SIPPING
Internet-Draft
Expires: September 1, 2003

E. Burger
SnowShore Networks, Inc.

March 3, 2003

Keypad Markup Language (KPML)
draft-burger-sipping-kpml-01

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 September 1, 2003.

Copyright Notice

Abstract

Keypad Markup Language (KPML) is a markup language used in conjunction with SIP and HTTP 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. 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

This document describes the Keypad Markup Language, KPML. KPML is a markup [7] that enables "dumb phones" to report on basic user key-press interactions.

This document refers to a "dumb phone" as a user device that does not have a display. Otherwise, it is actually a rather smart device. Most KPML implementations require the user device to be an http [2] client and interpret KPML markup.

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

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.

> NOTE: The spelunking community already took the name KML for their cave data base interchange format.

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 at 697 Hz and 1209 Hz transmitted.

A SIP-signaled [3] network transports end-to-end signaling with RFC2833 [9] packets. In RFC2833, the signaling application inserts RFC2833 named signal packets instead of generating tones. 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. Overview

KPML is a stateless, declarative markup. A KPML document contains a <pattern> tag with a series of <regex> tags. The <regex> tag has a value attribute which is a RFC3015 [4] (H.248) digit map.

> NOTE: We use H.248 digit maps instead of MGCP [13] digit maps because the former is an IETF standard and the latter is not.

> NOTE: We do not use SRGS [14] 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.

KPML 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 document 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 KPML 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 documents. However, as described above, once there is a match, the user device quarantines any keys the user enters subsequent to the match.

KPML documents 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 (http POST or SIP NOTIFY) 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.

Because it is not possible to know if the signaled digits are for local KPML processing or for other recipients of the media stream, the user device transmits the digits to the far end in real time, using either RFC2833 or by generating the appropriate tones.

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 "*".

3. HTTP Reporting

For HTTP reporting, each <regex> tag in the markup has an href attribute. When the user enters keypress(es) that match a <regex> tag, the user device issues an http POST to the URI specified by the href. The body of the POST is a report of the actual digits entered. This is so the user device can indicate what digit string matched a pattern with wildcards.

If the resulting document returned by the http POST is empty, the user device terminates the KPML session.

> NOTE: This is different than the behavior for VoiceXML as described in Basic Network Media Services with SIP [12], where an empty document results in the termination of the session.

If the KPML document includes "sip:" href targets, and the KPML interpreter does not support SIP Reporting, the KPML interpreter MUST reject the document in its entirety at interpretation time with the appropriate SIP error as described in ?????.

> NOTE: draft-jennings-sip-app-info-00.txt should cover document rejection. It does not right now. This draft should not address document rejection, other than the criteria for rejection. This draft focuses on KPML, not on the initiation mechanism.

4. SIP Reporting

For SIP reporting, the href attribute of the <regex> tag MUST be "sip:". When the user enters keypress(es) that match a <regex> tag, the user device will issue a SIP NOTIFY to the Contact of the original INVITE. A KPML interpreter MUST NOT direct the NOTIFY to other SIP endpoints. See the Security Considerations (Section 10) section for the rationale for this restriction.

> The reason one must specify a sip: scheme, and not simply make href optional, is to catch a HTTP-based script error where one forgets to specify the href tag. If href was optional, then this error would result in the user device generating a SIP NOTIFY, which would not be the desired action.

The specification of any scheme-specific part, that is, anything following the colon in "sip:", is an error. The interpreter MUST reject the request.

> NOTE: This greatly simplifies the security issues about who can send a NOTIFY to what dialog. Here we say simply that if someone asks you for service, you can tell them about it. However, you cannot tell someone else about it.

After reporting a SIP <regex>, the interpreter terminates the KPML session. To collect more digits, the requestor must issue a re-INVITE on the dialog.

> 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 [11] or MSCML [10].

If the KPML document includes "http:" href targets, and the KPML interpreter does not support HTTP Reporting, the KPML interpreter MUST reject the document in its entirety at interpretation time with the appropriate SIP error as described in ?????.

> NOTE: draft-jennings-sip-app-info-00.txt should cover document rejection. It does not right now. This draft should not address document rejection, other than the criteria for rejection. This draft focuses on KPML, not on the initiation mechanism.

5. Mixing HTTP and SIP

NOTE: So, now that Pandora's Box is open...

There is nothing to prevent mixing SIP and HTTP reporting requests in the same KPML document. While this may be a benefit, it has definite drawbacks.

The major drawback is that one cannot negotiate SIP-ness or HTTP-ness. As far as the endpoints are concerned it is all KPML. One could use "application/KPML+SIP+XML" and "application/KPML+HTTP+XML", but that is pretty ugly.

6. Examples

6.1 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 DTMF package, is the octothorpe key. In fact, KPML supports all digits, 1-9, *, #, A-D from the H.248 DTMF package.

6.2 VoiceXML Digit Collection

One could imagine a VoiceXML [11] platform that wants to have the user device signal the user's key presses, while the VoiceXML platform still streams prompts to the user device. Of course, by definition, the VoiceXML platform receives all of the user device's media. This is because the user hears prompts from the VoiceXML platform and the platform hears all of the user's utterances (e.g.,

for recording a message).

However, let us say that the VoiceXML platform would like to receive the stimulus in http, rather than in RFC2833. Moreover, the KPML mechanism enables the user device to immediately barge the prompt, saving at least a round-trip-time of latency.

In this example, a VoiceXML script builds a menu. The VoiceXML interpreter has pulls out a grammar definition similar to the following.

```
  <menu>
    <property name="inputmodes" value="dtmf"/>
    <prompt>
      For sports press 1, For weather press 2, For Stargazer
      astrophysics press 3.  To speak to a person press 0.
    </prompt>
    <choice dtmf="1"
            next="http://www.sports.example.com/vxml/start.vxml"/>
    <choice dtmf="2"
            next="http://www.weather.example.com/intro.vxml"/>
    <choice dtmf="3"
            next="http://www.stargazer.example.com/astronews.vxml"/>
    <choice dtmf="0"
            next="http://www.stargazer.example.com/transfer.vxml"/>
  </menu>
```

Figure 2 - VoiceXML Code

A browser could take the code in Figure 2 and make a KPML request similar to that shown in Figure 3.

```
<?xml version="1.0">
  <kpml version="1.0">
    <request>
      <pattern barge="yes">
        <regex value="1"
 href="http://app.carrier.net/vm/sess$143908143j?grx-idname-t1" />
        <regex value="2"
 href="http://app.carrier.net/vm/sess$143908143j?grx-idname-t2" />
        <regex value="3"
 href="http://app.carrier.net/vm/sess$143908143j?grx-idname-t3" />
        <regex value="0"
 href="http://app.carrier.net/vm/sess$143908143j?grx-idname-t4" />
      </pattern>
    </request>
  </kpml>
```

Figure 3 - VoiceXML KPML Example Code

Note the targets of the href's are opaque strings that have meaning only to the VoiceXML platform.

6.3 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="9xxxxxxxxx"
               href="http://app.carrier.net/pp/12?local-number10/>
        <regex value="91xxxxxxxxx"
               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).

6.4 Interactive Digit Collection

This is an example where one would probably be better off using a full scripting language such as VoiceXML or a device control language such as H.248.

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. 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.

6.5 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.

7. Report Body

For HTTP or SIP responses, 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 6.3) 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 will be an attribute to the <response> tag.

8. Formal Syntax

The following syntax specification uses the augmented Data Type Definition (DTD) as described in XML [7].

```
<!ELEMENT kpml (request | response)>
<!ATTLIST kpml version (1.0) #REQUIRED>

<!ELEMENT request (pattern)>

<!ELEMENT pattern (regex)>
<!ATTLIST pattern barge (yes | no) "yes">

<!ELEMENT regex (value | href)>
<!ATTLIST regex
        value CDATA #IMPLIED
        href  CDATA #REQUIRED>

<!ELEMENT response EMPTY>
<!ATTLIST response
        digits CDATA #IMPLIED>
```

Figure 8 - KPML DTD

And, if you prefer XML Schema [5], here it is.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
     <xs:element name="href">
            <xs:complexType/>
     </xs:element>
     <xs:element name="kpml">
            <xs:complexType>
                   <xs:choice>
                          <xs:element ref="request"/>
                          <xs:element ref="response"/>
                   </xs:choice>
                   <xs:attribute name="version" use="required">
                          <xs:simpleType>
                                 <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="1.0"/>
                                 </xs:restriction>
                          </xs:simpleType>
                   </xs:attribute>
            </xs:complexType>
     </xs:element>
     <xs:element name="pattern">
```

```
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="regex"/>
                        </xs:sequence>
                        <xs:attribute name="barge" default="yes">
                                <xs:simpleType>
                                        <xs:restriction base="xs:NMTOKEN">
                                                <xs:enumeration value="yes"/>
                                                <xs:enumeration value="no"/>
                                        </xs:restriction>
                                </xs:simpleType>
                        </xs:attribute>
                </xs:complexType>
        </xs:element>
        <xs:element name="regex">
                <xs:complexType>
                        <xs:choice>
                                <xs:element ref="value"/>
                                <xs:element ref="href"/>
                        </xs:choice>
                        <xs:attribute name="value" type="xs:string"/>
                        <xs:attribute name="href" type="xs:string"
use="required"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="request">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="pattern"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="response">
                <xs:complexType>
                        <xs:attribute name="digits" type="xs:string"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="value">
                <xs:complexType/>
        </xs:element>
   </xs:schema>
```

Figure 9 - XML Schema for KPML

9. IANA Considerations

9.1 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

9.2 Schema Registration

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

10. Security Considerations

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] 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.

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

[4] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and J. Segers, "Megaco Protocol Version 1.0", RFC 3015, November 2000.

[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.

Informative References

[7] 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.

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

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

[10] 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.

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

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

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

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

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, Steve Fisher and Eric Chueng of AT&T Laboratories helped immensely through many conversations and challenges.

Intellectual Property Statement

Full Copyright Statement

Acknowledgement

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