The FreeSWITCH Speech Phrase Management architecture provides a consistent framework for the management of language dependent voice prompting without the need to dig into the applications source code. A single application developed using the framework will work with the current languages implemented or new languages in the future.

Contents

- 1 Features
- 2 Overview
- 3 Usage
  - 3.1 Selecting the language
  - 3.2 Playing Prompts from the dialplan
  - 3.3 Playing Prompts from "C" application
  - 3.4 Playing Prompts from JavaScript Application
- 4 Phrases Section Primer
- 5 Action functions
- 6 Dial Plan Samples
- 7 Phrases Section Samples
- 8 Calling a macro from within a macro
- 9 Pitfalls

Features

- Multilingual Support
- No Source Code required to modify prompts
- Ability to select prompts using pattern matching in XML
- Integrated support for voice and TTS in the same application
- Custom phrases can be added at any time
- Switch voice libraries with one setting
- Only load the code for the languages you want to support (less code bloat).

Overview

There are several ways to speak prompts in FreeSWITCH but the Speech Phrase Management sub-system provides the most features and flexibility. Prompts are defined outside the application and can be modified to suit the specific implementation or language. When amounts, dates, numbers, or letters are annunciated the proper phrases to assemble and the ordering of those phrases is determined by the "mod_say_xx" module. Because different languages assemble the same phrases differently (and even use different words depending upon the type of object being referred to), a helper application is needed to do the job properly. This is the job of the mod_say_xx (mod_say_en, mod_say_fr, etc) module. Within this module are the necessary functions speak time, money, counts, spell letters, and digits. In order to support the english version (mod_say_en) the code expects the following prompt directories to exist in your base voice file path (for example "/var/sounds/freeswitch/en").

```
ascii/
```

```
phonetic-ascii/
digits/
time/
currency/
```

View a complete listing of the voice files used by mod_say_en. You can download a sample voice set from http://www.freeswitch.org/eg/en.tar.gz.

Usage

For each language you want to support you will need to load the appropriate "mod_say_xx" module in the modules section.

```
<load module="mod_say_en"/>
```

Also add to freeswitch.xml the following line for each language (example: German "de"):

```
 <X-PRE-PROCESS cmd="include" data="lang/de/*.xml"/>
```

Selecting the language

The language to use is selected by setting the "language" variable to the specific language code you want.

```
<action application="set" data="default_language=en"/>
```

Selects english as the current language to use.

Note: if you specify a specific language to use in the API call it will override the default_language channel variable setting. This is to support prompts that should be spoken in a particular language regardless of the users default language selection.

Playing Prompts from the dialplan

The "phrase" application will call the say API using the phrases defined in the "phrases" section of your freeswitch.xml file.

```
<action application="phrase" data="msgcount:10"/>
<action application="phrase" data="spell-phonetic:abc.012345 6789def#*"/>
<action application="phrase" data="spell:${caller_id_name}"/>
```

The data field passes two parameters:

- Phrase Macro Name to use
- Data to pass to the macro

The macro names are arbitrary but should be meaningful for documentation purposes. The data can be a literal as in the first two examples above or a string variable as in the third example.

The "playback" application can also be used in same way as "phrase" application.

```
<action application="set" data="playback_terminators=#"/>
<action application="playback" data="phrase:demo_ivr_main_menu"/>
<action application="playback" data="phrase:voicemail_message_count:16:new"/>
```

Playing Prompts from "C" application

```
status = switch_ivr_phrase_macro(session, "phrasename", "phrasedata", language, args);
```

Playing Prompts from JavaScript Application

```
function sayphrase(phrase, args)
{
    console_log("sayphrase: phrase=[" + phrase + "] args=[" + args + "]\n");
    var rtn = session.execute("phrase", phrase + "," + args);
    return(rtn);
}

if (session.ready()) {
    session.answer();
    session.execute("sleep","1000");
    sayphrase("msgcount", "10");
    session.hangup();
}
```

Phrases Section Primer

The Phrases section defines the construction and annunciation of phrases in various languages. The format of the XML is as follows;

```
<section name="phrases" description="Speech Phrase Management">
  <macros>
  ...
  </macros>
</section>
```

Defines the start and end of the phrases & macros section. All prompts should be defined in this section. This section is then sub-divided into languages with the "language" tag as follows.

```
<language name="en" sound_path="/var/sounds/phrases/en" tts_engine="cepstral" tts_voice="david">
  ...
</language>
```

- name - The name parameter defines the specific language these prompts belong to ("en" in the example above). This will cause the mod_say_en module to be used to annunciate any constructed phrases (like money, date, time, etc.)
- sound_path - The base path to the voice files for this language.
- tts_engine - The text-to-speech engine to use for any TTS spoken.

- tts_voice - The specific voice to use for TTS.

Within the language there are one or more macros defined.

```
<macro name="msgcount">
  ...
</macro>
```

The macro tag defines a specific macro name. This is the macro that will match the name in the "Phrases" application in the dialplan section.

Within a Macro there are one or more "input" patterns to be tested.

```
<input pattern="(.*)">
  ...
</input>
```

The pattern specified is a PCRE expression that the second parameter (the actual data to speak) of the "Phrases" application will be matched against. Using regex you can filter for specific conditions and even "scrub" the data to insure it is in the proper layout. You may have multiple input patterns and define different prompts for each. Example: Speak "You have 2 messages" versus "You have 1 message".

Note: Within a Macro all input patterns will be tested for possible matches unless the "break" action is used.

Within a "input" tag are one or more "match" tags or "nomatch" tags.

```
<match>
  ...
</match>
<nomatch>
  ...
</nomatch>
```

These define the actions to take if the input pattern is matched (or not matched).

Within a "match" (or "nomatch") tag one or more action tags follow.

```
<action function="execute" data="sleep(1000)"/>
<action function="play-file" data="vm-youhave.wav"/>
<action function="say" data="$1" method="pronounced" type="items"/>
```

These define the specific actions to take when this macro is applied. It usually consist of calling the "say" application passing the parsed data to be spoken. The possible actions are further defined in the section below.

Action functions

- execute - calls the FreeSWITCH execute API (you can execute any other API's)
- play-file - play a specific audio file or play a macro using *phrase:macro_name*
- say - Call the specific "type" say api as below. The method is used to modify the way the data is annunciated (counted, iterated, or pronounced).
    - spoken as general counts
        - number

        - ◊ items
        - ◊ persons
        - ◊ messages
    - ♦ Spoken as times
        - ◊ time_measurement
        - ◊ current_date
        - ◊ current_time
        - ◊ current_date_time
        - ◊ short_date_time
    - ♦ Spoken as an IP address
        - ◊ ip_address
    - ♦ Spelling
        - ◊ name_spelled
        - ◊ name_phonetic
    - ♦ Money related
        - ◊ currency
- speak-text - Speak some text using the TTS engine
- break - Stop parsing any more input patterns.

Dial Plan Samples

The sample dialplan extension below demonstrates speaking a number of the prompts in the "phrases" section.

```
<extension name="556">
  <condition field="destination_number" expression="^556$">
    <action application="answer"/>
    <action application="set" data="call_start_time=$strftime"/>
    <action application="sleep" data="500"/>
    <action application="phrase" data="spell,${caller_id_name}"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="spell-phonetic,abc.012345 6789def#*"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="saymoney,851920.11"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="spell,192.168.0.100"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="ip-addr,66.250.68.194"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="timespec,12:45:15"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="saydate,${strepoch(2006-03-23)}"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="saytime,${strepoch(2006-03-23 01:59)}"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="saydatetime,${strepoch(2006-03-23 12:34)}"/>
    <action application="sleep" data="500"/>
```

```
    <action application="phrase" data="msgcount,10"/>
    <action application="sleep" data="500"/>

    <action application="phrase" data="timeleft,3:30"/>
    <action application="sleep" data="500"/>
  </condition>
</extension>
```

Phrases Section Samples

This sample section defines the prompts to play for the examples above.

```
<section name="phrases" description="Speech Phrase Management">
  <macros>
    <language name="en" sound_path="/var/sounds/phrases/en" tts_engine="cepstral" tts_voice="dav
      <macro name="msgcount">
        <input pattern="(.*)">
          <match>
            <action function="execute" data="sleep(1000)"/>
            <action function="play-file" data="vm-youhave.wav"/>
            <action function="say" data="$1" method="pronounced" type="items"/>
          </match>
        </input>
        <input pattern="^1$">
          <match>
            <action function="play-file" data="vm-message.wav"/>
          </match>
          <nomatch>
            <action function="play-file" data="vm-messages.wav"/>
          </nomatch>
        </input>
      </macro>
      <macro name="saymoney">
        <input pattern="(.*)">
          <match>
            <action function="say" data="$1" method="pronounced" type="currency"/>
          </match>
        </input>
      </macro>
      <macro name="saydate">
        <input pattern="(.*)">
          <match>
            <action function="say" data="$1" method="pronounced" type="current_date"/>
          </match>
        </input>
      </macro>
      <macro name="ip-addr">
        <input pattern="(.*)">
          <match>
            <action function="say" data="$1" method="iterated" type="ip_address"/>
            <action function="say" data="$1" method="pronounced" type="ip_address"/>
          </match>
        </input>
      </macro>
      <macro name="saytime">
        <input pattern="(.*)">
          <match>
            <action function="say" data="$1" method="pronounced" type="current_time"/>
          </match>
```

```
      </input>
    </macro>
    <macro name="saydatetime">
      <input pattern="(.*)">
        <match>
          <action function="say" data="$1" method="pronounced" type="current_date_time"/>
        </match>
      </input>
    </macro>
    <macro name="timespec">
      <input pattern="(.*)">
        <match>
          <action function="say" data="$1" method="pronounced" type="time_measurement"/>
        </match>
      </input>
    </macro>
    <macro name="spell">
      <input pattern="(.*)">
        <match>
          <action function="say" data="$1" method="pronounced" type="name_spelled"/>
        </match>
      </input>
    </macro>
    <macro name="spell-phonetic">
      <input pattern="(.*)">
        <match>
          <action function="say" data="$1" method="pronounced" type="name_phonetic"/>
        </match>
      </input>
    </macro>
    <macro name="timeleft">
      <input pattern="(\d+):(\d+)">
        <match>
          <action function="say" data="$1:$2" method="pronounced" type="time_measurement"/>
        </match>
      </input>
    </macro>
    <macro name="tts-timeleft">
      <input pattern="(\d+):(\d+)">
        <match>
          <action function="speak-text" data="You have $1 minutes, $2 seconds remaining $strft
        </match>
        <nomatch>
          <action function="speak-text" data="That input was invalid."/>
        </nomatch>
      </input>
      <input pattern="(\d+) min (\d+) sec">
        <match>
          <action function="speak-text" data="You have $1 minutes, $2 seconds remaining $strft
          <action function="break"/>
        </match>
        <nomatch>
          <action function="speak-text" data="That input was invalid."/>
        </nomatch>
      </input>
    </macro>
  </language>
  <language name="fr" sound_path="/var/sounds/lang/fr/jean" tts_engine="cepstral" tts_voice="j
    <macro name="msgcount">
      <input pattern="(.*)">
        <match>
          <action function="play-file" data="tuas.wav"/>
```

```
            <action function="say" data="$1" method="pronounced" type="items"/>
            <action function="play-file" data="messages.wav"/>
          </match>
        </input>
      </macro>
      <macro name="timeleft">
        <input pattern="(\d+):(\d+)">
          <match>
            <action function="speak-text" data="il y a $1 minutes et de $2 secondes de restant"/
          </match>
        </input>
      </macro>
    </language>
  </macros>
</section>
```

Calling a macro from within a macro

```
<macro name="main_menu" pause="100">
  <input pattern="(.*)">
    <match>
      <action function="speak-text" data="Welcome to the FreeSWITCH System."/>
      <action function="play-file" data="phrase:main_menu_short"/>
    </match>
  </input>
</macro>
<macro name="main_menu_short" pause="100">
  <input pattern="(.*)">
    <match>
      <action function="speak-text" data="For English press 1."/>
      <action function="speak-text" data="To speak to the operator press 0."/>
    </match>
  </input>
</macro>
```

Pitfalls

I used the following for German prompts conf/lang/de/de.xml

```
 <include>
   <language name="de" sound-path="$${base_dir}/sounds/de/de/callie" tts-engine="cepstral" tts-v
     <X-PRE-PROCESS cmd="include" data="demo/demo.xml"/>
     <!--voicemail_de_tts is purely implemented with tts, we need a files based implementation t
     <!-- <X-PRE-PROCESS cmd="include" data="vm/tts.xml"/> -->
     <X-PRE-PROCESS cmd="include" data="vm/sounds.xml"/>  <!-- vm/tts.xml if you want to use tts
     <X-PRE-PROCESS cmd="include" data="dir/sounds.xml"/>  <!-- dir/tts.xml if you want to use t
   </language>
 </include>
```

Although <X-PRE-PROCESS cmd="include" data="vm/tts.xml"/> is commented, TTS is thus being used. So delete this line completely if you need voice prompts to be played as sound files.