Network Work Group
Internet Draft
Document: draft-burger-sipping-kpml-00.txt
Category: Standards Track
Expires: April 28, 2002

E. Burger
SnowShore Networks, Inc.

October 28, 2002

Keypad Markup Language (KPML)

Status of this Memo

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 digit presses. Note that this document specifies a hypothetical language that has no implementations.

Table of Contents

1. Conventions used in this document

In the narrative discussion, the "device" is a User Agent that will report stimulus. An "endpoint" is the system requesting a device to report stimulus. The "user" is an entity that stimulates the device to report the stimulus. In English, the device is a phone, the endpoint is an application server, and the user presses keys to generate stimulus.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [2].

2. Introduction

This document describes the Keypad Markup Language, KPML. KPML is a markup [3] that enables "dumb phones" to report on basic user key-press interactions.

This document refers to a "dumb phone" as a device that does not have a display. Otherwise, it is actually a rather smart device. KPML requires the device to be an http [4] client and interpret KPML markup.

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 endpoint being signaled, what is important is the fact of the stimulus, 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.

In a SIP-signaled [5] network, the preferred method of transporting end-to-end signaling is RFC 2833 [6]. In RFC 2833, the signaling endpoint inserts RFC 2833 signal packets instead of generating the tones. The receiving endpoint gets the tone information, which is what it wanted in the first place.

RFC 2833 is the "correct" answer for end-to-end signaling. It is the only method that can correlate the time the end user pressed a digit with the user's media. However, for various reasons, people request an out-of-band signaling method.

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

3. 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 RFC 3015 (H.248) [7] digit map.

> NOTE: We use Megaco digit maps instead of MGCP digit maps because the former is an IETF standard, while the latter is proprietary. If we have to play by the rules, we'll play by all of the rules.

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 device will issue a 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 device can indicated what digit string matched a pattern with wildcards.

It is possible that the page returned by the http POST is another

KPML document. In this situation, the device needs to decide what to do with user key presses collected between the time the device posted the last result and the fetch and interpretation of the next KPML document.

For many applications, the device needs to quarantine (buffer) those digits. Some applications use modal interfaces where the first few key presses determine what the following digits mean. For a novice user, the endpoint 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 device buffers digits and applies them immediately when the next KPML document arrives. Disabling barge by specifying "barge=no" means the device flushes any collected digits before collecting more digits and comparing them against the <pattern> tags.

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

Because it is not possible to know if the signaled digits may be for the far end, the device transmits the digits to the far end in real time, using either RFC 2833 or by generating the appropriate tones.

> NOTE: If KPML did not have this behavior, then a 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 device, because it is looking for the "*9" pattern. The user would probably press the "*" key again, hoping that the called IVR system just didn't hear the key press. At that point, the device would send both "*" entries, as "**" does not match "*9". However, that would not have the effect the user intended when they pressed "*".

4. Examples

4.1. 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 1 shows the KPML for long octothorpe.


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

4.2. Interactive Digit Collection

In this example, an application endpoint requests the device to send the user's signaling directly to the platform in HTTP, rather than monitoring the entire RTP stream. Figure 2 shows a voice mail menu,

where presumably the endpoint played a "Press K to keep the message, R to replay the message, and D to delete the message" prompt.

```
<?xml version="1.0">
  <kpml version="1.0">
    <pattern barge=off>
      <regex value="5"
        href="http://app.carrier.net/vm/sess$9awj08asd7?keep" />
      <regex value="7"
        href="http://app.carrier.net/vm/sess$9awj08asd7?replay" />
      <regex value="3"
        href="http://app.carrier.net/vm/sess$9awj08asd7?delete" />
    </pattern>
  </kpml>
```

Figure 2 - IVR KPML Example

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.

4.3. VoiceXML Digit Collection

One could imagine a VoiceXML platform that wants to have the device signal the user's key presses, while the VoiceXML platform still streams prompts to the device. Of course, by definition, the VoiceXML platform receives all of the 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 RFC 2833. KPML can do this, as the following example shows.

NOTE: Clearly I don't believe this is a useful use case. In particular, there is no way to indicate whether a future prompt is non-bargeable.

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 3 - VoiceXML Code


```
<?xml version="1.0">
  <kpml version="1.0">
    <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>
  </kpml>
```

Figure 4 - VoiceXML KPML Example

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

5. Formal Syntax

The following syntax specification uses the augmented Data Type Definition (DTD) as described in XML [3].

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

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

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

6. Security Considerations

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

7. References

1 Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.
INFORMATIVE

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

3 Bray, T. et. al., "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, http://www.w3.org/TR/REC-xml, October 2000.
NORMATIVE

4 Fielding, R. et. al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
INFORMATIVE

5 Rosenberg, J. et. al., "SIP: Session Initiation Protocol", RFC 3261, June 2002.
INFORMATIVE

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

7 Cuervo, F. et. al., "Megaco Protocol Version 1.0", RFC 3015, November 2000.
NORMATIVE

8. 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, Andy Spitzer, and Walter O'Connor were the primary contributors to that effort.

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

9. Acknowledgments

10. Author's Address

Eric Burger
SnowShore Networks, Inc.

285 Billerica Rd.
Chelmsford, MA 01824-4120
USA

E-mail: eburger@snowshore.com
Phone: +1 978/367-8400

Full Copyright Statement

Acknowledgement

The Internet Society currently provides funding for the RFC Editor function.

SnowShore Networks, Inc. is a member of the Internet Society.