procedure if the dialog underlying a subscription terminates. For example, a SUBSCRIBE-initiated dialog subscribes to the state of a different dialog (call) via the leg kpml parameter. That different call may terminate before the SUBSCRIBE-initiated dialog terminates. In this case, the end device MUST terminate the SUBSCRIBE-initiated dialog. This ensures reauthorization (if necessary) for attaching to subsequent call legs.

3.7.6 Bad Document

If the KPML document is not valid, the end device generates a KPML report with the KPML status code 501, Bad Document. The end device terminates the subscription by setting the subscription state to "terminated". If the subscription was on a SUBSCRIBE-initiated dialog, and there are no more active event id’s associated with the dialog, then the end point MUST consider the dialog terminated. If the subscription was on an INVITE-initiated dialog, then the end point MAY release KPML-specific resources related to the dialog, but it MUST NOT alter the state of the INVITE-initiated dialog.

3.7.7 One-Shot vs. Persistent Requests

A one-shot kpml subscription is one that the KPML document does not mark as persistent. If the end device detects a key press stimulus...
that triggers a one-shot KPML event, then the end device (notifier) 
MUST set the "Subscription-State" in the NOTIFY message to 
"terminated". At this point the end device MUST consider the 
subscription destroyed. The end device MUST quarantine digits per 
the controls specified in Section 4.1.

For persistent kpml subscriptions, the KPML document remains active 
for the lifetime of the subscription.

3.8 Subscriber Processing of NOTIFY Requests

3.8.1 No KPML Body

If there is no KPML body, it means the SUBSCRIBE was successful. 
This establishes the dialog if there are no quarantined digits to 
report.

3.8.2 KPML Body

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 4xx, then an 
error occurred with digit collection. The most likely cause is a 
timeout condition.

If there is a KPML document, and the KPML status code is 5xx, then an 
error occurred with the subscription. See Section 7 for more on the 
meaning of error codes.

The subscriber MUST be mindful of the subscription state. The end 
device may terminate the subscription at any time.

3.9 Handling of Forked Requests

The SUBSCRIBE behavior described in Section 3.6 ensures that it is 
only possible to have a subscription where there is an active (e.g., 
voice) dialog. Thus the case of multiple subscription installation 
cannot occur.

3.10 Rate of Notifications

The end device MUST NOT generate messages faster than 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".

Because there is no meaningful metric for throttling requests. In addition, the end device MUST reliably deliver notifications. Thus the end device SHOULD send NOTIFY messages over a congestion-controlled transport, such as TCP or SCTP.

End devices MUST at a minimum implement SIP over TCP.

3.11 State Agents

Not applicable.

4. Message Format - KPML

The Key Press Stimulus Protocol exchanges KPML messages. There are two, mutually exclusive elements to KPML: the request and response.

4.1 KPML Request

A KPML request document (message) contains a <request> entity containing a <pattern> tag with a series of <regex> tags. The <regex> element specifies a digit pattern for the device to report on. Section 5 describes the DRegex, or digit regular expression, language.

Some devices can buffer entered digits. Subsequent KPML requests first apply their patterns against the buffered digits. Some applications use modal interfaces where the first few key presses determine what the following digits 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.

The protocol provides a <flush> tag in the <pattern> element. The default is not to flush digits. Flushing digits means the user device flushes any buffered digits. This has the effect of ignoring digits entered before the KPML request. To flush digits, the KPML includes <flush>yes</flush>.

The End Device MUST be able to receive <flush>no</flush>. This directive is effectively a no-op.

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

If the user presses a key not matched by the <regex> tags, the user device MUST discard the key press from consideration against the
current or future KPML messages. However, as described above, once there is a match, the user device quarantines any key presses the user entered subsequent to the match.

NOTE: This behavior allows for applications to only receive digits that interest them. For example, a pre-paid application only wishes to monitor for a long pound. If the user enters other digits, presumably for other systems, the pre-paid application does not want notification of those digits. This feature is fundamentally different than the behavior of every system receiving every digit that TDM-based equipment provides.

The end device MAY support an inter-digit timeout value. This is the amount of time the end device 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 interdigit timer attribute to the <pattern> tag. The default is 4000 milliseconds. If the end device does not support the specification of an inter-digit timeout, the end device MUST silently ignore the specification. If the end device supports the specification of an inter-digit timeout, but not to the granularity specified by the value presented, the end device MUST round up the requested value to the closest value it can support.

KPML messages are independent. Thus it is not possible for the current document to know if a following document will enable barging or want the digits flushed. Therefore, the user device MUST quarantine all digits detected between the time of the report and the interpretation of the next script, if any. If the next script indicates a buffer flush, then the interpreter MUST flush all collected digits from consideration from KPML documents received on that dialog with the given event id. If the next script does not indicate flushing the quarantine digits, then the interpreter MUST apply the collected digits (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 3.7.2 If there is no match, the interpreter MUST flush all of the collected digits.

Unless there is a suppress indicator in the digit map, it is not possible to know if the signaled digits are for local KPML processing or for other recipients of the media stream. Thus, in the absence of a digit suppression indicator, the user device transmits the digits to the far end in real time, using either RFC2833, generating the appropriate tones, or both.

The section Digit Suppression (Section 4.1.2) describes the operation of the suppress indicator.
4.1.1 Pattern Matching

4.1.1.1 Inter-Digit Timing

The pattern matching logic works as follows. KPML endpoints MUST follow the logic presented in this section so that multiple implementations will perform deterministically on the same KPML document given the same key press input.

The pattern match algorithm matches the longest regular expression. This is the same mode as H.248.1 [13] and not the mode presented by MGCP [14]. The pattern match algorithm choice has an impact on determining when a pattern matches. Consider the following KPML document.

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

Figure 5: Greedy Matching

In Figure 5, 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.

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

Figure 6: Timeout Matching

Figure 6 is a typical NANP 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). From a user’s perspective, they do 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. Typically, it is the pound key (#). Now, consider the following snippet.

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

**Figure 7: Timeout Matching with Enter**

The problem with the approach in Figure 7 is that the digit collector will still look for a digit after the "#" in the seven-digit case. Worse yet, the "#" will appear in the returned dial string.

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

```xml
<request>
  <pattern enterkey="#">
    <regex>xxxxxxx</regex>
    <regex>xxxxxxxxxx</regex>
  </pattern>
</request>
```

**Figure 8: Timeout Matching with Enter Key**

In Figure 8 the enterkey parameter 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.

To address the various key press collection scenarios, we define three timers. The timers are the critical timer (criticaltimer), the inter-digit timer (interdigittimer), and the extra digit timer (extradigittimer). The critical timer is the time to wait for another digit if the collected digits can match a pattern. The extra timer is the time to wait after the longest match has occurred (presumably for the return key). The inter-digit timer inter-digit timer is the time to wait between digits in all other cases. Note there is no start timer, as that concept does not apply in the KPML context.

All of these timers are parameters to the `<pattern>` tag.

### 4.1.1.2 Intra-Digit Timing

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 long pound.

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

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 longtimer milliseconds, the Long modifier is true.

NOTE: It is a local matter at the end device to consider multiple presses of the same key during the longtimer period to be equivalent to the Long version of that key. This is to support end devices that do not generate continuous key press tones.

4.1.2 Digit Suppression

Under basic operation, a KPML endpoint will transmit in-band tones (RFC2833 [10] or actual tone) in parallel with digit reporting.

NOTE: If KPML did not have this behavior, then a user device 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 device, 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 device 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 effect a transfer.

For those situations, KPML adds a digit suppression tag, "pre", to the <regex> tag. There MUST NOT be more than one <pre> in any given
If there is only a single <pattern> and a single <regex>, the suppression processing is straightforward. The end-point passes digits until the stream matches the regular expression pre. At that point, the endpoint will continue collecting digits, but will suppress the generation or pass-through of any in-band digits.

If the endpoint suppressed digits, it MUST indicate this by including the attribute "suppressed" with a value of "yes" in the digit report.

Clearly, if the end device is processing the KPML document against quarantined digits, it is too late to suppress digits, as the end device has long sent the digits. This is a situation where there is a <pre> specification, but the "suppressed" attribute is not "yes" in the digit report.

A KPML endpoint MAY perform digit suppression. If it is not capable of digit suppression, it ignores the digit suppression attribute and will never send a suppressed indication in the digit report. In this case, it will match concatenated patterns of pre+value.

At some point in time, the endpoint will collect enough digits to the point it hits a <pre> pattern. The interdiggittimer attribute indicates how long to wait once the user enters digits before reporting a time-out error. If the interdiggittimer expires, the endpoint MUST issue a time-out report, transmit the suppressed digits on the media stream, and stop suppressing digit transmission.

Once the end device detects a match and it sends a NOTIFY request to report the digit string, the end device MUST stop digit suppression. Clearly, if subsequent digits match another <pre> expression, then the end device MUST start digit suppression.

After digit 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 endpoint receives "*8", it will stop forwarding digits. 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 endpoint to have a sensible inter-digit timer. This is because an errant dot (".") may suppress digit sending forever. See Section 4.1 for setting the inter-digit timer.

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 endpoint will hold digits, which will have obvious user interface issues in the case of a failure.

4.1.3 One-Shot and Persistent Triggers

The KPML document specifies if the patterns are to be persistent by setting the persistent attribute to the <pattern> tag to "true". Otherwise, the request will be a one-shot subscription. If the end device does not support persistent subscriptions, it returns a KPML document with the KPML result code set to 531. If there are digits in the quarantine buffer and the digits match an expression in the KPML document, the end device prepares the appropriate KPML document.

4.1.4 Multiple Patterns

Some end devices 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 end device sends back in the notification report upon a match in the digit map. In the case of multiple matches, the end device MUST chose the longest match in the KPML document. If multiple matches match the same length, the end device MUST chose the first expression listed in the subscription KPML document based on KPML document order.

If the end device does not support multiple regular expressions in a pattern request, the end device MUST return a KPML document with the KPML result code set to 532.

4.1.5 Monitoring Direction

By default, the end device monitors key presses emanating from the device. Given a dialog identifier of Call-ID, local-tag, and remote-tag, the end device 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 end point 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.
4.1.6 Multiple, Simultaneous Subscriptions

Some end devices may support multiple key press event notification subscriptions at the same time. In this situation, the end device 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. In the body of the SUBSCRIBE is a leg parameter that indicates which leg to monitor. Section 3.2 describes the dialog addressing mechanism in detail.

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

4.2 KPML Reports

When the user enters key press(es) that match a <regex> tag, the end device 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 end device 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. If your goal is to build an IVR session, we strongly suggest you investigate more appropriate technologies such as VoiceXML [8] or MSCML [9].

KPML reports have two mandatory attributes, code and text. These attributes describe the state of the KPML interpreter on the end device. Note the KPML 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 failure 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 end
device will most likely set the Subscription-State to terminated. This lets the SIP machinery know the subscription is no longer active.

4.2.1 Pattern Match Reports

If a pattern matches, the end device 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 (Section 4.1.2) and the end device 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 it 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.

4.2.2 KPML No Match Reports

There are a few circumstances in which the end device will emit a no match report. They are an immediate NOTIFY in response to SUBSCRIBE request (no digits detected yet), a request for service not supported by end device, or a failure of a digit map to match a string (timeout).

4.2.2.1 Immediate NOTIFY

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

If there are quarantined 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

5. DRegex Syntax

The Digit REGular EXpression (DRegex) syntax follows the Unix egrep and Java Regular Expression syntax.

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 describe the use of DRegex in KPML.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>digit</td>
<td>digit 0-9 and A-D</td>
</tr>
<tr>
<td>[digit selector]</td>
<td>Any digit in selector</td>
</tr>
<tr>
<td>[^digit selector]</td>
<td>Any digit NOT in selector</td>
</tr>
<tr>
<td>[digit-range]</td>
<td>Any digit in range</td>
</tr>
<tr>
<td>x</td>
<td>Any digit 0-9</td>
</tr>
<tr>
<td>.</td>
<td>Zero or more repetitions of</td>
</tr>
<tr>
<td></td>
<td>previous pattern</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{m}</td>
<td>m repetitions of previous</td>
</tr>
<tr>
<td></td>
<td>pattern</td>
</tr>
<tr>
<td>{m,}</td>
<td>m or more repetitions of</td>
</tr>
<tr>
<td></td>
<td>previous pattern</td>
</tr>
<tr>
<td>{,n}</td>
<td>At most n (including zero)</td>
</tr>
<tr>
<td></td>
<td>repetitions of previous pattern</td>
</tr>
<tr>
<td></td>
<td>at least m and at most n</td>
</tr>
<tr>
<td>{m,n}</td>
<td>repetitions of previous pattern</td>
</tr>
<tr>
<td></td>
<td>at least m and at most n</td>
</tr>
<tr>
<td>Ldigit</td>
<td>Match the digit if it is &quot;long&quot;</td>
</tr>
</tbody>
</table>
### 6. Formal Syntax

The following syntax in Figure 11 uses the XML Schema [4].

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) 
by Eric Burger (Snowshore Networks Inc.) -->
<xs:schema targetNamespace="urn:ietf:params:xml:ns:kpml"
  xmlns="urn:ietf:params:xml:ns:kpml"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="kpml">
    <xs:annotation>
      <xs:documentation>IETF Keypad Markup Language</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice>
        <xs:element name="request">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="pattern">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="flush" type="xs:string" minOccurs="0"/>
                    <xs:element name="regex" maxOccurs="unbounded">
                      <xs:complexType mixed="true">
                        <xs:sequence>
                          <xs:element name="pre" type="xs:string" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attribute name="tag" type="xs:string"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="persistent" type="xs:boolean"
use="optional"/>
<xs:attribute name="enterkey" type="xs:string"
use="optional"/>
<xs:attribute name="interdigittimer" type="xs:integer"
use="optional"/>
<xs:attribute name="criticaldigittimer" type="xs:integer"
use="optional"/>
<xs:attribute name="extradigittimer" type="xs:integer"
use="optional"/>
<xs:attribute name="longtimer" type="xs:integer"
use="optional"/>
</xs:complexType>
</xs:element>
<xs:element name="stream" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="response">
<xs:complexType>
<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="digits" type="xs:string" use="optional"/>
<xs:attribute name="tag" type="xs:string" use="optional"/>
</xs:complexType>
</xs:element>
</xs:choice>
<xs:attribute name="version" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:schema>

Figure 11: XML Schema for KPML

7. Enumeration of KPML Status Codes

KPML failure 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 (call leg) Not Found                             |
| 487  | Subscription Expired                                    |
| 501  | Bad Document                                            |
| 531  | Persistent Subscriptions Not Supported                  |
| 532  | Multiple or Alternate Regular Expressions Not Supported |
| 533  | Multiple Subscriptions on a Call Leg Not Supported      |
+-----------------+---------------------------------------------------------+

Table 3: KPML Failure Codes

8. IANA Considerations

8.1 MIME Media Type application/kpml+xml

MIME media type name: application
MIME subtype name: kpml+xml
Required parameters: none
Optional parameters: charset

charset This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in XML Media Types [5].

Encoding considerations: See RFC3023 [5].


Published specification: This document.

Applications which use this media type: Session-oriented applications that have primitive user interfaces.

Intended usage: COMMON
8.2URN Sub-Namespace Registration for urn:ietf:xml:ns:kpml

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

Registrant Contact: Eric Burger <eburger@ietf.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</title>
</head>
<body>
  <h1>Namespace for Key Press Markup Language</h1>
  <h2>urn:ietf:params:xml:ns:kpml</h2>
</body>
</html>

8.3KPML Schema Registration

Please register the XML Schema for KPML as referenced in Section 6.

9. Security Considerations

As an XML markup, all of the security considerations of RFC3023 [5] and RFC3406 [6] apply. Pay particular attention to the robustness requirements of parsing XML.

Key press information is potentially sensitive. 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. Thus if TLS is not used, S/MIME or other appropriate mechanism SHOULD be used.

End devices implementing this specification MUST implement TLS and SHOULD implement S/MIME at a minimum.
10. 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.

10.1 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 leg and shift the user’s focus to the application.

Figure 13 shows the KPML for long octothorpe. Note that the href is really on one line, but divided for clarity.

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

Figure 13: Long Octothorpe Example

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

10.2 Dial String Collection

In this example, the user device 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"?>
<kpm xmlns="urn:ietf:params:xml:ns:kpml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
    version="1.0">
    <request>
        <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>
</request>
</kpml>

Figure 14: 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. By following the rules described in Section 4.1.4, the KPML interpreter will pick the "9401xxxxxxx" string, as it occurs first in document order (both expressions match the same length). Figure 15 shows the response.

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

Figure 15: Dial String KPML Response

10.3 Interactive Digit Collection

This is an example where one would probably be better off using a full scripting language such as VoiceXML [8] or MSCML [9] or a device control language such as H.248.1 [13].

In this example, an application requests the user device to send the
user’s signaling directly to the platform in HTTP, rather than monitoring the entire RTP stream. Figure 16 shows a voice mail menu, where presumably the application played a “Press K to keep the message, R to replay the message, and D to delete the message” prompt. In addition, the application does not want the user to be able to barge the prompt.

```xml
<?xml version="1.0"?>
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd"
     version="1.0">
  <request>
    <pattern>
      <flush>yes</flush>
      <regex tag="keep">5</regex>
      <regex tag="replay">7</regex>
      <regex tag="delete">3</regex>
    </pattern>
  </request>
</kpml>
```

**Figure 16: IVR KPML Example Code**

NOTE: This usage of KPML is clearly inferior to using a device control protocol like H.248.1. From the application’s point of view, it has to do the low-level prompt-collect logic. Granted, it is relatively easy to change the key mappings for a given menu. However, often more of the call flow than a given menu mapping gets changed. Thus there would be little value in such a mapping to KPML. We STRONGLY suggest using a real scripting language such as VoiceXML or MSCML for this purpose.

11. Call Flow Example

11.1 INVITE-Initiated Dialog

This section describes a successful subscription and notification from an Application with an End Device ("User A") in an INVITE-Initiated dialog. Note the Application can be a Record-Route Proxy, a B2BUA, or another end device.
User A              Application

INVITE F1          Application Subscribes to "***" from User A
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100 TRYING F2</td>
<td>200 OK F7</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>180 F3</td>
<td>NOTIFY F8</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>200 OK F4</td>
<td>200 OK F9</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ACK F5</td>
<td>NOTIFY F10</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Media Session</td>
<td>200 OK F11</td>
</tr>
<tr>
<td>Application</td>
<td></td>
</tr>
</tbody>
</table>

Connection setup between User A and an Application subscribing to a DTMF event of "***" at User A.

F1 INVITE User A --> Application

INVITE sip:UserB@subB.example.com SIP/2.0
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
Max-Forwards: 70
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
Contact: <sip:UserA@client.subA.example.com>
Route: <sip:application.subA.example.com;lr>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Allow-Events: kpml
Supported: replaces
Content-Type: application/sdp
Content-Length: ...

v=0
o=UserA 2890844526 2890844526 IN IP4 client.subA.example.com
s=Session SDP
c=IN IP4 client.subA.example.com
t=3034423619 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

F2 100 Trying Application --> User A

SIP/2.0 100 Trying
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
;received=192.168.12.22
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
Content-Length: 0

F3 180 Ringing Application --> User A

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
;received=192.168.12.22
Record-Route: <sip:application.subA.example.com;lr>
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
Contact: <sip:UserB@client.subB.example.com>
Content Length: 0

F4 200 OK Application --> User A

SIP/2.0 200 OK
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bK74
;received=192.168.12.22
Record-Route: <sip:application.subA.example.com;lr>
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 INVITE
Contact: <sip:UserB@client.subB.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Type: application/sdp
Content-Length: ...

v=0
o=UserB 2890844527 2890844527 IN IP4 client.subB.example.com
s=Session SDP
c=IN IP4 client.subB.example.com
t=3034423619 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000

F5 ACK User A --> Application

ACK sip:UserB@subB.example.com SIP/2.0
Via: SIP/2.0/UDP client.subA.example.com:5060;branch=z9hG4bk74
Max-Forwards: 70
Route: <sip:application.subA.example.com;lr>
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 ACK
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY
Supported: replaces
Content-Length: 0

F6 SUBSCRIBE Application --> User A

SUBSCRIBE sip:UserA@subA.example.com SIP/2.0
Max-Forwards: 70
From: <sip:UserA@subA.example.com>;tag=567890
To: <sip:UserA@subA.example.com>;tag=1234567
Call-ID: 12345601@subA.example.com
CSeq: 1 SUBSCRIBE
Contact: <sip:UserB@client.subB.example.com>
Event: kpml
Expires: 7200
Accept: application/kpml+xml
Content-Type: application/kmpl+xml
Content-Length: ...

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

F7 200 OK User A --> Application

SIP/2.0 200 OK
To: <sip:UserA@subA.example.com>;tag=1234567
From: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 1 SUBSCRIBE
Contact: <sip:UserB@client.subB.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Length: 0

F8 NOTIFY User A --> Application

NOTIFY sip:UserB@subB.example.com SIP/2.0
Max-Forwards: 70
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 2 NOTIFY
Subscription-State: active;expires=3600
Content-Type: application/kpml+xml
Content-Length: ...
Event: kpml

<?xml version="1.0"?>
<kpml xmlns="urn:ietf:params:xml:ns:kpml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:kpml kpml.xsd" version="1.0">
  <response code="100" text="TRYING"/>
</kpml>

F9 200 OK Application --> User A

SIP/2.0 200 OK
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@subB.example.com>;tag=567890
F10 NOTIFY User A --> Application

```
NOTIFY sip:UserB@subB.example.com SIP/2.0
Max-Forwards: 70
From: <sip:UserA@subA.example.com>;tag=1234567
To: <sip:UserB@Application.example.com>;tag=567890
Call-ID: 12345601@subA.example.com
CSeq: 3 NOTIFY
Subscription-State: active;expires=3125
Content-Type: application/kpml+xml
Content-Length: ...
Event: kpml
```

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

F11 200 OK Application --> User A

```
SIP/2.0 200 OK
From: <sips:UserA@subA.net>;tag=1234567
To: <sips:UserB@Application.example.com>
Call-ID: 12345601@subA.com
JVD: CSeq: 3 NOTIFY
Contact: <sips:UserB@Application.example.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, SUBSCRIBE, NOTIFY
Supported: replaces
Content-Type: application/sdp
Content-Length: 0
```

11.2 Third-Party Subscription

Coming soon!
11.3 Remote-End Monitoring

Coming soon!

Normative References


Informative References


Authors' Addresses

Eric Burger
SnowShore Networks, Inc.
285 Billerica Rd.
Chelmsford, MA  01824-4120
USA

EMail: e.burger@ieee.org

Martin Dolly
AT&T Labs

EMail: mdolly@att.com
Appendix A. Contributors

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 I 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, the SnowShore Media Server Control Markup Language. Jeff Van Dyke and Andy Spitzer were the primary contributors to that effort.

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

Terence Lobo of SnowShore Networks made it all work.
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF’s procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Copyright (C) The Internet Society (2004). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
Acknowledgment

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