Powered by Universal Speech Solutions LLC

UniMRCP

Plugin Implementation Manual

Developer Guide

Revision: 49

Last Updated: May 20, 2017

Created by: Arsen Chaloyan

Table of Contents

1 Overview

This guide describes how to implement an MRCP resource engine plugin into the UniMRCP server. The document provides basic steps only and is not a complete reference.
The intended audience is speech application developers familiar with the C/C++ programming languages.

1.1 Applicable Versions

Unless explicitly stated, instructions provided in this guide are applicable to the following versions.

UniMRCP 1.0.0 and above

1.2 Installation and Configuration

For the installation and configuration of the library, see the Installation Guide and the Server Configuration Guide.

2 Architecture

The resource engine plugin is a run-time loadable library which implements the resource engine interface by using a 3-rd party ASR, TTS, or SVI engine. The resource engine interface is implemented based on the asynchronous event-driven model and is consistent for MRCP v1 and v2.

3 Structure

The resource engine plugin must declare its version number by using the following helper macro.

```
MRCP_PLUGIN_VERSION_DECLARE
```

The plugin may declare the logging routine of the server by declaring the following macro.

```
MRCP_PLUGIN_LOGGER_IMPLEMENT
```

The plugin must implement the creator function of the resource engine, which is the entry point for the run-time loadable library.

```
MRCP_PLUGIN_DECLARE(mrcp_engine_t*) mrcp_plugin_create(apr_pool_t *pool)
```

Note that the above function must be declared with the exact signature and the name in order to be found and loaded dynamically at run-time.

4 Interfaces

The plugin must implement the following interfaces.

4.1 The resource engine interface

The resource engine is an aggregation of the resource channels. The resource engine is created upon plugin creation and gets destroyed when the plugin is unloaded. The following methods of the resource engine need to be implemented in the plugin.

```
/** Table of MRCP engine virtual methods */
struct mrcp_engine_method_vtable_t {
	/** Virtual destroy */
	apt_bool_t (*destroy)(mrcp_engine_t *engine);
	/** Virtual open */
	apt_bool_t (*open)(mrcp_engine_t *engine);
	/** Virtual close */
	apt_bool_t (*close)(mrcp_engine_t *engine);
	/** Virtual channel create */
	mrcp_engine_channel_t* (*create_channel)(mrcp_engine_t *engine, apr_pool_t *pool);
};
```

4.2 The resource channel interface

The resource channel is created in the scope of the MRCP session and gets destroyed with the session termination. The following methods of the resource channel need to be implemented in the plugin.

```
/** Table of channel virtual methods */
struct mrcp_engine_channel_method_vtable_t {
	/** Virtual destroy */
	apt_bool_t (*destroy)(mrcp_engine_channel_t *channel);
	/** Virtual open */
	apt_bool_t (*open)(mrcp_engine_channel_t *channel);
	/** Virtual close */
	apt_bool_t (*close)(mrcp_engine_channel_t *channel);
```

```
	/** Virtual process_request */
	apt_bool_t (*process_request)(mrcp_engine_channel_t *channel, mrcp_message_t *request);
};
```

Note that every single request received from the server MUST be acknowledged by one and only one response sent by the plugin.
The methods of the engine channel MUST not block or hold the executable context.

4.3 The audio stream interface

The audio stream interface needs to be implemented in order to process audio data, for example, from the server to the ASR engine or, in reverse direction, from the TTS engine to the server.

```
/** Table of audio stream virtual methods */
struct mpf_audio_stream_vtable_t {
	/** Virtual destroy method */
	apt_bool_t (*destroy)(mpf_audio_stream_t *stream);

	/** Virtual open receiver method */
	apt_bool_t (*open_rx)(mpf_audio_stream_t *stream, mpf_codec_t *codec);
	/** Virtual close receiver method */
	apt_bool_t (*close_rx)(mpf_audio_stream_t *stream);
	/** Virtual read frame method */
	apt_bool_t (*read_frame)(mpf_audio_stream_t *stream, mpf_frame_t *frame);

	/** Virtual open transmitter method */
	apt_bool_t (*open_tx)(mpf_audio_stream_t *stream, mpf_codec_t *codec);
	/** Virtual close transmitter method */
	apt_bool_t (*close_tx)(mpf_audio_stream_t *stream);
	/** Virtual write frame method */
	apt_bool_t (*write_frame)(mpf_audio_stream_t *stream, const mpf_frame_t *frame);

	/** Virtual trace method */
	void (*trace)(mpf_audio_stream_t *stream, mpf_stream_direction_e direction, apt_text_stream_t *output);
};
```

The methods of the audio stream MUST not block or hold the executable contetx.

5 References

5.1 Implementation

- demo-synth – a simulated synthesizer plugin built for demonstration purposes only
- demo-recog – a simulated recognizer plugin built for demonstration purposes only
- demo-verifier – a simulated verifier plugin built for demonstration purposes only
- mrcp-recorder – a recorder plugin built for demonstration purposes only

5.2 Documentation

- UML Design Concepts – hierarchy, activity and sequence diagrams of the client stack
- API Reference – a documentation generated by Doxygen from the source code