Powered by Universal Speech Solutions LLC

UniMRCP

MRCP Message Usage

Developer Guide

Revision: 44

Last updated: May 20, 2017

Created by: Arsen Chaloyan

Table of Contents

1 Overview

This guide explains the MRCP message format and also describes related API functions.
The intended audience is speech application and server plugin 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

2 MRCP Message Format

The MRCP message consists of a start-line, a header section, and an optional message body.

```
mrcp-message = start-line
               header-section
               [body]
```

Based on the start line, an MRCP message can be either

- a request sent from the client to the server (MRCP_MESSAGE_TYPE_REQUEST)
- a response sent from the server to the client (MRCP_MESSAGE_TYPE_RESPONSE)
- an asynchronous event sent from the server to the client (MRCP_MESSAGE_TYPE_EVENT)

```
start-line = request-line / response-line / event-line
```

The header section is a collection of generic and resource specific header fields.

```
header-section = 1*(generic-header / resource-header)
resource-header = recognizer-header / synthesizer-header / recorder-header
```

The message body contains resource specific and message specific data.

3 Creating an MRCP Message

There are a number of functions for creation of MRCP requests, responses and events. Typically, the function mrcp_application_message_create() should be used for creation of MRCP messages in a user client application. This function creates a request message and initializes it based on the specified session and channel parameters.
The functions mrcp_response_create() and mrcp_event_create() should be used in server plugins for creation of MRCP responses and events respectively.

```
/**
 * Create an MRCP message.
 * @param pool the pool to allocate memory from
 */
MRCP_DECLARE(mrcp_message_t*) mrcp_message_create(apr_pool_t *pool);

/**
 * Create an MRCP request message.
 * @param resource the MRCP resource to use
 * @param resource the MRCP version to use
 * @param method_id the MRCP resource specific method identifier
 * @param pool the pool to allocate memory from
 */
MRCP_DECLARE(mrcp_message_t*) mrcp_request_create(const mrcp_resource_t
*resource, mrcp_version_e version, mrcp_method_id method_id, apr_pool_t *pool);

/**
 * Create an MRCP response message based on given request message.
 * @param request_message the MRCP request message to create a response for
 * @param pool the pool to allocate memory from
 */
MRCP_DECLARE(mrcp_message_t*) mrcp_response_create(const mrcp_message_t
*request_message, apr_pool_t *pool);

/**
 * Create an MRCP event message based on given requuest message.
 * @param request_message the MRCP request message to create an event for
 * @param event_id the MRCP resource specific event identifier
 * @param pool the pool to allocate memory from
 */
MRCP_DECLARE(mrcp_message_t*) mrcp_event_create(const mrcp_message_t
*request_message, mrcp_method_id event_id, apr_pool_t *pool);


/**
 * Create an MRCP request message.
 * @param session the session
 * @param channel the control channel
```

```
 * @param method_id the method identifier of MRCP message
 */
MRCP_DECLARE(mrcp_message_t*) mrcp_application_message_create(mrcp_session_t
*session, mrcp_channel_t *channel, mrcp_method_id method_id);
```

4 Operating on Header Fields

4.1 Setting the generic header fields using numeric identifiers

Get the generic header.

```
mrcp_generic_header_t *generic_header = mrcp_generic_header_get(message);
```

Set the Content-Type header field.

```
apt_string_assign(&generic_header->content_type,"application/synthesis+ssml",message->pool);
mrcp_generic_header_property_add(message,GENERIC_HEADER_CONTENT_TYPE);
```

Set the Content-Id header field.

```
apt_string_assign(&generic_header->content_id,"content-1",message->pool);
mrcp_generic_header_property_add(message,GENERIC_HEADER_CONTENT_ID);
```

Other header fields can be set similarly.

4.2 Getting the generic header fields using numeric identifiers

Get the generic header.

```
mrcp_generic_header_t *generic_header = mrcp_generic_header_get(message);
```

Check whether the Content-Type header field is set. If so, reference the value.

```
if(mrcp_generic_header_property_check(message,GENERIC_HEADER_CONTENT_TYPE) == TRUE) {
    generic_header->content_type
}
```

Check whether the Content-Id header field is set. If so, reference the value.

```
if(mrcp_generic_header_property_check(message,GENERIC_HEADER_CONTENT_ID) ==
TRUE) {
    generic_header->content_id
}
```

Other header fields can be referenced similarly.

4.3 Setting the resource header fields using numeric identifiers

Get the resource header.

```
mrcp_synth_header_t *synth_header = (mrcp_synth_header_t*)
mrcp_resource_header_get(message);
```

Note that the synthesizer header is used in this example. Other resource headers can be used instead in a similar manner.

Set the Voice-Age header field.

```
synth_header->voice_param.age = 28;
mrcp_resource_header_property_add(message,SYNTHESIZER_HEADER_VOICE_AGE);
```

Set the Speaker-Profile header field.

```
apt_string_assign(&synth_header->speaker_profile,"profile-1",message->pool);
mrcp_resource_header_property_add(message,SYNTHESIZER_HEADER_SPEAKER_PROFILE)
;
```

Other resource header fields can be set similarly.

4.4 Getting the resource header fields using numeric identifiers

Get the resource header.

```
mrcp_synth_header_t *synth_header = (mrcp_synth_header_t*)
mrcp_resource_header_get(message);
```

Check whether the Voice-Age header field is set. If so, reference the value.

```
if(mrcp_resource_header_property_check(message,SYNTHESIZER_HEADER_VOICE_AGE)
```

```
== TRUE) {
    synth_header->voice_param.age
}
```

Check whether the Speaker-Profile header field is set. If so, reference the value.

```
if(mrcp_resource_header_property_check(message,SYNTHESIZER_HEADER_SPEAKER_PRO
FILE) == TRUE) {
    synth_header->speaker_profile
}
```

Other resource header fields can be referenced similarly.

4.5 Setting header fields transparently using name-value pairs

1. Create a header field

There are a number of functions intended for creation of header fields. One function creates a header field using a name-value apt_str_t pair. The other creates a header field using given a name-value C string pair, and yet another one creates a header field by parsing a text line containing a name-value pair.

```
/**
 * Create a header field using given name and value APT strings.
 * @param name the name of the header field
 * @param name the value of the header field
 * @param pool the pool to allocate memory from
 */
APT_DECLARE(apt_header_field_t*) apt_header_field_create(const apt_str_t *name, const
apt_str_t *value, apr_pool_t *pool);
```

Example:

```
apt_str_t name, value;
apt_header_field_t *header_field;
apt_string_set(&name,"Content-Type");
apt_string_set(&value,"application/synthesis+ssml");
header_field = apt_header_field_create(&name,&value,message->pool);
```

```
/**
 * Create a header field using given name and value C strings.
 * @param name the name of the header field
 * @param name the value of the header field
```

```
 * @param pool the pool to allocate memory from
 */
APT_DECLARE(apt_header_field_t*) apt_header_field_create_c(const char *name, const char *value, apr_pool_t *pool);
```

Example:

```
apt header field t *header field = apt header field create c("Content-Type","application/synthesis+ssml",message->pool);
```

```
/**
 * Create a header field from entire line consisting of a name and value pair.
 * @param line the line, which consists of a name and value pair
 * @param separator the name and value separator
 * @param pool the pool to allocate memory from
 */
APT_DECLARE(apt_header_field_t*) apt_header_field_create_from_line(const apt_str_t *line, char separator, apr_pool_t *pool);
```

Example:

```
apt_str_t line;
apt_header_field_t *header_field;
apt_string_set(&line,"Content-Type=application/synthesis+ssml");
header_field = apt_header_field_create_from_line(&line,'=',message->pool);
```

2. Add the created header field to the message header.

```
mrcp_message_header_field_add(message,header_field);
```

4.6 Iterating on header fields

The header fields are stored in a ring (apr_ring_t), which, in its turn, is a generic double-linked list. The following code demonstrates how to transparently iterate on the header fields of the specified message.

```
apt_header_field_t *header_field = NULL;
while( (header_field = mrcp_message_next_header_field_get(message,header_field)) != NULL ) {
  printf("%s: %s\n", header_field->name.buf, header_field->value.buf);
}
```

5 Setting the Message Body

Typically, the message body is allocated from the message pool and has the same lifetime as the message itself.

```
apt_string_assign(&message->body,"Hello world",message->pool);
```