SIPPING
Internet-Draft
Expires: March 1, 2004

E. Burger
SnowShore Networks, Inc.

September 1, 2003

Keypad Stimulus Protocol (KPML)
draft-ietf-sipping-kpml-00

Status of this Memo

Copyright Notice

Abstract

The Keypress Stimulus Protocol uses the 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 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 3
2. Keypress Stimulus Protocol 4
2.1 Operation 4
3. Message Format - KPML 4
4. Digit Suppression 6
5. One-Shot and Persistant Triggers 7
6. Multiple, Simultaneous Markup 7
7. Reports 7
8. Examples 8
8.1 Monitoring for Octorhorpe 8
8.2 Dial String Collection 8
8.3 Interactive Digit Collection 9
8.4 SIP Request 10
9. Report Body 10
10. Formal Syntax 11
11. IANA Considerations 12
11.1 IANA Registration of MIME media type application/kpml+xml . 12
11.2 Schema Registration 13
12. Security Considerations 13
Normative References 13
Informative References 14
Author's Address 15
A. Contributors 15
B. Acknowledgements 15
Intellectual Property and Copyright Statements 17

1. Introduction

This document describes the Keypress Stimulus Protocol. The Keypress Stimulus Protocol exchanges messages over SIP with message bodies formed from the Keypad Markup Language, KPML. KPML is a markup [9] that enables "dumb phones" to report user key-press events. Colloquially, this mechanism provides for "digit reporting" or "DTMF reporting" in the signaling path.

We strongly discourage the use of non-validating XML parsers, as one can expect problems with future versions of KPML. That said, one could envision user devices that only accept SIP reporting and have a fixed parser, rather than a full XML parser. This means that 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 [10] and MSCML [11].

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 [2] network transports end-to-end signaling with RFC2833 [12] 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 is the only method that can correlate 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. Keypress Stimulus Protocol

2.1 Operation

The keypress stimulus protocol uses explicit subscription requests and notification requests.

> NOTE: The exact mechanism is coming in a later draft. The mechanism may or may not use the framework described by the applications interaction framework [13]. The protocol uses the mechanism described by the app-info [3] header.

If the user device receives a subscription for the same user instance, the user device MUST terminate the existing KPML request (if any) and replace it with the new request.

If the user device supports multiple, simultaneous KPML requests, the application must register the separate request at a different user instance at the user device. It is the responsibility of the application to ensure it uses unique user instance names.

If the user device does not support multiple, simultaneous KPML requests, it MUST reject the request with a 488 NOT ACCEPTABLE HERE. [Assuming SIP for requests.]

A KPML request can be persistent or one-shot. Persistent requests are active until either the dialog terminates, the client replaces them, or the client deletes them by sending a null document on the user instance.

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. Message Format - KPML

The Keypress Stimulus Protocol exchanges KPML messages. A KPML document (message) contains a <pattern> tag with a series of <regex> tags. The <regex> tag has a value attribute that is a H.248.1 [4] digit map.

> NOTE: We use RFC3525 digit maps instead of MGCP [14] digit maps because the former is an IETF standard and the latter is not.

> NOTE: We do not use SRGS [15] DTMF grammars because it is unlikely one would use KPML for independent digit collection in a browser context.

Interface attributes, such as the interdigit timeout and what constitutes a long key press, are implementation matters beyond the scope of this document.

For many applications, the user device needs to quarantine (buffer) 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 barge attribute to the <pattern> tag. The default is "barge=yes". Enabling barge means that the user device buffers digits and applies them immediately when the next KPML message arrives. Disabling barge by specifying "barge=no" means the user device flushes any collected digits before collecting more digits and comparing them against the <pattern> tags.

> NOTE: Quarantine and barge are separate actions. However, the barge action directly determines the quarantine action. Thus the protocol only specifies the barge action request.

If the user presses a key not matched by the <regex> tags, the user device discards 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 keys the user enters subsequent to the match.

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 has "barge=no", then the interpreter MUST flush all collected digits. If the next script has "barge=yes", then the interpreter MUST apply the collected digits against the digit maps presented by the script's <regex> tags. If there is a match, the interpreter MUST quarantine the remaining digits. 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 following section details the operation of the suppress

indicator.

4. Digit Suppression

Under basic operation, a KPML endpoint will transmit in-band tones (RFC2833 [12] 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 token to H.248.1 [4] digit maps. The digit suppression token is a "Q". There MUST NOT be more than one Q in any given <regex>.

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 up to the suppression token, Q. 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 suppresses digits, it MUST indicate this by including the attribute "suppressed" with a value of "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 token and will never send a suppressed indication in the digit report.

At some point in time, the endpoint will collect enough digits to the point it hits a suppression marker. The interdigittimer attribute indicates how long to wait once the user enters digits before reporting a time-out error. If the interdigittimer expires, the endpoint MUST issue a time-out report and transmit the suppressed digits on the media stream.

After digit suppression begins, it may become clear that a match will not occur. For example, take the regular expression "*8Qxxx[2-9]xxxxxx". 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.

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.

5. One-Shot and Persistant Triggers

NOTE: This section will be heavily dependent on the transport mechanism chosen, as the nature of the request could be determined by transport parameters rather than KPML parameters. KPML is probably a better route.

6. Multiple, Simultaneous Markup

NOTE: This section will be heavily dependent on the transport mechanism chosen. Most of the candidates will be able to a priori correlate KPML documents to results. Most of the protocol machinery will be around rules for simultaneous markup.

7. Reports

When the user enters keypress(es) that match a <regex> tag, the user device will issue a report.

> NOTE: Details coming soon.

After reporting, the interpreter terminates the KPML session unless the KPML document has a persistence indicator. Otherwise, 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 [10] or MSCML [11].

8. Examples

NOTE: This section will DEFINATELY change and be expanded.

8.1 Monitoring for Octorhorpe

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.

The following figure shows the KPML for long octothorpe. Note that the href is really on one line, but divided for clarity.

```
<?xml version="1.0">
  <kpml version="1.0">
    <request>
      <pattern>
        <regex value="ZF"
                href="http://app.example.net/cgi-bin/prepaid? \
                   session=19fsjcalksd&keypress=long-pound" />
      </pattern>
    </request>
  </kpml>
```

Figure 1 - Long Octothorpe Example

In this example, the parameter "session=19fsjcalksd" associates the http POST with the SIP call session. One can use other methods to associate the POST with a SIP call. The following examples will show these various methods.

The regex value Z indicates the following digit needs to be a long-duration key press. F, from the H.248.1 DTMF package, is the octothorpe key. In fact, KPML supports all digits, 1-9, *, #, A-D from the H.248 DTMF.1 package.

8.2 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">
  <kpml version="1.0">
    <request>
      <pattern>
        <regex value="0"
               href="http://app.carrier.net/pp/12?local-operator/>
        <regex value="00"
               href="http://app.carrier.net/pp/12?ld-operator/>
        <regex value="7xxx"
               href="http://app.carrier.net/pp/12?vpn/>
        <regex value="9xxxxxxx"
               href="http://app.carrier.net/pp/12?local-number7/>
        <regex value="9xxxxxxxxxx"
               href="http://app.carrier.net/pp/12?local-number10/>
        <regex value="91xxxxxxxxxx"
               href="http://app.carrier.net/pp/12?ddd/>
        <regex value="011x."
               href="http://app.carrier.net/pp/12?iddd/>
      </pattern>
    </request>
  </kpml>
```

Figure 4 - Dial String KPML Example Code

As before, the targets of the href's are opaque to KPML. Here the href's indicate the type of dial string, such as direct dial (ddd) or international direct dial (iddd).

8.3 Interactive Digit Collection

This is an example where one would probably be better off using a full scripting language such as VoiceXML [10] or MSCML [11] or a device control language such as H.248.1 [4].

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 5 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 version="1.0">
  <kpml version="1.0">
    <request>
      <pattern barge=off>
```

```
          <regex value="5"
      href="http://app.example.net/vm/sess$9awj08asd7?keep" />
          <regex value="7"
      href="http://app.example.net/vm/sess$9awj08asd7?replay" />
          <regex value="3"
      href="http://app.example.net/vm/sess$9awj08asd7?delete" />
        </pattern>
      </request>
    </kpml>
```

Figure 5 - IVR KPML Example Code

The target of the http post, "sess$9aej08asd7", identifies the SIP session.

NOTE: It is unclear if this usage of KPML is better than 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.

8.4 SIP Request

For example, the following figure is the example from Figure 1, but with SIP NOTIFY reporting.

```
  <?xml version="1.0">
    <kpml version="1.0">
      <request>
        <pattern>
          <regex value="ZF"
                  href="sip:" />
        </pattern>
      </request>
    </kpml>
```

Figure 6 - Long Octothorpe Example

The response body is identical to the response that Figure 1 would generate.

9. Report Body

The body of the response from the user device is a KPML response

form.

The <response> tag has an attribute, "digits". The digits attribute is the digit string. The digit string uses the conventional characters '*' and '#' for star and octothorpe respectively.

Figure 7 shows a sample response body to the example in the Dial String Collection (Section 8.2) section.

```
<?xml version="1.0">
  <kpml version="1.0">
    <response digits="0113224321234"/>
  </kpml>
```

Figure 7 - Response Body

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 keypress? 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.

10. Formal Syntax

The following syntax specification uses the XML Schema [5].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema 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:all>
              <xs:element name="pattern">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="regex"
```

```
                        maxOccurs="unbounded">
                         <xs:complexType>
                           <xs:attribute name="value"
                            type="xs:string" use="required"/>
                           <xs:attribute name="href"
                            type="xs:anyURI" use="required"/>
                         </xs:complexType>
                       </xs:element>
                     </xs:sequence>
                     <xs:attribute name="barge"
                      type="xs:boolean"
                       use="optional" default="true"/>
                     <xs:attribute name="interdigittimer"
                      type="xs:integer"
                       use="optional" default="100"/>
                   </xs:complexType>
                 </xs:element>
               </xs:all>
             </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:complexType>
           </xs:element>
         </xs:choice>
       </xs:complexType>
     </xs:element>
   </xs:schema>
```

Figure 9 - XML Schema for KPML

11. IANA Considerations

11.1 IANA Registration of 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 [6].

Encoding considerations: See RFC3023 [6].

Interoperability considerations: See RFC2023 [6] and this document.

Published specification: This document.

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

Intended usage: COMMON

11.2 Schema Registration

We really need a place to register the XML Schema. Where would that be?

12. Security Considerations

Since a KPML document directs a device to send results to an arbitrary URI, one could construct distributed denial of service attacks. For example, an errant application might send out KPML to numerous endpoints, all reporting to a single, unrelated href. Thus the policies for accepting KPML, such as access control lists ("only accept KPML from these hosts"), report limiting lists ("only send KPML responses to these hosts"), and other protections must be well thought out.

At the very least, the session startup protocol SHOULD be non-reputable and secure.

KPML presents no further security issues beyond the startup issues addressed in the companion documents to this document.

As an XML markup, all of the security considerations of RFC3023 [6] apply.

Normative References

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

[2] 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.

[3] Jennings, C., "SIP Support for Application Initiation", draft-jennings-sip-app-info-00 (work in progress), October 2002.

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

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

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

[7] 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.

[8] Burger, E., "The SIP APPINFO Method", draft-burger-sip-appinfo-00 (work in progress), 2003.

Informative References

[9] 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.

[10] World Wide Web Consortium, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C Working Draft , April 2002, <http://www.w3.org/TR/voicexml20/>.

[11] Burger, E., Van Dyke, J. and A. Spitzer, "SnowShore Media Server Control Markup Language and Protocol", draft-vandyke-mscml-00 (work in progress), November 2002.

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

[13] Rosenberg, J., "A Framework and Requirements for Application Interaction in SIP", draft-rosenberg-sipping-app-interaction-framework-00 (work in progress), November 2002.

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

[15] Hunt, A. and S. McGlashan, "Speech Recognition Grammar Specification Version 1.0", W3C CR CR-speech-grammar-20020626, June 2002.

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

[17] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C. and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[18] Van Dyke, J., Burger (Ed.), E. and A. Spitzer, "Basic Network Media Services with SIP", January 2003.

Author's Address

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

EMail: e.burger@ieee.org

Appendix A. Contributors

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 significantly 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 helped with the digit suppression logic and syntax.

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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