# Speech Phrase Management

## 0. About

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.

## 1. 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).

## 2. Overview

There are several ways to speak prompts in FreeSWITCH, but the Speech Phrase Management sub-system provides the most features and flexibility.

### 2.1 Language modules

Prompts are defined outside the application and can be modified to suit the specific implementation or language. When amounts, dates, numbers, or letters are enunciated, the proper phrases to assemble and the ordering of those phrases is determined by the **`mod_say_xx`** module (where `xx` stands for a two-letter language code, such as `en` ).

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` (e.g., `mod_say_en`, `mod_say_fr`) 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 certain prompt directories to exist in your base voice file path (for example at `/var/sounds/freeswitch/en`; see `sounds_dir` and `sound_prefix` in Global Variables).

Basic sounds should be installed during a vanilla install (at `/usr/share/freeswitch/sounds` on Debian 9), but, just in case, here are all the available sounds:

https://github.com/access-news/freeswitch-sounds

### 2.2 Configuration

#### 2.2.1 Load language modules

For each language you want to support you will need to load the appropriate `mod_say_xx` module in `conf/autoload_configs/modules.conf.xml`. (See Modules.)

| Language Activation |
| --- |
| `<load module="mod_say_en"/>` |

### 2.3 Specify phrase directories

See **3 Phrase primer** section below to read more on phrases.

Also specify the location of language-specific phrase directories for each language in `conf/freeswitch.xml` (e.g., "de" for German):

**Language Configuration**

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

See **6. Configuration files** in Default Configuration.

# 3. Phrase primer

The `phrases` section in `conf/freeswitch.xml` defines the construction and enunciation of phrases in various languages.

TODO By Tomas Bajarunas:

add optional name attribute for macros:

<macros name="optional_macros_name">

...

</macros>

I think that name later can be used when using phrases like this in dialplan:

<action application="playback" data="phrase:MyPhrase@optional_macros_name" />

from other phrases:

<action function="phrase" phrase="MyPhrase@optional_macro_name" data="some:data" />

## 3.1 Phrase macros

### 3.1.1 `macros` tag

The following XML snippet illustrates the structure to define phrase macros:

**Phrases**

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

All prompts should be defined in this section.

### 3.1.2 `language` tag

The <macros> section is then sub-divided into languages as follows.

**Language tags**

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

Where

- **`name`** - *Defines the specific language these prompts belong to.*
  In the above example it is `en` , that will cause the `mod_say_en` module to be used to enunciate 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.*

> See TTS page for available engines and voices.

### 3.1.3 `macro` tag

Within the language there are one or more macros defined:

**Macros**

```
<macro name="msgcount">
  <!-- inputs -->
</macro>
```

### 3.1.4 `input` tag

**Macros**

```
<macro name="msgcount">
  <input pattern="^\d+$">
    <!-- match and nomatch tags -->
  </input>
</macro>
```

`pattern` is a PCRE-compatible regular expression to match on the second argument to the `phrase` application (i.e., the actual data to speak).

For example, using the example below, the above macro pattern will match "130".

```
    <action application="phrase" data="msgcount,130"/>
```

Using regexes, you can filter for specific conditions, and even "scrub" the data to ensure it is in the proper layout.

> Within `macro` all `input` patterns will be tested for possible matches, unless the **`break` action** is used.
>
> See **3.2 Phrase macro actions** section below.

#### 3.1.4.1 Example

To achieve proper pluralization, you may define multiple `input` patterns, and use different prompts for each, such as "You have 2 messages" versus "You have 1 message".

**Macros**

```
<macro name="msgcount">
  <input pattern="^\d+$">
    <!-- ... plural prompt ... -->
  </input>
  <input pattern="^\d$">
    <!-- ... singular prompt ... -->
  </input>
</macro>
```

### 3.1.5 `match` and `nomatch` tags

Within a `input` tag there are one or more `match` and `nomatch` tags.

**Macros**

```
<macro name="msgcount">
  <input pattern="^\d+$">
    <match>
      <!-- actions -->
    </match>
    <nomatch>
      <!-- actions -->
    </nomatch>
  </input>
</macro>
```

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

#### 3.1.5.1 Example

```
<macro name="tts-timeleft">
  <input pattern="(\d+):(\d+)">
    <match>
      <!-- Speak the time in the format: -->
      <action function="speak-text" data="You have $1 minutes, $2 seconds remaining $strftime(%Y-%m-%d)"/>
    </match>
    <nomatch>
      <!-- The input wasn't in the format of 12:34 (or similar), hence: -->
      <action function="speak-text" data="That input was invalid."/>
    </nomatch>
  </input>
</macro>
```

3.1.6 `action` tag

Within a `match` and `nomatch` tag there are one or more actions.

**Matches and No matches**

```
<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 described in **3.2 Phrase macro actions** section below.

## 3.2 Phrase macro actions

**Syntax**

```
<action function=[phrase_macro_action] data=[arguments] [other_properties] />
```

Where `phrase_macro_action` can be:

| `phrase_macro_action` | Description |
|---|---|
| `execute` | Calls the FreeSWITCH `execute` API (you can execute any other API's).<br>TODO What is the `execute` API? |
| `play-file` | Play a specific audio file or play a macro in the form `phrase:macro_name` |

| | |
|---|---|
| `say` | Use the pre-recorded sound files to read or say various things like dates, times, digits, etc.<br><br>Requires the `type` and `method` properties whose values correspond to the `say_type` and `say_method` input values of `mod_dptools:say`. See **1. Syntax** section there. |
| `speak-text` | Speak some text using the TTS engine. |
| `break` | Stop parsing any more input patterns.<br><br>See **3.1.4 `input` tag** section. |

# 3. Usage

## 3.1 From XML Dialplan

### 3.1.1 Selecting the language

The language to use is selected by setting the `default_language` variable (see Channel Variables Catalog) to the specific language code you want.

**Language selection**

```
<!-- select English as the default language -->
<action application="set" data="default_language=en"/>
```

> **If you specify a specific language to use in the API call (see below methods), 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.

### 3.1.2 Playing prompts from the dialplan

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

**Invoke phrases in the Dialplan**

```
<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
  The macro names are arbitrary but should be meaningful for documentation purposes.
- **data** (i.e., arguments) to pass to the macro
  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.
>
> **Example using "playback"**
>
> ```
> <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"/>
> ```

## 3.2 Playing prompts from a C application

**Prompts form "C"**

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

### 3.3 Playing prompts from JavaScript application

**Prompts from JavaScript**

```
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();
}
```

## 4. Examples

### 4.1 Speaking a number

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

**Dialplan**

```
<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>
```

**Phrase macro definition for the above dialplan**

```
<section name="phrases" description="Speech Phrase Management">
   <macros>
     <language name="en" sound_path="/var/sounds/phrases/en" tts_engine="cepstral" tts_voice="david">
       <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 $strftime(%Y-%m-%d)"
/>
          </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 $strftime(%Y-%m-%d)"
/>
            <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="jean-pierre">
      <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>
```

### 4.2 Calling a macro from within a macro

**Calling a macro from 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>
```

## 5. Play as Sound Files

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

**Comment pre-process lines**

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

The FreeSWITCH parser will ignore comment strings for <X-PRE-PROCESS... lines, so to prevent the parser from reading the line at FS start add another character thus
<YX-PRE-PROCESS cmd="include" data="vm/tts.xml"/>
if you need voice prompts to be played as sound files.

## 6. See Also

- mod_dptools: phrase