UniMRCP

FreeSWITCH

AWS Lex and Polly

Usage Guide

Revision: 1

Created: September 28, 2018

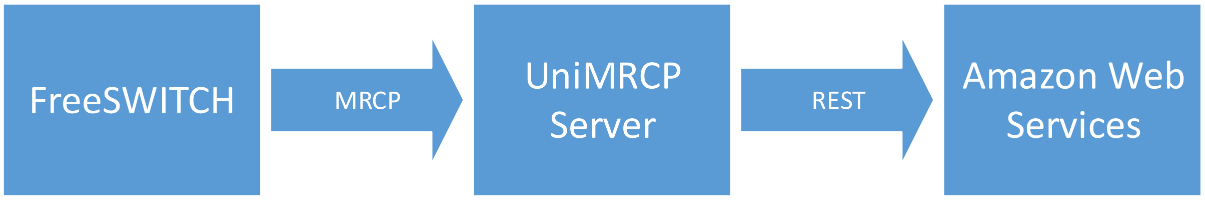
Last updated: September 28, 2018

Author: Arsen Chaloyan

Table of Contents

1 Overview

This guide describes how to utilize the Amazon Lex and Polly services with FreeSWITCH.



Note that the FreeSWITCH and the UniMRCP server typically reside on different hosts in a LAN, although both might be installed on the same host.

Installation of the FreeSWITCH and the UniMRCP server with the Lex and/or Polly plugins is not covered in this document. Visit the corresponding web pages for more information.

https://freeswitch.org/confluence/display/FREESWITCH/mod_unimrcp
http://unimrcp.org/lex
http://unimrcp.org/polly

1.1 Applicable Versions

Instructions provided in this guide are applicable to the following versions.

FreeSWITCH 1.4 and above

UniMRCP Lex Plugin 1.0.0 and above

UniMRCP Polly Plugin 1.0.0 and above

2 UniMRCP Module

2.1 Overview

The module *mod_unimrcp.so* provides an implementation of the ASR and TTS interfaces of FreeSWITCH, based on the UniMRCP client library.

2.2 Configuration Steps

This section outlines major configuration steps required for use of the module *mod_unimrcp.so* with the UniMRCP server.

Create a new MRCP profile (or modify an existing one) in the configuration directory *mrcp_profiles* of FreeSWITCH. In the following example, the FreeSWITCH/UniMRCP client is located on 10.0.0.1 and the UniMRCP server is on 10.0.0.2.

```
<include>
  <!-- UniMRCP Server MRCPv2 -->
  <profile name="uni2" version="2">
    <!--param name="client-ext-ip" value="auto"-->
    <param name="client-ip" value="10.0.0.1"/>
    <param name="client-port" value="16090"/>
    <param name="server-ip" value="10.0.0.2"/>
    <param name="server-port" value="8060"/>
    <!--param name="force-destination" value="1"/-->
    <param name="sip-transport" value="udp"/>
    <!--param name="ua-name" value="FreeSWITCH"/-->
    <!--param name="sdp-origin" value="FreeSWITCH"/-->
    <!--param name="rtp-ext-ip" value="auto"/-->
    <param name="rtp-ip" value="auto"/>
    <param name="rtp-port-min" value="14000"/>
    <param name="rtp-port-max" value="15000"/>
    <!-- enable/disable rtcp support -->
    <param name="rtcp" value="0"/>
    <!-- rtcp bye policies (rtcp must be enabled first)
           0 - disable rtcp bye
           1 - send rtcp bye at the end of session
           2 - send rtcp bye also at the end of each talkspurt (input)
    -->
    <param name="rtcp-bye" value="2"/>
    <!-- rtcp transmission interval in msec (set 0 to disable) -->
    <param name="rtcp-tx-interval" value="5000"/>
    <!-- period (timeout) to check for new rtcp messages in msec (set 0 to disable) -->
    <param name="rtcp-rx-resolution" value="1000"/>
    <!--param name="playout-delay" value="50"/-->
    <!--param name="max-playout-delay" value="200"/-->
```

```
        <!--param name="ptime" value="20"/-->
        <param name="codecs" value="PCMU PCMA L16/96/8000"/>

        <!-- Add any default MRCP params for SPEAK requests here -->
        <synthparams>
        </synthparams>

        <!-- Add any default MRCP params for RECOGNIZE requests here -->
        <recogparams>
          <!--param name="start-input-timers" value="false"/-->
        </recogparams>
      </profile>
    </include>
```

2.3 Usage Examples

Speech Synthesis

Use the speak application for synthesis.

```
<action application="speak" data="unimrcp:uni2|Ivy|Welcome to FreeSWITCH"/>
```

Place a test call and listen to the synthesized message.

Chat Bot

Lex is a conversational engine, which typically requires multiple interactions with the caller until a dialog is complete.
The following JavaScript application demonstrates a basic flow applicable to an ordinary Lex chat bot, which needs to be set up separately through the AWS Console. The application also requires the results to be returned in JSON format, which must be configured in umslex.xml.

```
// Profile to use
var profile = "unimrcp:uni2";
// Compose initial prompt command
var promptCmd = "say: Welcome to Amazon Lex. How can I help you?";
// Compose recognition command
var recogCmd = " detect:" + profile + " {start-input-timers=false,define-grammar=false,no-input-timeout=10000}builtin:speech/transcribe";
// Compose final message
var finalMessage = "Thank you. See you next time";

// Main recognition loop
while (session.ready()) {
```

```
                var command = promptCmd + recogCmd;
                // Play and detect speech
                session.execute("play_and_detect_speech", command);

                // Retrieve result
                var jsonResult = session.getVariable('detect_speech_result');
                if (typeof jsonResult != "undefined") {
                        if (jsonResult.indexOf("Completion-Cause:") == -1) {
                                // Parse JSON result
                                var result = JSON.parse(jsonResult);
                                if (typeof result != "undefined") {
                                        if (typeof result.transcript != "undefined") {
                                                // Compose next prompt command
                                                promptCmd = "say: " + result.message;
                                        }
                                        // Check dialog state
                                        var state = result.dialogstate;
                                        if (typeof state != "undefined") {
                                                if (state == "ReadyForFulfillment") {
                                                        break;
                                                }
                                        }
                                }
                        }
                        else {
                                consoleLog("INFO", "Recognition completed abnormally!\n");
                        }
                }
                else {
                        consoleLog("INFO", "No result!\n");
                        break;
                }
        }

        // Speak the final message
        session.speak(profile, "", finalMessage);
```

Assuming the provided JavaScript application is named lex.js and located in the script directory of FreeSWITCH, use the following dialplan to invoke the script.

```
        <action application="answer"/>
        <action application="set" data="tts_engine=unimrcp:uni2"/>
        <action application="javascript" data="lex.js"/>
```

Place a test call and follow the prompts until the dialog is complete.