Network Working Group
Request for Comments: 4463
Category: Informational

S. Shanmugham
Cisco Systems, Inc.
P. Monaco
Nuance Communications
B. Eberman
Speechworks Inc.

April 2006

A Media Resource Control Protocol (MRCP) Developed by Cisco, Nuance, and Speechworks

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

IESG Note

This RFC is not a candidate for any level of Internet Standard. The IETF disclaims any knowledge of the fitness of this RFC for any purpose and in particular notes that the decision to publish is not based on IETF review for such things as security, congestion control, or inappropriate interaction with deployed protocols. The RFC Editor has chosen to publish this document at its discretion. Readers of this document should exercise caution in evaluating its value for implementation and deployment. See RFC 3932 for more information.

Note that this document uses a MIME type 'application/mrcp' which has not been registered with the IANA, and is therefore not recognized as a standard IETF MIME type. The historical value of this document as an ancestor to ongoing standardization in this space, however, makes the publication of this document meaningful.

Abstract

This document describes a Media Resource Control Protocol (MRCP) that was developed jointly by Cisco Systems, Inc., Nuance Communications, and Speechworks, Inc. It is published as an RFC as input for further IETF development in this area.

MRCP controls media service resources like speech synthesizers, recognizers, signal generators, signal detectors, fax servers, etc., over a network. This protocol is designed to work with streaming protocols like RTSP (Real Time Streaming Protocol) or SIP (Session Initiation Protocol), which help establish control connections to external media streaming devices, and media delivery mechanisms like RTP (Real Time Protocol).

Table of Contents

1. Introduction

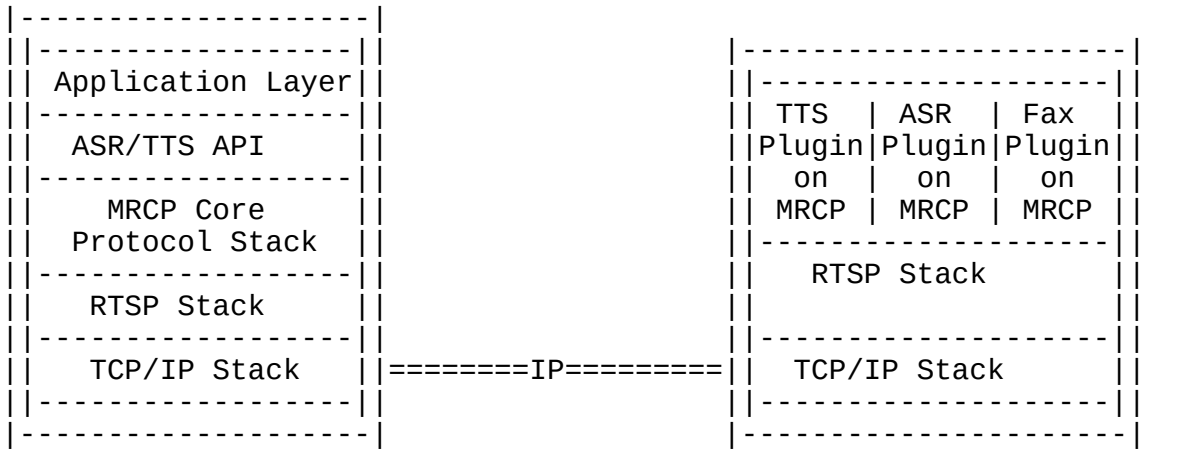
The Media Resource Control Protocol (MRCP) is designed to provide a mechanism for a client device requiring audio/video stream processing to control processing resources on the network. These media processing resources may be speech recognizers (a.k.a. Automatic-Speech-Recognition (ASR) engines), speech synthesizers (a.k.a. Text-To-Speech (TTS) engines), fax, signal detectors, etc. MRCP allows implementation of distributed Interactive Voice Response platforms, for example VoiceXML [6] interpreters. The MRCP protocol defines the requests, responses, and events needed to control the media processing resources. The MRCP protocol defines the state machine for each resource and the required state transitions for each request and server-generated event.

The MRCP protocol does not address how the control session is established with the server and relies on the Real Time Streaming Protocol (RTSP) [2] to establish and maintain the session. The session control protocol is also responsible for establishing the media connection from the client to the network server. The MRCP protocol and its messaging is designed to be carried over RTSP or another protocol as a MIME-type similar to the Session Description Protocol (SDP) [5].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [8].

2. Architecture

The system consists of a client that requires media streams generated or needs media streams processed and a server that has the resources or devices to process or generate the streams. The client establishes a control session with the server for media processing using a protocol such as RTSP. This will also set up and establish the RTP stream between the client and the server or another RTP endpoint. Each resource needed in processing or generating the stream is addressed or referred to by a URL. The client can now use MRCP messages to control the media resources and affect how they process or generate the media stream.



2.1. Resources and Services

The server is set up to offer a certain set of resources and services to the client. These resources are of 3 types.

Transmission Resources

These are resources that are capable of generating real-time streams, like signal generators that generate tones and sounds of certain frequencies and patterns, and speech synthesizers that generate spoken audio streams, etc.

Reception Resources

These are resources that receive and process streaming data like signal detectors and speech recognizers.

Dual Mode Resources

These are resources that both send and receive data like a fax resource, capable of sending or receiving fax through a two-way RTP stream.

2.2. Server and Resource Addressing

The server as a whole is addressed using a container URL, and the individual resources the server has to offer are reached by individual resource URLs within the container URL.

RTSP Example:

A media server or container URL like,

```
rtsp://mediaserver.com/media/
```

may contain one or more resource URLs of the form,

```
rtsp://mediaserver.com/media/speechrecognizer/
rtsp://mediaserver.com/media/speechsynthesizer/
rtsp://mediaserver.com/media/fax/
```

3. MRCP Protocol Basics

The message format for MRCP is text based, with mechanisms to carry embedded binary data. This allows data like recognition grammars, recognition results, synthesizer speech markup, etc., to be carried in the MRCP message between the client and the server resource. The protocol does not address session control management, media management, reliable sequencing, and delivery or server or resource addressing. These are left to a protocol like SIP or RTSP. MRCP addresses the issue of controlling and communicating with the resource processing the stream, and defines the requests, responses, and events needed to do that.

3.1. Establishing Control Session and Media Streams

The control session between the client and the server is established using a protocol like RTSP. This protocol will also set up the appropriate RTP streams between the server and the client, allocating ports and setting up transport parameters as needed. Each control

session is identified by a unique session-id. The format, usage, and life cycle of the session-id is in accordance with the RTSP protocol. The resources within the session are addressed by the individual resource URLs.

The MRCP protocol is designed to work with and tunnel through another protocol like RTSP, and augment its capabilities. MRCP relies on RTSP headers for sequencing, reliability, and addressing to make sure that messages get delivered reliably and in the correct order and to the right resource. The MRCP messages are carried in the RTSP message body. The media server delivers the MRCP message to the appropriate resource or device by looking at the session-level message headers and URL information. Another protocol, such as SIP [4], could be used for tunneling MRCP messages.

3.2. MRCP over RTSP

RTSP supports both TCP and UDP mechanisms for the client to talk to the server and is differentiated by the RTSP URL. All MRCP based media servers MUST support TCP for transport and MAY support UDP.

In RTSP, the ANNOUNCE method/response MUST be used to carry MRCP request/responses between the client and the server. MRCP messages MUST NOT be communicated in the RTSP SETUP or TEARDOWN messages.

Currently all RTSP messages are request/responses and there is no support for asynchronous events in RTSP. This is because RTSP was designed to work over TCP or UDP and, hence, could not assume reliability in the underlying protocol. Hence, when using MRCP over RTSP, an asynchronous event from the MRCP server is packaged in a server-initiated ANNOUNCE method/response communication. A future RTSP extension to send asynchronous events from the server to the client would provide an alternate vehicle to carry such asynchronous MRCP events from the server.

An RTSP session is created when an RTSP SETUP message is sent from the client to a server and is addressed to a server URL or any one of its resource URLs without specifying a session-id. The server will establish a session context and will respond with a session-id to the client. This sequence will also set up the RTP transport parameters between the client and the server, and then the server will be ready to receive or send media streams. If the client wants to attach an additional resource to an existing session, the client should send that session’s ID in the subsequent SETUP message.

When a media server implementing MRCP over RTSP receives a PLAY, RECORD, or PAUSE RTSP method from an MRCP resource URL, it should respond with an RTSP 405 "Method not Allowed" response. For these resources, the only allowed RTSP methods are SETUP, TEARDOWN, DESCRIBE, and ANNOUNCE.

Example 1:

```
C->S:  ANNOUNCE rtsp://media.server.com/media/synthesizer RTSP/1.0
       CSeq:4
       Session:12345678
       Content-Type:application/mrcp
       Content-Length:223

       SPEAK 543257 MRCP/1.0
       Voice-gender:neutral
       Voice-category:teenager
       Prosody-volume:medium
       Content-Type:application/synthesis+ssml
       Content-Length:104

       <?xml version="1.0"?>
       <speak>
        <paragraph>
          <sentence>You have 4 new messages.</sentence>
          <sentence>The first is from <say-as
          type="name">Stephanie Williams</say-as>
          and arrived at <break/>
          <say-as type="time">3:45pm</say-as>.</sentence>

          <sentence>The subject is <prosody
          rate="-20%">ski trip</prosody></sentence>
        </paragraph>
       </speak>

S->C:  RTSP/1.0 200 OK
       CSeq: 4
       Session:12345678
       RTP-Info:url=rtsp://media.server.com/media/synthesizer;
                seq=9810092;rtptime=3450012
       Content-Type:application/mrcp
       Content-Length:52

       MRCP/1.0 543257 200 IN-PROGRESS

S->C:  ANNOUNCE rtsp://media.server.com/media/synthesizer RTSP/1.0
       CSeq:6
       Session:12345678
```

```
          Content-Type:application/mrcp
          Content-Length:123

          SPEAK-COMPLETE 543257 COMPLETE MRCP/1.0

   C->S:  RTSP/1.0 200 OK
          CSeq:6
```

For the sake of brevity, most examples from here on show only the MRCP messages and do not show the RTSP message and headers in which they are tunneled. Also, RTSP messages such as response that are not carrying an MRCP message are also left out.

3.3. Media Streams and RTP Ports

A single set of RTP/RTCP ports is negotiated and shared between the MRCP client and server when multiple media processing resources, such as automatic speech recognition (ASR) engines and text to speech (TTS) engines, are used for a single session. The individual resource instances allocated on the server under a common session identifier will feed from/to that single RTP stream.

The client can send multiple media streams towards the server, differentiated by using different synchronized source (SSRC) identifier values. Similarly the server can use multiple Synchronized Source (SSRC) identifier values to differentiate media streams originating from the individual transmission resource URLs if more than one exists. The individual resources may, on the other hand, work together to send just one stream to the client. This is up to the implementation of the media server.

4. Notational Conventions

Since many of the definitions and syntax are identical to HTTP/1.1, this specification only points to the section where they are defined rather than copying it. For brevity, [HX.Y] refers to Section X.Y of the current HTTP/1.1 specification (RFC 2616 [1]).

All the mechanisms specified in this document are described in both prose and an augmented Backus-Naur form (ABNF) similar to that used in [H2.1]. It is described in detail in RFC 4234 [3].

The ABNF provided along with the descriptive text is informative in nature and may not be complete. The complete message format in ABNF form is provided in Appendix A and is the normative format definition.

5. MRCP Specification

The MRCP PDU is textual using an ISO 10646 character set in the UTF-8 encoding (RFC 3629 [12]) to allow many different languages to be represented. However, to assist in compact representations, MRCP also allows other character sets such as ISO 8859-1 to be used when desired. The MRCP protocol headers and field names use only the US-ASCII subset of UTF-8. Internationalization only applies to certain fields like grammar, results, speech markup, etc., and not to MRCP as a whole.

Lines are terminated by CRLF, but receivers SHOULD be prepared to also interpret CR and LF by themselves as line terminators. Also, some parameters in the PDU may contain binary data or a record spanning multiple lines. Such fields have a length value associated with the parameter, which indicates the number of octets immediately following the parameter.

The whole MRCP PDU is encoded in the body of the session level message as a MIME entity of type application/mrcp. The individual MRCP messages do not have addressing information regarding which resource the request/response are to/from. Instead, the MRCP message relies on the header of the session level message carrying it to deliver the request to the appropriate resource, or to figure out who the response or event is from.

The MRCP message set consists of requests from the client to the server, responses from the server to the client and asynchronous events from the server to the client. All these messages consist of a start-line, one or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and an optional message body.

```
generic-message =   start-line
                    message-header
                    CRLF
                    [ message-body ]

message-body    =   *OCTET

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

The message-body contains resource-specific and message-specific data that needs to be carried between the client and server as a MIME entity. The information contained here and the actual MIME-types used to carry the data are specified later when addressing the specific messages.

If a message contains data in the message body, the header fields will contain content-headers indicating the MIME-type and encoding of the data in the message body.

5.1. Request

An MRCP request consists of a Request line followed by zero or more parameters as part of the message headers and an optional message body containing data specific to the request message.

The Request message from a client to the server includes, within the first line, the method to be applied, a method tag for that request, and the version of protocol in use.

```
request-line   =    method-name SP request-id SP
                     mrcp-version CRLF
```

The request-id field is a unique identifier created by the client and sent to the server. The server resource should use this identifier in its response to this request. If the request does not complete with the response, future asynchronous events associated with this request MUST carry the request-id.

```
request-id    =    1*DIGIT
```

The method-name field identifies the specific request that the client is making to the server. Each resource supports a certain list of requests or methods that can be issued to it, and will be addressed in later sections.

```
method-name    =    synthesizer-method
               /    recognizer-method
```

The mrcp-version field is the MRCP protocol version that is being used by the client.

```
mrcp-version   =    "MRCP" "/" 1*DIGIT "." 1*DIGIT
```

5.2. Response

After receiving and interpreting the request message, the server resource responds with an MRCP response message. It consists of a status line optionally followed by a message body.

```
response-line  =    mrcp-version SP request-id SP status-code SP
                    request-state CRLF
```

The mrcp-version field used here is similar to the one used in the Request Line and indicates the version of MRCP protocol running on the server.

The request-id used in the response MUST match the one sent in the corresponding request message.

The status-code field is a 3-digit code representing the success or failure or other status of the request.

The request-state field indicates if the job initiated by the Request is PENDING, IN-PROGRESS, or COMPLETE. The COMPLETE status means that the Request was processed to completion and that there will be no more events from that resource to the client with that request-id. The PENDING status means that the job has been placed on a queue and will be processed in first-in-first-out order. The IN-PROGRESS status means that the request is being processed and is not yet complete. A PENDING or IN-PROGRESS status indicates that further Event messages will be delivered with that request-id.

```
request-state    =  "COMPLETE"
                 /  "IN-PROGRESS"
                 /  "PENDING"
```

5.2.1. Status Codes

The status codes are classified under the Success(2XX) codes and the Failure(4XX) codes.

5.2.1.1. Success 2xx

200	Success
201	Success with some optional parameters ignored.

5.2.1.2. Failure 4xx

401	Method not allowed
402	Method not valid in this state
403	Unsupported Parameter
404	Illegal Value for Parameter
405	Not found (e.g., Resource URI not initialized or doesn't exist)
406	Mandatory Parameter Missing
407	Method or Operation Failed (e.g., Grammar compilation failed in the recognizer. Detailed cause codes MAY BE available through a resource specific header field.)
408	Unrecognized or unsupported message entity

```
409       Unsupported Parameter Value
421-499   Resource specific Failure codes
```

5.3. Event

The server resource may need to communicate a change in state or the occurrence of a certain event to the client. These messages are used when a request does not complete immediately and the response returns a status of PENDING or IN-PROGRESS. The intermediate results and events of the request are indicated to the client through the event message from the server. Events have the request-id of the request that is in progress and is generating these events and status value. The status value is COMPLETE if the request is done and this was the last event, else it is IN-PROGRESS.

```
event-line      =  event-name SP request-id SP request-state SP
                   mrcp-version CRLF
```

The mrcp-version used here is identical to the one used in the Request/Response Line and indicates the version of MRCP protocol running on the server.

The request-id used in the event should match the one sent in the request that caused this event.

The request-state indicates if the Request/Command causing this event is complete or still in progress, and is the same as the one mentioned in Section 5.2. The final event will contain a COMPLETE status indicating the completion of the request.

The event-name identifies the nature of the event generated by the media resource. The set of valid event names are dependent on the resource generating it, and will be addressed in later sections.

```
event-name      =  synthesizer-event
                /  recognizer-event
```

5.4. Message Headers

MRCP header fields, which include general-header (Section 5.4) and resource-specific-header (Sections 7.4 and 8.4), follow the same generic format as that given in Section 2.1 of RFC 2822 [7]. Each header field consists of a name followed by a colon (":") and the field value. Field names are case-insensitive. The field value MAY be preceded by any amount of linear whitespace (LWS), though a single SP is preferred. Header fields can be extended over multiple lines by preceding each extra line with at least one SP or HT.

```
message-header =    1*(generic-header / resource-header)
```

The order in which header fields with differing field names are received is not significant. However, it is "good practice" to send general-header fields first, followed by request-header or response-header fields, and ending with the entity-header fields.

Multiple message-header fields with the same field-name MAY be present in a message if and only if the entire field value for that header field is defined as a comma-separated list (i.e., #(values)).

It MUST be possible to combine the multiple header fields into one "field-name:field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. Therefore, the order in which header fields with the same field-name are received is significant to the interpretation of the combined field value, and thus a proxy MUST NOT change the order of these field values when a message is forwarded.

Generic Headers

```
generic-header      =    active-request-id-list
                    /    proxy-sync-id
                    /    content-id
                    /    content-type
                    /    content-length
                    /    content-base
                    /    content-location
                    /    content-encoding
                    /    cache-control
                    /    logging-tag
```

All headers in MRCP will be case insensitive, consistent with HTTP and RTSP protocol header definitions.

5.4.1. Active-Request-Id-List

In a request, this field indicates the list of request-ids to which it should apply. This is useful when there are multiple Requests that are PENDING or IN-PROGRESS and you want this request to apply to one or more of these specifically.

In a response, this field returns the list of request-ids that the operation modified or were in progress or just completed. There could be one or more requests that returned a request-state of PENDING or IN-PROGRESS. When a method affecting one or more PENDING

or IN-PROGRESS requests is sent from the client to the server, the response MUST contain the list of request-ids that were affected in this header field.

The active-request-id-list is only used in requests and responses, not in events.

For example, if a STOP request with no active-request-id-list is sent to a synthesizer resource (a wildcard STOP) that has one or more SPEAK requests in the PENDING or IN-PROGRESS state, all SPEAK requests MUST be cancelled, including the one IN-PROGRESS. In addition, the response to the STOP request would contain the request-id of all the SPEAK requests that were terminated in the active-request-id-list. In this case, no SPEAK-COMPLETE or RECOGNITION-COMPLETE events will be sent for these terminated requests.

```
active-request-id-list  =  "Active-Request-Id-List" ":" request-id
                           *("," request-id) CRLF
```

5.4.2. Proxy-Sync-Id

When any server resource generates a barge-in-able event, it will generate a unique Tag and send it as a header field in an event to the client. The client then acts as a proxy to the server resource and sends a BARGE-IN-OCCURRED method (Section 7.10) to the synthesizer server resource with the Proxy-Sync-Id it received from the server resource. When the recognizer and synthesizer resources are part of the same session, they may choose to work together to achieve quicker interaction and response. Here, the proxy-sync-id helps the resource receiving the event, proxied by the client, to decide if this event has been processed through a direct interaction of the resources.

```
proxy-sync-id    =  "Proxy-Sync-Id" ":" 1*ALPHA CRLF
```

5.4.3. Accept-Charset

See [H14.2]. This specifies the acceptable character set for entities returned in the response or events associated with this request. This is useful in specifying the character set to use in the Natural Language Semantics Markup Language (NLSML) results of a RECOGNITON-COMPLETE event.

5.4.4. Content-Type

See [H14.17]. Note that the content types suitable for MRCP are restricted to speech markup, grammar, recognition results, etc., and are specified later in this document. The multi-part content type "multi-part/mixed" is supported to communicate multiple of the above mentioned contents, in which case the body parts cannot contain any MRCP specific headers.

5.4.5. Content-Id

This field contains an ID or name for the content, by which it can be referred to. The definition of this field conforms to RFC 2392 [14], RFC 2822 [7], RFC 2046 [13] and is needed in multi-part messages. In MRCP whenever the content needs to be stored, by either the client or the server, it is stored associated with this ID. Such content can be referenced during the session in URI form using the session:URI scheme described in a later section.

5.4.6. Content-Base

The content-base entity-header field may be used to specify the base URI for resolving relative URLs within the entity.

```
content-base      = "Content-Base" ":" absoluteURI CRLF
```

Note, however, that the base URI of the contents within the entity-body may be redefined within that entity-body. An example of this would be a multi-part MIME entity, which in turn can have multiple entities within it.

5.4.7. Content-Encoding

The content-encoding entity-header field is used as a modifier to the media-type. When present, its value indicates what additional content coding has been applied to the entity-body, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the content-type header field. Content-encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

```
content-encoding =  "Content-Encoding" ":"
                    *WSP content-coding
                    *(*WSP "," *WSP content-coding *WSP )
                    CRLF

content-coding   =  token
```

```
token           =  1*(alphanum / "-" / "." / "!" / "%" / "*"
                      / "_" / "+" / "`" / "'" / "~" )
```

Content coding is defined in [H3.5]. An example of its use is

```
Content-Encoding:gzip
```

If multiple encodings have been applied to an entity, the content codings MUST be listed in the order in which they were applied.

5.4.8. Content-Location

The content-location entity-header field MAY BE used to supply the resource location for the entity enclosed in the message when that entity is accessible from a location separate from the requested resource's URI.

```
content-location =  "Content-Location" ":" ( absoluteURI /
                        relativeURI ) CRLF
```

The content-location value is a statement of the location of the resource corresponding to this particular entity at the time of the request. The media server MAY use this header field to optimize certain operations. When providing this header field, the entity being sent should not have been modified from what was retrieved from the content-location URI.

For example, if the client provided a grammar markup inline, and it had previously retrieved it from a certain URI, that URI can be provided as part of the entity, using the content-location header field. This allows a resource like the recognizer to look into its cache to see if this grammar was previously retrieved, compiled, and cached. In which case, it might optimize by using the previously compiled grammar object.

If the content-location is a relative URI, the relative URI is interpreted relative to the content-base URI.

5.4.9. Content-Length

This field contains the length of the content of the message body (i.e., after the double CRLF following the last header field). Unlike HTTP, it MUST be included in all messages that carry content beyond the header portion of the message. If it is missing, a default value of zero is assumed. It is interpreted according to [H14.13].

5.4.10. Cache-Control

If the media server plans on implementing caching, it MUST adhere to the cache correctness rules of HTTP 1.1 (RFC2616), when accessing and caching HTTP URI. In particular, the expires and cache-control headers of the cached URI or document must be honored and will always take precedence over the Cache-Control defaults set by this header field. The cache-control directives are used to define the default caching algorithms on the media server for the session or request. The scope of the directive is based on the method it is sent on. If the directives are sent on a SET-PARAMS method, it SHOULD apply for all requests for documents the media server may make in that session. If the directives are sent on any other messages, they MUST only apply to document requests the media server needs to make for that method. An empty cache-control header on the GET-PARAMS method is a request for the media server to return the current cache-control directives setting on the server.

```
cache-control  =    "Cache-Control" ":" *WSP cache-directive
                    *( *WSP "," *WSP cache-directive *WSP )
                    CRLF

cache-directive =   "max-age" "=" delta-seconds
                  /   "max-stale" "=" delta-seconds
                  /   "min-fresh" "=" delta-seconds

delta-seconds       = 1*DIGIT
```

Here, delta-seconds is a time value to be specified as an integer number of seconds, represented in decimal, after the time that the message response or data was received by the media server.

These directives allow the media server to override the basic expiration mechanism.

max-age

Indicates that the client is OK with the media server using a response whose age is no greater than the specified time in seconds. Unless a max-stale directive is also included, the client is not willing to accept the media server using a stale response.

min-fresh

Indicates that the client is willing to accept the media server using a response whose freshness lifetime is no less than its current age plus the specified time in seconds. That is, the

client wants the media server to use a response that will still be fresh for at least the specified number of seconds.

max-stale

Indicates that the client is willing to accept the media server using a response that has exceeded its expiration time. If max-stale is assigned a value, then the client is willing to accept the media server using a response that has exceeded its expiration time by no more than the specified number of seconds. If no value is assigned to max-stale, then the client is willing to accept the media server using a stale response of any age.

The media server cache MAY BE requested to use stale response/data without validation, but only if this does not conflict with any "MUST"-level requirements concerning cache validation (e.g., a "must-revalidate" cache-control directive) in the HTTP 1.1 specification pertaining the URI.

If both the MRCP cache-control directive and the cached entry on the media server include "max-age" directives, then the lesser of the two values is used for determining the freshness of the cached entry for that request.

5.4.11. Logging-Tag

This header field MAY BE sent as part of a SET-PARAMS/GET-PARAMS method to set the logging tag for logs generated by the media server. Once set, the value persists until a new value is set or the session is ended. The MRCP server should provide a mechanism to subset its output logs so that system administrators can examine or extract only the log file portion during which the logging tag was set to a certain value.

MRCP clients using this feature should take care to ensure that no two clients specify the same logging tag. In the event that two clients specify the same logging tag, the effect on the MRCP server's output logs in undefined.

```
logging-tag    =    "Logging-Tag" ":" 1*ALPHA CRLF
```

6. Media Server

The capability of media server resources can be found using the RTSP DESCRIBE mechanism. When a client issues an RTSP DESCRIBE method for a media resource URI, the media server response MUST contain an SDP description in its body describing the capabilities of the media server resource. The SDP description MUST contain at a minimum the media header (m-line) describing the codec and other media related features it supports. It MAY contain another SDP header as well, but support for it is optional.

The usage of SDP messages in the RTSP message body and its application follows the SIP RFC 2543 [4], but is limited to media-related negotiation and description.

6.1. Media Server Session

As discussed in Section 3.2, a client/server should share one RTSP session-id for the different resources it may use under the same session. The client MUST allocate a set of client RTP/RTCP ports for a new session and MUST NOT send a Session-ID in the SETUP message for the first resource. The server then creates a Session-ID and allocates a set of server RTP/RTCP ports and responds to the SETUP message.

If the client wants to open more resources with the same server under the same session, it will send the session-id (that it got in the earlier SETUP response) in the SETUP for the new resource. A SETUP message with an existing session-id tells the server that this new resource will feed from/into the same RTP/RTCP stream of that existing session.

If the client wants to open a resource from a media server that is not where the first resource came from, it will send separate SETUP requests with no session-id header field in them. Each server will allocate its own session-id and return it in the response. Each of them will also come back with their own set of RTP/RTCP ports. This would be the case when the synthesizer engine and the recognition engine are on different servers.

The RTSP SETUP method SHOULD contain an SDP description of the media stream being set up. The RTSP SETUP response MUST contain an SDP description of the media stream that it expects to receive and send on that session.

The SDP description in the SETUP method from the client SHOULD describe the required media parameters like codec, Named Signaling Event (NSE) payload types, etc. This could have multiple media

headers (i.e., m-lines) to allow the client to provide the media server with more than one option to choose from.

The SDP description in the SETUP response should reflect the media parameters that the media server will be using for the stream. It should be within the choices that were specified in the SDP of the SETUP method, if one was provided.

Example:

C->S:

```
SETUP rtsp://media.server.com/recognizer/ RTSP/1.0
CSeq:1
Transport:RTP/AVP;unicast;client_port=46456-46457
Content-Type:application/sdp
Content-Length:190

v=0
o=- 123 456 IN IP4 10.0.0.1
s=Media Server
p=+1-888-555-1212
c=IN IP4 0.0.0.0
t=0 0
m=audio 46456 RTP/AVP 0 96
a=rtpmap:0 pcmu/8000
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-15
```

S->C:

```
RTSP/1.0 200 OK
CSeq:1
Session:0a030258_00003815_3bc4873a_0001_0000
Transport:RTP/AVP;unicast;client_port=46456-46457;
           server_port=46460-46461
Content-Length:190
Content-Type:application/sdp

v=0
o=- 3211724219 3211724219 IN IP4 10.3.2.88
s=Media Server
c=IN IP4 0.0.0.0
t=0 0
m=audio 46460 RTP/AVP 0 96
a=rtpmap:0 pcmu/8000
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-15
```

If an SDP description was not provided in the RTSP SETUP method, then the media server may decide on parameters of the stream but MUST specify what it chooses in the SETUP response. An SDP announcement is only returned in a response to a SETUP message that does not specify a Session. That is, the server will not return an SDP announcement for the synthesizer SETUP of a session already established with a recognizer.

```
C->S:

  SETUP rtsp://media.server.com/recognizer/ RTSP/1.0
  CSeq:1
  Transport:RTP/AVP;unicast;client_port=46498

S->C:

  RTSP/1.0 200 OK
  CSeq:1
  Session:0a030258_000039dc_3bc48a13_0001_0000
  Transport:RTP/AVP;unicast; client_port=46498;
            server_port=46502-46503
  Content-Length:193
  Content-Type:application/sdp

  v=0
  o=- 3211724947 3211724947 IN IP4 10.3.2.88
  s=Media Server
  c=IN IP4 0.0.0.0
  t=0 0
  m=audio 46502 RTP/AVP 0 101
  a=rtpmap:0 pcmu/8000
  a=rtpmap:101 telephone-event/8000
  a=fmtp:101 0-15
```

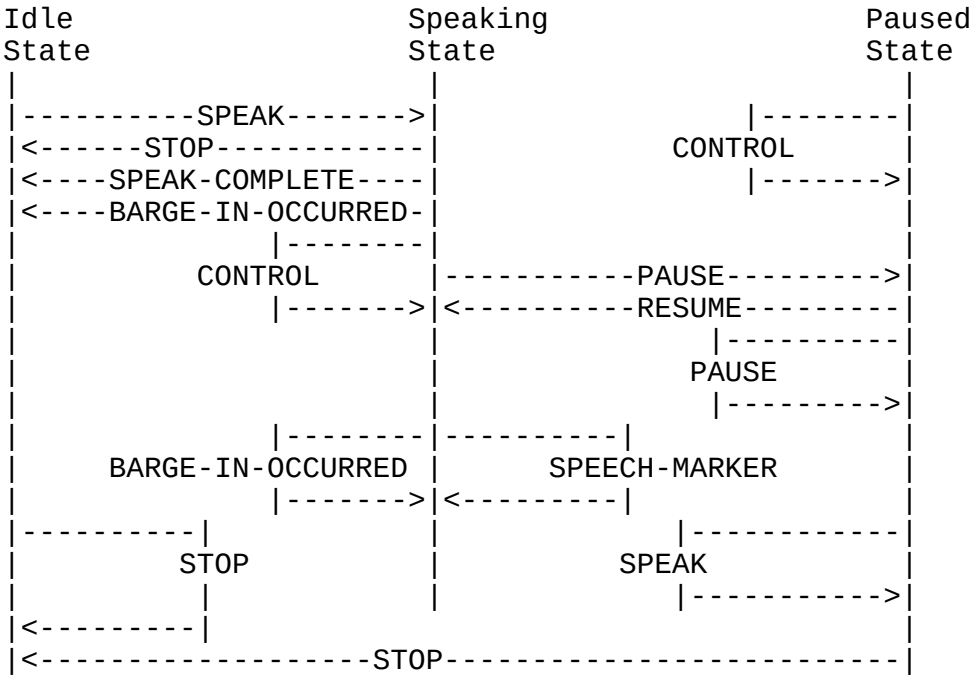
7. Speech Synthesizer Resource

This resource is capable of converting text provided by the client and generating a speech stream in real-time. Depending on the implementation and capability of this resource, the client can control parameters like voice characteristics, speaker speed, etc.

The synthesizer resource is controlled by MRCP requests from the client. Similarly, the resource can respond to these requests or generate asynchronous events to the server to indicate certain conditions during the processing of the stream.

7.1. Synthesizer State Machine

The synthesizer maintains states because it needs to correlate MRCP requests from the client. The state transitions shown below describe the states of the synthesizer and reflect the request at the head of the queue. A SPEAK request in the PENDING state can be deleted or stopped by a STOP request and does not affect the state of the resource.



7.2. Synthesizer Methods

The synthesizer supports the following methods.

```
synthesizer-method  =  "SET-PARAMS"
                    /  "GET-PARAMS"
                    /  "SPEAK"
                    /  "STOP"
                    /  "PAUSE"
                    /  "RESUME"
                    /  "BARGE-IN-OCCURRED"
                    /  "CONTROL"
```

7.3. Synthesizer Events

The synthesizer may generate the following events.

```
synthesizer-event   =  "SPEECH-MARKER"
                    /  "SPEAK-COMPLETE"
```

7.4. Synthesizer Header Fields

A synthesizer message may contain header fields containing request options and information to augment the Request, Response, or Event of the message with which it is associated.

```
synthesizer-header  =  jump-target       ; Section 7.4.1
                    /  kill-on-barge-in  ; Section 7.4.2
                    /  speaker-profile   ; Section 7.4.3
                    /  completion-cause  ; Section 7.4.4
                    /  voice-parameter   ; Section 7.4.5
                    /  prosody-parameter ; Section 7.4.6
                    /  vendor-specific   ; Section 7.4.7
                    /  speech-marker     ; Section 7.4.8
                    /  speech-language   ; Section 7.4.9
                    /  fetch-hint        ; Section 7.4.10
                    /  audio-fetch-hint  ; Section 7.4.11
                    /  fetch-timeout     ; Section 7.4.12
                    /  failed-uri        ; Section 7.4.13
                    /  failed-uri-cause  ; Section 7.4.14
                    /  speak-restart     ; Section 7.4.15
                    /  speak-length      ; Section 7.4.16
```

Parameter	Support	Methods/Events/Response
jump-target	MANDATORY	SPEAK, CONTROL
logging-tag	MANDATORY	SET-PARAMS, GET-PARAMS
kill-on-barge-in	MANDATORY	SPEAK
speaker-profile	OPTIONAL	SET-PARAMS, GET-PARAMS, SPEAK, CONTROL
completion-cause	MANDATORY	SPEAK-COMPLETE
voice-parameter	MANDATORY	SET-PARAMS, GET-PARAMS, SPEAK, CONTROL
prosody-parameter	MANDATORY	SET-PARAMS, GET-PARAMS, SPEAK, CONTROL
vendor-specific	MANDATORY	SET-PARAMS, GET-PARAMS
speech-marker	MANDATORY	SPEECH-MARKER
speech-language	MANDATORY	SET-PARAMS, GET-PARAMS, SPEAK
fetch-hint	MANDATORY	SET-PARAMS, GET-PARAMS, SPEAK
audio-fetch-hint	MANDATORY	SET-PARAMS, GET-PARAMS, SPEAK
fetch-timeout	MANDATORY	SET-PARAMS, GET-PARAMS, SPEAK

failed-uri	MANDATORY	Any
failed-uri-cause	MANDATORY	Any
speak-restart	MANDATORY	CONTROL
speak-length	MANDATORY	SPEAK, CONTROL

7.4.1. Jump-Target

This parameter MAY BE specified in a CONTROL method and controls the jump size to move forward or rewind backward on an active SPEAK request. A + or - indicates a relative value to what is being currently played. This MAY BE specified in a SPEAK request to indicate an offset into the speech markup that the SPEAK request should start speaking from. The different speech length units supported are dependent on the synthesizer implementation. If it does not support a unit or the operation, the resource SHOULD respond with a status code of 404 "Illegal or Unsupported value for parameter".

```
jump-target         =    "Jump-Size" ":" speech-length-value CRLF
speech-length-value =    numeric-speech-length
                    /    text-speech-length
text-speech-length  =    1*ALPHA SP "Tag"
numeric-speech-length=   ("+" / "-") 1*DIGIT SP
                         numeric-speech-unit
numeric-speech-unit =    "Second"
                    /    "Word"
                    /    "Sentence"
                    /    "Paragraph"
```

7.4.2. Kill-On-Barge-In

This parameter MAY BE sent as part of the SPEAK method to enable kill-on-barge-in support. If enabled, the SPEAK method is interrupted by DTMF input detected by a signal detector resource or by the start of speech sensed or recognized by the speech recognizer resource.

```
kill-on-barge-in    =    "Kill-On-Barge-In" ":" boolean-value CRLF
boolean-value       =    "true" / "false"
```

If the recognizer or signal detector resource is on, the same server as the synthesizer, the server should be intelligent enough to recognize their interactions by their common RTSP session-id and work with each other to provide kill-on-barge-in support. The client needs to send a BARGE-IN-OCCURRED method to the synthesizer resource when it receives a barge-in-able event from the synthesizer resource

or signal detector resource. These resources MAY BE local or distributed. If this field is not specified, the value defaults to "true".

7.4.3. Speaker Profile

This parameter MAY BE part of the SET-PARAMS/GET-PARAMS or SPEAK request from the client to the server and specifies the profile of the speaker by a URI, which may be a set of voice parameters like gender, accent, etc.

```
speaker-profile     =    "Speaker-Profile" ":" uri CRLF
```

7.4.4. Completion Cause

This header field MUST be specified in a SPEAK-COMPLETE event coming from the synthesizer resource to the client. This indicates the reason behind the SPEAK request completion.

```
completion-cause    =    "Completion-Cause" ":" 1*DIGIT SP 1*ALPHA
                         CRLF
```

Cause-Code	Cause-Name	Description
000	normal	SPEAK completed normally.
001	barge-in	SPEAK request was terminated because of barge-in.
002	parse-failure	SPEAK request terminated because of a failure to parse the speech markup text.
003	uri-failure	SPEAK request terminated because, access to one of the URIs failed.
004	error	SPEAK request terminated prematurely due to synthesizer error.
005	language-unsupported	Language not supported.

7.4.5. Voice-Parameters

This set of parameters defines the voice of the speaker.

```
voice-parameter     =    "Voice-" voice-param-name ":"
                         voice-param-value CRLF
```

voice-param-name is any one of the attribute names under the voice element specified in W3C's Speech Synthesis Markup Language Specification [9]. The voice-param-value is any one of the value choices of the corresponding voice element attribute specified in the above section.

These header fields MAY BE sent in SET-PARAMS/GET-PARAMS request to define/get default values for the entire session or MAY BE sent in the SPEAK request to define default values for that speak request. Furthermore, these attributes can be part of the speech text marked up in Speech Synthesis Markup Language (SSML).

These voice parameter header fields can also be sent in a CONTROL method to affect a SPEAK request in progress and change its behavior on the fly. If the synthesizer resource does not support this operation, it should respond back to the client with a status of unsupported.

7.4.6. Prosody-Parameters

This set of parameters defines the prosody of the speech.

```
prosody-parameter   =    "Prosody-" prosody-param-name ":"
                         prosody-param-value CRLF
```

prosody-param-name is any one of the attribute names under the prosody element specified in W3C's Speech Synthesis Markup Language Specification [9]. The prosody-param-value is any one of the value choices of the corresponding prosody element attribute specified in the above section.

These header fields MAY BE sent in SET-PARAMS/GET-PARAMS request to define/get default values for the entire session or MAY BE sent in the SPEAK request to define default values for that speak request. Furthermore, these attributes can be part of the speech text marked up in SSML.

The prosody parameter header fields in the SET-PARAMS or SPEAK request only apply if the speech data is of type text/plain and does not use a speech markup format.

These prosody parameter header fields MAY also be sent in a CONTROL method to affect a SPEAK request in progress and to change its behavior on the fly. If the synthesizer resource does not support this operation, it should respond back to the client with a status of unsupported.

7.4.7. Vendor-Specific Parameters

This set of headers allows for the client to set vendor-specific parameters.

```
vendor-specific         = "Vendor-Specific-Parameters" ":"
                          vendor-specific-av-pair
                          *[";" vendor-specific-av-pair] CRLF

vendor-specific-av-pair = vendor-av-pair-name "="
                          vendor-av-pair-value
```

This header MAY BE sent in the SET-PARAMS/GET-PARAMS method and is used to set vendor-specific parameters on the server side. The vendor-av-pair-name can be any vendor-specific field name and conforms to the XML vendor-specific attribute naming convention. The vendor-av-pair-value is the value to set the attribute to and needs to be quoted.

When asking the server to get the current value of these parameters, this header can be sent in the GET-PARAMS method with the list of vendor-specific attribute names to get separated by a semicolon.

7.4.8. Speech Marker

This header field contains a marker tag that may be embedded in the speech data. Most speech markup formats provide mechanisms to embed marker fields between speech texts. The synthesizer will generate SPEECH-MARKER events when it reaches these marker fields. This field SHOULD be part of the SPEECH-MARKER event and will contain the marker tag values.

```
speech-marker =          "Speech-Marker" ":" 1*ALPHA CRLF
```

7.4.9. Speech Language

This header field specifies the default language of the speech data if it is not specified in the speech data. The value of this header field should follow RFC 3066 [16] for its values. This MAY occur in SPEAK, SET-PARAMS, or GET-PARAMS request.

```
speech-language          =    "Speech-Language" ":" 1*ALPHA CRLF
```

7.4.10. Fetch Hint

When the synthesizer needs to fetch documents or other resources like speech markup or audio files, etc., this header field controls URI access properties. This defines when the synthesizer should retrieve content from the server. A value of "prefetch" indicates a file may be downloaded when the request is received, whereas "safe" indicates a file that should only be downloaded when actually needed. The default value is "prefetch". This header field MAY occur in SPEAK, SET-PARAMS, or GET-PARAMS requests.

```
fetch-hint              =    "Fetch-Hint" ":" 1*ALPHA CRLF
```

7.4.11. Audio Fetch Hint

When the synthesizer needs to fetch documents or other resources like speech audio files, etc., this header field controls URI access properties. This defines whether or not the synthesizer can attempt to optimize speech by pre-fetching audio. The value is either "safe" to say that audio is only fetched when it is needed, never before; "prefetch" to permit, but not require the platform to pre-fetch the audio; or "stream" to allow it to stream the audio fetches. The default value is "prefetch". This header field MAY occur in SPEAK, SET-PARAMS, or GET-PARAMS requests.

```
audio-fetch-hint        =    "Audio-Fetch-Hint" ":" 1*ALPHA CRLF
```

7.4.12. Fetch Timeout

When the synthesizer needs to fetch documents or other resources like speech audio files, etc., this header field controls URI access properties. This defines the synthesizer timeout for resources the media server may need to fetch from the network. This is specified in milliseconds. The default value is platform-dependent. This header field MAY occur in SPEAK, SET-PARAMS, or GET-PARAMS.

```
fetch-timeout           =    "Fetch-Timeout" ":" 1*DIGIT CRLF
```

7.4.13. Failed URI

When a synthesizer method needs a synthesizer to fetch or access a URI, and the access fails, the media server SHOULD provide the failed URI in this header field in the method response.

```
failed-uri              =    "Failed-URI" ":" Url CRLF
```

7.4.14. Failed URI Cause

When a synthesizer method needs a synthesizer to fetch or access a URI, and the access fails, the media server SHOULD provide the URI specific or protocol-specific response code through this header field in the method response. This field has been defined as alphanumeric to accommodate all protocols, some of which might have a response string instead of a numeric response code.

```
failed-uri-cause        =    "Failed-URI-Cause" ":" 1*ALPHA CRLF
```

7.4.15. Speak Restart

When a CONTROL jump backward request is issued to a currently speaking synthesizer resource and the jumps beyond the start of the speech, the current SPEAK request re-starts from the beginning of its speech data and the response to the CONTROL request would contain this header indicating a restart. This header MAY occur in the CONTROL response.

```
speak-restart       =    "Speak-Restart" ":" boolean-value CRLF
```

7.4.16. Speak Length

This parameter MAY BE specified in a CONTROL method to control the length of speech to speak, relative to the current speaking point in the currently active SPEAK request. A "-" value is illegal in this field. If a field with a Tag unit is specified, then the media must speak until the tag is reached or the SPEAK request complete, whichever comes first. This MAY BE specified in a SPEAK request to indicate the length to speak in the speech data and is relative to the point in speech where the SPEAK request starts. The different speech length units supported are dependent on the synthesizer implementation. If it does not support a unit or the operation, the resource SHOULD respond with a status code of 404 "Illegal or Unsupported value for parameter".

```
speak-length        =    "Speak-Length" ":" speech-length-value
                         CRLF
```

7.5. Synthesizer Message Body

A synthesizer message may contain additional information associated with the Method, Response, or Event in its message body.

7.5.1. Synthesizer Speech Data

Marked-up text for the synthesizer to speak is specified as a MIME entity in the message body. The message to be spoken by the synthesizer can be specified inline (by embedding the data in the message body) or by reference (by providing the URI to the data). In either case, the data and the format used to markup the speech needs to be supported by the media server.

All media servers MUST support plain text speech data and W3C's Speech Synthesis Markup Language [9] at a minimum and, hence, MUST support the MIME types text/plain and application/synthesis+ssml at a minimum.

If the speech data needs to be specified by URI reference, the MIME type text/uri-list is used to specify the one or more URIs that will list what needs to be spoken. If a list of speech URIs is specified, speech data provided by each URI must be spoken in the order in which the URI are specified.

If the data to be spoken consists of a mix of URI and inline speech data, the multipart/mixed MIME-type is used and embedded with the MIME-blocks for text/uri-list, application/synthesis+ssml or text/plain. The character set and encoding used in the speech data may be specified according to standard MIME-type definitions. The multi-part MIME-block can contain actual audio data in .wav or Sun audio format. This is used when the client has audio clips that it may have recorded, then stored in memory or a local device, and that it currently needs to play as part of the SPEAK request. The audio MIME-parts can be sent by the client as part of the multi-part MIME-block. This audio will be referenced in the speech markup data that will be another part in the multi-part MIME-block according to the multipart/mixed MIME-type specification.

Example 1:

```
Content-Type:text/uri-list
Content-Length:176

http://www.cisco.com/ASR-Introduction.sml
http://www.cisco.com/ASR-Document-Part1.sml
http://www.cisco.com/ASR-Document-Part2.sml
http://www.cisco.com/ASR-Conclusion.sml
```

Example 2:

```
Content-Type:application/synthesis+ssml
Content-Length:104

<?xml version="1.0"?>
```

```
      <speak>
      <paragraph>
               <sentence>You have 4 new messages.</sentence>
               <sentence>The first is from <say-as
               type="name">Stephanie Williams</say-as>
               and arrived at <break/>
               <say-as type="time">3:45pm</say-as>.</sentence>

               <sentence>The subject is <prosody
               rate="-20%">ski trip</prosody></sentence>
      </paragraph>
      </speak>
```

Example 3:

```
      Content-Type:multipart/mixed; boundary="--break"

      --break
      Content-Type:text/uri-list
      Content-Length:176

      http://www.cisco.com/ASR-Introduction.sml
      http://www.cisco.com/ASR-Document-Part1.sml
      http://www.cisco.com/ASR-Document-Part2.sml
      http://www.cisco.com/ASR-Conclusion.sml

      --break
      Content-Type:application/synthesis+ssml
      Content-Length:104

      <?xml version="1.0"?>
      <speak>
      <paragraph>
               <sentence>You have 4 new messages.</sentence>
               <sentence>The first is from <say-as
               type="name">Stephanie Williams</say-as>
               and arrived at <break/>
               <say-as type="time">3:45pm</say-as>.</sentence>

               <sentence>The subject is <prosody
               rate="-20%">ski trip</prosody></sentence>
      </paragraph>
      </speak>
       --break
```

7.6. SET-PARAMS

The SET-PARAMS method, from the client to server, tells the synthesizer resource to define default synthesizer context parameters, like voice characteristics and prosody, etc. If the server accepted and set all parameters, it MUST return a Response-Status of 200. If it chose to ignore some optional parameters, it MUST return 201.

If some of the parameters being set are unsupported or have illegal values, the server accepts and sets the remaining parameters and MUST respond with a Response-Status of 403 or 404, and MUST include in the response the header fields that could not be set.

Example:

```
  C->S:SET-PARAMS 543256 MRCP/1.0
      Voice-gender:female
      Voice-category:adult
      Voice-variant:3

  S->C:MRCP/1.0 543256 200 COMPLETE
```

7.7. GET-PARAMS

The GET-PARAMS method, from the client to server, asks the synthesizer resource for its current synthesizer context parameters, like voice characteristics and prosody, etc. The client SHOULD send the list of parameters it wants to read from the server by listing a set of empty parameter header fields. If a specific list is not specified then the server SHOULD return all the settable parameters including vendor-specific parameters and their current values. The wild card use can be very intensive as the number of settable parameters can be large depending on the vendor. Hence, it is RECOMMENDED that the client does not use the wildcard GET-PARAMS operation very often.

Example:

```
  C->S:GET-PARAMS 543256 MRCP/1.0
       Voice-gender:
       Voice-category:
       Voice-variant:
       Vendor-Specific-Parameters:com.mycorp.param1;
                   com.mycorp.param2

  S->C:MRCP/1.0 543256 200 COMPLETE
       Voice-gender:female
       Voice-category:adult
       Voice-variant:3
```

```
Vendor-Specific-Parameters:com.mycorp.param1="Company Name";
                com.mycorp.param2="124324234@mycorp.com"
```

7.8. SPEAK

The SPEAK method from the client to the server provides the synthesizer resource with the speech text and initiates speech synthesis and streaming. The SPEAK method can carry voice and prosody header fields that define the behavior of the voice being synthesized, as well as the actual marked-up text to be spoken. If specific voice and prosody parameters are specified as part of the speech markup text, it will take precedence over the values specified in the header fields and those set using a previous SET-PARAMS request.

When applying voice parameters, there are 3 levels of scope. The highest precedence are those specified within the speech markup text, followed by those specified in the header fields of the SPEAK request and, hence, apply for that SPEAK request only, followed by the session default values that can be set using the SET-PARAMS request and apply for the whole session moving forward.

If the resource is idle and the SPEAK request is being actively processed, the resource will respond with a success status code and a request-state of IN-PROGRESS.

If the resource is in the speaking or paused states (i.e., it is in the middle of processing a previous SPEAK request), the status returns success and a request-state of PENDING. This means that this SPEAK request is in queue and will be processed after the currently active SPEAK request is completed.

For the synthesizer resource, this is the only request that can return a request-state of IN-PROGRESS or PENDING. When the text to be synthesized is complete, the resource will issue a SPEAK-COMPLETE event with the request-id of the SPEAK message and a request-state of COMPLETE.

Example:

```
  C->S:SPEAK 543257 MRCP/1.0
       Voice-gender:neutral
       Voice-category:teenager
       Prosody-volume:medium
       Content-Type:application/synthesis+ssml
       Content-Length:104
```

```
          <?xml version="1.0"?>
          <speak>
          <paragraph>
            <sentence>You have 4 new messages.</sentence>
            <sentence>The first is from <say-as
            type="name">Stephanie Williams</say-as>
            and arrived at <break/>
            <say-as type="time">3:45pm</say-as>.</sentence>

            <sentence>The subject is <prosody
            rate="-20%">ski trip</prosody></sentence>
          </paragraph>
          </speak>

     S->C:MRCP/1.0 543257 200 IN-PROGRESS

     S->C:SPEAK-COMPLETE 543257 COMPLETE MRCP/1.0
          Completion-Cause:000 normal
```

7.9. STOP

The STOP method from the client to the server tells the resource to stop speaking if it is speaking something.

The STOP request can be sent with an active-request-id-list header field to stop the zero or more specific SPEAK requests that may be in queue and return a response code of 200(Success). If no active-request-id-list header field is sent in the STOP request, it will terminate all outstanding SPEAK requests.

If a STOP request successfully terminated one or more PENDING or IN-PROGRESS SPEAK requests, then the response message body contains an active-request-id-list header field listing the SPEAK request-ids that were terminated. Otherwise, there will be no active-request-id-list header field in the response. No SPEAK-COMPLETE events will be sent for these terminated requests.

If a SPEAK request that was IN-PROGRESS and speaking was stopped, the next pending SPEAK request, if any, would become IN-PROGRESS and move to the speaking state.

If a SPEAK request that was IN-PROGRESS and in the paused state was stopped, the next pending SPEAK request, if any, would become IN-PROGRESS and move to the paused state.

```
   Example:
     C->S:SPEAK 543258 MRCP/1.0
          Content-Type:application/synthesis+ssml
          Content-Length:104

          <?xml version="1.0"?>
          <speak>
          <paragraph>
            <sentence>You have 4 new messages.</sentence>
            <sentence>The first is from <say-as
            type="name">Stephanie Williams</say-as>
            and arrived at <break/>
            <say-as type="time">3:45pm</say-as>.</sentence>

            <sentence>The subject is <prosody
            rate="-20%">ski trip</prosody></sentence>
          </paragraph>
          </speak>

     S->C:MRCP/1.0 543258 200 IN-PROGRESS

     C->S:STOP 543259 200 MRCP/1.0

     S->C:MRCP/1.0 543259 200 COMPLETE
          Active-Request-Id-List:543258
```

7.10. BARGE-IN-OCCURRED

The BARGE-IN-OCCURRED method is a mechanism for the client to communicate a barge-in-able event it detects to the speech resource.

This event is useful in two scenarios,

1. The client has detected some events like DTMF digits or other barge-in-able events and wants to communicate that to the synthesizer.

2. The recognizer resource and the synthesizer resource are in different servers. In which case the client MUST act as a Proxy and receive event from the recognition resource, and then send a BARGE-IN-OCCURRED method to the synthesizer. In such cases, the BARGE-IN-OCCURRED method would also have a proxy-sync-id header field received from the resource generating the original event.

If a SPEAK request is active with kill-on-barge-in enabled, and the BARGE-IN-OCCURRED event is received, the synthesizer should stop streaming out audio. It should also terminate any speech requests queued behind the current active one, irrespective of whether they

have barge-in enabled or not. If a barge-in-able prompt was playing and it was terminated, the response MUST contain the request-ids of all SPEAK requests that were terminated in its active-request-id-list. There will be no SPEAK-COMPLETE events generated for these requests.

If the synthesizer and the recognizer are on the same server, they could be optimized for a quicker kill-on-barge-in response by having them interact directly based on a common RTSP session-id. In these cases, the client MUST still proxy the recognition event through a BARGE-IN-OCCURRED method, but the synthesizer resource may have already stopped and sent a SPEAK-COMPLETE event with a barge-in completion cause code. If there were no SPEAK requests terminated as a result of the BARGE-IN-OCCURRED method, the response would still be a 200 success, but MUST not contain an active-request-id-list header field.

```
C->S:SPEAK 543258 MRCP/1.0
     Voice-gender:neutral
     Voice-category:teenager
     Prosody-volume:medium
     Content-Type:application/synthesis+ssml
     Content-Length:104

     <?xml version="1.0"?>
     <speak>
     <paragraph>
       <sentence>You have 4 new messages.</sentence>
       <sentence>The first is from <say-as
       type="name">Stephanie Williams</say-as>
       and arrived at <break/>
       <say-as type="time">3:45pm</say-as>.</sentence>
       <sentence>The subject is <prosody
       rate="-20%">ski trip</prosody></sentence>
     </paragraph>
     </speak>

S->C:MRCP/1.0 543258 200 IN-PROGRESS

C->S:BARGE-IN-OCCURRED 543259 200 MRCP/1.0
     Proxy-Sync-Id:987654321

S->C:MRCP/1.0 543259 200 COMPLETE
     Active-Request-Id-List:543258
```

7.11. PAUSE

The PAUSE method from the client to the server tells the resource to pause speech, if it is speaking something. If a PAUSE method is issued on a session when a SPEAK is not active, the server SHOULD respond with a status of 402 or "Method not valid in this state". If a PAUSE method is issued on a session when a SPEAK is active and paused, the server SHOULD respond with a status of 200 or "Success". If a SPEAK request was active, the server MUST return an active-request-id-list header with the request-id of the SPEAK request that was paused.

```
C->S:SPEAK 543258 MRCP/1.0
     Voice-gender:neutral
     Voice-category:teenager
     Prosody-volume:medium
     Content-Type:application/synthesis+ssml
     Content-Length:104

     <?xml version="1.0"?>
     <speak>
     <paragraph>
       <sentence>You have 4 new messages.</sentence>
       <sentence>The first is from <say-as
       type="name">Stephanie Williams</say-as>
       and arrived at <break/>
       <say-as type="time">3:45pm</say-as>.</sentence>

       <sentence>The subject is <prosody
       rate="-20%">ski trip</prosody></sentence>
     </paragraph>
     </speak>

S->C:MRCP/1.0 543258 200 IN-PROGRESS

C->S:PAUSE 543259 MRCP/1.0

S->C:MRCP/1.0 543259 200 COMPLETE
     Active-Request-Id-List:543258
```

7.12. RESUME

The RESUME method from the client to the server tells a paused synthesizer resource to continue speaking. If a RESUME method is issued on a session when a SPEAK is not active, the server SHOULD respond with a status of 402 or "Method not valid in this state". If a RESUME method is issued on a session when a SPEAK is active and speaking (i.e., not paused), the server SHOULD respond with a status

of 200 or "Success". If a SPEAK request was active, the server MUST return an active-request-id-list header with the request-id of the SPEAK request that was resumed

Example:

```
  C->S:SPEAK 543258 MRCP/1.0
       Voice-gender:neutral
       Voice-category:teenager
       Prosody-volume:medium
       Content-Type:application/synthesis+ssml
       Content-Length:104

       <?xml version="1.0"?>
       <speak>
       <paragraph>
           <sentence>You have 4 new messages.</sentence>
           <sentence>The first is from <say-as
           type="name">Stephanie Williams</say-as>
           and arrived at <break/>
           <say-as type="time">3:45pm</say-as>.</sentence>

           <sentence>The subject is <prosody
           rate="-20%">ski trip</prosody></sentence>
       </paragraph>
       </speak>

    S->C:MRCP/1.0 543258 200 IN-PROGRESS

    C->S:PAUSE 543259 MRCP/1.0

    S->C:MRCP/1.0 543259 200 COMPLETE
       Active-Request-Id-List:543258

    C->S:RESUME 543260 MRCP/1.0

    S->C:MRCP/1.0 543260 200 COMPLETE
       Active-Request-Id-List:543258
```

7.13. CONTROL

The CONTROL method from the client to the server tells a synthesizer that is speaking to modify what it is speaking on the fly. This method is used to make the synthesizer jump forward or backward in what it is being spoken, change speaker rate and speaker parameters, etc. It affects the active or IN-PROGRESS SPEAK request. Depending on the implementation and capability of the synthesizer resource, it may allow this operation or one or more of its parameters.

When a CONTROL to jump forward is issued and the operation goes beyond the end of the active SPEAK method's text, the request succeeds. A SPEAK-COMPLETE event follows the response to the CONTROL method. If there are more SPEAK requests in the queue, the synthesizer resource will continue to process the next SPEAK method. When a CONTROL to jump backwards is issued and the operation jumps to the beginning of the speech data of the active SPEAK request, the response to the CONTROL request contains the speak-restart header.

These two behaviors can be used to rewind or fast-forward across multiple speech requests, if the client wants to break up a speech markup text into multiple SPEAK requests.

If a SPEAK request was active when the CONTROL method was received, the server MUST return an active-request-id-list header with the Request-id of the SPEAK request that was active.

Example:

```
  C->S:SPEAK 543258 MRCP/1.0
       Voice-gender:neutral
       Voice-category:teenager
       Prosody-volume:medium
       Content-Type:application/synthesis+ssml
       Content-Length:104

       <?xml version="1.0"?>
       <speak>
       <paragraph>
         <sentence>You have 4 new messages.</sentence>
         <sentence>The first is from <say-as
         type="name">Stephanie Williams</say-as>
         and arrived at <break/>
         <say-as type="time">3:45pm</say-as>.</sentence>

         <sentence>The subject is <prosody
         rate="-20%">ski trip</prosody></sentence>
       </paragraph>
       </speak>

  S->C:MRCP/1.0 543258 200 IN-PROGRESS

  C->S:CONTROL 543259 MRCP/1.0
       Prosody-rate:fast

  S->C:MRCP/1.0 543259 200 COMPLETE
       Active-Request-Id-List:543258

  C->S:CONTROL 543260 MRCP/1.0
```

```
          Jump-Size:-15 Words

     S->C:MRCP/1.0 543260 200 COMPLETE
          Active-Request-Id-List:543258
```

7.14. SPEAK-COMPLETE

This is an Event message from the synthesizer resource to the client indicating that the SPEAK request was completed. The request-id header field WILL match the request-id of the SPEAK request that initiated the speech that just completed. The request-state field should be COMPLETE indicating that this is the last Event with that request-id, and that the request with that request-id is now complete. The completion-cause header field specifies the cause code pertaining to the status and reason of request completion such as the SPEAK completed normally or because of an error or kill-on-barge-in, etc.

Example:

```
  C->S:SPEAK 543260 MRCP/1.0
       Voice-gender:neutral
       Voice-category:teenager
       Prosody-volume:medium
       Content-Type:application/synthesis+ssml
       Content-Length:104

       <?xml version="1.0"?>
       <speak>
       <paragraph>
         <sentence>You have 4 new messages.</sentence>
         <sentence>The first is from <say-as
         type="name">Stephanie Williams</say-as>
         and arrived at <break/>
         <say-as type="time">3:45pm</say-as>.</sentence>

         <sentence>The subject is <prosody
         rate="-20%">ski trip</prosody></sentence>
       </paragraph>
       </speak>

  S->C:MRCP/1.0 543260 200 IN-PROGRESS

  S->C:SPEAK-COMPLETE 543260 COMPLETE MRCP/1.0

       Completion-Cause:000 normal
```

7.15. SPEECH-MARKER

This is an event generated by the synthesizer resource to the client when it hits a marker tag in the speech markup it is currently processing. The request-id field in the header matches the SPEAK request request-id that initiated the speech. The request-state field should be IN-PROGRESS as the speech is still not complete and there is more to be spoken. The actual speech marker tag hit, describing where the synthesizer is in the speech markup, is returned in the speech-marker header field.

Example:

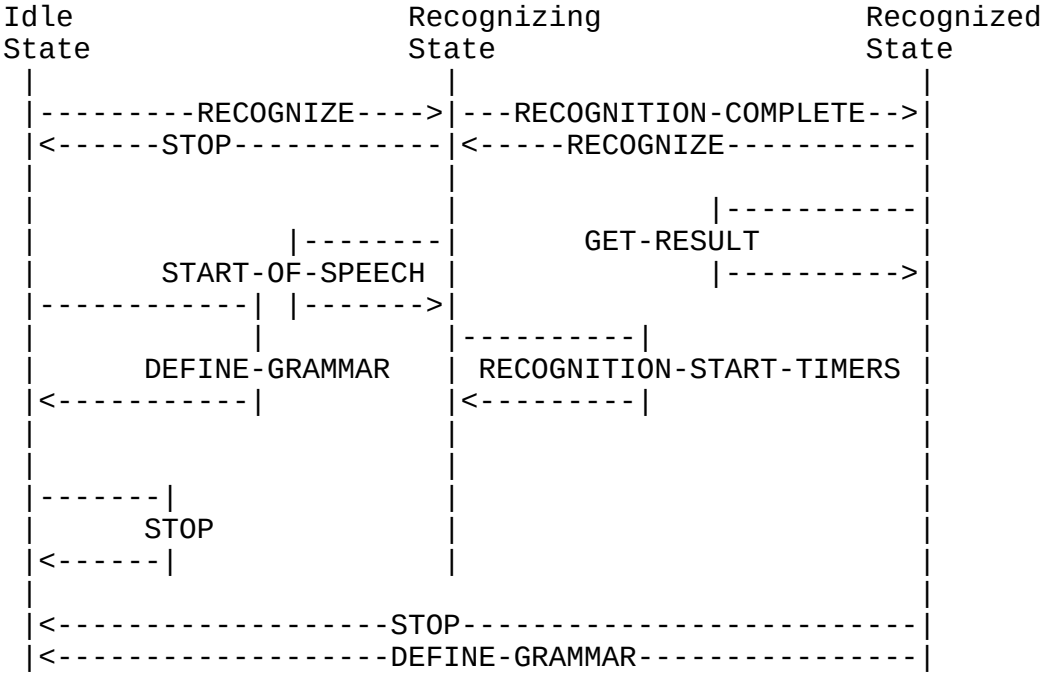
```
  C->S:SPEAK 543261 MRCP/1.0
       Voice-gender:neutral
       Voice-category:teenager
       Prosody-volume:medium
       Content-Type:application/synthesis+ssml
       Content-Length:104

       <?xml version="1.0"?>
       <speak>
       <paragraph>
         <sentence>You have 4 new messages.</sentence>
         <sentence>The first is from <say-as
         type="name">Stephanie Williams</say-as>
         and arrived at <break/>
         <say-as type="time">3:45pm</say-as>.</sentence>
         <mark name="here"/>
         <sentence>The subject is
            <prosody rate="-20%">ski trip</prosody>
         </sentence>
         <mark name="ANSWER"/>
       </paragraph>
       </speak>

  S->C:MRCP/1.0 543261 200 IN-PROGRESS

  S->C:SPEECH-MARKER 543261 IN-PROGRESS MRCP/1.0
       Speech-Marker:here

  S->C:SPEECH-MARKER 543261 IN-PROGRESS MRCP/1.0
       Speech-Marker:ANSWER

  S->C:SPEAK-COMPLETE 543261 COMPLETE MRCP/1.0
       Completion-Cause:000 normal
```

8. Speech Recognizer Resource

The speech recognizer resource is capable of receiving an incoming voice stream and providing the client with an interpretation of what was spoken in textual form.

8.1. Recognizer State Machine

The recognizer resource is controlled by MRCP requests from the client. Similarly, the resource can respond to these requests or generate asynchronous events to the server to indicate certain conditions during the processing of the stream. Hence, the recognizer maintains states to correlate MRCP requests from the client. The state transitions are described below.



8.2. Recognizer Methods

The recognizer supports the following methods.

```
  recognizer-method   =    SET-PARAMS
                      /    GET-PARAMS
                      /    DEFINE-GRAMMAR
                      /    RECOGNIZE
                      /    GET-RESULT
                      /    RECOGNITION-START-TIMERS
                      /    STOP
```

8.3. Recognizer Events

The recognizer may generate the following events.

```
recognizer-event    =    START-OF-SPEECH
                    /    RECOGNITION-COMPLETE
```

8.4. Recognizer Header Fields

A recognizer message may contain header fields containing request options and information to augment the Method, Response, or Event message it is associated with.

```
recognizer-header   =    confidence-threshold     ; Section 8.4.1
                    /    sensitivity-level        ; Section 8.4.2
                    /    speed-vs-accuracy        ; Section 8.4.3
                    /    n-best-list-length       ; Section 8.4.4
                    /    no-input-timeout         ; Section 8.4.5
                    /    recognition-timeout      ; Section 8.4.6
                    /    waveform-url             ; Section 8.4.7
                    /    completion-cause         ; Section 8.4.8
                    /    recognizer-context-block ; Section 8.4.9
                    /    recognizer-start-timers  ; Section 8.4.10
                    /    vendor-specific          ; Section 8.4.11
                    /    speech-complete-timeout  ; Section 8.4.12
                    /    speech-incomplete-timeout; Section 8.4.13
                    /    dtmf-interdigit-timeout  ; Section 8.4.14
                    /    dtmf-term-timeout        ; Section 8.4.15
                    /    dtmf-term-char           ; Section 8.4.16
                    /    fetch-timeout            ; Section 8.4.17
                    /    failed-uri               ; Section 8.4.18
                    /    failed-uri-cause         ; Section 8.4.19
                    /    save-waveform            ; Section 8.4.20
                    /    new-audio-channel        ; Section 8.4.21
                    /    speech-language          ; Section 8.4.22
```

Parameter	Support	Methods/Events
confidence-threshold	MANDATORY	SET-PARAMS, RECOGNIZE GET-RESULT
sensitivity-level	Optional	SET-PARAMS, GET-PARAMS, RECOGNIZE
speed-vs-accuracy	Optional	SET-PARAMS, GET-PARAMS, RECOGNIZE
n-best-list-length	Optional	SET-PARAMS, GET-PARAMS, RECOGNIZE, GET-RESULT
no-input-timeout	MANDATORY	SET-PARAMS, GET-PARAMS, RECOGNIZE

recognition-timeout	MANDATORY	SET-PARAMS, GET-PARAMS, RECOGNIZE
waveform-url	MANDATORY	RECOGNITION-COMPLETE
completion-cause	MANDATORY	DEFINE-GRAMMAR, RECOGNIZE, RECOGNITON-COMPLETE
recognizer-context-block	Optional	SET-PARAMS, GET-PARAMS
recognizer-start-timers	MANDATORY	RECOGNIZE
vendor-specific	MANDATORY	SET-PARAMS, GET-PARAMS
speech-complete-timeout	MANDATORY	SET-PARAMS, GET-PARAMS RECOGNIZE
speech-incomplete-timeout	MANDATORY	SET-PARAMS, GET-PARAMS RECOGNIZE
dtmf-interdigit-timeout	MANDATORY	SET-PARAMS, GET-PARAMS RECOGNIZE
dtmf-term-timeout	MANDATORY	SET-PARAMS, GET-PARAMS RECOGNIZE
dtmf-term-char	MANDATORY	SET-PARAMS, GET-PARAMS RECOGNIZE
fetch-timeout	MANDATORY	SET-PARAMS, GET-PARAMS RECOGNIZE, DEFINE-GRAMMAR
failed-uri	MANDATORY	DEFINE-GRAMMAR response, RECOGNITION-COMPLETE
failed-uri-cause	MANDATORY	DEFINE-GRAMMAR response, RECOGNITION-COMPLETE
save-waveform	MANDATORY	SET-PARAMS, GET-PARAMS, RECOGNIZE
new-audio-channel	MANDATORY	RECOGNIZE
speech-language	MANDATORY	SET-PARAMS, GET-PARAMS, RECOGNIZE, DEFINE-GRAMMAR

8.4.1. Confidence Threshold

When a recognition resource recognizes or matches a spoken phrase with some portion of the grammar, it associates a confidence level with that conclusion. The confidence-threshold parameter tells the recognizer resource what confidence level should be considered a successful match. This is an integer from 0-100 indicating the recognizer's confidence in the recognition. If the recognizer determines that its confidence in all its recognition results is less than the confidence threshold, then it MUST return no-match as the recognition result. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS. The default value for this field is platform specific.

```
confidence-threshold =    "Confidence-Threshold" ":" 1*DIGIT CRLF
```

8.4.2. Sensitivity Level

To filter out background noise and not mistake it for speech, the recognizer may support a variable level of sound sensitivity. The sensitivity-level parameter allows the client to set this value on the recognizer. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS. A higher value for this field means higher sensitivity. The default value for this field is platform specific.

```
sensitivity-level   =    "Sensitivity-Level" ":" 1*DIGIT CRLF
```

8.4.3. Speed Vs Accuracy

Depending on the implementation and capability of the recognizer resource, it may be tunable towards Performance or Accuracy. Higher accuracy may mean more processing and higher CPU utilization, meaning less calls per media server and vice versa. This parameter on the resource can be tuned by the speed-vs-accuracy header. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS. A higher value for this field means higher speed. The default value for this field is platform specific.

```
speed-vs-accuracy   =     "Speed-Vs-Accuracy" ":" 1*DIGIT CRLF
```

8.4.4. N Best List Length

When the recognizer matches an incoming stream with the grammar, it may come up with more than one alternative match because of confidence levels in certain words or conversation paths. If this header field is not specified, by default, the recognition resource will only return the best match above the confidence threshold. The client, by setting this parameter, could ask the recognition resource to send it more than 1 alternative. All alternatives must still be above the confidence-threshold. A value greater than one does not guarantee that the recognizer will send the requested number of alternatives. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS. The minimum value for this field is 1. The default value for this field is 1.

```
n-best-list-length  =    "N-Best-List-Length" ":" 1*DIGIT CRLF
```

8.4.5. No Input Timeout

When recognition is started and there is no speech detected for a certain period of time, the recognizer can send a RECOGNITION-COMPLETE event to the client and terminate the recognition operation. The no-input-timeout header field can set this timeout value. The value is in milliseconds. This header field MAY occur in RECOGNIZE,

SET-PARAMS, or GET-PARAMS. The value for this field ranges from 0 to MAXTIMEOUT, where MAXTIMEOUT is platform specific. The default value for this field is platform specific.

```
no-input-timeout    =    "No-Input-Timeout" ":" 1*DIGIT CRLF
```

8.4.6. Recognition Timeout

When recognition is started and there is no match for a certain period of time, the recognizer can send a RECOGNITION-COMPLETE event to the client and terminate the recognition operation. The recognition-timeout parameter field sets this timeout value. The value is in milliseconds. The value for this field ranges from 0 to MAXTIMEOUT, where MAXTIMEOUT is platform specific. The default value is 10 seconds. This header field MAY occur in RECOGNIZE, SET-PARAMS or GET-PARAMS.

```
recognition-timeout =    "Recognition-Timeout" ":" 1*DIGIT CRLF
```

8.4.7. Waveform URL

If the save-waveform header field is set to true, the recognizer MUST record the incoming audio stream of the recognition into a file and provide a URI for the client to access it. This header MUST be present in the RECOGNITION-COMPLETE event if the save-waveform header field was set to true. The URL value of the header MUST be NULL if there was some error condition preventing the server from recording. Otherwise, the URL generated by the server SHOULD be globally unique across the server and all its recognition sessions. The URL SHOULD BE available until the session is torn down.

```
waveform-url        =    "Waveform-URL" ":" Url CRLF
```

8.4.8. Completion Cause

This header field MUST be part of a RECOGNITION-COMPLETE event coming from the recognizer resource to the client. This indicates the reason behind the RECOGNIZE method completion. This header field MUST BE sent in the DEFINE-GRAMMAR and RECOGNIZE responses, if they return with a failure status and a COMPLETE state.

Cause-Code	Cause-Name	Description
000	success	RECOGNIZE completed with a match or DEFINE-GRAMMAR succeeded in downloading and compiling the grammar

001	no-match	RECOGNIZE completed, but no match was found
002	no-input-timeout	RECOGNIZE completed without a match due to a no-input-timeout
003	recognition-timeout	RECOGNIZE completed without a match due to a recognition-timeout
004	gram-load-failure	RECOGNIZE failed due grammar load failure.
005	gram-comp-failure	RECOGNIZE failed due to grammar compilation failure.
006	error	RECOGNIZE request terminated prematurely due to a recognizer error.
007	speech-too-early	RECOGNIZE request terminated because speech was too early.
008	too-much-speech-timeout	RECOGNIZE request terminated because speech was too long.
009	uri-failure	Failure accessing a URI.
010	language-unsupported	Language not supported.

8.4.9. Recognizer Context Block

This parameter MAY BE sent as part of the SET-PARAMS or GET-PARAMS request. If the GET-PARAMS method contains this header field with no value, then it is a request to the recognizer to return the recognizer context block. The response to such a message MAY contain a recognizer context block as a message entity. If the server returns a recognizer context block, the response MUST contain this header field and its value MUST match the content-id of that entity.

If the SET-PARAMS method contains this header field, it MUST contain a message entity containing the recognizer context data, and a content-id matching this header field.

This content-id should match the content-id that came with the context data during the GET-PARAMS operation.

```
recognizer-context-block =    "Recognizer-Context-Block" ":"
                              1*ALPHA CRLF
```

8.4.10. Recognition Start Timers

This parameter MAY BE sent as part of the RECOGNIZE request. A value of false tells the recognizer to start recognition, but not to start the no-input timer yet. The recognizer should not start the timers until the client sends a RECOGNITION-START-TIMERS request to the recognizer. This is useful in the scenario when the recognizer and synthesizer engines are not part of the same session. Here, when a kill-on-barge-in prompt is being played, you want the RECOGNIZE request to be simultaneously active so that it can detect and implement kill-on-barge-in. But at the same time, you don't want the recognizer to start the no-input timers until the prompt is finished. The default value is "true".

```
recognizer-start-timers  =    "Recognizer-Start-Timers" ":"
                              boolean-value CRLF
```

8.4.11. Vendor Specific Parameters

This set of headers allows the client to set Vendor Specific parameters.

This header can be sent in the SET-PARAMS method and is used to set vendor-specific parameters on the server. The vendor-av-pair-name can be any vendor-specific field name and conforms to the XML vendor-specific attribute naming convention. The vendor-av-pair-value is the value to set the attribute to, and needs to be quoted.

When asking the server to get the current value of these parameters, this header can be sent in the GET-PARAMS method with the list of vendor-specific attribute names to get separated by a semicolon. This header field MAY occur in SET-PARAMS or GET-PARAMS.

8.4.12. Speech Complete Timeout

This header field specifies the length of silence required following user speech before the speech recognizer finalizes a result (either accepting it or throwing a nomatch event). The speech-complete-timeout value is used when the recognizer currently has a complete match of an active grammar, and specifies how long it should wait for more input before declaring a match. By contrast, the incomplete timeout is used when the speech is an incomplete match to an active grammar. The value is in milliseconds.

```
speech-complete-timeout = "Speech-Complete-Timeout" ":"
                          1*DIGIT CRLF
```

A long speech-complete-timeout value delays the result completion and, therefore, makes the computer's response slow. A short speech-complete-timeout may lead to an utterance being broken up inappropriately. Reasonable complete timeout values are typically in the range of 0.3 seconds to 1.0 seconds. The value for this field ranges from 0 to MAXTIMEOUT, where MAXTIMEOUT is platform specific. The default value for this field is platform specific. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS.

8.4.13. Speech Incomplete Timeout

This header field specifies the required length of silence following user speech, after which a recognizer finalizes a result. The incomplete timeout applies when the speech prior to the silence is an incomplete match of all active grammars. In this case, once the timeout is triggered, the partial result is rejected (with a nomatch event). The value is in milliseconds. The value for this field ranges from 0 to MAXTIMEOUT, where MAXTIMEOUT is platform specific. The default value for this field is platform specific.

```
speech-incomplete-timeout = "Speech-Incomplete-Timeout" ":"
                            1*DIGIT CRLF
```

The speech-incomplete-timeout also applies when the speech prior to the silence is a complete match of an active grammar, but where it is possible to speak further and still match the grammar. By contrast, the complete timeout is used when the speech is a complete match to an active grammar and no further words can be spoken.

A long speech-incomplete-timeout value delays the result completion and, therefore, makes the computer's response slow. A short speech-incomplete-timeout may lead to an utterance being broken up inappropriately.

The speech-incomplete-timeout is usually longer than the speech-complete-timeout to allow users to pause mid-utterance (for example, to breathe). This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS.

8.4.14. DTMF Interdigit Timeout

This header field specifies the inter-digit timeout value to use when recognizing DTMF input. The value is in milliseconds. The value for this field ranges from 0 to MAXTIMEOUT, where MAXTIMEOUT is platform specific. The default value is 5 seconds. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS.

```
dtmf-interdigit-timeout = "DTMF-Interdigit-Timeout" ":"
                          1*DIGIT CRLF
```

8.4.15. DTMF Term Timeout

This header field specifies the terminating timeout to use when recognizing DTMF input. The value is in milliseconds. The value for this field ranges from 0 to MAXTIMEOUT, where MAXTIMEOUT is platform specific. The default value is 10 seconds. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS.

```
dtmf-term-timeout   =    "DTMF-Term-Timeout" ":" 1*DIGIT CRLF
```

8.4.16. DTMF-Term-Char

This header field specifies the terminating DTMF character for DTMF input recognition. The default value is NULL which is specified as an empty header field. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS.

```
dtmf-term-char      =    "DTMF-Term-Char" ":" CHAR CRLF
```

8.4.17. Fetch Timeout

When the recognizer needs to fetch grammar documents, this header field controls URI access properties. This defines the recognizer timeout for completing the fetch of the resources the media server needs from the network. The value is in milliseconds. The value for this field ranges from 0 to MAXTIMEOUT, where MAXTIMEOUT is platform specific. The default value for this field is platform specific. This header field MAY occur in RECOGNIZE, SET-PARAMS, or GET-PARAMS.

8.4.18. Failed URI

When a recognizer method needs a recognizer to fetch or access a URI, and the access fails, the media server SHOULD provide the failed URI in this header field in the method response.

8.4.19. Failed URI Cause

When a recognizer method needs a recognizer to fetch or access a URI, and the access fails, the media server SHOULD provide the URI-specific or protocol-specific response code through this header field in the method response. This field has been defined as alphanumeric to accommodate all protocols, some of which might have a response string instead of a numeric response code.

8.4.20. Save Waveform

This header field allows the client to indicate to the recognizer that it MUST save the audio stream that was recognized. The recognizer MUST then record the recognized audio and make it available to the client in the form of a URI returned in the waveform-uri header field in the RECOGNITION-COMPLETE event. If there was an error in recording the stream or the audio clip is otherwise not available, the recognizer MUST return an empty waveform-uri header field. The default value for this fields is "false".

```
save-waveform       =    "Save-Waveform" ":" boolean-value CRLF
```

8.4.21. New Audio Channel

This header field MAY BE specified in a RECOGNIZE message and allows the client to tell the media server that, from that point on, it will be sending audio data from a new audio source, channel, or speaker. If the recognition resource had collected any line statistics or information, it MUST discard it and start fresh for this RECOGNIZE. This helps in the case where the client MAY want to reuse an open recognition session with the media server for multiple telephone calls.

```
new-audio-channel   =    "New-Audio-Channel" ":" boolean-value CRLF
```

8.4.22. Speech Language

This header field specifies the language of recognition grammar data within a session or request, if it is not specified within the data. The value of this header field should follow RFC 3066 [16] for its values. This MAY occur in DEFINE-GRAMMAR, RECOGNIZE, SET-PARAMS, or GET-PARAMS request.

8.5. Recognizer Message Body

A recognizer message may carry additional data associated with the method, response, or event. The client may send the grammar to be recognized in DEFINE-GRAMMAR or RECOGNIZE requests. When the grammar is sent in the DEFINE-GRAMMAR method, the server should be able to download compile and optimize the grammar. The RECOGNIZE request MUST contain a list of grammars that need to be active during the recognition. The server resource may send the recognition results in the RECOGNITION-COMPLETE event or the GET-RESULT response. This data will be carried in the message body of the corresponding MRCP message.

8.5.1. Recognizer Grammar Data

Recognizer grammar data from the client to the server can be provided inline or by reference. Either way, they are carried as MIME entities in the message body of the MRCP request message. The grammar specified inline or by reference specifies the grammar used to match in the recognition process and this data is specified in one of the standard grammar specification formats like W3C's XML or ABNF or Sun's Java Speech Grammar Format, etc. All media servers MUST support W3C's XML based grammar markup format [11] (MIME-type application/grammar+xml) and SHOULD support the ABNF form (MIME-type application/grammar).

When a grammar is specified in-line in the message, the client MUST provide a content-id for that grammar as part of the content headers. The server MUST store the grammar associated with that content-id for the duration of the session. A stored grammar can be overwritten by defining a new grammar with the same content-id. Grammars that have been associated with a content-id can be referenced through a special "session:" URI scheme.

Example:

```
  session:help@root-level.store
```

If grammar data needs to be specified by external URI reference, the MIME-type text/uri-list is used to list the one or more URI that will specify the grammar data. All media servers MUST support the HTTP URI access mechanism.

If the data to be defined consists of a mix of URI and inline grammar data, the multipart/mixed MIME-type is used and embedded with the MIME-blocks for text/uri-list, application/grammar or application/grammar+xml. The character set and encoding used in the grammar data may be specified according to standard MIME-type definitions.

When more than one grammar URI or inline grammar block is specified in a message body of the RECOGNIZE request, it is an active list of grammar alternatives to listen. The ordering of the list implies the precedence of the grammars, with the first grammar in the list having the highest precedence.

Example 1:

```
    Content-Type:application/grammar+xml
    Content-Id:request1@form-level.store
    Content-Length:104

    <?xml version="1.0"?>
```

```
      <!-- the default grammar language is US English -->
      <grammar xml:lang="en-US" version="1.0">

      <!-- single language attachment to tokens -->
      <rule id="yes">
                 <one-of>
                     <item xml:lang="fr-CA">oui</item>
                     <item xml:lang="en-US">yes</item>
                 </one-of>
         </rule>

      <!-- single language attachment to a rule expansion -->
         <rule id="request">
                 may I speak to
                 <one-of xml:lang="fr-CA">
                     <item>Michel Tremblay</item>
                     <item>Andre Roy</item>
                 </one-of>
         </rule>

         <!-- multiple language attachment to a token -->
         <rule id="people1">
                 <token lexicon="en-US,fr-CA"> Robert </token>
         </rule>

         <!-- the equivalent single-language attachment expansion -->
         <rule id="people2">
                 <one-of>
                     <item xml:lang="en-US">Robert</item>
                     <item xml:lang="fr-CA">Robert</item>
                 </one-of>
         </rule>

         </grammar>
```

Example 2:

```
   Content-Type:text/uri-list
   Content-Length:176

   session:help@root-level.store
   http://www.cisco.com/Directory-Name-List.grxml
   http://www.cisco.com/Department-List.grxml
   http://www.cisco.com/TAC-Contact-List.grxml
   session:menu1@menu-level.store
```

Example 3:

```
   Content-Type:multipart/mixed; boundary="--break"
```

```
      --break
      Content-Type:text/uri-list
      Content-Length:176
      http://www.cisco.com/Directory-Name-List.grxml
      http://www.cisco.com/Department-List.grxml
      http://www.cisco.com/TAC-Contact-List.grxml

      --break
      Content-Type:application/grammar+xml
      Content-Id:request1@form-level.store
      Content-Length:104

      <?xml version="1.0"?>

      <!-- the default grammar language is US English -->
      <grammar xml:lang="en-US" version="1.0">

      <!-- single language attachment to tokens -->
      <rule id="yes">
                  <one-of>
                      <item xml:lang="fr-CA">oui</item>
                      <item xml:lang="en-US">yes</item>
                  </one-of>
         </rule>

      <!-- single language attachment to a rule expansion -->
         <rule id="request">
                  may I speak to
                  <one-of xml:lang="fr-CA">
                      <item>Michel Tremblay</item>
                      <item>Andre Roy</item>
                  </one-of>
         </rule>

         <!-- multiple language attachment to a token -->
         <rule id="people1">
                  <token lexicon="en-US,fr-CA"> Robert </token>
         </rule>

         <!-- the equivalent single-language attachment expansion -->

         <rule id="people2">
                  <one-of>
                      <item xml:lang="en-US">Robert</item>
                      <item xml:lang="fr-CA">Robert</item>
                  </one-of>
         </rule>
```

```
        </grammar>
      --break
```

8.5.2. Recognizer Result Data

Recognition result data from the server is carried in the MRCP message body of the RECOGNITION-COMPLETE event or the GET-RESULT response message as MIME entities. All media servers MUST support W3C's Natural Language Semantics Markup Language (NLSML) [10] as the default standard for returning recognition results back to the client, and hence MUST support the MIME-type application/x-nlsml.

Example 1:

```
Content-Type:application/x-nlsml
Content-Length:104

<?xml version="1.0"?>
<result grammar="http://theYesNoGrammar">
    <interpretation>
        <instance>
            <myApp:yes_no>
                <response>yes</response>
            </myApp:yes_no>
        </instance>
        <input>ok</input>
    </interpretation>
</result>
```

8.5.3. Recognizer Context Block

When the client has to change recognition servers within a call, this is a block of data that the client MAY collect from the first media server and provide to the second media server. This may be because the client needs different language support or because the media server issued an RTSP RE-DIRECT. Here, the first recognizer may have collected acoustic and other data during its recognition. When we switch recognition servers, communicating this data may allow the second recognition server to provide better recognition based on the acoustic data collected by the previous recognizer. This block of data is vendor-specific and MUST be carried as MIME-type application/octets in the body of the message.

This block of data is communicated in the SET-PARAMS and GET-PARAMS method/response messages. In the GET-PARAMS method, if an empty recognizer-context-block header field is present, then the recognizer should return its vendor-specific context block in the message body as a MIME-entity with a specific content-id. The content-id value should also be specified in the recognizer-context-block header field

in the GET-PARAMS response. The SET-PARAMS request wishing to provide this vendor-specific data should send it in the message body as a MIME-entity with the same content-id that it received from the GET-PARAMS. The content-id should also be sent in the recognizer-context-block header field of the SET-PARAMS message.

Each automatic speech recognition (ASR) vendor choosing to use this mechanism to handoff recognizer context data among its servers should distinguish its vendor-specific block of data from other vendors by choosing a unique content-id that they should recognize.

8.6. SET-PARAMS

The SET-PARAMS method, from the client to the server, tells the recognizer resource to set and modify recognizer context parameters like recognizer characteristics, result detail level, etc. In the following sections some standard parameters are discussed. If the server resource does not recognize an OPTIONAL parameter, it MUST ignore that field. Many of the parameters in the SET-PARAMS method can also be used in another method like the RECOGNIZE method. But the difference is that when you set something like the sensitivity-level using the SET-PARAMS, it applies for all future requests, whenever applicable. On the other hand, when you pass sensitivity-level in a RECOGNIZE request, it applies only to that request.

Example:

```
C->S:SET-PARAMS 543256 MRCP/1.0
     Sensitivity-Level:20
     Recognition-Timeout:30
     Confidence-Threshold:85

S->C:MRCP/1.0 543256 200 COMPLETE
```

8.7. GET-PARAMS

The GET-PARAMS method, from the client to the server, asks the recognizer resource for its current default parameters, like sensitivity-level, n-best-list-length, etc. The client can request specific parameters from the server by sending it one or more empty parameter headers with no values. The server should then return the settings for those specific parameters only. When the client does not send a specific list of empty parameter headers, the recognizer should return the settings for all parameters. The wild card use can be very intensive as the number of settable parameters can be large depending on the vendor. Hence, it is RECOMMENDED that the client does not use the wildcard GET-PARAMS operation very often.

```
Example:
  C->S:GET-PARAMS 543256 MRCP/1.0
       Sensitivity-Level:
       Recognition-Timeout:
       Confidence-threshold:

  S->C:MRCP/1.0 543256 200 COMPLETE
       Sensitivity-Level:20
       Recognition-Timeout:30
       Confidence-Threshold:85
```

8.8. DEFINE-GRAMMAR

The DEFINE-GRAMMAR method, from the client to the server, provides a grammar and tells the server to define, download if needed, and compile the grammar.

If the server resource is in the recognition state, the DEFINE-GRAMMAR request MUST respond with a failure status.

If the resource is in the idle state and is able to successfully load and compile the grammar, the status MUST return a success code and the request-state MUST be COMPLETE.

If the recognizer could not define the grammar for some reason, say the download failed or the grammar failed to compile, or the grammar was in an unsupported form, the MRCP response for the DEFINE-GRAMMAR method MUST contain a failure status code of 407, and a completion-cause header field describing the failure reason.

```
Example:
  C->S:DEFINE-GRAMMAR 543257 MRCP/1.0
       Content-Type:application/grammar+xml
       Content-Id:request1@form-level.store
       Content-Length:104

       <?xml version="1.0"?>

       <!-- the default grammar language is US English -->
       <grammar xml:lang="en-US" version="1.0">

       <!-- single language attachment to tokens -->
       <rule id="yes">
           <one-of>
               <item xml:lang="fr-CA">oui</item>
               <item xml:lang="en-US">yes</item>
           </one-of>
       </rule>
```

```
          <!-- single language attachment to a rule expansion -->
          <rule id="request">
              may I speak to
              <one-of xml:lang="fr-CA">
                  <item>Michel Tremblay</item>
                  <item>Andre Roy</item>
              </one-of>
          </rule>

          </grammar>

     S->C:MRCP/1.0 543257 200 COMPLETE
          Completion-Cause:000 success


     C->S:DEFINE-GRAMMAR 543258 MRCP/1.0
          Content-Type:application/grammar+xml
          Content-Id:helpgrammar@root-level.store
          Content-Length:104

          <?xml version="1.0"?>

          <!-- the default grammar language is US English -->
          <grammar xml:lang="en-US" version="1.0">

          <rule id="request">
              I need help
          </rule>

          </grammar>

     S->C:MRCP/1.0 543258 200 COMPLETE
          Completion-Cause:000 success

     C->S:DEFINE-GRAMMAR 543259 MRCP/1.0
          Content-Type:application/grammar+xml
          Content-Id:request2@field-level.store
          Content-Length:104
          <?xml version="1.0"?>

          <!-- the default grammar language is US English -->
          <grammar xml:lang="en-US" version="1.0">

          <rule id="request">
              I need help
          </rule>

     S->C:MRCP/1.0 543258 200 COMPLETE
```

```
          Completion-Cause:000 success

     C->S:DEFINE-GRAMMAR 543259 MRCP/1.0
          Content-Type:application/grammar+xml
          Content-Id:request2@field-level.store
          Content-Length:104

          <?xml version="1.0"?>

               <grammar xml:lang="en">

               <import uri="session:politeness@form-level.store"
                       name="polite"/>

               <rule id="basicCmd" scope="public">
               <example> please move the window </example>
               <example> open a file </example>

               <ruleref import="polite#startPolite"/>
               <ruleref uri="#command"/>
               <ruleref import="polite#endPolite"/>
               </rule>

               <rule id="command">
               <ruleref uri="#action"/> <ruleref uri="#object"/>
               </rule>

               <rule id="action">
                    <choice>
                    <item weight="10" tag="OPEN">   open </item>
                    <item weight="2"  tag="CLOSE">  close </item>
                    <item weight="1"  tag="DELETE"> delete </item>
                    <item weight="1"  tag="MOVE">   move </item>
                    </choice>
               </rule>

               <rule id="object">
               <count number="optional">
                    <choice>
                         <item> the </item>
                         <item> a </item>
                    </choice>
               </count>
               <choice>
                    <item> window </item>
                    <item> file </item>
                    <item> menu </item>
               </choice>
```

```
               </rule>

               </grammar>

     S->C:MRCP/1.0 543259 200 COMPLETE
          Completion-Cause:000 success

     C->S:RECOGNIZE 543260 MRCP/1.0
          N-Best-List-Length:2
          Content-Type:text/uri-list
          Content-Length:176

          session:request1@form-level.store
          session:request2@field-level.store
          session:helpgramar@root-level.store

     S->C:MRCP/1.0 543260 200 IN-PROGRESS

     S->C:START-OF-SPEECH 543260 IN-PROGRESS MRCP/1.0

     S->C:RECOGNITION-COMPLETE 543260 COMPLETE MRCP/1.0
          Completion-Cause:000 success
          Waveform-URL:http://web.media.com/session123/audio.wav
          Content-Type:applicationt/x-nlsml
          Content-Length:276

          <?xml version="1.0"?>
          <result x-model="http://IdentityModel"
            xmlns:xf="http://www.w3.org/2000/xforms"
            grammar="session:request1@form-level.store">
               <interpretation>
                    <xf:instance name="Person">
                      <Person>
                          <Name> Andre Roy </Name>
                      </Person>
                    </xf:instance>
                    <input>   may I speak to Andre Roy </input>
               </interpretation>
          </result>
```

8.9. RECOGNIZE

The RECOGNIZE method from the client to the server tells the recognizer to start recognition and provides it with a grammar to match for. The RECOGNIZE method can carry parameters to control the sensitivity, confidence level, and the level of detail in results provided by the recognizer. These parameters override the current defaults set by a previous SET-PARAMS method.

If the resource is in the recognition state, the RECOGNIZE request MUST respond with a failure status.

If the resource is in the Idle state and was able to successfully start the recognition, the server MUST return a success code and a request-state of IN-PROGRESS. This means that the recognizer is active and that the client should expect further events with this request-id.

If the resource could not start a recognition, it MUST return a failure status code of 407 and contain a completion-cause header field describing the cause of failure.

For the recognizer resource, this is the only request that can return request-state of IN-PROGRESS, meaning that recognition is in progress. When the recognition completes by matching one of the grammar alternatives or by a time-out without a match or for some other reason, the recognizer resource MUST send the client a RECOGNITON-COMPLETE event with the result of the recognition and a request-state of COMPLETE.

For large grammars that can take a long time to compile and for grammars that are used repeatedly, the client could issue a DEFINE-GRAMMAR request with the grammar ahead of time. In such a case, the client can issue the RECOGNIZE request and reference the grammar through the "session:" special URI. This also applies in general if the client wants to restart recognition with a previous inline grammar.

Note that since the audio and the messages are carried over separate communication paths there may be a race condition between the start of the flow of audio and the receipt of the RECOGNIZE method. For example, if audio flow is started by the client at the same time as the RECOGNIZE method is sent, either the audio or the RECOGNIZE will arrive at the recognizer first. As another example, the client may chose to continuously send audio to the Media server and signal the Media server to recognize using the RECOGNIZE method. A number of mechanisms exist to resolve this condition and the mechanism chosen is left to the implementers of recognizer Media servers.

Example:

```
  C->S:RECOGNIZE 543257 MRCP/1.0
       Confidence-Threshold:90
       Content-Type:application/grammar+xml
       Content-Id:request1@form-level.store
       Content-Length:104

       <?xml version="1.0"?>
```

```
        <!-- the default grammar language is US English -->
        <grammar xml:lang="en-US" version="1.0">

        <!-- single language attachment to tokens -->
        <rule id="yes">
                <one-of>
                        <item xml:lang="fr-CA">oui</item>
                        <item xml:lang="en-US">yes</item>
                </one-of>
            </rule>

        <!-- single language attachment to a rule expansion -->
            <rule id="request">
                may I speak to
                <one-of xml:lang="fr-CA">
                        <item>Michel Tremblay</item>
                        <item>Andre Roy</item>
                </one-of>
            </rule>

          </grammar>

     S->C:MRCP/1.0 543257 200 IN-PROGRESS

     S->C:START-OF-SPEECH 543257 IN-PROGRESS MRCP/1.0

     S->C:RECOGNITION-COMPLETE 543257 COMPLETE MRCP/1.0

          Completion-Cause:000 success
          Waveform-URL:http://web.media.com/session123/audio.wav
          Content-Type:application/x-nlsml
          Content-Length:276

          <?xml version="1.0"?>
          <result x-model="http://IdentityModel"
            xmlns:xf="http://www.w3.org/2000/xforms"
            grammar="session:request1@form-level.store">
              <interpretation>
                  <xf:instance name="Person">
                      <Person>
                          <Name> Andre Roy </Name>
                      </Person>
                  </xf:instance>
                    <input>   may I speak to Andre Roy </input>
              </interpretation>
          </result>
```

8.10. STOP

The STOP method from the client to the server tells the resource to stop recognition if one is active. If a RECOGNIZE request is active and the STOP request successfully terminated it, then the response header contains an active-request-id-list header field containing the request-id of the RECOGNIZE request that was terminated. In this case, no RECOGNITION-COMPLETE event will be sent for the terminated request. If there was no recognition active, then the response MUST NOT contain an active-request-id-list header field. Either way,method the response MUST contain a status of 200(Success).

Example:

```
  C->S:RECOGNIZE 543257 MRCP/1.0
       Confidence-Threshold:90
       Content-Type:application/grammar+xml
       Content-Id:request1@form-level.store
       Content-Length:104

       <?xml version="1.0"?>

       <!-- the default grammar language is US English -->
       <grammar xml:lang="en-US" version="1.0">

       <!-- single language attachment to tokens -->
       <rule id="yes">
                <one-of>
                         <item xml:lang="fr-CA">oui</item>
                         <item xml:lang="en-US">yes</item>
                </one-of>
            </rule>

       <!-- single language attachment to a rule expansion -->
            <rule id="request">
                may I speak to
                <one-of xml:lang="fr-CA">
                         <item>Michel Tremblay</item>
                         <item>Andre Roy</item>
                </one-of>
            </rule>

       </grammar>

  S->C:MRCP/1.0 543257 200 IN-PROGRESS

  C->S:STOP 543258 200 MRCP/1.0
```

```
     S->C:MRCP/1.0 543258 200 COMPLETE
          Active-Request-Id-List:543257
```

8.11. GET-RESULT

The GET-RESULT method from the client to the server can be issued when the recognizer is in the recognized state. This request allows the client to retrieve results for a completed recognition. This is useful if the client decides it wants more alternatives or more information. When the media server receives this request, it should re-compute and return the results according to the recognition constraints provided in the GET-RESULT request.

The GET-RESULT request could specify constraints like a different confidence-threshold, or n-best-list-length. This feature is optional and the automatic speech recognition (ASR) engine may return a status of unsupported feature.

Example:

```
  C->S:GET-RESULT 543257 MRCP/1.0
       Confidence-Threshold:90

  S->C:MRCP/1.0 543257 200 COMPLETE
       Content-Type:application/x-nlsml
       Content-Length:276

       <?xml version="1.0"?>
       <result x-model="http://IdentityModel"
         xmlns:xf="http://www.w3.org/2000/xforms"
         grammar="session:request1@form-level.store">
           <interpretation>
               <xf:instance name="Person">
                   <Person>
                       <Name> Andre Roy </Name>

                   </Person>
               </xf:instance>
                         <input>   may I speak to Andre Roy </input>
           </interpretation>
       </result>
```

8.12. START-OF-SPEECH

This is an event from the recognizer to the client indicating that it has detected speech. This event is useful in implementing kill-on-barge-in scenarios when the synthesizer resource is in a different session than the recognizer resource and, hence, is not aware of an incoming audio source. In these cases, it is up to the client to act

as a proxy and turn around and issue the BARGE-IN-OCCURRED method to the synthesizer resource. The recognizer resource also sends a unique proxy-sync-id in the header for this event, which is sent to the synthesizer in the BARGE-IN-OCCURRED method to the synthesizer.

This event should be generated irrespective of whether the synthesizer and recognizer are in the same media server or not.

8.13. RECOGNITION-START-TIMERS

This request is sent from the client to the recognition resource when it knows that a kill-on-barge-in prompt has finished playing. This is useful in the scenario when the recognition and synthesizer engines are not in the same session. Here, when a kill-on-barge-in prompt is being played, you want the RECOGNIZE request to be simultaneously active so that it can detect and implement kill-on-barge-in. But at the same time, you don't want the recognizer to start the no-input timers until the prompt is finished. The parameter recognizer-start-timers header field in the RECOGNIZE request will allow the client to say if the timers should be started or not. The recognizer should not start the timers until the client sends a RECOGNITION-START-TIMERS method to the recognizer.

8.14. RECOGNITON-COMPLETE

This is an Event from the recognizer resource to the client indicating that the recognition completed. The recognition result is sent in the MRCP body of the message. The request-state field MUST be COMPLETE indicating that this is the last event with that request-id, and that the request with that request-id is now complete. The recognizer context still holds the results and the audio waveform input of that recognition until the next RECOGNIZE request is issued. A URL to the audio waveform MAY BE returned to the client in a waveform-url header field in the RECOGNITION-COMPLETE event. The client can use this URI to retrieve or playback the audio.

Example:

```
  C->S:RECOGNIZE 543257 MRCP/1.0
       Confidence-Threshold:90
       Content-Type:application/grammar+xml
       Content-Id:request1@form-level.store
       Content-Length:104

       <?xml version="1.0"?>

       <!-- the default grammar language is US English -->
       <grammar xml:lang="en-US" version="1.0">
```

```
          <!-- single language attachment to tokens -->
          <rule id="yes">
                   <one-of>
                            <item xml:lang="fr-CA">oui</item>
                            <item xml:lang="en-US">yes</item>
                   </one-of>
               </rule>

          <!-- single language attachment to a rule expansion -->
               <rule id="request">
                   may I speak to
                   <one-of xml:lang="fr-CA">
                            <item>Michel Tremblay</item>
                            <item>Andre Roy</item>
                   </one-of>
               </rule>

          </grammar>

     S->C:MRCP/1.0 543257 200 IN-PROGRESS

     S->C:START-OF-SPEECH 543257 IN-PROGRESS MRCP/1.0

     S->C:RECOGNITION-COMPLETE 543257 COMPLETE MRCP/1.0
          Completion-Cause:000 success
          Waveform-URL:http://web.media.com/session123/audio.wav
          Content-Type:application/x-nlsml
          Content-Length:276

          <?xml version="1.0"?>
          <result x-model="http://IdentityModel"
            xmlns:xf="http://www.w3.org/2000/xforms"
            grammar="session:request1@form-level.store">
              <interpretation>
                  <xf:instance name="Person">
                      <Person>
                          <Name> Andre Roy </Name>
                      </Person>
                  </xf:instance>
                            <input>   may I speak to Andre Roy </input>
              </interpretation>
          </result>
```

8.15. DTMF Detection

Digits received as DTMF tones will be delivered to the automatic speech recognition (ASR) engine in the RTP stream according to RFC 2833 [15]. The automatic speech recognizer (ASR) needs to support RFC 2833 [15] to recognize digits. If it does not support RFC 2833 [15], it will have to process the audio stream and extract the audio tones from it.

9. Future Study

Various sections of the recognizer could be distributed into Digital Signal Processors (DSPs) on the Voice Browser/Gateway or IP Phones. For instance, the gateway might perform voice activity detection to reduce network bandwidth and CPU requirement of the automatic speech recognition (ASR) server. Such extensions are deferred for further study and will not be addressed in this document.

10. Security Considerations

The MRCP protocol may carry sensitive information such as account numbers, passwords, etc. For this reason it is important that the client have the option of secure communication with the server for both the control messages as well as the media, though the client is not required to use it. If all MRCP communications happens in a trusted domain behind a firewall, this may not be necessary. If the client or server is deployed in an insecure network, communication happening across this insecure network needs to be protected. In such cases, the following additional security functionality MUST be supported on the MRCP server. MRCP servers MUST implement Transport Layer Security (TLS) to secure the RTSP communication, i.e., the RTSP stack SHOULD support the rtsps: URI form. MRCP servers MUST support Secure Real-Time Transport Protocol (SRTP) as an option to send and receive media.

11. RTSP-Based Examples

The following is an example of a typical session of speech synthesis and recognition between a client and the server.

Opening the synthesizer. This is the first resource for this session. The server and client agree on a single Session ID 12345678 and set of RTP/RTCP ports on both sides.

```
C->S:SETUP rtsp://media.server.com/media/synthesizer RTSP/1.0
     CSeq:2
     Transport:RTP/AVP;unicast;client_port=46456-46457
     Content-Type:application/sdp
```

```
          Content-Length:190

          v=0
          o=- 123 456 IN IP4 10.0.0.1
          s=Media Server
          p=+1-888-555-1212
          c=IN IP4 0.0.0.0
          t=0 0
          m=audio 0 RTP/AVP 0 96
          a=rtpmap:0 pcmu/8000
          a=rtpmap:96 telephone-event/8000
          a=fmtp:96 0-15

     S->C:RTSP/1.0 200 OK
          CSeq:2
          Transport:RTP/AVP;unicast;client_port=46456-46457;
                    server_port=46460-46461
          Session:12345678
          Content-Length:190
          Content-Type:application/sdp

          v=0
          o=- 3211724219 3211724219 IN IP4 10.3.2.88
          s=Media Server
          c=IN IP4 0.0.0.0
          t=0 0
          m=audio 46460 RTP/AVP 0 96
          a=rtpmap:0 pcmu/8000
          a=rtpmap:96 telephone-event/8000
          a=fmtp:96 0-15
```

Opening a recognizer resource. Uses the existing session ID and ports.

```
     C->S:SETUP rtsp://media.server.com/media/recognizer RTSP/1.0
          CSeq:3
          Transport:RTP/AVP;unicast;client_port=46456-46457;
                     mode=record;ttl=127
          Session:12345678

     S->C:RTSP/1.0 200 OK
          CSeq:3
          Transport:RTP/AVP;unicast;client_port=46456-46457;
                     server_port=46460-46461;mode=record;ttl=127
          Session:12345678
```

An ANNOUNCE message with the MRCP SPEAK request initiates speech.

```
C->S:ANNOUNCE rtsp://media.server.com/media/synthesizer RTSP/1.0
     CSeq:4
     Session:12345678
     Content-Type:application/mrcp
     Content-Length:456

     SPEAK 543257 MRCP/1.0
     Kill-On-Barge-In:false
     Voice-gender:neutral
     Voice-category:teenager
     Prosody-volume:medium
     Content-Type:application/synthesis+ssml
     Content-Length:104

     <?xml version="1.0"?>
     <speak>
     <paragraph>
              <sentence>You have 4 new messages.</sentence>
              <sentence>The first is from <say-as
              type="name">Stephanie Williams</say-as> <mark
     name="Stephanie"/>
              and arrived at <break/>
              <say-as type="time">3:45pm</say-as>.</sentence>

              <sentence>The subject is <prosody
              rate="-20%">ski trip</prosody></sentence>
     </paragraph>
     </speak>

S->C:RTSP/1.0 200 OK
     CSeq:4
     Session:12345678
     RTP-Info:url=rtsp://media.server.com/media/synthesizer;
                seq=9810092;rtptime=3450012
     Content-Type:application/mrcp
     Content-Length:456

     MRCP/1.0 543257 200 IN-PROGRESS
```

The synthesizer hits the special marker in the message to be spoken and faithfully informs the client of the event.

```
S->C:ANNOUNCE rtsp://media.server.com/media/synthesizer RTSP/1.0
     CSeq:5
     Session:12345678
```

```
          Content-Type:application/mrcp
          Content-Length:123

          SPEECH-MARKER 543257 IN-PROGRESS MRCP/1.0
          Speech-Marker:Stephanie
     C->S:RTSP/1.0 200 OK
          CSeq:5
```

The synthesizer finishes with the SPEAK request.

```
     S->C:ANNOUNCE rtsp://media.server.com/media/synthesizer RTSP/1.0
          CSeq:6
          Session:12345678
          Content-Type:application/mrcp
          Content-Length:123

          SPEAK-COMPLETE 543257 COMPLETE MRCP/1.0


     C->S:RTSP/1.0 200 OK
          CSeq:6
```

The recognizer is issued a request to listen for the customer choices.

```
     C->S:ANNOUNCE rtsp://media.server.com/media/recognizer RTSP/1.0
          CSeq:7
          Session:12345678

          RECOGNIZE 543258 MRCP/1.0
          Content-Type:application/grammar+xml
          Content-Length:104

          <?xml version="1.0"?>

          <!-- the default grammar language is US English -->
          <grammar xml:lang="en-US" version="1.0">

          <!-- single language attachment to a rule expansion -->
               <rule id="request">
                   Can I speak to
                   <one-of xml:lang="fr-CA">
                            <item>Michel Tremblay</item>
                            <item>Andre Roy</item>
                   </one-of>
               </rule>

          </grammar>
```

```
  S->C:RTSP/1.0 200 OK
       CSeq:7
       Content-Type:application/mrcp
       Content-Length:123

       MRCP/1.0 543258 200 IN-PROGRESS
```

The client issues the next MRCP SPEAK method in an ANNOUNCE message, asking the user the question. It is generally RECOMMENDED when playing a prompt to the user with kill-on-barge-in and asking for input, that the client issue the RECOGNIZE request ahead of the SPEAK request for optimum performance and user experience. This way, it is guaranteed that the recognizer is online before the prompt starts playing and the user's speech will not be truncated at the beginning (especially for power users).

```
  C->S:ANNOUNCE rtsp://media.server.com/media/synthesizer RTSP/1.0
       CSeq:8 Session:12345678 Content-Type:application/mrcp
       Content-Length:733

       SPEAK 543259 MRCP/1.0
       Kill-On-Barge-In:true
       Content-Type:application/synthesis+ssml
       Content-Length:104

       <?xml version="1.0"?>
       <speak>
       <paragraph>
                <sentence>Welcome to ABC corporation.</sentence>
                <sentence>Who would you like Talk to.</sentence>
       </paragraph>
       </speak>

  S->C:RTSP/1.0 200 OK
       CSeq:8
       Content-Type:application/mrcp
       Content-Length:123

       MRCP/1.0 543259 200 IN-PROGRESS
```

Since the last SPEAK request had Kill-On-Barge-In set to "true", the message synthesizer is interrupted when the user starts speaking, and the client is notified.

Now, since the recognition and synthesizer resources are in the same session, they worked with each other to deliver kill-on-barge-in. If the resources were in different sessions, it would have taken a few more messages before the client got the SPEAK-COMPLETE event from the

synthesizer resource. Whether the synthesizer and recognizer are in the same session or not, the recognizer MUST generate the START-OF-SPEECH event to the client.

The client should have then blindly turned around and issued a BARGE-IN-OCCURRED method to the synthesizer resource. The synthesizer, if kill-on-barge-in was enabled on the current SPEAK request, would have then interrupted it and issued SPEAK-COMPLETE event to the client. In this example, since the synthesizer and recognizer are in the same session, the client did not issue the BARGE-IN-OCCURRED method to the synthesizer and assumed that kill-on-barge-in was implemented between the two resources in the same session and worked.

The completion-cause code differentiates if this is normal completion or a kill-on-barge-in interruption.

```
S->C:ANNOUNCE rtsp://media.server.com/media/recognizer RTSP/1.0
     CSeq:9
     Session:12345678
     Content-Type:application/mrcp
     Content-Length:273

     START-OF-SPEECH 543258 IN-PROGRESS MRCP/1.0

C->S:RTSP/1.0 200 OK
     CSeq:9

S->C:ANNOUNCE rtsp://media.server.com/media/synthesizer RTSP/1.0
     CSeq:10
     Session:12345678
     Content-Type:application/mrcp
     Content-Length:273

     SPEAK-COMPLETE 543259 COMPLETE MRCP/1.0
     Completion-Cause:000 normal

C->S:RTSP/1.0 200 OK
     CSeq:10
```

The recognition resource matched the spoken stream to a grammar and generated results. The result of the recognition is returned by the server as part of the RECOGNITION-COMPLETE event.

```
S->C:ANNOUNCE rtsp://media.server.com/media/recognizer RTSP/1.0
     CSeq:11
     Session:12345678
     Content-Type:application/mrcp
```

```
          Content-Length:733

          RECOGNITION-COMPLETE 543258 COMPLETE MRCP/1.0
          Completion-Cause:000 success
          Waveform-URL:http://web.media.com/session123/audio.wav
          Content-Type:application/x-nlsml
          Content-Length:104

          <?xml version="1.0"?>
          <result x-model="http://IdentityModel"
            xmlns:xf="http://www.w3.org/2000/xforms"
            grammar="session:request1@form-level.store">
              <interpretation>
                  <xf:instance name="Person">
                      <Person>
                          <Name> Andre Roy </Name>
                      </Person>
                  </xf:instance>
                            <input>   may I speak to Andre Roy </input>
              </interpretation>
          </result>

     C->S:RTSP/1.0 200 OK
          CSeq:11

     C->S:TEARDOWN rtsp://media.server.com/media/synthesizer RTSP/1.0
          CSeq:12
          Session:12345678

     S->C:RTSP/1.0 200 OK
          CSeq:12
```

We are done with the resources and are tearing them down. When the last of the resources for this session are released, the Session-ID and the RTP/RTCP ports are also released.

```
     C->S:TEARDOWN rtsp://media.server.com/media/recognizer RTSP/1.0
          CSeq:13
          Session:12345678

     S->C:RTSP/1.0 200 OK
          CSeq:13
```

12. Informative References

[1] Fielding, R., Gettys, J., Mogul, J., Frystyk. H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext transfer protocol -- HTTP/1.1", RFC 2616, June 1999.

[2] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998

[3] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[4] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[5] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[6] World Wide Web Consortium, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C Candidate Recommendation, March 2004.

[7] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[8] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[9] World Wide Web Consortium, "Speech Synthesis Markup Language (SSML) Version 1.0", W3C Candidate Recommendation, September 2004.

[10] World Wide Web Consortium, "Natural Language Semantics Markup Language (NLSML) for the Speech Interface Framework", W3C Working Draft, 30 May 2001.

[11] World Wide Web Consortium, "Speech Recognition Grammar Specification Version 1.0", W3C Candidate Recommendation, March 2004.

[12] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[13] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[14] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998.

[15] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[16] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.

Appendix A. ABNF Message Definitions

```
   ALPHA          =  %x41-5A / %x61-7A   ; A-Z / a-z

   CHAR           =  %x01-7F     ; any 7-bit US-ASCII character,
                                 ;    excluding NUL

   CR             =  %x0D        ; carriage return

   CRLF           =  CR LF       ; Internet standard newline

   DIGIT          =  %x30-39     ; 0-9

   DQUOTE         =  %x22        ; " (Double Quote)

   HEXDIG         =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

   HTAB           =  %x09        ; horizontal tab

   LF             =  %x0A        ; linefeed

   OCTET          =  %x00-FF     ; 8 bits of data

   SP             =  %x20        ; space

   WSP            =  SP / HTAB   ; white space

   LWS            =  [*WSP CRLF] 1*WSP ; linear whitespace

   SWS            =  [LWS] ; sep whitespace

   UTF8-NONASCII  =  %xC0-DF 1UTF8-CONT
                  /  %xE0-EF 2UTF8-CONT
                  /  %xF0-F7 3UTF8-CONT
                  /  %xF8-Fb 4UTF8-CONT
                  /  %xFC-FD 5UTF8-CONT

   UTF8-CONT      =  %x80-BF

   param          =  *pchar

   quoted-string  =  SWS DQUOTE *(qdtext / quoted-pair )
                     DQUOTE

   qdtext         =  LWS / %x21 / %x23-5B / %x5D-7E
                     / UTF8-NONASCII
```

```
quoted-pair    =  "\" (%x00-09 / %x0B-0C
                  / %x0E-7F)

token          =  1*(alphanum / "-" / "." / "!" / "%" / "*"
                   / "_" / "+" / "`" / "'" / "~" )

reserved       =  ";" / "/" / "?" / ":" / "@" / "&" / "="
                  / "+" / "$" / ","

mark           =  "-" / "_" / "." / "!" / "~" / "*" / "'"
                  / "(" / ")"

unreserved     =  alphanum / mark

char           =  unreserved / escaped /
                  ":" / "@" / "&" / "=" / "+" / "$" / ","

alphanum       =  ALPHA / DIGIT

escaped        =  "%" HEXDIG HEXDIG

absoluteURI    =  scheme ":" ( hier-part / opaque-part )

relativeURI    =  ( net-path / abs-path / rel-path )
                  [ "?" query ]

hier-part      =  ( net-path / abs-path ) [ "?" query ]

net-path       =  "//" authority [ abs-path ]

abs-path       =  "/" path-segments

rel-path       =  rel-segment [ abs-path ]

rel-segment    =  1*( unreserved / escaped / ";" / "@"
                  / "&" / "=" / "+" / "$" / "," )

opaque-part    =  uric-no-slash *uric

uric           =  reserved / unreserved / escaped

uric-no-slash  =  unreserved / escaped / ";" / "?" / ":"
                  / "@" / "&" / "=" / "+" / "$" / ","

path-segments  =  segment *( "/" segment )

segment        =  *pchar *( ";" param )
```

```
scheme          =  ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )

authority       =  srvr / reg-name

srvr            =  [ [ userinfo "@" ] hostport ]

reg-name        =  1*( unreserved / escaped / "$" / ","
                   / ";" / ":" / "@" / "&" / "=" / "+" )

query           =  *uric

userinfo        =  ( user ) [ ":" password ] "@"

user            =  1*( unreserved / escaped
                     / user-unreserved )

user-unreserved  =  "&" / "=" / "+" / "$" / "," / ";"
                     / "?" / "/"

password         =  *( unreserved / escaped /
                     "&" / "=" / "+" / "$" / "," )

hostport         =  host [ ":" port ]

host             =  hostname / IPv4address / IPv6reference

hostname         =  *( domainlabel "." ) toplabel [ "." ]

domainlabel      =  alphanum
                     / alphanum *( alphanum / "-" ) alphanum

toplabel         =    ALPHA / ALPHA *( alphanum / "-" )
                     alphanum

IPv4address      =    1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "."
                     1*3DIGIT

IPv6reference    =    "[" IPv6address "]"

IPv6address      =    hexpart [ ":" IPv4address ]

hexpart          =    hexseq / hexseq "::" [ hexseq ] / "::"
                     [ hexseq ]

hexseq           =    hex4 *( ":" hex4)

hex4             =    1*4HEXDIG
```

```
   port           =    1*DIGIT

   generic-message =   start-line
                       message-header
                       CRLF
                       [ message-body ]

   message-body   =    *OCTET

   start-line     =    request-line / status-line / event-line

   request-line   =    method-name SP request-id SP
                                mrcp-version CRLF

   status-line    =    mrcp-version SP request-id SP
                       status-code SP request-state CRLF

   event-line     =    event-name SP request-id SP
                       request-state SP mrcp-version CRLF

   message-header =    1*(generic-header / resource-header)

   generic-header =    active-request-id-list
                  /    proxy-sync-id
                  /    content-id
                  /    content-type
                  /    content-length
                  /    content-base
                  /    content-location
                  /    content-encoding
                  /    cache-control
                  /    logging-tag
   ; -- content-id is as defined in RFC 2392 and RFC 2046

   mrcp-version   =    "MRCP" "/" 1*DIGIT "." 1*DIGIT

   request-id     =    1*DIGIT

   status-code    =    1*DIGIT

   active-request-id-list =  "Active-Request-Id-List" ":"
                           request-id *("," request-id) CRLF

   proxy-sync-id  =    "Proxy-Sync-Id" ":" 1*ALPHA CRLF

   content-length =    "Content-Length" ":" 1*DIGIT CRLF

   content-base   =    "Content-Base" ":" absoluteURI CRLF
```

```
content-type   =    "Content-Type" ":" media-type

media-type     =    type "/" subtype *( ";" parameter )

type           =    token

subtype        =    token

parameter      =    attribute "=" value

attribute      =    token

value          =    token / quoted-string

content-encoding =  "Content-Encoding" ":"
                    *WSP content-coding
                    *(*WSP "," *WSP content-coding *WSP )
                    CRLF

content-coding   =  token


content-location =  "Content-Location" ":"
                    ( absoluteURI / relativeURI )  CRLF

cache-control  =    "Cache-Control" ":"
                    *WSP cache-directive
                    *( *WSP "," *WSP cache-directive *WSP )
                    CRLF

cache-directive =   "max-age" "=" delta-seconds
                /   "max-stale" "=" delta-seconds
                /   "min-fresh" "=" delta-seconds

logging-tag    =    "Logging-Tag" ":" 1*ALPHA CRLF


resource-header =   recognizer-header
                    /    synthesizer-header

method-name    =    synthesizer-method
                    /    recognizer-method

event-name     =    synthesizer-event
                    /    recognizer-event
```

```
   request-state  =    "COMPLETE"
                  /    "IN-PROGRESS"
                  /    "PENDING"

   synthesizer-method = "SET-PARAMS"
                  /    "GET-PARAMS"
                  /    "SPEAK"
                  /    "STOP"
                  /    "PAUSE"
                  /    "RESUME"
                  /    "BARGE-IN-OCCURRED"
                  /    "CONTROL"

   synthesizer-event = "SPEECH-MARKER"
                  /    "SPEAK-COMPLETE"

   synthesizer-header =     jump-target
                      /     kill-on-barge-in
                      /     speaker-profile
                      /     completion-cause
                      /     voice-parameter
                      /     prosody-parameter
                      /     vendor-specific
                      /     speech-marker
                      /     speech-language
                      /     fetch-hint
                      /     audio-fetch-hint
                      /     fetch-timeout
                      /     failed-uri
                      /     failed-uri-cause
                      /     speak-restart
                      /     speak-length

   recognizer-method = "SET-PARAMS"
                      /    "GET-PARAMS"
                      /    "DEFINE-GRAMMAR"
                      /    "RECOGNIZE"
                      /    "GET-RESULT"
                      /    "RECOGNITION-START-TIMERS"
                      /    "STOP"

   recognizer-event  =      "START-OF-SPEECH"
                     /      "RECOGNITION-COMPLETE"

   recognizer-header =      confidence-threshold
                     /      sensitivity-level
                     /      speed-vs-accuracy
                     /      n-best-list-length
```

```
                    /      no-input-timeout
                    /      recognition-timeout
                    /      waveform-url
                    /      completion-cause
                    /      recognizer-context-block
                    /      recognizer-start-timers
                    /      vendor-specific
                    /      speech-complete-timeout
                    /      speech-incomplete-timeout
                    /      dtmf-interdigit-timeout
                    /      dtmf-term-timeout
                    /      dtmf-term-char
                    /      fetch-timeout
                    /      failed-uri
                    /      failed-uri-cause
                    /      save-waveform
                    /      new-audio-channel
                    /      speech-language

   jump-target       =  "Jump-Size" ":" speech-length-value CRLF

   speech-length-value =    numeric-speech-length
                     /      text-speech-length

   text-speech-length =     1*ALPHA SP "Tag"

   numeric-speech-length =("+" / "-") 1*DIGIT SP
                         numeric-speech-unit

   numeric-speech-unit =    "Second"
                       /    "Word"
                       /    "Sentence"
                       /    "Paragraph"

   delta-seconds  =    1*DIGIT

   kill-on-barge-in =  "Kill-On-Barge-In" ":" boolean-value CRLF

   boolean-value  =    "true" / "false"

   speaker-profile =    "Speaker-Profile" ":" absoluteURI CRLF

   completion-cause =  "Completion-Cause" ":" 1*DIGIT SP
                       1*ALPHA CRLF

   voice-parameter =   "Voice-" voice-param-name ":"
                       voice-param-value CRLF
```

```
   voice-param-name =  1*ALPHA

   voice-param-value = 1*alphanum

   prosody-parameter = "Prosody-" prosody-param-name ":"
                        prosody-param-value CRLF

   prosody-param-name =     1*ALPHA

   prosody-param-value = 1*alphanum

   vendor-specific =   "Vendor-Specific-Parameters" ":"
                      vendor-specific-av-pair
                       *[";" vendor-specific-av-pair] CRLF

   vendor-specific-av-pair = vendor-av-pair-name "="
                             vendor-av-pair-value

   vendor-av-pair-name = 1*ALPHA

   vendor-av-pair-value = 1*alphanum

   speech-marker  =    "Speech-Marker" ":" 1*ALPHA CRLF

   speech-language =   "Speech-Language" ":" 1*ALPHA CRLF

   fetch-hint     =    "Fetch-Hint" ":" 1*ALPHA CRLF

   audio-fetch-hint =  "Audio-Fetch-Hint" ":" 1*ALPHA CRLF

   fetch-timeout  =    "Fetch-Timeout" ":" 1*DIGIT CRLF

   failed-uri     =    "Failed-URI" ":" absoluteURI CRLF

   failed-uri-cause =  "Failed-URI-Cause" ":" 1*ALPHA CRLF

   speak-restart  =    "Speak-Restart" ":" boolean-value CRLF

   speak-length   =    "Speak-Length" ":" speech-length-value
                       CRLF
   confidence-threshold =   "Confidence-Threshold" ":"
                            1*DIGIT CRLF

   sensitivity-level = "Sensitivity-Level" ":" 1*DIGIT CRLF

   speed-vs-accuracy = "Speed-Vs-Accuracy" ":" 1*DIGIT CRLF

   n-best-list-length = "N-Best-List-Length" ":" 1*DIGIT CRLF
```

```
   no-input-timeout =  "No-Input-Timeout" ":" 1*DIGIT CRLF

   recognition-timeout = "Recognition-Timeout" ":" 1*DIGIT CRLF

   waveform-url   =    "Waveform-URL" ":" absoluteURI CRLF

   recognizer-context-block = "Recognizer-Context-Block" ":"
                       1*ALPHA CRLF

   recognizer-start-timers = "Recognizer-Start-Timers" ":"
                       boolean-value CRLF

   speech-complete-timeout = "Speech-Complete-Timeout" ":"
                       1*DIGIT CRLF

   speech-incomplete-timeout = "Speech-Incomplete-Timeout" ":"
                       1*DIGIT CRLF

   dtmf-interdigit-timeout = "DTMF-Interdigit-Timeout" ":"
                             1*DIGIT CRLF

   dtmf-term-timeout = "DTMF-Term-Timeout" ":" 1*DIGIT CRLF

   dtmf-term-char =    "DTMF-Term-Char" ":" CHAR CRLF

   save-waveform  =    "Save-Waveform" ":" boolean-value CRLF

   new-audio-channel = "New-Audio-Channel" ":"
                       boolean-value CRLF
```

Appendix B. Acknowledgements

Andre Gillet (Nuance Communications)
Andrew Hunt (SpeechWorks)
Aaron Kneiss (SpeechWorks)
Kristian Finlator (SpeechWorks)
Martin Dragomirecky (Cisco Systems, Inc.)
Pierre Forgues (Nuance Communications)
Suresh Kaliannan (Cisco Systems, Inc.)
Corey Stohs (Cisco Systems, Inc.)
Dan Burnett (Nuance Communications)

Authors' Addresses

Saravanan Shanmugham
Cisco Systems, Inc.
170 W. Tasman Drive
San Jose, CA 95134

EMail: sarvi@cisco.com

Peter Monaco
Nuasis Corporation
303 Bryant St.
Mountain View, CA 94041

EMail: peter.monaco@nuasis.com

Brian Eberman
Speechworks, Inc.
695 Atlantic Avenue
Boston, MA 02111

EMail: brian.eberman@speechworks.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).