```
SPEECHSC                                                    D. Burnett
Internet-Draft                                                   Voxeo
Intended status: Standards Track                        S. Shanmugham
Expires: January 12, 2012                         Cisco Systems, Inc.
                                                         July 11, 2011
```

Media Resource Control Protocol Version 2 (MRCPv2)
draft-ietf-speechsc-mrcpv2-25

Abstract

   The MRCPv2 protocol allows client hosts to control media service
   resources such as speech synthesizers, recognizers, verifiers and
   identifiers residing in servers on the network. MRCPv2 is not a
   "stand-alone" protocol - it relies on other protocols, such as
   Session Initiation Protocol (SIP) to rendezvous MRCPv2 clients and
   servers and manage sessions between them, and the Session Description
   Protocol (SDP) to describe, discover and exchange capabilities. It
   also depends on SIP and SDP to establish the media sessions and
   associated parameters between the media source or sink and the media
   server. Once this is done, the MRCPv2 protocol exchange operates
   over the control session established above, allowing the client to
   control the media processing resources on the speech resource server.

Status of this Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at http://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on January 12, 2012.

Table of Contents

1. Introduction

   The MRCPv2 protocol is designed to allow a client device to control
   media processing resources on the network. Some of these media
   processing resources include speech recognition engines, speech
   synthesis engines, speaker verification and speaker identification
   engines. MRCPv2 enables the implementation of distributed
   Interactive Voice Response platforms using VoiceXML
   [W3C.REC-voicexml20-20040316] browsers or other client applications
   while maintaining separate back-end speech processing capabilities on
   specialized speech processing servers. MRCPv2 is based on the
   earlier Media Resource Control Protocol (MRCP) [RFC4463] developed
   jointly by Cisco Systems, Inc., Nuance Communications, and
   Speechworks Inc.

   The protocol requirements of SPEECHSC [RFC4313] include that the
   solution be capable of reaching a media processing server and setting
   up communication channels to the media resources, and sending and
   receiving control messages and media streams to/from the server. The
   Session Initiation Protocol (SIP) [RFC3261] meets these requirements.

   The proprietary version of MRCP ran over the Real Time Streaming
   Protocol (RTSP) [RFC2326]. At the time work on MRCPv2 was begun, the
   consensus was that this use of RTSP would break the RTSP protocol or
   cause backward-compatibility problems, something forbidden by Section
   3.2 of the above mentioned requirements document. This is the reason
   why MRCPv2 does not run over RTSP.

   MRCPv2 leverages these capabilities by building upon SIP and the
   Session Description Protocol (SDP) [RFC4566]. MRCPv2 uses SIP to
   setup and tear down media and control sessions with the server. In
   addition, the client can use a SIP re-INVITE method (an INVITE dialog
   sent within an existing SIP Session) to change the characteristics of
   these media and control session while maintaining the SIP dialog
   between the client and server. SDP is used to describe the
   parameters of the media sessions associated with that dialog. It is
   mandatory to support SIP as the session establishment protocol to
   ensure interoperability. Other protocols can be used for session
   establishment by prior agreement. This document only describes the
   use of SIP and SDP.

   MRCPv2 uses SIP and SDP to create the speech client/server dialog and
   set up the media channels to the server. It also uses SIP and SDP to
   establish MRCPv2 control sessions between the client and the server
   for each media processing resource required for that dialog. The
   MRCPv2 protocol exchange between the client and the media resource is
   carried on that control session. MRCPv2 protocol exchanges do not
   change the state of the SIP dialog, the media sessions, or other

parameters of the dialog initiated via SIP. It controls and affects
the state of the media processing resource associated with the MRCPv2
session(s).

MRCPv2 defines the messages to control the different media processing
resources and the state machines required to guide their operation.
It also describes how these messages are carried over a transport
layer protocol such as the Transmission Control Protocol (TCP)
[RFC0793] or the Transport Layer Security (TLS) Protocol [RFC5246]
(Note: the Stream Control Transmission Protocol (SCTP) [RFC4960] is a
viable transport for MRCPv2 as well, but the mapping onto SCTP is not
described in this specification).

2. Document Conventions

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 [RFC2119].

Since many of the definitions and syntax are identical to those for
the HTTP/1.1 (Hypertext Transfer Protocol (HTTP/1.1) [RFC2616]), this
specification refers to the section where they are defined rather
than copying it. For brevity, [HX.Y] is to be taken to refer to
Section X.Y of RFC 2616.

All the mechanisms specified in this document are described in both
prose and an augmented Backus-Naur form (ABNF [RFC5234]).

The complete message format in ABNF form is provided in Section 15
and is the normative format definition. Note that productions may be
duplicated within the main body of the document for reading
convenience. If a production in the body of the text conflicts with
one in the normative definition, the latter rules.

2.1. Definitions

Media Resource
            An entity on the speech processing server that can be
            controlled through the MRCPv2 protocol.
MRCP Server
            Aggregate of one or more "Media Resource" entities on
            a Server, exposed through the MRCPv2 protocol
            ("Server" for short).

MRCP Client

> An entity controlling one or more Media Resources
> through the MRCPv2 protocol ("Client" for short).

DTMF

> Dual Tone Multi-Frequency; a method of transmitting
> key presses in-band, either as actual tones (Q.23
> [Q.23]) or as named tone events (RFC 4733 [RFC4733]).

Endpointing

> The process of automatically detecting the beginning
> and end of speech in an audio stream. This is
> critical both for speech recognition and for automated
> recording as one would find in voice mail systems.

Hotword Mode

> A mode of speech recognition where a stream of
> utterances is evaluated for match against a small set
> of command words. This is generally employed to
> either trigger some action, or to control the
> subsequent grammar to be used for further recognition

2.2. State-Machine Diagrams

The state-machine diagrams in this document do not show every
possible method call. Rather, they reflect the state of the resource
based on the methods that have moved to IN-PROGRESS or COMPLETE
states (see Section 5.3). Note that since PENDING requests
essentially have not affected the resource yet and are in queue to be
processed, they are not reflected in the state-machine diagrams.

2.3. URI Schemes

This document defines many protocol headers that contain URIs
(Uniform Resource Identifier (URI) [RFC3986]) or lists of URIs for
referencing media. The entire document, including the Security
Considerations section (Section 12), assumes that HTTP or HTTP over
TLS (HTTPS) [RFC2818] will be used as the URI addressing scheme
unless otherwise stated. However, implementations MAY support other
schemes (such as "file") provided they have addressed any security
considerations described in this document and any others particular
to the specific scheme. For example, implementations where the
client and server both reside on the same physical hardware and the
file system is secured by traditional user-level file access controls
could be reasonable candidates for supporting the "file" scheme.

3. Architecture

A system using MRCPv2 consists of a client that requires the
generation and/or consumption of media streams and a media resource

server that has the resources or "engines" to process these streams
as input or generate these streams as output. The client uses SIP
and SDP to establish an MRCPv2 control channel with the server to use
its media processing resources. MRCPv2 servers are addressed using
SIP URIs.

The session initiation protocol (SIP) uses SDP with the offer/answer
model described in RFC3264 [RFC3264] to set up the MRCPv2 control
channels and describe their characteristics. A separate MRCPv2
session is needed to control each of the media processing resources
associated with the SIP dialog between the client and server. Within
a SIP dialog, the individual resource control channels for the
different resources are added or removed through SDP offer/answer
carried in a SIP re-INVITE transaction.

The server, through the SDP exchange, provides the client with an
unambiguous channel identifier and a TCP port number. The client MAY
then open a new TCP connection with the server on this port number.
Multiple MRCPv2 channels can share a TCP connection between the
client and the server. All MRCPv2 messages exchanged between the
client and the server carry the specified channel identifier that the
server MUST ensure is unambiguous among all MRCPv2 control channels
that are active on that server. The client uses this channel
identifier to indicate the media processing resource associated with
that channel. For information on message framing, see Section 5.

The session initiation protocol (SIP) also establishes the media
sessions between the client (or other source/sink of media) and the
MRCPv2 server using SDP m-lines. One or more media processing
resources may share a media session under a SIP session, or each
media processing resource may have its own media session.

The following diagram shows the general architecture of a system that
uses MRCPv2. To simplify the diagram only a few resources are shown.

```
        MRCPv2 client                   MRCPv2 Media Resource Server
 |-------------------|             |------------------------------------|
 ||-----------------||             ||----------------------------------||
 || Application Layer||             ||Synthesis|Recognition|Verification||
 ||-----------------||             || Engine  |  Engine   |  Engine    ||
 ||Media Resource API||             ||   ||    |    ||     |    ||      ||
 ||-----------------||             ||Synthesis|Recognizer |  Verifier  ||
 || SIP  |  MRCPv2   ||             ||Resource | Resource  |  Resource  ||
 ||Stack |           ||             ||    Media Resource Management     ||
 ||      |           ||             ||----------------------------------||
 ||-----------------||             ||  SIP   |        MRCPv2           ||
 ||   TCP/IP Stack   ||---MRCPv2---|| Stack  |                         ||
 ||                  ||             ||----------------------------------||
 ||-----------------||----SIP-----||         TCP/IP Stack              ||
 |-------------------|             ||                                  ||
          |                        ||----------------------------------||
         SIP                        |------------------------------------|
          |                              /
 |-------------------|           RTP
 |                   |             /
 | Media Source/Sink |------------/
 |                   |
 |-------------------|
```
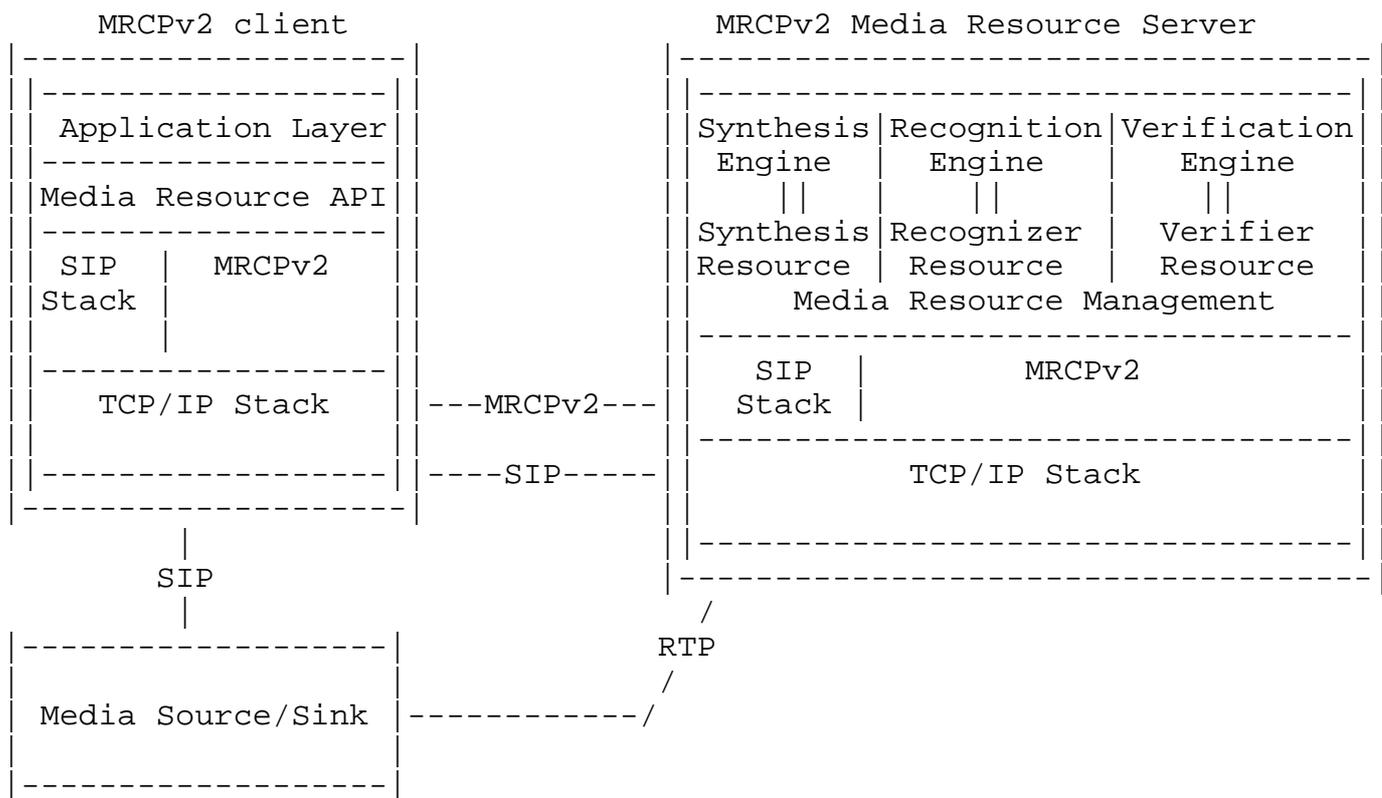
                    Figure 1: Architectural Diagram

3.1. MRCPv2 Media Resource Types

   An MRCPv2 server may offer one or more of the following media
   processing resources to its clients.
   Basic Synthesizer
               A speech synthesizer resource with very limited
               capabilities, that can generate its media stream
               exclusively from concatenated audio clips. The speech
               data is described using a limited subset of the Speech
               Synthesis Markup Language (SSML)
               [W3C.REC-speech-synthesis-20040907] elements. A basic
               synthesizer MUST support the SSML tags <speak>,
               <audio>, <say-as> and <mark>.
   Speech Synthesizer
               A full capability speech synthesis resource capable of
               rendering speech from text. Such a synthesizer MUST
               have full SSML [W3C.REC-speech-synthesis-20040907]
               support.

Recorder

A resource capable of recording audio and providing a URI pointer to the recording. A recorder MUST provide some endpointing capabilities for suppressing silence at the beginning and end of a recording, and MAY also suppress silence in the middle of a recording. If such suppression is done, the recorder MUST maintain timing metadata to indicate the actual time stamps of the recorded media.

DTMF Recognizer

A recognition resource capable of extracting and interpreting Dual-Tone Multi-Frequency (DTMF) [Q.23] digits in a media stream and matching them against a supplied digit grammar It could also do a semantic interpretation based on semantic tags in the grammar.

Speech Recognizer

A full speech recognition resource that is capable of receiving a media stream containing audio and interpreting it to recognition results. It also has a natural language semantic interpreter to post-process the recognized data according to the semantic data in the grammar and provide semantic results along with the recognized input. The recognizer may also support enrolled grammars, where the client can enroll and create new personal grammars for use in future recognition operations.

Speaker Verifier

A resource capable of verifying the authenticity of a claimed identity by matching a media stream containing spoken input to a pre-existing voiceprint. This may also involve matching the caller's voice against more than one voiceprint, also called multi-verification or speaker identification.

3.2. Server and Resource Addressing

The MRCPv2 server is a generic SIP server, and is thus addressed by a SIP URI (RFC 3261 [RFC3261]).

For example:

    sip:mrcpv2@example.net

4. MRCPv2 Protocol Basics

MRCPv2 requires a connection-oriented transport layer protocol such as TCP or SCTP to guarantee reliable sequencing and delivery of

MRCPv2 control messages between the client and the server. In order
to meet the requirements for security enumerated in SpeechSC
Requirements [RFC4313], clients and servers MUST implement TLS as
well. One or more connections between the client and the server can
be shared among different MRCPv2 channels to the server. The
individual messages carry the channel identifier to differentiate
messages on different channels. MRCPv2 protocol encoding is text
based with mechanisms to carry embedded binary data. This allows
arbitrary data like recognition grammars, recognition results,
synthesizer speech markup etc. to be carried in MRCPv2 messages. For
information on message framing, see Section 5.

4.1. Connecting to the Server

MRCPv2 employs a session establishment and management protocol such
as SIP in conjunction with SDP. The client reaches an MRCPv2 server
using conventional INVITE and other SIP requests for establishing,
maintaining, and terminating SIP dialogs. The SDP offer/answer
exchange model over SIP is used to establish a resource control
channel for each resource. The SDP offer/answer exchange is also
used to establish media sessions between the server and the source or
sink of audio.

4.2. Managing Resource Control Channels

The client needs a separate MRCPv2 resource control channel to
control each media processing resource under the SIP dialog. A
unique channel identifier string identifies these resource control
channels. The channel identifier is an unambiguous, opaque string
followed by an "@", then by a string token specifying the type of
resource. The server generates the channel identifier and MUST make
sure it does not clash with the identifier of any other MRCP channel
currently allocated by that server. MRCPv2 defines the following
IANA-registered types of media processing resources. Additional
resource types, their associated methods/events and state machines
may be added as described below in Section 13.

```
+---------------+----------------------+-------------+
| Resource Type | Resource Description | Described in |
+---------------+----------------------+-------------+
| speechrecog   | Speech Recognizer    | Section 9    |
| dtmfrecog     | DTMF Recognizer      | Section 9    |
| speechsynth   | Speech Synthesizer   | Section 8    |
| basicsynth    | Basic Synthesizer    | Section 8    |
| speakverify   | Speaker Verification | Section 11   |
| recorder      | Speech Recorder      | Section 10   |
+---------------+----------------------+-------------+
```

Resource Types

The SIP INVITE or re-INVITE transaction and the SDP offer/answer
exchange it carries contain m-lines describing the resource control
channel to be allocated. There MUST be one SDP m-line for each
MRCPv2 resource to be used in the session. This m-line MUST have a
media type field of "application" and a transport type field of
either "TCP/MRCPv2" or "TCP/TLS/MRCPv2". (The usage of SCTP with
MRCPv2 may be addressed in a future specification). The port number
field of the m-line MUST contain the "discard" port of the transport
protocol (port 9 for TCP) in the SDP offer from the client and MUST
contain the TCP listen port on the server in the SDP answer. The
client may then either set up a TCP or TLS connection to that server
port or share an already established connection to that port. Since
MRCPv2 allows multiple sessions to share the same TCP connection,
multiple m-lines in a single SDP document may share the same port
field value; MRCPv2 servers MUST NOT assume any relationship between
resources using the same port other than the sharing of the
communication channel.

MRCPv2 resources do not use the port or format field of the m-line to
distinguish themselves from other resources using the same channel.
The client MUST specify the resource type identifier in the resource
attribute associated with the control m-line of the SDP offer. The
server MUST respond with the full Channel-Identifier (which includes
the resource type identifier and an unambiguous string) in the
"channel" attribute associated with the control m-line of the SDP
answer. To remain backwards compatible with conventional SDP usage,
the format field of the m-line MUST have the arbitrarily-selected
value of "1".

When the client wants to add a media processing resource to the
session, it issues a new SDP offer, according to the procedures of
RFC 3264 [RFC3264], in a SIP re-INVITE request. The SDP offer/answer
exchange carried by this SIP transaction contains one or more
additional control m-lines for the new resources to be allocated to
the session. The server, on seeing the new m-line, allocates the
resources (if they are available) and responds with a corresponding
control m-line in the SDP answer carried in the SIP response. If the
new resources are not available, the re-INVITE receives an error
message, and existing media processing going on before the re-INVITE
will continue as it was before.

MRCPv2 clients and servers using TCP as a transport protocol MUST use
the procedures specified in RFC 4145 [RFC4145] for setting up the TCP
connection, with the considerations described hereby. Similarly,
MRCPv2 clients and servers using TCP/TLS as a transport protocol MUST
use the procedures specified in RFC 4572 [RFC4572] for setting up the

TLS connection, with the considerations described hereby. The
a=setup attribute, as described in RFC 4145 [RFC4145], MUST be
"active" for the offer from the client and MUST be "passive" for the
answer from the MRCPv2 server. The a=connection attribute MUST have
a value of "new" on the very first control m-line offer from the
client to an MRCPv2 server. Subsequent control m-line offers from
the client to the MRCP server MAY contain "new" or "existing",
depending on whether the client wants to set up a new connection or
share an existing connection, respectively. If the client specifies
a value of "new", the server MUST respond with a value of "new". If
the client specifies a value of "existing", the server MAY respond
with a value of "existing" if it prefers to share an existing
connection or can answer with a value of "new", in which case the
client MUST initiate a new transport connection.

When the client wants to de-allocate the resource from this session,
it issues a new SDP offer, according to RFC 3264 [RFC3264], where the
control m-line port MUST be set to 0. This SDP offer is sent in a
SIP re-INVITE request. This de-allocates the associated MRCPv2
identifier and resource. The server MUST NOT close the TCP, SCTP or
TLS connection if it is currently being shared among multiple MRCP
channels. When all MRCP channels that may be sharing the connection
are released and/or the associated SIP dialog is terminated, the
client or server terminates the connection.

When the client wants to tear down the whole session and all its
resources, it MUST issue a SIP BYE request to close the SIP session.
This will de-allocate all the control channels and resources
allocated under the session.

All servers MUST support TLS. Servers MAY support TCP without TLS in
physically secure environments. It is up to the client, through the
SDP offer, to choose which transport it wants to use for an MRCPv2
session. Aside from the exceptions given above, when using TCP the
m-lines MUST conform to RFC4145 [RFC4145], which describes the usage
of SDP for connection-oriented transport. When using TLS the SDP
m-line for the control stream MUST conform to comedia over TLS
[RFC4572], which specifies the usage of SDP for establishing a secure
connection-oriented transport over TLS.

4.3. SIP session example

This first example shows the power of using SIP to route to the
appropriate resource. In the example, note the use of a request to a
domain's speech server service in the INVITE to
mresources@example.com. The SIP routing machinery in the domain
locates the actual server, mresources@server.example.com, which gets
returned in the 200 OK. Note that "cmid" is defined in Section 4.4.

   This example exchange adds a resource control channel for a
   synthesizer. Since a synthesizer also generates an audio stream,
   this interaction also creates a receive-only Real-Time Protocol (RTP)
   [RFC3550] media session for the server to send audio to. The SIP
   dialog with the media source/sink is independent of MRCP and is not
   shown.

   C->S: INVITE sip:mresources@example.com SIP/2.0
           Via:SIP/2.0/TCP client.atlanta.example.com:5060;
            branch=z9hG4bK74bf1
           Max-Forwards:6
           To:MediaServer <sip:mresources@example.com>
           From:sarvi <sip:sarvi@example.com>;tag=1928301774
           Call-ID:a84b4c76e66710
           CSeq:314161 INVITE
           Contact:<sip:sarvi@client.example.com>
           Content-Type:application/sdp
           Content-Length:...

           v=0
           o=sarvi 2890844526 2890844526 IN IP4 192.0.2.12
           s=-
           c=IN IP4 192.0.2.12
           t=0 0
           m=application 9 TCP/MRCPv2 1
           a=setup:active
           a=connection:new
           a=resource:speechsynth
           a=cmid:1
           m=audio 49170 RTP/AVP 0
           a=rtpmap:0 pcmu/8000
           a=recvonly
           a=mid:1

   S->C: SIP/2.0 200 OK
           Via:SIP/2.0/TCP client.atlanta.example.com:5060;
            branch=z9hG4bK74bf1;received=192.0.32.10
           To:MediaServer <sip:mresources@example.com>;tag=62784
           From:sarvi <sip:sarvi@example.com>;tag=1928301774
           Call-ID:a84b4c76e66710
           CSeq:314161 INVITE
           Contact:<sip:mresources@server.example.com>
           Content-Type:application/sdp
           Content-Length:...

           v=0
           o=- 2890842808 2890842808 IN IP4 192.0.2.11

```
           s=-
           c=IN IP4 192.0.2.11
           t=0 0
           m=application 32416 TCP/MRCPv2 1
           a=setup:passive
           a=connection:new
           a=channel:32AECB234338@speechsynth
           a=cmid:1
           m=audio 48260 RTP/AVP 0
           a=rtpmap:0 pcmu/8000
           a=sendonly
           a=mid:1
```

```
   C->S:   ACK sip:mresources@server.example.com SIP/2.0
           Via:SIP/2.0/TCP client.atlanta.example.com:5060;
            branch=z9hG4bK74bf2
           Max-Forwards:6
           To:MediaServer <sip:mresources@example.com>;tag=62784
           From:Sarvi <sip:sarvi@example.com>;tag=1928301774
           Call-ID:a84b4c76e66710
           CSeq:314161 ACK
           Content-Length:0
```

               Example: Add Synthesizer Control Channel

   This example exchange continues from the previous figure and
   allocates an additional resource control channel for a recognizer.
   Since a recognizer would need to receive an audio stream for
   recognition, this interaction also updates the audio stream to
   sendrecv, making it a 2-way RTP media session.

```
   C->S:   INVITE sip:mresources@server.example.com SIP/2.0
           Via:SIP/2.0/TCP client.atlanta.example.com:5060;
            branch=z9hG4bK74bf3
           Max-Forwards:6
           To:MediaServer <sip:mresources@example.com>;tag=62784
           From:sarvi <sip:sarvi@example.com>;tag=1928301774
           Call-ID:a84b4c76e66710
           CSeq:314162 INVITE
           Contact:<sip:sarvi@client.example.com>
           Content-Type:application/sdp
           Content-Length:...

           v=0
           o=sarvi 2890844526 2890844527 IN IP4 192.0.2.12
           s=-
           c=IN IP4 192.0.2.12
```

```
         t=0 0
         m=application 9 TCP/MRCPv2 1
         a=setup:active
         a=connection:existing
         a=resource:speechsynth
         a=cmid:1
         m=audio 49170 RTP/AVP 0 96
         a=rtpmap:0 pcmu/8000
         a=rtpmap:96 telephone-event/8000
         a=fmtp:96 0-15
         a=sendrecv
         a=mid:1
         m=application 9 TCP/MRCPv2 1
         a=setup:active
         a=connection:existing
         a=resource:speechrecog
         a=cmid:1


   S->C:  SIP/2.0 200 OK
         Via:SIP/2.0/TCP client.atlanta.example.com:5060;
          branch=z9hG4bK74bf3;received=192.0.32.10
         To:MediaServer <sip:mresources@example.com>;tag=62784
         From:sarvi <sip:sarvi@example.com>;tag=1928301774
         Call-ID:a84b4c76e66710
         CSeq:314162 INVITE
         Contact:<sip:mresources@server.example.com>
         Content-Type:application/sdp
         Content-Length:...

         v=0
         o=- 2890842808 2890842809 IN IP4 192.0.2.11
         s=-
         c=IN IP4 192.0.2.11
         t=0 0
         m=application 32416 TCP/MRCPv2 1
         a=setup:passive
         a=connection:existing
         a=channel:32AECB234338@speechsynth
         a=cmid:1
         m=audio 48260 RTP/AVP 0 96
         a=rtpmap:0 pcmu/8000
         a=rtpmap:96 telephone-event/8000
         a=fmtp:96 0-15
         a=sendrecv
         a=mid:1
         m=application 32416 TCP/MRCPv2 1
         a=setup:passive
```

```
                a=connection:existing
                a=channel:32AECB234338@speechrecog
                a=cmid:1


    C->S:   ACK sip:mresources@server.example.com SIP/2.0
            Via:SIP/2.0/TCP client.atlanta.example.com:5060;
             branch=z9hG4bK74bf4
            Max-Forwards:6
            To:MediaServer <sip:mresources@example.com>;tag=62784
            From:Sarvi <sip:sarvi@example.com>;tag=1928301774
            Call-ID:a84b4c76e66710
            CSeq:314162 ACK
            Content-Length:0


                        Add Recognizer example
```

This example exchange continues from the previous figure and de-
allocates the recognizer channel. Since a recognizer no longer needs
to receive an audio stream, this interaction also updates the RTP
media session to recvonly.

```
    C->S:   INVITE sip:mresources@server.example.com SIP/2.0
            Via:SIP/2.0/TCP client.atlanta.example.com:5060;
             branch=z9hG4bK74bf5
            Max-Forwards:6
            To:MediaServer <sip:mresources@example.com>;tag=62784
            From:sarvi <sip:sarvi@example.com>;tag=1928301774
            Call-ID:a84b4c76e66710
            CSeq:314163 INVITE
            Contact:<sip:sarvi@client.example.com>
            Content-Type:application/sdp
            Content-Length:...

            v=0
            o=sarvi 2890844526 2890844528 IN IP4 192.0.2.12
            s=-
            c=IN IP4 192.0.2.12
            t=0 0
            m=application 9 TCP/MRCPv2 1
            a=resource:speechsynth
            a=cmid:1
            m=audio 49170 RTP/AVP 0
            a=rtpmap:0 pcmu/8000
            a=recvonly
            a=mid:1
            m=application 0 TCP/MRCPv2 1
            a=resource:speechrecog
```

```
     a=cmid:1


  S->C:  SIP/2.0 200 OK
         Via:SIP/2.0/TCP client.atlanta.example.com:5060;
          branch=z9hG4bK74bf5;received=192.0.32.10
         To:MediaServer <sip:mresources@example.com>;tag=62784
         From:sarvi <sip:sarvi@example.com>;tag=1928301774
         Call-ID:a84b4c76e66710
         CSeq:314163 INVITE
         Contact:<sip:mresources@server.example.com>
         Content-Type:application/sdp
         Content-Length:...

         v=0
         o=- 2890842808 2890842810 IN IP4 192.0.2.11
         s=-
         c=IN IP4 192.0.2.11
         t=0 0
         m=application 32416 TCP/MRCPv2 1
         a=channel:32AECB234338@speechsynth
         a=cmid:1
         m=audio 48260 RTP/AVP 0
         a=rtpmap:0 pcmu/8000
         a=sendonly
         a=mid:1
         m=application 0 TCP/MRCPv2 1
         a=channel:32AECB234338@speechrecog
         a=cmid:1


  C->S:  ACK sip:mresources@server.example.com SIP/2.0
         Via:SIP/2.0/TCP client.atlanta.example.com:5060;
          branch=z9hG4bK74bf6
         Max-Forwards:6
         To:MediaServer <sip:mresources@example.com>;tag=62784
         From:Sarvi <sip:sarvi@example.com>;tag=1928301774
         Call-ID:a84b4c76e66710
         CSeq:314163 ACK
         Content-Length:0
```

                    Deallocate Recognizer example

4.4. Media Streams and RTP Ports

   Since MRCPv2 resources either generate or consume media streams, the
   client or the server needs to associate media sessions with their
   corresponding resource or resources. More than one resource could be

associated with a single media session or each resource could be
assigned a separate media session. Also note that more than one
media session can be associated with a single resource if need be,
but this scenario is not useful for the current set of resources.
For example, a synthesizer and a recognizer could be associated to
the same media session (m=audio line), if it is opened in "sendrecv"
mode. Alternatively, the recognizer could have its own "sendonly"
audio session and the synthesizer could have its own "recvonly" audio
session.

The association between control channels and their corresponding
media sessions is established using a new "resource channel media
identifier" media-level attribute ("cmid"). Valid values of this
attribute are the values of the "mid" attribute defined in RFC 5888
[RFC5888]. If there is more than 1 audio m-line, then each audio
m-line MUST have a "mid" attribute. Each control m-line MAY have one
or more "cmid" attributes that match the resource control channel to
the "mid" attributes of the audio m-lines it is associated with.
Note that if a control m-line does not have a "cmid" attribute it
will not be associated with any media. The operations on such a
resource will hence be limited. For example, if it was a recognizer
resource, the RECOGNIZE method requires an associated media to
process while the INTERPRET method does not. The formatting of the
"cmid" attribute is described by the following ABNF:

```
cmid-attribute = "a=cmid:" identification-tag
identification-tag = token
```

To allow this flexible mapping of media sessions to MRCPv2 control
channels, a single audio m-line can be associated with multiple
resources or each resource can have its own audio m-line. For
example, if the client wants to allocate a recognizer and a
synthesizer and associate them with a single 2-way audio stream, the
SDP offer would contain two control m-lines and a single audio m-line
with an attribute of "sendrecv". Each of the control m-lines would
have a "cmid" attribute whose value matches the "mid" of the audio
m-line. If, on the other hand, the client wants to allocate a
recognizer and a synthesizer each with its own separate audio stream,
the SDP offer would carry two control m-lines (one for the recognizer
and another for the synthesizer) and two audio m-lines (one with the
attribute "sendonly" and another with attribute "recvonly"). The
"cmid" attribute of the recognizer control m-line would match the
"mid" value of the "sendonly" audio m-line and the "cmid" attribute
of the synthesizer control m-line would match the "mid" attribute of
the "recvonly" m-line.

When a server receives media (e.g. audio) on a media session that is
associated with more than one media processing resource, it is the

responsibility of the server to receive and fork the media to the
resources that need to consume it. If multiple resources in an
MRCPv2 session are generating audio (or other media) to be sent on a
single associated media session, it is the responsibility of the
server to either multiplex the multiple streams onto the single RTP
session or contain an embedded RTP mixer (see RFC 3550 [RFC3550]) to
combine the multiple streams into one. In the former case, the media
stream will contain RTP packets generated by different sources, and
hence the packets will have different Synchronization Source
identifiers (SSRCs). In the latter case, the RTP packets will
contain multiple Contributing Source Identifiers (CSRCs)
corresponding to the original streams before being combined by the
mixer. An MRCPv2 implementation MUST either multiplex or mix unless
it cannot correctly do either, in which case the server MUST disallow
the client from associating multiple such resources to a single audio
stream by rejecting the SDP offer with a SIP 488 "Not Acceptable"
error. Note that there is a large installed base that will return a
SIP 501 "Not Implemented" error in this case. To facilitate
interoperability with this installed base, new implementations should
consider adding configuration to treat a 501 in this context as a 488
when it is received from an element known to be a legacy
implementation.

4.5. MRCPv2 Message Transport

   The MRCPv2 messages defined in this document are transported over a
   TCP, TLS or SCTP (in the future) connection between the client and
   the server. The method for setting up this transport connection and
   the resource control channel is discussed in Section 4.1 and
   Section 4.2. Multiple resource control channels between a client and
   a server that belong to different SIP dialogs can share one or more
   TLS, TCP or SCTP connections between them; the server and client MUST
   support this mode of operation. The individual MRCPv2 messages carry
   the MRCPv2 channel identifier in their Channel-Identifier header
   field, which MUST be used to differentiate MRCPv2 messages from
   different resource channels (see Section 6.2.1 for details). All
   MRCPv2 servers MUST support TLS. Servers MAY support TCP without TLS
   in physically secure environments. It is up to the client to choose
   which mode of transport it wants to use for an MRCPv2 session.

   Most examples from here on show only the MRCPv2 messages and do not
   show the SIP messages that may have been used to establish the MRCPv2
   control channel.

4.6. MRCPv2 Session Termination

   If an MRCP client notices that the underlying connection has been
   closed for one of its MRCP channels, and it has not previously

initiated a re-INVITE to close that channel, it MUST send a BYE to
close down the SIP dialog and all other MRCP channels. If an MRCP
server notices that the underlying connection has been closed for one
of its MRCP channels, and it has not previously received and accepted
a re-INVITE closing that channel, then it MUST send a BYE to close
down the SIP dialog and all other MRCP channels.

5. MRCPv2 Specification

   MRCPv2 messages are textual using the ISO 10646 character set in the
   UTF-8 encoding (RFC3629 [RFC3629]) to allow many different languages
   to be represented. However, to assist in compact representations,
   MRCPv2 also allows message bodies to be represented in other
   character sets such as ISO 8859-1 [ISO.8859-1.1987]. This may be
   useful for languages such as Chinese where the default character set
   for most documents is not UTF-8. The MRCPv2 protocol headers (the
   first line of an MRCP message) and header 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
   MRCPv2 as a whole.

   Lines are terminated by CRLF. Also, some parameters in the message
   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.

5.1. Common Protocol Elements

   The MRCPv2 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, 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 / response-line / event-line

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

resource-header   =       synthesizer-header
                      /   recognizer-header
                      /   recorder-header
                      /   verifier-header
```

The message-body contains resource-specific and message-specific
data. The actual Media Types used to carry the data are specified
later in the sections defining the individual messages. Generic
header fields are described in Section 6.2.

If a message contains a message body, the message MUST contain
content-headers indicating the Media Type and encoding of the data in
the message body.

Request, response and event messages (described in following
sections) include the version of MRCP that the message conforms to.
Version compatibility rules follow [H3.1] regarding version ordering,
compliance requirements, and upgrading of version numbers. The
version information is indicated by "MRCP" (as opposed to "HTTP" in
[H3.1]) or "MRCP/2.0" (as opposed to "HTTP/1.1" in [H3.1]). To be
compliant with this specification, clients and servers sending MRCPv2
messages MUST indicate an mrcp-version of "MRCP/2.0". ABNF
productions using mrcp-version can be found in Section 5.2,
Section 5.3, and Section 5.5.

```
mrcp-version      =     "MRCP" "/" 1*2DIGIT "." 1*2DIGIT
```

The message-length field specifies the length of the message in
octets, including the start-line, and MUST be the 2nd token from the
beginning of the message. This is to make the framing and parsing of
the message simpler to do. This field specifies the length of the
message including data that may be encoded into the body of the
message. Note that this value MAY be printed as a fixed-length
integer that is zero-padded in front in order to eliminate or reduce
inefficiency in cases where the message-length value would change as
a result of the length of the message-length token itself. This

value, as with all lengths in MRCP, is to be interpreted as a base-10
number. In particular, leading zeros do not indicate that the value
is to be interpreted as a base-8 number.

```
message-length =    1*19DIGIT
```

The following sample MRCP exchange demonstrates proper message-length
values. The values for message-length have been removed from all
other examples in the specification and replaced by '...' to reduce
confusion in the case of minor message-length computation errors in
those examples.

```
C->S:   MRCP/2.0 877 INTERPRET 543266
        Channel-Identifier:32AECB23433801@speechrecog
        Interpret-Text:may I speak to Andre Roy
        Content-Type:application/srgs+xml
        Content-ID:<request1@form-level.store>
        Content-Length:661

        <?xml version="1.0"?>
        <!-- the default grammar language is US English -->
        <grammar xmlns="http://www.w3.org/2001/06/grammar"
                xml:lang="en-US" version="1.0" root="request">
        <!-- 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/2.0 82 543266 200 IN-PROGRESS
        Channel-Identifier:32AECB23433801@speechrecog

S->C:   MRCP/2.0 634 INTERPRETATION-COMPLETE 543266 200 COMPLETE
        Channel-Identifier:32AECB23433801@speechrecog
        Completion-Cause:000 success
        Content-Type:application/nlsml+xml
```

```
        Content-Length:441

        <?xml version="1.0"?>
        <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                xmlns:ex="http://www.example.com/example"
                grammar="session:request1@form-level.store">
           <interpretation>
              <instance name="Person">
                 <ex:Person>
                     <ex:Name> Andre Roy </ex:Name>
                 </ex:Person>
              </instance>
              <input>   may I speak to Andre Roy </input>
           </interpretation>
        </result>
```

All MRCPv2 messages, responses and events MUST carry the Channel-
Identifier header field so the server or client can differentiate
messages from different control channels that may share the same
transport connection.

In the resource-specific header field descriptions in sections 8-11,
a header field is disallowed on a method (request, response, or
event) for that resource unless specifically listed as being allowed.
Also, the phrasing "This header field MAY occur on method X"
indicates that the header field is allowed on that method but is not
required to be used in every instance of that method.

5.2. Request

An MRCPv2 request consists of a Request line followed by the message
header section 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 the protocol in use.

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

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

The message-length field specifies the length of the message,
including the start-line.

Details about the mrcp-version and message-length fields are given in

Section 5.1.

The method-name field identifies the specific request that the client
is making to the server. Each resource supports a subset of the
MRCPv2 methods. The subset for each resource is defined in the
section of the specification for the corresponding resource.

```
method-name    =    generic-method
               /    synthesizer-method
               /    recognizer-method
               /    recorder-method
               /    verifier-method
```

The request-id field is a unique identifier representable as an
unsigned 32 bit integer created by the client and sent to the server.
Consecutive requests within an MRCP session MUST utilize
monotonically increasing request-id's. The request-id space is
linear, (i.e. not mod(32)) so the space does not wrap and validity
can be checked with a simple unsigned comparison operation. The
client may choose any initial value for its first request, but a
small integer is RECOMMENDED to avoid exhausting the space in long
sessions. If the server receives duplicate or out-of-order requests
the server MUST reject the request with a response code of 410.
Since request-id's are scoped to the MRCP session, they are unique
across all TCP connections and all resource channels in the session.

The server resource MUST use the client-assigned identifier in its
response to the request. If the request does not complete
synchronously, future asynchronous events associated with this
request MUST carry the client-assigned request-id.

```
request-id     =    1*10DIGIT
```

5.3. Response

After receiving and interpreting the request message for a method,
the server resource responds with an MRCPv2 response message. The
response consists of a response line followed by the message header
section and an optional message body containing data specific to the
method.

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

The mrcp-version field MUST contain the version of the request if
supported; otherwise, it must contain the highest version of the
MRCPv2 protocol supported by the server.

The message-length field specifies the length of the message, including the start-line.

Details about the mrcp-version and message-length fields are given in Section 5.1.

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.

```
status-code     =     3DIGIT
```

The request-state field indicates if the action 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 or other messages from that resource to the client with that request-id. The PENDING status means that the request 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 may be delivered with that request-id.

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

5.4. Status Codes

The status codes are classified under the Success (2XX) codes, Client Failure (4XX) codes, and Server Failure (5XX).

                            Success Codes

```
+------------+----------------------------------------------------+
| Code       | Meaning                                            |
+------------+----------------------------------------------------+
| 200        | Success                                            |
| 201        | Success with some optional header fields ignored   |
+------------+----------------------------------------------------+
```

                            Success 2xx

Client Failure 4xx Codes

+-----------+--+
| Code | Meaning |
+-----------+--+
401	Method not allowed
402	Method not valid in this state
403	Unsupported header field
404	Illegal value for header field. This is the error
	for a syntax violation.
405	Resource not allocated for this session or does not
	exist
406	Mandatory Header Field Missing
407	Method or Operation Failed (e.g., Grammar
	compilation failed in the recognizer. Detailed
	cause codes might be available through a resource
	specific header.)
408	Unrecognized or unsupported message entity
409	Unsupported Header Field Value. This is a value
	that is syntactically legal but exceeds the
	implementation's capabilities or expectations.
410	Non-Monotonic or Out of order sequence number in
	request.
411-420	Reserved for future assignment
+-----------+--+

Client Failure 4xx

Server Failure 5xx Codes

+-----------+---------------------------------+
| Code | Meaning |
+-----------+---------------------------------+
501	Server Internal Error
502	Protocol Version not supported
503	Reserved for future assignment
504	Message too large
+-----------+---------------------------------+

Server Failure 4xx

5.5. Events

   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. The event message consists of an event
header line followed by the message header section and an optional
message body containing data specific to the event message. The
header line has the request-id of the corresponding request and
status value. The request-state value is COMPLETE if the request is
done and this was the last event, else it is IN-PROGRESS.

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

The mrcp-version used here is identical to the one used in the
Request/Response Line and indicates the version of the MRCPv2
protocol running on the server.

The message-length field specifies the length of the message,
including the start-line.

Details about the mrcp-version and message-length fields are given in
Section 5.1.

The event-name identifies the nature of the event generated by the
media resource. The set of valid event names depends on the resource
generating it. See the corresponding resource-specific section of
the document.

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

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

The request-state indicates whether the Request/Command causing this
event is complete or still in progress, and is the same as the one
mentioned in Section 5.3. The final event for a request has a
COMPLETE status indicating the completion of the request.

6. MRCPv2 Generic Methods, Headers, and Result Structure

   MRCPv2 supports a set of methods and header fields that are common to
   all resources. These are discussed here; resource-specific methods
   and header fields are discussed in the corresponding resource-
   specific section of the document.

6.1. Generic Methods

   MRCPv2 supports two generic methods for reading and writing the state
   associated with a resource.

   generic-method   =   "SET-PARAMS"
                       /   "GET-PARAMS"

   These are described in the following sub-sections.

6.1.1. SET-PARAMS

   The "SET-PARAMS" method, from the client to the server, tells the
   MRCPv2 resource to define parameters for the session, such as voice
   characteristics and prosody on synthesizers, recognition timers on
   recognizers, etc. If the server accepts and sets all parameters it
   MUST return a response status-code of 200. If it chooses to ignore
   some optional header fields that can be safely ignored without
   affecting operation of the server it MUST return 201.

   If one or more of the header fields being sent is incorrect, error
   403, 404, or 409 MUST be returned as follows:
   o If one or more of the header fields being set has an illegal
      value, the server MUST reject the request with a 404 Illegal Value
      for Header Field.
   o If one or more of the header fields being set is unsupported for
      the resource, the server MUST reject the request with a 403
      Unsupported Header Field, except as described in the next
      paragraph.
   o If one or more of the header fields being set has an unsupported
      value, the server MUST reject the request with a 409 Unsupported
      Header Field Value, except as described in the next paragraph.

   If both error 404 and another error have occurred, only error 404
   MUST be returned. If both errors 403 and 409 have occurred, but not
   error 404, only error 403 MUST be returned.

   If error 403, 404, or 409 is returned, the response MUST include the
   bad or unsupported header fields and their values exactly as they
   were sent from the client. Session parameters modified using
   "SET-PARAMS" do not override parameters explicitly specified on
   individual requests or requests that are in-PROGRESS.

```
C->S:   MRCP/2.0 ... SET-PARAMS 543256
        Channel-Identifier:32AECB23433802@speechsynth
        Voice-gender:female
        Voice-variant:3

S->C:   MRCP/2.0 ... 543256 200 COMPLETE
        Channel-Identifier:32AECB23433802@speechsynth
```

6.1.2. GET-PARAMS

The "GET-PARAMS" method, from the client to the server, asks the
MRCPv2 resource for its current session parameters, such as voice
characteristics and prosody on synthesizers, recognition-timer on
recognizers, etc. For every header field the client sends in the
request without a value, the server MUST include the corresponding
header fields and their values in the response. If no parameter
header fields are specified by the client then the server MUST return
all the settable parameters and their values in the corresponding
header section of the response, including vendor-specific parameters.
Such wild-card parameter requests can be very processing-intensive,
since the number of settable parameters can be large depending on the
implementation. Hence, it is RECOMMENDED that the client not use the
wildcard "GET-PARAMS" operation very often. Note that "GET-PARAMS"
returns header field values that apply to the whole session and not
values that have a request level scope. For example, Input-Waveform-
URI is a request-level header field and thus would not be returned by
GET-PARAMS.

If all of the header fields requested are supported, the server MUST
return a response status-code of 200. If some of the header fields
being retrieved are unsupported for the resource, the server MUST
reject the request with a 403 Unsupported Header Field. Such a
response MUST include the unsupported header fields exactly as they
were sent from the client, without values.

```
C->S:   MRCP/2.0 ... GET-PARAMS 543256
        Channel-Identifier:32AECB23433802@speechsynth
        Voice-gender:
        Voice-variant:
        Vendor-Specific-Parameters:com.example.param1;
                    com.example.param2

S->C:   MRCP/2.0 ... 543256 200 COMPLETE
        Channel-Identifier:32AECB23433802@speechsynth
        Voice-gender:female
        Voice-variant:3
        Vendor-Specific-Parameters:com.example.param1="Company Name";
                    com.example.param2="124324234@example.com"
```

6.2. Generic Message Headers

   All MRCPv2 header fields, which include both the generic-headers
   defined in the following sub-sections and the resource-specific
   header fields defined later, follow the same generic format as that
   given in Section 3.1 of RFC5322 [RFC5322]. Each header field
   consists of a name followed by a colon (":") and the value. Header
   field names are case-insensitive. The value MAY be preceded by any
   amount of LWS (linear white space), though a single SP (space) is
   preferred. Header fields may extend over multiple lines by preceding
   each extra line with at least one SP or HT (horizontal tab).

   generic-field = field-name ":" [field-value]
   field-name   = token
   field-value   = *LWS field-content *(CRLF 1*LWS field-content)
   field-content = <the OCTETs making up the field-value
                     and consisting of either *TEXT or combinations
                     of token, separators, and quoted-string>

   The field-content does not include any leading or trailing LWS (i.e.
   linear white space occurring before the first non-whitespace
   character of the field-value or after the last non-whitespace
   character of the field-value). Such leading or trailing LWS MAY be
   removed without changing the semantics of the field value. Any LWS
   that occurs between field-content MAY be replaced with a single SP
   before interpreting the field value or forwarding the message
   downstream.

   MRCPv2 servers and clients MUST NOT depend on header field order. 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. However, MRCPv2 servers and clients MUST be prepared
   to process the header fields in any order. The only exception to
   this rule is when there are multiple header fields with the same name
   in a message.

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

   Since vendor-specific parameters may be order-dependent, it MUST be
   possible to combine multiple header fields of the same name into one
   "name:value" pair without changing the semantics of the message, by
   appending each subsequent value to the first, each separated by a
   comma. The order in which header fields with the same name are
   received is therefore significant to the interpretation of the
   combined header field value, and thus an intermediary MUST NOT change
   the order of these values when a message is forwarded.

```
generic-header       =     channel-identifier
                     /     accept
                     /     active-request-id-list
                     /     proxy-sync-id
                     /     accept-charset
                     /     content-type
                     /     content-id
                     /     content-base
                     /     content-encoding
                     /     content-location
                     /     content-length
                     /     fetch-timeout
                     /     cache-control
                     /     logging-tag
                     /     set-cookie
                     /     set-cookie2
                     /     vendor-specific
```

6.2.1. Channel-Identifier

All MRCPv2 requests, responses and events MUST contain the Channel-
Identifier header field. The value is allocated by the server when a
control channel is added to the session and communicated to the
client by the "a=channel" attribute in the SDP answer from the
server. The header field value consists of 2 parts separated by the
'@' symbol. The first part is an unambiguous string identifying the
MRCPv2 session. The second part is a string token which specifies
one of the media processing resource types listed in Section 3.1.
The unambiguous string (first part) MUST be unique among the resource
instances managed by the server and is common to all resource
channels with that server established through a single SIP dialog.

```
channel-identifier  = "Channel-Identifier" ":" channel-id CRLF
channel-id          = 1*alphanum "@" 1*alphanum
```

6.2.2. Accept

The Accept header field follows the syntax defined in [H14.1]. The
semantics are also identical, with the exception that if no Accept
header field is present, the server MUST assume a default value that
is specific to the resource type that is being controlled. This
default value can be changed for a resource on a session by sending
this header field in a SET-PARAMS method. The current default value
of this header field for a resource in a session can be found through
a GET-PARAMS method. This header field MAY occur on any request.

6.2.3. Active-Request-Id-List

   In a request, this header field indicates the list of request-ids to
   which the request applies. This is useful when there are multiple
   requests that are PENDING or IN-PROGRESS and the client wants this
   request to apply to one or more of these specifically.

   In a response, this header field returns the list of request-ids that
   the method modified or affected. There could be one or more requests
   in 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 or modified by this command in its
   header section.

   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 which has one or more "SPEAK" requests
   in the PENDING or IN-PROGRESS state, all "SPEAK" requests MUST be
   cancelled, including the one IN-PROGRESS. The response to the "STOP"
   request contains in the Active-Request-Id-List value the request-ids
   of all the "SPEAK" requests that were terminated. After sending the
   STOP response, the server MUST NOT send any SPEAK-COMPLETE or
   RECOGNITION-COMPLETE events for the terminated requests.

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

6.2.4. Proxy-Sync-Id

   When any server resource generates a barge-in-able event, it also
   generates a unique tag. The tag is sent as this header field's value
   in an event to the client. The client then acts as a intermediary
   among the server resources and sends a BARGE-IN-OCCURRED method 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, intermediated
   by the client, to decide if this event has been processed through a
   direct interaction of the resources. This header field MAY occur
   only on events and the BARGE-IN-OCCURRED method. The name of this
   header field contains the word 'proxy' only for historical reasons
   and does not imply that a proxy server is involved.

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

6.2.5. Accept-Charset

   See [H14.2]. This specifies the acceptable character sets 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 Semantic Markup Language (NLSML) results of a
   "RECOGNITION-COMPLETE" event. This header field is only used on
   requests.

6.2.6. Content-Type

   See [H14.17]. MRCPv2 supports a restricted set of registered Media
   Types for content, including speech markup, grammar, and recognition
   results. The content types applicable to each MRCPv2 resource-type
   are specified in the corresponding section of the document and are
   registered in the MIME Media Types registry maintained by IANA. The
   multi-part content type "multi-part/mixed" is supported to
   communicate multiple of the above mentioned contents, in which case
   the body parts MUST NOT contain any MRCPv2 specific header fields.
   This header field MAY occur on all messages.

   content-type   =   "Content-Type" ":" media-type-value CRLF

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

   type   =   token

   subtype   =   token

   parameter   =   attribute "=" value

   attribute   =   token

   value   =   token / quoted-string

6.2.7. Content-ID

   This header field contains an ID or name for the content by which it
   can be referenced. This header field operates according to the
   specification in RFC 2392 [RFC2392] and is required for content
   disambiguation in multi-part messages. In MRCPv2 whenever the
   associated content is stored, by either the client or the server, it
   MUST be retrievable using this ID. Such content can be referenced
   later in a session by addressing it with the "session" URI scheme
   described in Section 13.6. This header field MAY occur on all
   messages.

6.2.8. Content-Base

The content-base entity-header may be used to specify the base URI
for resolving relative URIs 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 multi-part media, which in turn can have multiple entities
within it. This header field MAY occur on all messages.

6.2.9. Content-Encoding

The Content-Encoding entity-header is used as a modifier to the
Content-Type. When present, its value indicates what additional
content encoding 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. Note that the SDP session
can be used to determine accepted encodings (see Section 7). This
header field MAY occur on all messages.

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

Content-Encoding 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
encodings MUST be listed in the order in which they were applied.

6.2.10. Content-Location

The Content-Location entity-header 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. Refer to [H14.14].

```
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. This header field is provided for optimization purposes
only. The receiver of this header field MAY assume that the entity
being sent is identical to what would have been retrieved or might
already have been 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 this 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. This header field MAY
occur on all messages.

6.2.11. Content-Length

This header field contains the length of the content of the message
body (i.e. after the double CRLF (carriage return, then line feed)
following the last header field). Unlike in HTTP, it MUST be
included in all messages that carry content beyond the header
section. If it is missing, a default value of zero is assumed.
Otherwise, it is interpreted according to [H14.13]. When a message
having no use for a message body contains one, i.e. the Content-
Length is non-zero, the receiver MUST ignore the content of the
message body. This header field MAY occur on all messages.

   content-length = "Content-Length" ":" 1*19DIGIT CRLF

6.2.12. Fetch Timeout

When the recognizer or synthesizer needs to fetch documents or other
resources this header field controls the corresponding URI access
properties. This defines the timeout for content that the server may
need to fetch over the network. The value is interpreted to be in
milliseconds and ranges from 0 to an implementation-specific maximum
value. The default value for this header field is implementation-
specific. This header field MAY occur in "DEFINE-GRAMMAR",
"RECOGNIZE", "SPEAK", "SET-PARAMS" or "GET-PARAMS".

   fetch-timeout   =   "Fetch-Timeout" ":" 1*19DIGIT CRLF

6.2.13. Cache-Control

   If the server implements content caching, it MUST adhere to the cache
   correctness rules of HTTP 1.1 [RFC2616] when accessing and caching
   stored content. In particular, the "expires" and "cache-control"
   header fields of the cached URI or document MUST be honored and 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 server for the session or request. The scope of
   the directive is based on the method it is sent on. If the directive
   is sent on a "SET-PARAMS" method, it applies for all requests for
   external documents the server makes during that session, unless
   overridden by a Cache-Control header field on an individual request.
   If the directives are sent on any other requests they apply only to
   external document requests the server makes for that request. An
   empty Cache-Control header field on the "GET-PARAMS" method is a
   request for the server to return the current Cache-Control directives
   setting on the server. This header field MAY occur only on requests.

       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*19DIGIT

   Here delta-seconds is a decimal time value specifying the number of
   seconds since the instant the message response or data was received
   by the server.

   The different cache-directive options allow the client to ask the
   server to override the default cache expiration mechanisms:
   max-age   Indicates that the client can tolerate the server
                  using content 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 a response based on stale data.
   min-fresh   Indicates that the client is willing to accept a
                  server response with cached data whose expiration is
                  no less than its current age plus the specified time
                  in seconds. If the server's cache time to live
                  exceeds the client-supplied min-fresh value, the
                  server MUST NOT utilize cached content.

max-stale   Indicates that the client is willing to allow a server
                   to utilize cached data that has exceeded its
                   expiration time. If "max-stale" is assigned a value,
                   then the client is willing to allow the server to use
                   cached data 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 allow the server to use stale data of any
                   age.

The 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 to
the corresponding URI).

If both the MRCPv2 Cache-Control directive and the cached entry on
the 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.

6.2.14. Logging-Tag

This header field MAY be sent as part of a "SET-PARAMS"/"GET-PARAMS"
method to set or retrieve the logging tag for logs generated by the
server. Once set, the value persists until a new value is set or the
session ends. The MRCPv2 server MAY 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.

It is RECOMMENDED that clients have some identifying information in
the logging tag, so that one can determine which client request
generated a given log message at the server.

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

6.2.15. Set-Cookie and Set-Cookie2

Since the associated HTTP client on an MRCPv2 server fetches
documents for processing on behalf of the MRCPv2 client, the cookie
store in the HTTP client of the MRCPv2 server is treated as an
extension of the cookie store in the HTTP client of the MRCPv2
client. This requires that the MRCPv2 client and server be able to
synchronize their common cookie store as needed. To enable the
MRCPv2 client to push its stored cookies to the MRCPv2 server and get
new cookies from the MRCPv2 server stored back to the MRCPv2 client,
the Set-Cookie and Set-Cookie2 entity-header fields MAY be included

   in MRCPv2 requests to update the cookie store on a server and be
   returned in final MRCPv2 responses or events to subsequently update
   the client's own cookie store. The stored cookies on the server
   persist for the duration of the MRCPv2 session and MUST be destroyed
   at the end of the session. To ensure support for the type of cookie
   header field dictated by the HTTP origin server, MRCPv2 clients and
   servers MUST support both the Set-Cookie and Set-Cookie2 entity
   header fields.

```
   set-cookie      =       "Set-Cookie:" cookies CRLF
   cookies         =       cookie *("," *LWS cookie)
   cookie          =       attribute "=" value *(";" cookie-av)
   cookie-av       =       "Comment" "=" value
                   /       "Domain" "=" value
                   /       "Max-Age" "=" value
                   /       "Path" "=" value
                   /       "Secure"
                   /       "Version" "=" 1*19DIGIT
                   /       "Age" "=" delta-seconds

   set-cookie2     =       "Set-Cookie2:" cookies2 CRLF
   cookies2        =       cookie2 *("," *LWS cookie2)
   cookie2         =       attribute "=" value *(";" cookie-av2)
   cookie-av2      =       "Comment" "=" value
                   /       "CommentURL" "=" DQUOTE uri DQUOTE
                   /       "Discard"
                   /       "Domain" "=" value
                   /       "Max-Age" "=" value
                   /       "Path" "=" value
                   /       "Port" [ "=" DQUOTE portlist DQUOTE ]
                   /       "Secure"
                   /       "Version" "=" 1*19DIGIT
                   /       "Age" "=" delta-seconds
   portlist        =       portnum *("," *LWS portnum)
   portnum         =       1*19DIGIT
```

   The Set-Cookie and Set-Cookie2 header fields are specified in RFC
   2109 [RFC2109] and RFC 2965 [RFC2965], respectively. The "Age"
   attribute is introduced in this specification to indicate the age of
   the cookie and is optional. An MRCPv2 client or server MUST
   calculate the age of the cookie according to the age calculation
   rules in the HTTP/1.1 specification [RFC2616] and append the "Age"
   attribute accordingly.

   The MRCPv2 client or server MUST supply defaults for the "Domain" and
   "Path" attributes if omitted by the HTTP origin server as specified
   in RFC 2109 (set-cookie) and RFC 2965 (set-cookie2). Note that there
   is no leading dot present in the "Domain" attribute value in this

case. Although an explicitly specified "Domain" value received via
the HTTP protocol may be modified to include a leading dot, an MRCPv2
client or server MUST NOT modify the "Domain" value when received via
the MRCPv2 protocol.

An MRCPv2 client or server MAY combine multiple cookie header fields
of the same type into a single "field-name:field-value" pair as
described in Section 6.2.

The Set-Cookie and Set-Cookie2 header fields MAY be specified in any
request that subsequently results in the server performing an HTTP
access. When a server receives new cookie information from an HTTP
origin server, and assuming the cookie store is modified according
RFC 2109 or RFC 2965, the server MUST return the new cookie
information in the MRCPv2 COMPLETE response or event as appropriate
to allow the client to update its own cookie store.

The "SET-PARAMS" request MAY specify the Set-Cookie and Set-Cookie2
header fields to update the cookie store on a server. The GET-PARAMS
request MAY be used to return the entire cookie store of "Set-Cookie"
or "Set-Cookie2" type cookies to the client.

6.2.16. Vendor Specific Parameters

This set of header fields allows for the client to set or retrieve
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 "="
                              value

vendor-av-pair-name      = 1*UTFCHAR
```

Header fields of this form MAY be sent in any method (request) and
are used to manage implementation-specific parameters on the server
side. The vendor-av-pair-name follows the reverse Internet Domain
Name convention (see Section 13.1.6 for syntax and registration
information). The value of the vendor attribute is specified after
the "=" symbol and MAY be quoted. For example:

com.example.companyA.paramxyz=256
com.example.companyA.paramabc=High
com.example.companyB.paramxyz=Low

When used in GET-PARAMS to get the current value of these parameters
from the server, this header field value may contain a semicolon-
separated list of implementation-specific attribute names.

6.3. Generic Result Structure

Result data from the server for the Recognizer and Verifier resources
is carried as a typed media entity in the MRCPv2 message body of
various events. The Natural Language Semantics Markup Language
(NLSML), an XML markup based on an early draft from the W3C, is the
default standard for returning results back to the client. Hence,
all servers implementing these resource types MUST support the Media
Type application/nlsml+xml. The Extensible MultiModal Annotation
(EMMA) [W3C.REC-emma-20090210] format can be used to return results
as well. This can be done by negotiating the format at session
establishment time with SDP (a=resultformat:application/emma+xml) or
with SIP (Allow/Accept). With SIP, for example, if a client wants
results in EMMA, an MRCPv2 server can route the request to another
server that supports EMMA by inspecting the SIP header fields, rather
than having to introspect into the SDP.

MRCPv2 uses this representation to convey content among the clients
and servers that generate and make use of the markup. MRCPv2 uses
NSLML specifically to convey recognition, enrollment, and
verification results between the corresponding resource on the MRCPv2
server and the MRCPv2 client. Details of this result format are
fully described in Section 6.3.1.

```
Content-Type:application/nlsml+xml
Content-Length:...

<?xml version="1.0"?>
<result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
        xmlns:ex="http://www.example.com/example"
        grammar="http://theYesNoGrammar">
    <interpretation>
        <instance>
                <ex:response>yes</ex:response>
        </instance>
        <input>ok</input>
    </interpretation>
</result>
```

                          Result Example

6.3.1. Natural Language Semantics Markup Language

   The Natural Language Semantics Markup Language (NLSML) is an XML data
   structure with elements and attributes designed to carry result
   information from recognizer (including enrollment) and verifier
   resources. The normative definition of NLSML is the RelaxNG schema
   in Section 16.1. Note that the elements and attributes of this
   format are defined in the MRCPv2 namespace. In the result structure,
   they must either be prefixed by a namespace prefix declared within
   the result or must be children of an element identified as belonging
   to the respective namespace. For details on how to use XML
   Namespaces, see [W3C.REC-xml-names11-20040204]. Section 2 of
   [W3C.REC-xml-names11-20040204] provides details on how to declare
   namespaces and namespace prefixes.

   The root element of NLSML is <result>. Optional child elements are
   <interpretation>, <enrollment-result>, and <verification-result>, at
   least one of which must be present. A single <result> may contain
   all of the optional child elements. Details of the <result> and
   <interpretation> elements and their subelements and attributes can be
   found in Section 9.6. Details of the <enrollment-result> element and
   its subelements can be found in Section 9.7. Details of the
   <verification-result> element and its subelements can be found in
   Section 11.5.2.

7. Resource Discovery

   Server resources may be discovered and their capabilities learned by
   clients through standard SIP machinery. The client MAY issue a SIP
   OPTIONS transaction to a server, which has the effect of requesting
   the capabilities of the server. The server MUST respond to such a
   request with an SDP-encoded description of its capabilities according
   to RFC3264 [RFC3264]. The MRCPv2 capabilities are described by a
   single m-line containing the media type "application" and transport
   type "TCP/TLS/MRCPv2" or "TCP/MRCPv2". There MUST be one "resource"
   attribute for each media resource that the server supports with the
   resource type identifier as its value.

   The SDP description MUST also contain m-lines describing the audio
   capabilities and the coders the server supports.

In this example, the client uses the SIP OPTIONS method to query the
capabilities of the MRCPv2 server.

C->S:
```
OPTIONS sip:mrcp@server.example.com SIP/2.0
Via:SIP/2.0/TCP client.atlanta.example.com:5060;
 branch=z9hG4bK74bf7
Max-Forwards:6
To:<sip:mrcp@example.com>
From:Sarvi <sip:sarvi@example.com>;tag=1928301774
Call-ID:a84b4c76e66710
CSeq:63104 OPTIONS
Contact:<sip:sarvi@client.example.com>
Accept:application/sdp
Content-Length:0
```

S->C:
```
SIP/2.0 200 OK
Via:SIP/2.0/TCP client.atlanta.example.com:5060;
 branch=z9hG4bK74bf7;received=192.0.32.10
To:<sip:mrcp@example.com>;tag=62784
From:Sarvi <sip:sarvi@example.com>;tag=1928301774
Call-ID:a84b4c76e66710
CSeq:63104 OPTIONS
Contact:<sip:mrcp@server.example.com>
Allow:INVITE, ACK, CANCEL, OPTIONS, BYE
Accept:application/sdp
Accept-Encoding:gzip
Accept-Language:en
Supported:foo
Content-Type:application/sdp
Content-Length:...

v=0
o=sarvi 2890844536 2890842811 IN IP4 192.0.2.12
s=-
i=MRCPv2 server capabilities
c=IN IP4 192.0.2.12/127
t=0 0
m=application 0 TCP/TLS/MRCPv2 1
a=resource:speechsynth
a=resource:speechrecog
a=resource:speakverify
m=audio 0 RTP/AVP 0 3
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
```

Using SIP OPTIONS for MRCPv2 Server Capability Discovery

8. Speech Synthesizer Resource

   This resource processes text markup provided by the client and
   generates a stream of synthesized speech in real-time. Depending
   upon the server implementation and capability of this resource, the
   client can also dictate parameters of the synthesized speech such as
   voice characteristics, speaker speed, etc.

   The synthesizer resource is controlled by MRCPv2 requests from the
   client. Similarly, the resource can respond to these requests or
   generate asynchronous events to the client to indicate conditions of
   interest to the client during the generation of the synthesized
   speech stream.

   This section applies for the following resource types:
   o speechsynth
   o basicsynth

   The capabilities of these resources are defined in Section 3.1.

8.1. Synthesizer State Machine

   The synthesizer maintains a state machine to process MRCPv2 requests
   from the client. The state transitions shown below describe the
   states of the synthesizer and reflect the state of the request at the
   head of the synthesizer resource queue. A "SPEAK" request in the
   PENDING state can be deleted or stopped by a "STOP" request without
   affecting the state of the resource.
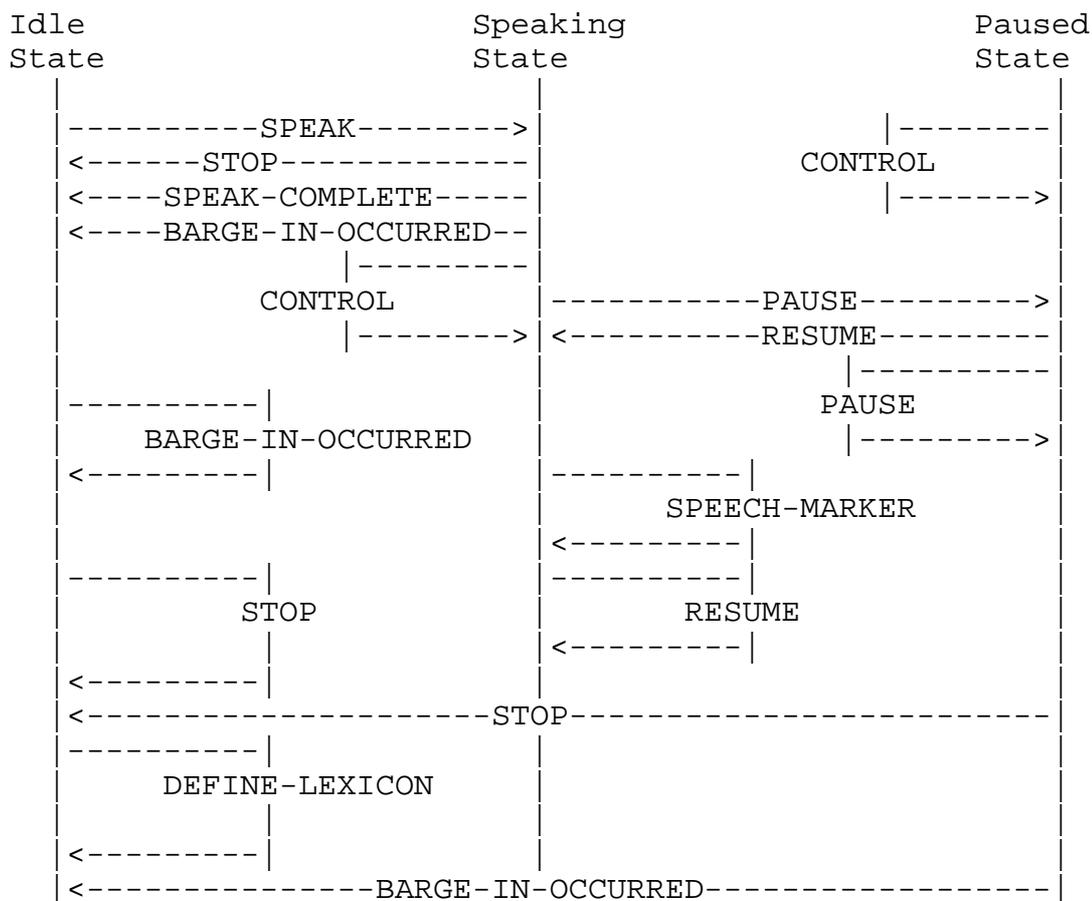
```
         Idle                    Speaking                   Paused
         State                    State                     State
           |                        |                        |
           |----------SPEAK-------->|            |---------|
           |<------STOP-------------|          CONTROL      |
           |<----SPEAK-COMPLETE-----|            |-------->|
           |<----BARGE-IN-OCCURRED--|                        |
           |             |---------|                        |
           |           CONTROL      |----------PAUSE--------->|
           |             |-------->|<----------RESUME---------|
           |                        |                        |---------|
           |----------|            |                      PAUSE       |
           |     BARGE-IN-OCCURRED  |            |-------->|
           |<---------|            |----------|                        |
           |                        |    SPEECH-MARKER              |
           |                        |<---------|                        |
           |----------|            |----------|                        |
           |     STOP               |    RESUME                        |
           |          |            |<---------|                        |
           |<---------|            |                        |
           |<-------------------STOP------------------------|
           |----------|            |                        |
           |     DEFINE-LEXICON     |                        |
           |          |            |                        |
           |<---------|            |                        |
           |<--------------BARGE-IN-OCCURRED----------------|

                        Synthesizer State Machine
```

8.2. Synthesizer Methods

   The synthesizer supports the following methods.

```
   synthesizer-method   =   "SPEAK"
                        /   "STOP"
                        /   "PAUSE"
                        /   "RESUME"
                        /   "BARGE-IN-OCCURRED"
                        /   "CONTROL"
                        /   "DEFINE-LEXICON"
```

8.3. Synthesizer Events

   The synthesizer may generate the following events.

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

8.4. Synthesizer Header Fields

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

   synthesizer-header = jump-size
                       / kill-on-barge-in
                       / speaker-profile
                       / completion-cause
                       / completion-reason
                       / voice-parameter
                       / prosody-parameter
                       / speech-marker
                       / speech-language
                       / fetch-hint
                       / audio-fetch-hint
                       / failed-uri
                       / failed-uri-cause
                       / speak-restart
                       / speak-length
                       / load-lexicon
                       / lexicon-search-order

8.4.1. Jump-Size

   This header field MAY be specified in a CONTROL method and controls
   the amount to jump forward or backward in an active "SPEAK" request.
   A + or - indicates a relative value to what is being currently
   played. This header field MAY also be specified in a "SPEAK" request
   as a desired offset into the synthesized speech. In this case, the
   synthesizer MUST begin speaking from this amount of time into the
   speech markup. Note that an offset that extends beyond the end of
   the produced speech will result in audio of length zero. The
   different speech length units supported are dependent on the
   synthesizer implementation. If the synthesizer resource does not
   support a unit or the operation, the resource MUST respond with a
   status-code of 409 "Unsupported Header Field Value".

```
jump-size                =    "Jump-Size" ":" speech-length-value CRLF

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

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

numeric-speech-length =       ("+" / "-") positive-speech-length

positive-speech-length =      1*19DIGIT SP numeric-speech-unit

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

8.4.2. Kill-On-Barge-In

This header field 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 CRLF
```

The client MUST send a BARGE-IN-OCCURRED method to the synthesizer
resource when it receives a barge-in-able event from any source.
This source could be a synthesizer resource or signal detector
resource and MAY be either local or distributed. If this header
field is not specified in a "SPEAK" request or explicitly set by a
"SET-PARAMS", the default value for this header field is "true".

If the recognizer or signal detector resource is on the same server
as the synthesizer and both are part of the same session, the server
MAY work with both to provide internal notification to the
synthesizer so that audio may be stopped without having to wait for
the client's BARGE-IN-OCCURRED event.

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).

8.4.3. Speaker Profile

   This header field MAY be part of the "SET-PARAMS"/"GET-PARAMS" or
   "SPEAK" request from the client to the server and specifies a URI
   which references the profile of the speaker. Speaker profiles are
   collections of voice parameters like gender, accent etc.

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

8.4.4. 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 the "SPEAK" request completed.

   completion-cause   =   "Completion-Cause" ":" 3DIGIT SP
                               1*VCHAR 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.
006	lexicon-load-failure	Lexicon loading failed.
007	cancelled	A prior SPEAK request failed
		while this one was still in
		the queue.
   +------------+----------------------+------------------------------+

              Synthesizer Resource Compleion Cause Codes

8.4.5. Completion Reason

   This header field MAY be specified in a "SPEAK-COMPLETE" event coming
   from the synthesizer resource to the client. This contains the
   reason text behind the "SPEAK" request completion. This header field
   communicates text describing the reason for the failure, such as an
   error in parsing the speech markup text.

   completion-reason   =   "Completion-Reason" ":"
                            quoted-string CRLF

   The completion reason text is provided for client use in logs and for
   debugging and instrumentation purposes. Clients MUST NOT interpret
   the completion reason text.

8.4.6. Voice-Parameter

   This set of header fields defines the voice of the speaker.

   voice-parameter   =   voice-gender
                       /   voice-age
                       /   voice-variant
                       /   voice-name

   voice-gender   =   "Voice-Gender:" voice-gender-value CRLF
   voice-gender-value =   "male"
                       /   "female"
                       /   "neutral"
   voice-age   =   "Voice-Age:" 1*3DIGIT CRLF
   voice-variant   =   "Voice-Variant:" 1*19DIGIT CRLF
   voice-name   =   "Voice-Name:"
                            1*UTFCHAR *(1*WSP 1*UTFCHAR) CRLF

   The Voice- parameters are derived from the similarly-named attributes
   of the voice element specified in W3C's Speech Synthesis Markup
   Language Specification (SSML) [W3C.REC-speech-synthesis-20040907].
   Legal values for these parameters are as defined in that
   specification.

   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.
   Note that SSML content can itself set these values internal to the
   SSML document, of course.

   Voice parameter header fields MAY 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
MUST reject the request with a status-code of 403 "Unsupported Header
Field".

8.4.7. Prosody-Parameters

   This set of header fields defines the prosody of the speech.

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

   prosody-param-name   =   1*VCHAR

   prosody-param-value   =   1*VCHAR

   prosody-param-name is any one of the attribute names under the
   prosody element specified in W3C's Speech Synthesis Markup Language
   Specification [W3C.REC-speech-synthesis-20040907]. 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 change its
   behavior on the fly. If the synthesizer resource does not support
   this operation, it MUST respond back to the client with a status-code
   of 403 "Unsupported Header Field".

8.4.8. Speech Marker

   This header field contains timestamp information in a "timestamp"
   field. This is a Network Time Protocol (NTP) [RFC5905] timestamp, a
   64 bit number in decimal form. It MUST be synced with the Real-Time
   Protocol (RTP) [RFC3550] timestamp of the media stream through the
   Real-Time Control Protocol (RTCP) [RFC3550].

   Markers are bookmarks that are defined within the markup. Most

speech markup formats provide mechanisms to embed marker fields within speech texts. The synthesizer generates SPEECH-MARKER events when it reaches these marker fields. This header field MUST be part of the SPEECH-MARKER event and contain the marker tag value after the timestamp, separated by a semicolon. In these events the timestamp marks the time the text corresponding to the marker was emitted as speech by the synthesizer.

This header field MUST also be returned in responses to STOP, CONTROL, and BARGE-IN-OCCURRED methods, in the "SPEAK-COMPLETE" event, and in an IN-PROGRESS SPEAK response. In these messages, if any markers have been encountered for the current SPEAK, the marker tag value MUST be the last embedded marker encountered. If no markers have yet been encountered for the current SPEAK, only the timestamp is REQUIRED. Note than in these events the purpose of this header field is to provide timestamp information associated with important events within the lifecycle of a request (start of SPEAK processing, end of SPEAK processing, receipt of CONTROL/STOP/ BARGE-IN-OCCURRED).

```
timestamp           =    "timestamp" "=" time-stamp-value

time-stamp-value    =    1*20DIGIT

speech-marker       =    "Speech-Marker" ":"
                         timestamp
                         [";" 1*(UTFCHAR / %x20)] CRLF
```

8.4.9. Speech Language

This header field specifies the default language of the speech data if the language is not specified in the markup. The value of this header field MUST follow RFC 5646 [RFC5646] for its values. The header field MAY occur in "SPEAK", "SET-PARAMS" or "GET-PARAMS" requests.

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

8.4.10. Fetch Hint

When the synthesizer needs to fetch documents or other resources like speech markup or audio files, this header field controls the corresponding URI access properties. This provides client policy on when the synthesizer should retrieve content from the server. A value of "prefetch" indicates the content MAY be downloaded when the request is received, whereas "safe" indicates that content MUST NOT be downloaded until actually referenced. The default value is "prefetch". This header field MAY occur in "SPEAK", "SET-PARAMS" or

"GET-PARAMS" requests.

```
fetch-hint           =    "Fetch-Hint" ":" ("prefetch" / "safe") CRLF
```

8.4.11. Audio Fetch Hint

When the synthesizer needs to fetch documents or other resources like
speech audio files, this header field controls the corresponding URI
access properties. This provides client policy whether or not the
synthesizer may attempt to optimize speech by pre-fetching audio.
The value is either "safe" to say that audio is only fetched when it
is referenced, never before; "prefetch" to permit, but not require
the implementation 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" ":"
                         ("prefetch" / "safe" / "stream") CRLF
```

8.4.12. Failed URI

When a synthesizer method needs a synthesizer to fetch or access a
URI and the access fails, the server SHOULD provide the failed URI in
this header field in the method response, unless there are multiple
URI failures, in which case one of the failed URIs MUST be provided
in this header field in the method response.

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

8.4.13. Failed URI Cause

When a synthesizer method needs a synthesizer to fetch or access a
URI and the access fails the server MUST provide the URI-specific or
protocol-specific response code for the URI in the Failed-URI header
field in the method response through this header field. The value
encoding is UTF-8 (RFC3629 [RFC3629]) to accommodate any access
protocol, some of which might have a response string instead of a
numeric response code.
```
failed-uri-cause    =    "Failed-URI-Cause" ":" 1*UTFCHAR CRLF
```

8.4.14. Speak Restart

When a CONTROL request to jump backward is issued to a currently
speaking synthesizer resource, and the target jump point is before
the start of the current "SPEAK" request, the current "SPEAK" request
MUST restart from the beginning of its speech data and the response
to the CONTROL request MUST contain this header field with a value of

"true" indicating a restart.

```
speak-restart        =    "Speak-Restart" ":" BOOLEAN CRLF
```

8.4.15. Speak Length

   This header field 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. If numeric, the value MUST be
   a positive integer. If a header field with a Tag unit is specified,
   then the speech output continues until the tag is reached or the
   "SPEAK" request complete, whichever comes first. This header field
   MAY be specified in a "SPEAK" request to indicate the length to speak
   from the speech data and is relative to the point in speech that the
   "SPEAK" request starts. The different speech length units supported
   are synthesizer implementation dependent. If a server does not
   support the specified unit, the resource MUST respond with a status-
   code of 409 "Unsupported Header Field Value".

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

positive-length-value =    positive-speech-length
                      /    text-speech-length

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

positive-speech-length =   1*19DIGIT SP numeric-speech-unit

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

8.4.16. Load-Lexicon

   This header field is used to indicate whether a lexicon has to be
   loaded or unloaded. The default value for this header field is
   "true". This header field MAY be specified in a DEFINE-LEXICON
   method.

```
load-lexicon          =    "Load-Lexicon" ":" BOOLEAN CRLF
```

8.4.17. Lexicon-Search-Order

   This header field is used to specify a list of active pronunciation
   lexicon URIs and the search order among the active lexicons.
   Lexicons specified within the SSML document take precedence over the
   lexicons specified in this header field. This header field MAY be
   specified in the SPEAK, SET-PARAMS, and GET-PARAMS methods.

   lexicon-search-order =   "Lexicon-Search-Order" ":"
            "<" absoluteURI ">" *(" " "<" absoluteURI ">") CRLF

8.5. Synthesizer Message Body

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

8.5.1. Synthesizer Speech Data

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

   All MRCPv2 servers containing synthesizer resources MUST support both
   plain text speech data and W3C's Speech Synthesis Markup Language
   [W3C.REC-speech-synthesis-20040907] and hence MUST support the Media
   Types text/plain and application/ssml+xml. Other formats MAY be
   supported.

   If the speech data is to be fetched by URI reference, the Media Type
   text/uri-list (see RFC2483 [RFC2483]) is used to indicate one or
   more URIs that, when dereferenced, will contain the content 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 URIs are
   specified in the content.

   A mix of URI and inline speech data MAY be indicated through the
   multipart/mixed Media Type, and MRCPv2 clients and servers MUST
   support the multipart/mixed Media Type. Embedded within the
   multipart there MAY be content for the text/uri-list, application/
   ssml+xml and/or text/plain media types. The character set and
   encoding used in the speech data is specified according to standard
   Media Type definitions. The multi-part content MAY also contain
   actual audio data. Clients may have recorded audio clips stored in
   memory or on a local device and wish to play it as part of the
   "SPEAK" request. The audio portions MAY be sent by the client as

part of the multi-part content block. This audio is referenced in
the speech markup data that is another part in the multi-part content
block according to the multipart/mixed Media Type specification.

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

http://www.example.com/ASR-Introduction.ssml
http://www.example.com/ASR-Document-Part1.ssml
http://www.example.com/ASR-Document-Part2.ssml
http://www.example.com/ASR-Conclusion.ssml
```

URI List Example


```
Content-Type:application/ssml+xml
Content-Length:...

<?xml version="1.0"?>
    <speak version="1.0"
           xmlns="http://www.w3.org/2001/10/synthesis"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
               http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
           xml:lang="en-US">
      <p>
        <s>You have 4 new messages.</s>
        <s>The first is from Aldine Turnbet
        and arrived at <break/>
        <say-as interpret-as="vxml:time">0345p</say-as>.</s>

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

SSML Example

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

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

http://www.example.com/ASR-Introduction.ssml
http://www.example.com/ASR-Document-Part1.ssml
http://www.example.com/ASR-Document-Part2.ssml
http://www.example.com/ASR-Conclusion.ssml

--break
Content-Type:application/ssml+xml
Content-Length:...

<?xml version="1.0"?>
    <speak version="1.0"
          xmlns="http://www.w3.org/2001/10/synthesis"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
              http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
          xml:lang="en-US">
      <p>
        <s>You have 4 new messages.</s>
        <s>The first is from Stephanie Williams
        and arrived at <break/>
        <say-as interpret-as="vxml:time">0345p</say-as>.</s>

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

                         Multipart Example

8.5.2. Lexicon Data

   Synthesizer lexicon data from the client to the server can be
   provided inline or by reference. Either way they are carried as
   typed media in the message body of the MRCPv2 request message (see
   Section 8.14).

   When a lexicon is specified in-line in the message, the client MUST
   provide a Content-ID for that lexicon as part of the content header
   fields. The server MUST store the lexicon associated with that
   Content-ID for the duration of the session. A stored lexicon can be
   overwritten by defining a new lexicon with the same Content-ID.

Lexicons that have been associated with a Content-ID can be
referenced through the "session" URI scheme (see Section 13.6).

If lexicon data is specified by external URI reference, the Media
Type text/uri-list (see RFC2483 [RFC2483]) is used to list the one
or more URIs that may be dereferenced to obtain the lexicon data.
All MRCPv2 servers MUST support the HTTP and HTTPS uri access
mechanisms, and MAY support other mechanisms.

If the data in the message body consists of a mix of URI and inline
lexicon data the multipart/mixed Media Type is used. The character
set and encoding used in the lexicon data may be specified according
to standard Media Type definitions.

8.6. SPEAK Method

The "SPEAK" Request provides the synthesizer resource with the speech
text and initiates speech synthesis and streaming. The "SPEAK"
method MAY carry voice and prosody header fields that alter the
behavior of the voice being synthesized, as well as a typed media
message body containing the actual marked-up text to be spoken.

The SPEAK method implementation MUST do a fetch of all external URIs
that are part of that operation. If caching is implemented, this URI
fetching MUST conform to the cache control hints and parameter header
fields associated with the method in deciding whether it is to be
fetched from cache or from the external server. If these hints/
parameters are not specified in the method, the values set for the
session using SET-PARAMS/GET-PARAMS apply. If it was not set for the
session their default values apply.

When applying voice parameters there are 3 levels of precedence. 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 which can be set using the "SET-PARAMS"
request and apply for subsequent methods invoked during the session.

If the resource was idle at the time the "SPEAK" request arrived at
the server and the "SPEAK" method is being actively processed, the
resource responds immediately with a success status code and a
request-state of IN-PROGRESS.

If the resource is in the speaking or paused state when the "SPEAK"
method arrives at the server, i.e. it is in the middle of processing
a previous "SPEAK" request, the status returns success with a
request-state of PENDING. The server places the "SPEAK" request in
the synthesizer resource request queue. The request queue operates

strictly FIFO: requests are processed serially in order of receipt.
If the current SPEAK fails, all SPEAK methods in the pending queue
are cancelled and each generates a SPEAK-COMPLETE event with a
Completion-Cause of "cancelled".

For the synthesizer resource, "SPEAK" is the only method that can
return a request-state of IN-PROGRESS or PENDING. When the text has
been synthesized and played into the media stream, the resource
issues a "SPEAK-COMPLETE" event with the request-id of the "SPEAK"
request and a request-state of COMPLETE.

```
C->S: MRCP/2.0 ... SPEAK 543257
      Channel-Identifier:32AECB23433802@speechsynth
      Voice-gender:neutral
      Voice-Age:25
      Prosody-volume:medium
      Content-Type:application/ssml+xml
      Content-Length:...

      <?xml version="1.0"?>
         <speak version="1.0"
            xmlns="http://www.w3.org/2001/10/synthesis"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
            xml:lang="en-US">
         <p>
          <s>You have 4 new messages.</s>
          <s>The first is from Stephanie Williams and arrived at
             <break/>
             <say-as interpret-as="vxml:time">0345p</say-as>.
             </s>
          <s>The subject is
                 <prosody rate="-20%">ski trip</prosody>
          </s>
         </p>
        </speak>


S->C: MRCP/2.0 ... 543257 200 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857206027059


S->C: MRCP/2.0 ... SPEAK-COMPLETE 543257 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Completion-Cause:000 normal
      Speech-Marker:timestamp=857206027059
```

SPEAK Example

8.7. STOP

   The "STOP" method from the client to the server tells the synthesizer
   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 status-code of 200 (Success). If no
   Active-Request-Id-List header field is sent in the "STOP" request the
   server terminates all outstanding "SPEAK" requests.

   If a "STOP" request successfully terminated one or more PENDING or
   IN-PROGRESS "SPEAK" requests, then the response MUST contain an
   Active-Request-Id-List header field enumerating the "SPEAK" request-
   ids that were terminated. Otherwise there is no Active-Request-Id-
   List header field in the response. No "SPEAK-COMPLETE" events are
   sent for such terminated requests.

   If a "SPEAK" request that was IN-PROGRESS and speaking was stopped,
   the next pending "SPEAK" request, if any, becomes IN-PROGRESS at the
   resource and enters the speaking state.

   If a "SPEAK" request that was IN-PROGRESS and paused was stopped, the
   next pending "SPEAK" request, if any, becomes IN-PROGRESS and enters
   the paused state.

```
C->S: MRCP/2.0 ... SPEAK 543258
      Channel-Identifier:32AECB23433802@speechsynth
      Content-Type:application/ssml+xml
      Content-Length:...

      <?xml version="1.0"?>
        <speak version="1.0"
             xmlns="http://www.w3.org/2001/10/synthesis"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                 http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
             xml:lang="en-US">
         <p>
          <s>You have 4 new messages.</s>
          <s>The first is from Stephanie Williams and arrived at
             <break/>
             <say-as interpret-as="vxml:time">0345p</say-as>.</s>
          <s>The subject is
              <prosody rate="-20%">ski trip</prosody></s>
         </p>
        </speak>


S->C: MRCP/2.0 ... 543258 200 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857206027059

C->S: MRCP/2.0 ... STOP 543259
      Channel-Identifier:32AECB23433802@speechsynth

S->C: MRCP/2.0 ... 543259 200 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Active-Request-Id-List:543258
      Speech-Marker:timestamp=857206039059
```

                          STOP Example

8.8. BARGE-IN-OCCURRED

   The BARGE-IN-OCCURRED method, when used with the synthesizer
   resource, provides a client which has detected a barge-in-able event
   a means to communicate the occurrence of the event to the synthesizer
   resource.

   This method is useful in two scenarios,
   1. The client has detected DTMF digits in the input media or some
       other barge-in-able event and wants to communicate that to the
       synthesizer resource.

2. The recognizer resource and the synthesizer resource are in
    different servers. In this case the client acts as an
    intermediary for the two servers. It receives an event from the
    recognition resource and sends a BARGE-IN-OCCURRED request 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 (see
Section 8.4.2), and the BARGE-IN-OCCURRED event is received, the
synthesizer MUST immediately stop streaming out audio. It MUST 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 "SPEAK" request was playing and it was terminated, the
response MUST contain the an Active-Request-Id-List header field
listing the request-ids of all "SPEAK" requests that were terminated.
The server generates no "SPEAK-COMPLETE" events for these requests.

If there were no "SPEAK" requests terminated by the synthesizer
resource as a result of the BARGE-IN-OCCURRED method, the server
responds to the BARGE-IN-OCCURRED with a status-code of 200 success,
and the response MUST NOT contain an Active-Request-Id-List header
field.

If the synthesizer and recognizer resources are part of the same
MRCPv2 session, they can be optimized for a quicker kill-on-barge-in
response if the recognizer and synthesizer interact directly. In
these cases, the client MUST still react to a START-OF-INPUT event
from the recognizer by invoking the BARGE-IN-OCCURRED method to the
synthesizer. The client MUST invoke the BARGE-IN-OCCURRED if it has
any outstanding requests to the synthesizer resource in either the
PENDING or IN-PROGRESS state.

```
C->S: MRCP/2.0 ... SPEAK 543258
      Channel-Identifier:32AECB23433802@speechsynth
      Voice-gender:neutral
      Voice-Age:25
      Prosody-volume:medium
      Content-Type:application/ssml+xml
      Content-Length:...

      <?xml version="1.0"?>
        <speak version="1.0"
             xmlns="http://www.w3.org/2001/10/synthesis"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
             xml:lang="en-US">
        <p>
         <s>You have 4 new messages.</s>
         <s>The first is from Stephanie Williams and arrived at
            <break/>
            <say-as interpret-as="vxml:time">0345p</say-as>.</s>
         <s>The subject is
            <prosody rate="-20%">ski trip</prosody></s>
        </p>
       </speak>

S->C: MRCP/2.0 ... 543258 200 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857206027059

C->S: MRCP/2.0 ... BARGE-IN-OCCURRED 543259
      Channel-Identifier:32AECB23433802@speechsynth
      Proxy-Sync-Id:987654321

S->C:MRCP/2.0 ... 543259 200 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Active-Request-Id-List:543258
      Speech-Marker:timestamp=857206039059
```

                    BARGE-IN-OCCURED Example

8.9. PAUSE

The PAUSE method from the client to the server tells the synthesizer
resource to pause speech output if it is speaking something. If a
PAUSE method is issued on a session when a "SPEAK" is not active the
server MUST respond with a status-code of 402 "Method not valid in
this state". If a PAUSE method is issued on a session when a "SPEAK"
is active and paused the server MUST respond with a status-code of

200 "Success". If a "SPEAK" request was active the server MUST
return an Active-Request-Id-List header field whose value contains
the request-id of the "SPEAK" request that was paused.

```
C->S: MRCP/2.0 ... SPEAK 543258
      Channel-Identifier:32AECB23433802@speechsynth
      Voice-gender:neutral
      Voice-Age:25
      Prosody-volume:medium
      Content-Type:application/ssml+xml
      Content-Length:...

      <?xml version="1.0"?>
        <speak version="1.0"
            xmlns="http://www.w3.org/2001/10/synthesis"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
            xml:lang="en-US">
       <p>
        <s>You have 4 new messages.</s>
        <s>The first is from Stephanie Williams and arrived at
            <break/>
            <say-as interpret-as="vxml:time">0345p</say-as>.</s>

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

S->C: MRCP/2.0 ... 543258 200 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857206027059

C->S: MRCP/2.0 ... PAUSE 543259
      Channel-Identifier:32AECB23433802@speechsynth

S->C: MRCP/2.0 ... 543259 200 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Active-Request-Id-List:543258
```

                         PAUSE Example

8.10. RESUME

The RESUME method from the client to the server tells a paused
synthesizer resource to resume speaking. If a RESUME request is
issued on a session with no active "SPEAK" request, the server MUST

respond with a status-code of 402 "Method not valid in this state".
If a RESUME request is issued on a session with an active "SPEAK"
request that is speaking (i.e., not paused) the server MUST respond
with a status-code of 200 "Success". If a "SPEAK" request was paused
the server MUST return an Active-Request-Id-List header field whose
value contains the request-id of the "SPEAK" request that was
resumed.

```
C->S: MRCP/2.0 ... SPEAK 543258
      Channel-Identifier:32AECB23433802@speechsynth
      Voice-gender:neutral
      Voice-age:25
      Prosody-volume:medium
      Content-Type:application/ssml+xml
      Content-Length:...

      <?xml version="1.0"?>
        <speak version="1.0"
             xmlns="http://www.w3.org/2001/10/synthesis"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                 http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
             xml:lang="en-US">
       <p>
        <s>You have 4 new messages.</s>
        <s>The first is from Stephanie Williams and arrived at
           <break/>
           <say-as interpret-as="vxml:time">0345p</say-as>.</s>
        <s>The subject is
           <prosody rate="-20%">ski trip</prosody></s>
       </p>
      </speak>

S->C: MRCP/2.0 ... 543258 200 IN-PROGRESS@speechsynth
      Channel-Identifier:32AECB23433802
      Speech-Marker:timestamp=857206027059

C->S: MRCP/2.0 ... PAUSE 543259
      Channel-Identifier:32AECB23433802@speechsynth

S->C: MRCP/2.0 ... 543259 200 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Active-Request-Id-List:543258

C->S: MRCP/2.0 ... RESUME 543260
      Channel-Identifier:32AECB23433802@speechsynth

S->C: MRCP/2.0 ... 543260 200 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Active-Request-Id-List:543258
```

                        RESUME Example

8.11. 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 request the synthesizer to jump forward or backward
   in what it is speaking, change speaker rate, speaker parameters, etc.
   It affects only the currently IN-PROGRESS "SPEAK" request. Depending
   on the implementation and capability of the synthesizer resource it
   may or may not support the various modifications indicated by header
   fields in the CONTROL request.

   When a client invokes a CONTROL method to jump forward and the
   operation goes beyond the end of the active "SPEAK" method's text,
   the CONTROL request still succeeds. The active "SPEAK" request
   completes and returns a "SPEAK-COMPLETE" event following the response
   to the CONTROL method. If there are more "SPEAK" requests in the
   queue, the synthesizer resource starts at the beginning of the next
   "SPEAK" request in the queue.

   When a client invokes a CONTROL method to jump backward and the
   operation jumps to the beginning or beyond the beginning of the
   speech data of the active "SPEAK" method, the CONTROL request still
   succeeds. The response to the CONTROL request contains the speak-
   restart header field, and the active "SPEAK" request restarts from
   the beginning of its speech data.

   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 to 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 field with
   the Request-id of the "SPEAK" request that was active.

```
C->S: MRCP/2.0 ... SPEAK 543258
      Channel-Identifier:32AECB23433802@speechsynth
      Voice-gender:neutral
      Voice-age:25
      Prosody-volume:medium
      Content-Type:application/ssml+xml
      Content-Length:...

      <?xml version="1.0"?>
        <speak version="1.0"
            xmlns="http://www.w3.org/2001/10/synthesis"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
              http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
            xml:lang="en-US">
        <p>
         <s>You have 4 new messages.</s>
         <s>The first is from Stephanie Williams
            and arrived at <break/>
            <say-as interpret-as="vxml:time">0345p</say-as>.</s>

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


S->C: MRCP/2.0 ... 543258 200 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857205016059

C->S: MRCP/2.0 ... CONTROL 543259
      Channel-Identifier:32AECB23433802@speechsynth
      Prosody-rate:fast

S->C: MRCP/2.0 ... 543259 200 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Active-Request-Id-List:543258
      Speech-Marker:timestamp=857206027059

C->S: MRCP/2.0 ... CONTROL 543260
      Channel-Identifier:32AECB23433802@speechsynth
      Jump-Size:-15 Words

S->C: MRCP/2.0 ... 543260 200 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Active-Request-Id-List:543258
      Speech-Marker:timestamp=857206039059
```

                         CONTROL Example

8.12. SPEAK-COMPLETE

   This is an Event message from the synthesizer resource to the client
   indicating that the corresponding "SPEAK" request was completed. The
   request-id header field matches the request-id of the "SPEAK" request
   that initiated the speech that just completed. The request-state
   field is set to COMPLETE by the server, indicating that this is the
   last event with the corresponding request-id. 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, kill-on-barge-in etc.

   C->S: MRCP/2.0 ... SPEAK 543260
         Channel-Identifier:32AECB23433802@speechsynth
         Voice-gender:neutral
         Voice-age:25
         Prosody-volume:medium
         Content-Type:application/ssml+xml
         Content-Length:...

         <?xml version="1.0"?>
           <speak version="1.0"
               xmlns="http://www.w3.org/2001/10/synthesis"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                   http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
               xml:lang="en-US">
          <p>
           <s>You have 4 new messages.</s>
           <s>The first is from Stephanie Williams
              and arrived at <break/>
              <say-as interpret-as="vxml:time">0345p</say-as>.</s>
           <s>The subject is
              <prosody rate="-20%">ski trip</prosody></s>
          </p>
          </speak>

   S->C: MRCP/2.0 ... 543260 200 IN-PROGRESS
         Channel-Identifier:32AECB23433802@speechsynth
         Speech-Marker:timestamp=857206027059

   S->C: MRCP/2.0 ... SPEAK-COMPLETE 543260 COMPLETE
         Channel-Identifier:32AECB23433802@speechsynth
         Completion-Cause:000 normal
         Speech-Marker:timestamp=857206039059

                       SPEAK-COMPLETE Example

8.13. SPEECH-MARKER

   This is an event generated by the synthesizer resource to the client
   when the synthesizer encounters a marker tag in the speech markup it
   is currently processing. The request-id field in the header field
   matches the corresponding "SPEAK" request. The request-state field
   indicates IN-PROGRESS as the speech is still not complete. The value
   of the speech marker tag hit, describing where the synthesizer is in
   the speech markup, is returned in the speech-marker header field,
   along with an NTP timestamp indicating the instant in the output
   speech stream that the marker was encountered. The SPEECH-MARKER
   event MUST also be generated with a null marker value and output NTP
   timestamp when a SPEAK request in Pending-State (i.e. in the queue)
   changes state to IN-PROGRESS and starts speaking. The NTP timestamp
   MUST be synchronized with the RTP timestamp used to generate the
   speech stream through standard RTCP machinery.

```
C->S: MRCP/2.0 ... SPEAK 543261
      Channel-Identifier:32AECB23433802@speechsynth
      Voice-gender:neutral
      Voice-age:25
      Prosody-volume:medium
      Content-Type:application/ssml+xml
      Content-Length:...

      <?xml version="1.0"?>
        <speak version="1.0"
             xmlns="http://www.w3.org/2001/10/synthesis"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                 http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
             xml:lang="en-US">
       <p>
        <s>You have 4 new messages.</s>
        <s>The first is from Stephanie Williams
           and arrived at <break/>
           <say-as interpret-as="vxml:time">0345p</say-as>.</s>
           <mark name="here"/>
        <s>The subject is
           <prosody rate="-20%">ski trip</prosody>
        </s>
        <mark name="ANSWER"/>
       </p>
      </speak>


S->C: MRCP/2.0 ... 543261 200 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857205015059

S->C: MRCP/2.0 ... SPEECH-MARKER 543261 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857206027059;here

S->C: MRCP/2.0 ... SPEECH-MARKER 543261 IN-PROGRESS
      Channel-Identifier:32AECB23433802@speechsynth
      Speech-Marker:timestamp=857206039059;ANSWER

S->C: MRCP/2.0 ... SPEAK-COMPLETE 543261 COMPLETE
      Channel-Identifier:32AECB23433802@speechsynth
      Completion-Cause:000 normal
      Speech-Marker:timestamp=857207689259;ANSWER
```

                        SPEECH-MARKER Example

8.14. DEFINE-LEXICON

   The DEFINE-LEXICON method, from the client to the server, provides a
   lexicon and tells the server to load or unload the lexicon (see
   Section 8.4.16). The media type of the lexicon is provided in the
   Content-Type header (see Section 8.5.2). One such media type is PLS
   [W3C.REC-pronunciation-lexicon-20081014].

   If the server resource is in the speaking or paused state, the server
   MUST respond with a failure status-code of 402 "Method not valid in
   this state".

   If the resource is in the idle state and is able to successfully
   load/unload the lexicon the status MUST return a 200 "Success"
   status-code and the request-state MUST be COMPLETE.

   If the synthesizer could not define the lexicon for some reason, for
   example because the download failed or the lexicon was in an
   unsupported form, the server MUST respond with a failure status-code
   of 407, and a Completion-Cause header field describing the failure
   reason.

9. Speech Recognizer Resource

   The speech recognizer resource receives an incoming voice stream and
   provides the client with an interpretation of what was spoken in
   textual form.

   The recognizer resource is controlled by MRCPv2 requests from the
   client. The recognizer resource can both respond to these requests
   and generate asynchronous events to the client to indicate conditions
   of interest during the processing of the method.

   This section applies to the following resource types.
   1. speechrecog
   2. dtmfrecog

   The difference between the above two resources is in their level of
   support for recognition grammars. The "dtmfrecog" resource type is
   capable of recognizing only DTMF digits and hence accepts only DTMF
   grammars. It only generates barge-in for DTMF inputs and ignores
   speech. The "speechrecog" resource type can recognize regular speech
   as well as DTMF digits and hence MUST support grammars describing
   either speech or DTMF. This resource generates barge-in events for
   speech and/or DTMF. By analyzing the grammars that are activated by
   the RECOGNIZE method, it determines if a barge-in should occur for
   speech and/or DTMF. When the recognizer decides it needs to generate

barge-in it also generates a START-OF-INPUT event to the client. The
recognition resource may support recognition in the normal or hotword
modes or both (although note that a single speechrecog resource does
not perform normal and hotword mode recognition simultaneously). For
implementations where a single recognition resource does not support
both modes, or simultaneous normal and hotword recognition is
desired, the two modes can be invoked through separate resources
allocated to the same SIP dialog (with different MRCP session
identifiers) and share the RTP audio feed.

The capabilities of the recognition resource are enumerated below:

Normal Mode Recognition Normal mode recognition tries to match all
    of the speech or DTMF against the grammar and returns a no-match
    status if the input fails to match or the method times out.
Hotword Mode Recognition Hotword mode is where the recognizer looks
    for a match against specific speech grammar or DTMF sequence and
    ignores speech or DTMF that does not match. The recognition
    completes only for a successful match of grammar or if the client
    cancels the request or if there is a a non-input or recognition
    timeout.
Voice Enrolled Grammars A recognition resource may optionally
    support Voice Enrolled Grammars. With this functionality,
    enrollment is performed using a person's voice. For example, a
    list of contacts can be created and maintained by recording the
    person's names using the caller's voice. This technique is
    sometimes also called speaker-dependent recognition.
Interpretation A recognition resource may be employed strictly for
    its natural language interpretation capabilities by supplying it
    with a text string as input instead of speech. In this mode the
    resource takes text as input and produces an "interpretation" of
    the input according to the supplied grammar.

Voice Enrollment has the concept of an enrollment session. A session
to add a new phrase to a personal grammar involves the initial
enrollment followed by a repeat of enough utterances before
committing the new phrase to the personal grammar. Each time an
utterance is recorded, it is compared for similarity with the other
samples and a clash test is performed against other entries in the
personal grammar to ensure there are no similar and confusable
entries.

Enrollment is done using a recognizer resource. Controlling which
utterances are to be considered for enrollment of a new phrase is
done by setting a header field (see Section 9.4.39) in the Recognize
request.

Interpretation is accomplished through the INTERPRET method

(Section 9.20) and the interpret-text header field (Section 9.4.30).

9.1. Recognizer State Machine

   The recognizer resource maintains a state machine to process MRCPv2
   requests from the client.

```
 Idle                      Recognizing               Recognized
 State                     State                     State
   |                         |                         |
   |---------RECOGNIZE---->|---RECOGNITION-COMPLETE-->|
   |<------STOP-----------|<-----RECOGNIZE----------|
   |                         |                         |
   |          |--------|     |          |----------|   |
   |        START-OF-INPUT   |        GET-RESULT       |
   |          |------->|     |          |--------->|   |
   |------------|           |                         |
   |        DEFINE-GRAMMAR   |----------|              |
   |<-----------|           | START-INPUT-TIMERS      |
   |                         |<--------|               |
   |------|                  |                         |
   |   INTERPRET             |                         |
   |<-----|                  |------|                  |
   |                         |   RECOGNIZE             |
   |-------|                 |<-----|                  |
   |       STOP              |                         |
   |<------|                 |                         |
   |<-----------------STOP------------------------|
   |<----------------DEFINE-GRAMMAR---------------|
```
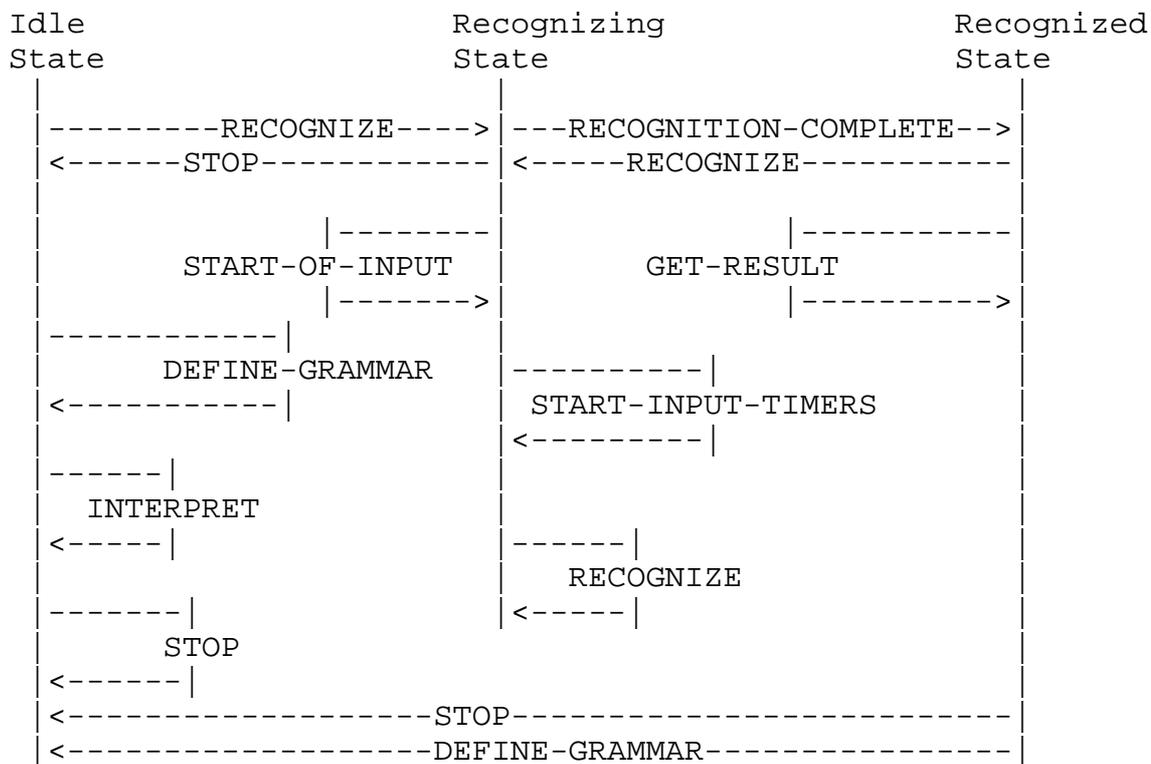
                    Recognizer State Machine

   If a recognition resource supports voice enrolled grammars, starting
   an enrollment session does not change the state of the recognizer
   resource. Once an enrollment session is started, then utterances are
   enrolled by calling the RECOGNIZE method repeatedly. The state of
   the speech recognizer resource goes from IDLE to RECOGNIZING state
   each time RECOGNIZE is called.

9.2. Recognizer Methods

   The recognizer supports the following methods.

```
recognizer-method       =    recog-only-method
                        /    enrollment-method

recog-only-method       =    "DEFINE-GRAMMAR"
                        /    "RECOGNIZE"
                        /    "INTERPRET"
                        /    "GET-RESULT"
                        /    "START-INPUT-TIMERS"
                        /    "STOP"
```

It is OPTIONAL for a recognizer resource to support voice enrolled
grammars. If the recognizer resource does support voice enrolled
grammars it MUST support the following methods.

```
enrollment-method       =    "START-PHRASE-ENROLLMENT"
                        /    "ENROLLMENT-ROLLBACK"
                        /    "END-PHRASE-ENROLLMENT"
                        /    "MODIFY-PHRASE"
                        /    "DELETE-PHRASE"
```

9.3. Recognizer Events

The recognizer may generate the following events.

```
recognizer-event        =    "START-OF-INPUT"
                        /    "RECOGNITION-COMPLETE"
                        /    "INTERPRETATION-COMPLETE"
```

9.4. 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     =   recog-only-header
                      /   enrollment-header

recog-only-header     =   confidence-threshold
                      /   sensitivity-level
                      /   speed-vs-accuracy
                      /   n-best-list-length
                      /   no-input-timeout
                      /   input-type
                      /   recognition-timeout
                      /   waveform-uri
                      /   input-waveform-uri
                      /   completion-cause
                      /   completion-reason
                      /   recognizer-context-block
                      /   start-input-timers
                      /   speech-complete-timeout
                      /   speech-incomplete-timeout
                      /   dtmf-interdigit-timeout
                      /   dtmf-term-timeout
                      /   dtmf-term-char
                      /   failed-uri
                      /   failed-uri-cause
                      /   save-waveform
                      /   media-type
                      /   new-audio-channel
                      /   speech-language
                      /   ver-buffer-utterance
                      /   recognition-mode
                      /   cancel-if-queue
                      /   hotword-max-duration
                      /   hotword-min-duration
                      /   interpret-text
                      /   dtmf-buffer-time
                      /   clear-dtmf-buffer
                      /   early-no-match
```

If a recognition resource supports voice enrolled grammars, the
following header fields are also used.

```
enrollment-header      =    num-min-consistent-pronunciations
                       /    consistency-threshold
                       /    clash-threshold
                       /    personal-grammar-uri
                       /    enroll-utterance
                       /    phrase-id
                       /    phrase-nl
                       /    weight
                       /    save-best-waveform
                       /    new-phrase-id
                       /    confusable-phrases-uri
                       /    abort-phrase-enrollment
```

For enrollment-specific header fields that can appear as part of
"SET-PARAMS" or "GET-PARAMS" methods, the following general rule
applies: the START-PHRASE-ENROLLMENT method must be invoked before
these header fields may be set through the "SET-PARAMS" method or
retrieved through the "GET-PARAMS" method.

Note that the Waveform-URI header field of the Recognizer resource
can also appear in the response to the END-PHRASE-ENROLLMENT method.

9.4.1. 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 match. The confidence-threshold header field tells the
recognizer resource what confidence level the client considers a
successful match. This is a float value between 0.0-1.0 indicating
the recognizer's confidence in the recognition. If the recognizer
determines that there is no candidate match with a confidence that is
greater 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 header
field is implementation specific, as is the interpretation of any
specific value for this header field. Although values for servers
from different vendors are not comparable, it is expected that
clients will tune this value over time for a given server.

```
confidence-threshold      =  "Confidence-Threshold" ":" FLOAT CRLF
```

9.4.2. 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 header field is a float value between 0.0 and 1.0
and allows the client to set the sensitivity level for the
recognizer. This header field MAY occur in RECOGNIZE, "SET-PARAMS"

or "GET-PARAMS". A higher value for this header field means higher
sensitivity. The default value for this header field is
implementation specific, as is the interpretation of any specific
value for this header field. Although values for servers from
different vendors are not comparable, it is expected that clients
will tune this value over time for a given server.

```
sensitivity-level        =  "Sensitivity-Level" ":" FLOAT CRLF
```

9.4.3. 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
fewer active sessions per server and vice versa. The value is a
float between 0.0 and 1.0. A value of 0.0 means fastest recognition.
A value of 1.0 means best accuracy. This header field MAY occur in
RECOGNIZE, "SET-PARAMS" or "GET-PARAMS". The default value for this
header field is implementation specific. Although values for servers
from different vendors are not comparable, it is expected that
clients will tune this value over time for a given server.

```
speed-vs-accuracy        =  "Speed-Vs-Accuracy" ":" FLOAT CRLF
```

9.4.4. 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
returns only the best match above the confidence threshold. The
client, by setting this header field, can 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 provide the requested
number of alternatives. This header field MAY occur in RECOGNIZE,
"SET-PARAMS" or "GET-PARAMS". The minimum value for this header
field is 1. The default value for this header field is 1.

```
n-best-list-length       =  "N-Best-List-Length" ":" 1*19DIGIT CRLF
```

9.4.5. Input Type

When the recognizer detects barge-in-able input and generates a
START-OF-INPUT event, that event MUST carry this header field to
specify whether the input that caused the barge-in was DTMF or
speech.

```
input-type            = "Input-Type" ":"  inputs CRLF
inputs                = "speech" / "dtmf"
```

9.4.6. 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 with a Completion-Cause of "no-input-
   timeout" and terminate the recognition operation. The client can use
   the no-input-timeout header field to set this timeout. The value is
   in milliseconds and may range from 0 to an implementation specific
   maximum value. This header field MAY occur in RECOGNIZE,
   "SET-PARAMS" or "GET-PARAMS". The default value is implementation
   specific.

```
no-input-timeout         = "No-Input-Timeout" ":" 1*19DIGIT CRLF
```

9.4.7. 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 header field allows the client to set this
   timeout value. The value is in milliseconds. The value for this
   header field ranges from 0 to an implementation specific maximum
   value. The default value is 10 seconds. This header field MAY occur
   in RECOGNIZE, SET-PARAMS or GET-PARAMS.

```
recognition-timeout      = "Recognition-Timeout" ":" 1*19DIGIT CRLF
```

9.4.8. Waveform URI

   If the Save-Waveform header field is set to true, the recognizer MUST
   record the incoming audio stream of the recognition into a stored
   form and provide a URI for the client to access it. This header
   field MUST be present in the RECOGNITION-COMPLETE event if the Save-
   Waveform header field was set to true. The value of the header field
   MUST be empty if there was some error condition preventing the server
   from recording. Otherwise, the URI generated by the server MUST be
   unambiguous across the server and all its recognition sessions. The
   content associated with the URI MUST be available to the client until
   the MRCPv2 session terminates.

   Similarly, if the Save-Best-Waveform header field is set to true, the
   recognizer MUST save the audio stream for the best repetition of the
   phrase that was used during the enrollment session. The recognizer
   MUST then record the recognized audio and make it available to the
   client by returning a URI in the Waveform-URI header field in the

response to the END-PHRASE-ENROLLMENT method. The value of the
header field MUST be empty if there was some error condition
preventing the server from recording. Otherwise, the URI generated
by the server MUST be unambiguous across the server and all its
recognition sessions. The content associated with the URI MUST be
available to the client until the MRCPv2 session terminates. See the
discussion on the sensitivity of saved waveforms in Section 12.

The server MUST also return the size in octets and the duration in
milliseconds of the recorded audio wave-form as parameters associated
with the header field.

```
waveform-uri            = "Waveform-URI" ":" ["<" uri ">"
                          ";" "size" "=" 1*19DIGIT
                          ";" "duration" "=" 1*19DIGIT] CRLF
```

9.4.9. Media Type

This header field MAY be specified in the SET-PARAMS, GET-PARAMS or
the RECOGNIZE methods and tells the server resource the Media Type in
which to store captured audio or video such as the one captured and
returned by the Waveform-URI header field.

```
media-type              = "Media-Type" ":" media-type-value
                          CRLF
```

9.4.10. Input-Waveform-URI

This optional header field specifies a URI pointing to audio content
to be processed by the RECOGNIZE operation. This enables the client
to request recognition from a specified buffer or audio file.

```
input-waveform-uri      = "Input-Waveform-URI" ":" uri CRLF
```

9.4.11. Completion Cause

This header field MUST be part of a RECOGNITION-COMPLETE, event
coming from the recognizer resource to the client. It 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. In the ABNF
below, the 'cause-code' contains a numerical value selected from the
Cause-Code column of the following table. The 'cause-name' contains
the corresponding token selected from the Cause-Name column.

```
completion-cause        = "Completion-Cause" ":" cause-code SP
                          cause-name CRLF
cause-code              = 3DIGIT
```

```
cause-name                 =  *VCHAR
```

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	hotword-maxtime	RECOGNIZE in hotword mode completed without a match due to a recognition-timeout
004	grammar-load-failure	RECOGNIZE failed due grammar load failure.
005	grammar-compilation-failure	RECOGNIZE failed due to grammar compilation failure.
006	recognizer-error	RECOGNIZE request terminated prematurely due to a recognizer error.
007	speech-too-early	RECOGNIZE request terminated because speech was too early. This happens when the audio stream is already "in-speech" when the RECOGNIZE request was received.
008	success-maxtime	RECOGNIZE request terminated because speech was too long but whatever was spoken till that point was a full match.
009	uri-failure	Failure accessing a URI.
010	language-unsupported	Language not supported.
011	cancelled	A new RECOGNIZE cancelled this one, or a prior RECOGNIZE failed while this one was still in the queue.

012	semantics-failure	Recognition succeeded but semantic interpretation of the recognized input failed. The RECOGNITION-COMPLETE event MUST contain the Recognition result with only input text and no interpretation.
013	partial-match	Speech Incomplete timeout expired before there was a full match. But whatever that was spoken till that point was a partial match to one or more grammars.
014	partial-match-maxtime	The Recognition-Timer expired before full match was achieved. But whatever was spoken till that point was a partial match to one or more grammars.
015	no-match-maxtime	The Recognition-Timer expired. Whatever was spoken till that point either did not match any of the grammars. This cause could also be returned if the recognizer does not support detecting partial grammar matches.
016	grammar-definition-failure	any DEFINE-GRAMMAR error other than grammar-load-failure and grammar-compilation-failure.

9.4.12. Completion Reason

   This header field MAY be specified in a RECOGNITION-COMPLETE event
   coming from the recognizer resource to the client. This contains the
   reason text behind the RECOGNIZE request completion. The server uses
   this header field to communicate text describing the reason for the
   failure, such as the specific error encountered in parsing a grammar
   markup.

   The completion reason text is provided for client use in logs and for
   debugging and instrumentation purposes. Clients MUST NOT interpret
   the completion reason text.

```
completion-reason          =  "Completion-Reason" ":"
                              quoted-string CRLF
```

9.4.13. Recognizer Context Block

This header field 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 typed media message body.
If the server returns a recognizer context block, the response MUST
contain this header field and its value MUST match the Content-ID of
the corresponding media block.

If the "SET-PARAMS" method contains this header field, it MUST also
contain a message body containing the recognizer context data, and a
Content-ID matching this header field value. This Content-ID MUST
match the Content-ID that came with the context data during the
"GET-PARAMS" operation.

An implementation choosing to use this mechanism to hand off
recognizer context data between servers MUST distinguish its
implementation-specific block of data by using an IANA-registered
content type in the IANA Media Type vendor tree.

```
recognizer-context-block  =  "Recognizer-Context-Block" ":"
                              [1*VCHAR] CRLF
```

9.4.14. Start Input Timers

This header field 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 MUST NOT start the
timers until the client sends a START-INPUT-TIMERS request to the
recognizer. This is useful in the scenario when the recognizer and
synthesizer engines are not part of the same session. In such
configurations, when a kill-on-barge-in prompt is being played (see
Section 8.4.2), the client wants the RECOGNIZE request to be
simultaneously active so that it can detect and implement kill-on-
barge-in. However, the recognizer SHOULD NOT start the no-input
timers until the prompt is finished. The default value is "true".

```
start-input-timers  =  "Start-Input-Timers" ":" BOOLEAN CRLF
```

9.4.15. 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 generating a nomatch event). The speech-complete-
   timeout value applies when the recognizer currently has a complete
   match against an active grammar, and specifies how long the
   recognizer MUST 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*19DIGIT CRLF

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

9.4.16. 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
   Completion-Cause of "partial-match"). The value is in milliseconds.
   The value for this header field ranges from 0 to an implementation
   specific maximum value. The default value for this header field is
   implementation specific.

   speech-incomplete-timeout = "Speech-Incomplete-Timeout" ":" 1*19DIGIT
                               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 spoken words can continue to
   represent a match.

   A long speech-incomplete-timeout value delays the result to the
   client and therefore makes the application's response to a user 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".

9.4.17. 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 header field ranges from 0 to an implementation specific maximum
value. 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*19DIGIT CRLF

9.4.18. DTMF Term Timeout

This header field specifies the terminating timeout to use when
recognizing DTMF input. The DTMF-Term-Timeout applies only when no
additional input is allowed by the grammar; otherwise, the DTMF-
Interdigit-Timeout applies. The value is in milliseconds. The value
for this header field ranges from 0 to an implementation specific
maximum value. 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*19DIGIT CRLF

9.4.19. DTMF-Term-Char

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

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

9.4.20. Failed URI

When a recognizer needs to fetch or access a URI and the access fails
the server SHOULD provide the failed URI in this header field in the
method response, unless there are multiple URI failures, in which
case one of the failed URIs MUST be provided in this header field in
the method response.

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

9.4.21. Failed URI Cause

   When a recognizer method needs a recognizer to fetch or access a URI
   and the access fails the server MUST provide the URI specific or
   protocol specific response code for the URI in the Failed-URI header
   field through this header field in the method response. The value
   encoding is UTF-8 (RFC3629 [RFC3629]) to accommodate any access
   protocol, some of which might have a response string instead of a
   numeric response code.

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

9.4.22. Save Waveform

   This header field allows the client to request the recognizer
   resource to save the audio input to the recognizer. The recognizer
   resource MUST then attempt to record the recognized audio, without
   endpointing, 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
   content is otherwise not available, the recognizer MUST return an
   empty Waveform-URI header field. The default value for this field is
   "false". This header field MAY occur in RECOGNIZE, "SET-PARAMS" or
   "GET-PARAMS". See the discussion on the sensitivity of saved
   waveforms in Section 12.

   save-waveform   =   "Save-Waveform" ":" BOOLEAN CRLF

9.4.23. New Audio Channel

   This header field MAY be specified in a RECOGNIZE request and allows
   the client to tell the server that, from this point on, further input
   audio comes from a different audio source, channel or speaker. If
   the recognition resource had collected any input statistics or
   adaptation state, the recognition resource MUST do what is
   appropriate for the specific recognition technology, which includes
   but is not limited to discarding any collected input statistics or
   adaptation state before starting the RECOGNIZE request. Note that if
   there are multiple resources that are sharing a media stream and are
   collecting or using this data, and the client issues this header
   field to one of the resources, the reset operation applies to all
   resources that use the shared media stream. This helps in a number
   of use cases, including where the client wishes to reuse an open
   recognition session with an existing media session for multiple
   telephone calls.

   new-audio-channel   =   "New-Audio-Channel" ":" BOOLEAN
                                CRLF

9.4.24. 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 MUST follow RFC 5646 [RFC5646] for its
   values. This MAY occur in DEFINE-GRAMMAR, RECOGNIZE, "SET-PARAMS" or
   "GET-PARAMS" request.

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

9.4.25. Ver-Buffer-Utterance

   This header field lets the client request the server to buffer the
   utterance associated with this recognition request into a buffer
   available to a co-resident verifier resource. The buffer is shared
   across resources within a session and is allocated when a verifier
   resource is added to this session. The client MUST NOT send this
   header field unless a verifier resource is instantiated for the
   session. The buffer is released when the verifier resource is
   released from the session.

9.4.26. Recognition-Mode

   This header field specifies what mode the RECOGNIZE method will
   operate in. The value choices are "normal" or "hotword". If the
   value is "normal", the RECOGNIZE starts matching speech and DTMF to
   the grammars specified in the RECOGNIZE request. If any portion of
   the speech does not match the grammar, the RECOGNIZE command
   completes with a no-match status. Timers may be active to detect
   speech in the audio (see Section 9.4.14), so the RECOGNIZE method may
   complete because of a timeout waiting for speech. If the value of
   this header field is "hotword", the RECOGNIZE method operates in
   hotword mode, where it only looks for the particular keywords or DTMF
   sequences specified in the grammar and ignores silence or other
   speech in the audio stream. The default value for this header field
   is "normal". This header field MAY occur on the RECOGNIZE method.

   recognition-mode   =   "Recognition-Mode" ":"
                                 normal-value / hotword-value CRLF
   normal-value   =   "normal"
   hotword-value   =   "hotword"

9.4.27. Cancel-If-Queue

   This header field specifies what will happen if the client attempts
   to invoke another RECOGNIZE method when this RECOGNIZE request is
   already in progress for the resource. The value for this header

field is Boolean. A value of "true" means the server MUST terminate
this RECOGNIZE request, with a Completion-Cause of "cancelled", if
the client issues another RECOGNIZE request for the same resource. A
value of "false" for this header field indicates to the server that
this RECOGNIZE request will continue to completion and if the client
issues more RECOGNIZE requests to the same resource, they are queued.
When the currently active RECOGNIZE request is stopped or completes
with a successful match, the first RECOGNIZE method in the queue
becomes active. If the current RECOGNIZE fails, all RECOGNIZE
methods in the pending queue are cancelled and each generates a
RECOGNITION-COMPLETE event with a Completion-Cause of "cancelled".
This header field MUST be present in every RECOGNIZE request. There
is no default value.

```
cancel-if-queue          =   "Cancel-If-Queue" ":" BOOLEAN CRLF
```

9.4.28. Hotword-Max-Duration

This header field MAY be sent in a hotword mode RECOGNIZE request.
It specifies the maximum length of an utterance (in seconds) that
will be considered for Hotword recognition. This header field, along
with Hotword-Min-Duration, can be used to tune performance by
preventing the recognizer from evaluating utterances that are too
short or too long to be one of the hotwords in the grammar(s). The
value is in milliseconds. The default is implementation dependent.
If present in a RECOGNIZE request specifying a mode other than
"hotword", the header field is ignored.

```
hotword-max-duration     =   "Hotword-Max-Duration" ":" 1*19DIGIT
                             CRLF
```

9.4.29. Hotword-Min-Duration

This header field MAY be sent in a hotword mode RECOGNIZE request.
It specifies the minimum length of an utterance (in seconds) that
will be considered for Hotword recognition. This header field, along
with Hotword-Max-Duration, can be used to tune performance by
preventing the recognizer from evaluating utterances that are too
short or too long to be one of the hotwords in the grammar(s). The
value is in milliseconds. The default value is implementation
dependent. If present in a RECOGNIZE request specifying a mode other
than "hotword", the header field is ignored.

```
hotword-min-duration     =   "Hotword-Min-Duration" ":" 1*19DIGIT CRLF
```

9.4.30. Interpret-Text

   The value of this header field is used to provide a pointer to the
   text for which a natural language interpretation is desired. The
   value is either a URI or text. If the value is a URI, it MUST be a
   Content-ID that refers to an entity of type text/plain in the body of
   the message. Otherwise, the server MUST treat the value as the text
   to be interpreted. This header field MUST be used when invoking the
   INTERPRET method.

   interpret-text   =   "Interpret-Text" ":" 1*VCHAR CRLF

9.4.31. DTMF-Buffer-Time

   This header field MAY be specified in a GET-PARAMS or SET-PARAMS
   method and is used to specify the size in time, in milliseconds, of
   the typeahead buffer for the recognizer. This is the buffer that
   collects DTMF digits as they are pressed even when there is no
   RECOGNIZE command active. When a subsequent RECOGNIZE method is
   received it MAY look to this buffer to match the RECOGNIZE request.
   If the digits in the buffer is not sufficient then it can continue to
   listen to more digits to match the grammar. The default size of this
   DTMF buffer is platform specific.

   dtmf-buffer-time =   "DTMF-Buffer-Time" ":" 1*19DIGIT CRLF

9.4.32. Clear-DTMF-Buffer

   This header field MAY be specified in a RECOGNIZE method and is used
   to tell the recognizer to clear the DTMF type-ahead buffer before
   starting the recognize. The default value of this header field is
   FALSE, which does not clear the typeahead buffer before starting the
   RECOGNIZE method. If this header field is specified to be TRUE, then
   the recognize will clear the DTMF buffer before starting recognition.
   This means digits pressed by the caller before the RECOGNIZE command
   was issued are discarded.

   clear-dtmf-buffer = "Clear-DTMF-Buffer" ":" BOOLEAN CRLF

9.4.33. Early-No-Match

   This header field MAY be specified in a RECOGNIZE method and is used
   to tell the recognizer that it MUST NOT wait for the end of speech
   before processing the collected speech to match active grammars. A
   value of TRUE indicates the recognizer MUST do early matching. The
   default value for this header field if not specified is FALSE. If
   the recognizer does not support the processing of the collected audio
   before the end of speech this header field can be safely ignored.

```
early-no-match  = "Early-No-Match" ":" BOOLEAN CRLF
```

9.4.34. Num-Min-Consistent-Pronunciations

This header field MAY be specified in a START-PHRASE-ENROLLMENT,
"SET-PARAMS", or "GET-PARAMS" method and is used to specify the
minimum number of consistent pronunciations that must be obtained to
voice enroll a new phrase. The minimum value is 1. The default
value is implementation specific and MAY be greater than 1.

```
num-min-consistent-pronunciations  =
            "Num-Min-Consistent-Pronunciations" ":" 1*19DIGIT CRLF
```

9.4.35. Consistency-Threshold

This header field MAY be sent as part of the START-PHRASE-ENROLLMENT,
"SET-PARAMS", or "GET-PARAMS" method. Used during voice-enrollment,
this header field specifies how similar to a previously enrolled
pronunciation of the same phrase an utterance needs to be in order to
be considered "consistent." The higher the threshold, the closer the
match between an utterance and previous pronunciations must be for
the pronunciation to be considered consistent. The range for this
threshold is a float value between is 0.0 to 1.0. The default value
for this header field is implementation specific.

```
consistency-threshold    = "Consistency-Threshold" ":" FLOAT CRLF
```

9.4.36. Clash-Threshold

This header field MAY be sent as part of the START-PHRASE-ENROLLMENT,
SET-PARAMS, or "GET-PARAMS" method. Used during voice-enrollment,
this header field specifies how similar the pronunciations of two
different phrases can be before they are considered to be clashing.
For example, pronunciations of phrases such as "John Smith" and "Jon
Smits" may be so similar that they are difficult to distinguish
correctly. A smaller threshold reduces the number of clashes
detected. The range for this threshold is float value between 0.0
and 1.0. The default value for this header field is implementation
specific. Clash testing can be turned off completely by setting the
Clash-Threshold header field value to 0.

```
clash-threshold          = "Clash-Threshold" ":" FLOAT CRLF
```

9.4.37. Personal-Grammar-URI

This header field specifies the speaker-trained grammar to be used or
referenced during enrollment operations. Phrases are added to this
grammar during enrollment. For example, a contact list for user

"Jeff" could be stored at the Personal-Grammar-URI
"http://myserver.example.com/myenrollmentdb/jeff-list". The
generated grammar syntax MAY be implementation specific. There is no
default value for this header field. This header field MAY be sent
as part of the START-PHRASE-ENROLLMENT, SET-PARAMS, or "GET-PARAMS"
method.

       personal-grammar-uri   = "Personal-Grammar-URI" ":" uri CRLF

9.4.38. Enroll-Utterance

   This header field MAY be specified in the RECOGNIZE method. If this
   header field is set to "true" and an Enrollment is active, the
   RECOGNIZE command MUST add the collected utterance to the personal
   grammar that is being enrolled. The way in which this occurs is
   engine-specific and may be an area of future standardization. The
   default value for this header field is "false".

       enroll-utterance   = "Enroll-Utterance" ":" BOOLEAN CRLF

9.4.39. Phrase-Id

   This header field in a request identifies a phrase in an existing
   personal grammar for which enrollment is desired. It is also
   returned to the client in the RECOGNIZE complete event. This header
   field MAY occur in START-PHRASE-ENROLLMENT, MODIFY-PHRASE or DELETE-
   PHRASE requests. There is no default value for this header field.

       phrase-id   = "Phrase-ID" ":" 1*VCHAR CRLF

9.4.40. Phrase-NL

   This string specifies the interpreted text to be returned when the
   phrase is recognized. This header field MAY occur in START-PHRASE-
   ENROLLMENT and MODIFY-PHRASE requests. There is no default value for
   this header field.

       phrase-nl   = "Phrase-NL" ":" 1*UTFCHAR CRLF

9.4.41. Weight

   The value of this header field represents the occurrence likelihood
   of a phrase in an enrolled grammar. When using grammar enrollment,
   the system is essentially constructing a grammar segment consisting
   of a list of possible match phrases. This can be thought of to be
   similar to the dynamic construction of a <one-of> tag in the W3C
   grammar specification. Each enrolled-phrase becomes an item in the
   list that can be matched against spoken input similar to the <item>

within a <one-of> list. This header field allows you to assign a
weight to the phrase (i.e., <item> entry) in the <one-of> list that
is enrolled. Grammar weights are normalized to a sum of one at
grammar compilation time, so a weight value of 1 for each phrase in
an enrolled grammar list indicates all items in that list have the
same weight. This header field MAY occur in START-PHRASE-ENROLLMENT
and MODIFY-PHRASE requests. The default value for this header field
is implementation specific.

```
weight                    =  "Weight" ":" weight-value CRLF
weight-value              =  FLOAT
```

9.4.42. Save-Best-Waveform

This header field allows the client to request the recognizer
resource to save the audio stream for the best repetition of the
phrase that was used during the enrollment session. The recognizer
MUST attempt to 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 response to the END-PHRASE-ENROLLMENT method. If there
was an error in recording the stream or the audio data is otherwise
not available, the recognizer MUST return an empty Waveform-URI
header field. This header field MAY occur in the START-PHRASE-
ENROLLMENT, SET-PARAMS, and GET-PARAMS methods.

```
save-best-waveform  =  "Save-Best-Waveform" ":" BOOLEAN CRLF
```

9.4.43. New-Phrase-Id

This header field replaces the id used to identify the phrase in a
personal grammar. The recognizer returns the new id when using an
enrollment grammar. This header field MAY occur in MODIFY-PHRASE
requests.

```
new-phrase-id             =  "New-Phrase-ID" ":" 1*VCHAR CRLF
```

9.4.44. Confusable-Phrases-URI

This header field specifies a grammar that defines invalid phrases
for enrollment. For example, typical applications do not allow an
enrolled phrase that is also a command word. This header field MAY
occur in RECOGNIZE requests that are part of an enrollment session.

```
confusable-phrases-uri  =  "Confusable-Phrases-URI" ":" uri CRLF
```

9.4.45. Abort-Phrase-Enrollment

   This header field can optionally be specified in the END-PHRASE-
   ENROLLMENT method to abort the phrase enrollment, rather than
   committing the phrase to the personal grammar.

   abort-phrase-enrollment = "Abort-Phrase-Enrollment" ":"
                               BOOLEAN CRLF

9.5. Recognizer Message Body

   A recognizer message may carry additional data associated with the
   request, response or event. The client may provide the grammar to be
   recognized in DEFINE-GRAMMAR or RECOGNIZE requests. When one or more
   grammars are specified using the DEFINE-GRAMMAR method, the server
   MUST attempt to fetch, compile and optimize the grammar before
   returning a response to the DEFINE-GRAMMAR method. A RECOGNIZE
   request MUST completely specify the grammars to be active during the
   recognition operation, except when the RECOGNIZE method is being used
   to enroll a grammar. During grammar enrollment, such grammars are
   optional. The server resource may send the recognition results in
   the RECOGNITION-COMPLETE event or the GET-RESULT response. Grammars
   and recognition results are carried in the message body of the
   corresponding MRCPv2 messages.

9.5.1. Recognizer Grammar Data

   Recognizer grammar data from the client to the server can be provided
   inline or by reference. Either way, grammar data is carried as typed
   media entities in the message body of the RECOGNIZE or DEFINE-GRAMMAR
   request. All MRCPv2 servers MUST accept grammars in the XML form
   (Media Type application/srgs+xml) of the W3C's XML-based Speech
   Grammar Markup Format (SRGS) [W3C.REC-speech-grammar-20040316] and
   MAY accept grammars in other formats. Examples include but are not
   limited to:
   o   the ABNF form (Media Type application/srgs) of SRGS
   o   Sun's Java Speech Grammar Format (JSGF)
       [refs.javaSpeechGrammarFormat]
   Additionally, MRCPv2 servers MAY support the Semantic Interpretation
   for Speech Recognition (SISR)
   [W3C.REC-semantic-interpretation-20070405] specification.

   When a grammar is specified inline in the request, the client MUST
   provide a Content-ID for that grammar as part of the content header
   fields. If there is no space on the server to store the inline
   grammar, the request MUST return with a Completion-Cause code of 016
   "grammar-definition-failure". Otherwise, the server MUST associate
   the inline grammar block with that Content-ID and MUST store it on

the server for the duration of the session. However, if the
Content-ID is redefined later in the session through a subsequent
DEFINE-GRAMMAR, the inline grammar previously associated with the
Content-ID MUST be freed. If the Content-ID is redefined through a
subsequent DEFINE-GRAMMAR with an empty message body (i.e. no grammar
definition), then in addition to freeing any grammar previously
associated with the Content-ID the server MUST clear all bindings and
associations to the Content-ID. Unless and until subsequently
redefined, this URI MUST be interpreted by the server as one that has
never been set.

Grammars that have been associated with a Content-ID can be
referenced through the "session" URI scheme (see Section 13.6). For
example:
session:help@root-level.store

Grammar data MAY be specified using external URI references. To do
so, the client uses a body of Media Type text/uri-list (see RFC 2483
[RFC2483]) to list the one or more URIs that point to the grammar
data. The client can use a body of Media Type text/grammar-ref-list
(see Section 13.5.1) if it wants to assign weights to the list of
grammar URI. All MRCPv2 servers MUST support grammar access using
the HTTP and HTTPS URI schemes.

If the grammar data the client wishes to be used on a request
consists of a mix of URI and inline grammar data the client uses the
multipart/mixed Media Type to enclose the text/uri-list, application/
srgs or application/srgs+xml content entities. The character set and
encoding used in the grammar data are specified using to standard
Media Type definitions.

When more than one grammar URI or inline grammar block is specified
in a message body of the RECOGNIZE request, the server interprets
this as a list of grammar alternatives to match against.

```
Content-Type:application/srgs+xml
Content-ID:<request1@form-level.store>
Content-Length:...

<?xml version="1.0"?>

<!-- the default grammar language is US English -->
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="en-US" version="1.0" root="request">

<!-- 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>
```

                        SRGS Grammar Example

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

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

                         Grammar Reference Example


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

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

http://www.example.com/Directory-Name-List.grxml
http://www.example.com/Department-List.grxml
http://www.example.com/TAC-Contact-List.grxml

--break
Content-Type:application/srgs+xml
Content-ID:<request1@form-level.store>
Content-Length:...

<?xml version="1.0"?>

<!-- the default grammar language is US English -->
<grammar xmlns="http://www.w3.org/2001/06/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--
```

Mixed Grammar Reference Example

9.5.2. Recognizer Result Data

Recognition results are returned to the client in the message body of
the RECOGNITION-COMPLETE event or the GET-RESULT response message as
described in Section 6.3). Element and attribute descriptions for
the recognition portion of the NLSML format are provided in
Section 9.6 with a normative definition of the schema in
Section 16.1.

```
Content-Type:application/nlsml+xml
Content-Length:...

<?xml version="1.0"?>
<result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
        xmlns:ex="http://www.example.com/example"
        grammar="http://www.example.com/theYesNoGrammar">
    <interpretation>
        <instance>
                <ex:response>yes</ex:response>
        </instance>
        <input>ok</input>
    </interpretation>
</result>
```

Result Example

9.5.3. Enrollment Result Data

   Enrollment results are returned to the client in the message body of
   the RECOGNITION-COMPLETE event as described in Section 6.3). Element
   and attribute descriptions for the enrollment portion of the NLSML
   format are provided in Section 9.7 with a normative definition of the
   schema in Section 16.2.

9.5.4. Recognizer Context Block

   When a client changes servers while operating on the behalf of the
   same incoming communication session, this header field allows the
   client to collect a block of opaque data from one server and provide
   it to another server. This capability is desirable if the client
   needs different language support or because the server issued a
   redirect. Here the first recognizer resource may have collected
   acoustic and other data during its execution of recognition methods.
   After a server switch, communicating this data may allow the
   recognition resource on the new server to provide better recognition.
   This block of data is implementation-specific and MUST be carried as
   Media 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, if
   any, in the message body as an entity of Media Type application/
   octets with a specific Content-ID. The Content-ID value MUST 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 MUST send it in the message body as a typed
   entity with the same Content-ID that it received from the
   "GET-PARAMS". The Content-ID MUST also be sent in the recognizer-
   context-block header field of the "SET-PARAMS" message.

   Each speech recognition implementation choosing to use this mechanism
   to hand off recognizer context data among servers MUST distinguish
   its implementation-specific block of data from other implementations
   by choosing a Content-ID that is recognizable among the participating
   servers and unlikely to collide with values chosen by another
   implementation.

9.6. Recognizer Results

   The recognizer portion of NLSML (see Section 6.3.1) represents
   information automatically extracted from a user's utterances by a
   semantic interpretation component, where "utterance" is to be taken
   in the general sense of a meaningful user input in any modality

supported by the MRCPv2 implementation.

9.6.1. Markup Functions

   MRCPv2 recognition resources employ the Natural Language Semantics
   Markup Language (NLSML) to interpret natural language speech input
   and to format the interpretation for consumption by an MRCPv2 client.

   The elements of the markup fall into the following general functional
   categories: Interpretation, Side Information, and Multi-Modal
   Integration.

9.6.1.1. Interpretation

   Elements and attributes represent the semantics of a user's
   utterance, including the <result>, <interpretation>, and <instance>
   elements. The <result> element contains the full result of
   processing one utterance. It may contain multiple <interpretation>
   elements if the interpretation of the utterance results in multiple
   alternative meanings due to uncertainty in speech recognition or
   natural language understanding. There are at least two reasons for
   providing multiple interpretations:
   1. the client application might have additional information, for
       example, information from a database, that would allow it to
       select a preferred interpretation from among the possible
       interpretations returned from the semantic interpreter.
   2. a client-based dialog manager (e.g. VoiceXML
       [W3C.REC-voicexml20-20040316]) that was unable to select between
       several competing interpretations could use this information to
       go back to the user and find out what was intended. For example,
       it could issue a "SPEAK" request to a synthesizer resource to
       emit "Did you say 'Boston' or 'Austin'?"

9.6.1.2. Side Information

   These are elements and attributes representing additional information
   about the interpretation, over and above the interpretation itself.
   Side information includes:
   1. Whether an interpretation was achieved (the <nomatch> element)
       and the system's confidence in an interpretation (the
       "confidence" attribute of <interpretation>).
   2. Alternative interpretations (<interpretation>)
   3. Input formats and Automatic Speech Recognition (ASR) information:
       the <input> element, representing the input to the semantic
       interpreter.

9.6.1.3. Multi-Modal Integration

   When more than one modality is available for input, the
   interpretation of the inputs need to be coordinated. The "mode"
   attribute of <input> supports this by indicating whether the
   utterance was input by speech, dtmf, pointing, etc. The
   "timestamp_start" and "timestamp_end" attributes of <interpretation>
   also provide for temporal coordination by indicating when inputs
   occurred.

9.6.2. Overview of Recognizer Result Elements and their Relationships

   The recognizer elements in NLSML fall into two categories:
   1. description of the input that was processed.
   2. description of the meaning which was extracted from the input.
   Next to each element are its attributes. In addition, some elements
   can contain multiple instances of other elements. For example, a
   <result> can contain multiple <interpretations>, each of which is
   taken to be an alternative. Similarly, <input> can contain multiple
   child <input> elements which are taken to be cumulative. To
   illustrate the basic usage of these elements, as a simple example,
   consider the utterance "ok" (interpreted as "yes"). The example
   illustrates how that utterance and its interpretation would be
   represented in the NLSML markup.

   <?xml version="1.0"?>
   <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
           xmlns:ex="http://www.example.com/example"
           grammar="http://www.example.com/theYesNoGrammar">
     <interpretation>
       <instance>
          <ex:response>yes</ex:response>
        </instance>
       <input>ok</input>
     </interpretation>
   </result>

   This example includes only the minimum required information. There
   is an overall <result> element which includes one interpretation and
   an input element. The interpretation contains the application-
   specific element "<response>" which is the semantically interpreted
   result.

9.6.3. Elements and Attributes

9.6.3.1. RESULT Root Element

   The root element of the markup is <result>. The <result> element
   includes one or more <interpretation> elements. Multiple
   interpretations can result from ambiguities in the input or in the
   semantic interpretation. If the "grammar" attribute does not apply
   to all of the interpretations in the result it can be overridden for
   individual interpretations at the <interpretation> level.

   Attributes:
   1. grammar: The grammar or recognition rule matched by this result.
       The format of the grammar attribute will match the rule reference
       semantics defined in the grammar specification. Specifically,
       the rule reference is in the external XML form for grammar rule
       references. The markup interpreter needs to know the grammar
       rule that is matched by the utterance because multiple rules may
       be simultaneously active. The value is the grammar URI used by
       the markup interpreter to specify the grammar. The grammar can
       be overridden by a grammar attribute in the <interpretation>
       element if the input was ambiguous as to which grammar it
       matched. If all interpretation elements within the result
       element contain carry their own grammar attributes, the attribute
       can be dropped from the result element.

   <?xml version="1.0"?>
   <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
           grammar="http://www.example.com/grammar">
     <interpretation>
     ....
     </interpretation>
   </result>

9.6.3.2. INTERPRETATION Element

   An <interpretation> element contains a single semantic
   interpretation.

   Attributes:
   1. confidence: A float value from 0.0-1.0 indicating the semantic
       analyzer's confidence in this interpretation. A value of 1.0
       indicates maximum confidence. The values are implementation-
       dependent, but are intended to align with the value
       interpretation for the confidence MRCPv2 header field defined in
       Section 9.4.1. This attribute is optional.
   2. grammar: The grammar or recognition rule matched by this
       interpretation (if needed to override the grammar specification
       at the <interpretation> level.) This attribute is only needed
       under <interpretation> if it is necessary to override a grammar

that was defined at the <result> level.) Note that the grammar
attribute for the interpretation element is OPTIONAL if and only
if the grammar attribute is specified in the result element.

Interpretations MUST be sorted best-first by some measure of
"goodness". The goodness measure is "confidence" if present,
otherwise, it is some implementation-specific indication of quality.

The grammar is expected to be specified most frequently at the
<result> level. However, it can be overridden at the
<interpretation> level because it is possible that different
interpretations may match different grammar rules.

The <interpretation> element includes an optional <input> element
which contains the input being analyzed, and an <instance> element
containing the interpretation of the utterance.

```
<interpretation confidence="0.75"
               grammar="http://www.example.com/grammar">
    ...
</interpretation>
```

9.6.3.3. INSTANCE Element

The <instance> element contains the interpretation of the utterance.
When the Semantic Interpretation for Speech Recognition format is
used, the <instance> element contains the XML serialization of the
result using the approach defined in that specification. When there
is semantic markup in the grammar that does not create semantic
objects, but instead only does a semantic translation of a portion of
the input, such as translating "coke" to "coca-cola", the instance
contains the whole input but with the translation applied. The NLSML
looks like the markup in Figure 2 below. If there are no semantic
objects created, nor any semantic translation the instance value is
the same as the input value.

Attributes:
1. confidence: Each element of the instance may have a confidence
    attribute, defined in the NLSML namespace. The confidence
    attribute contains a float value in the range from 0.0-1.0
    reflecting the system's confidence in the analysis of that slot.
    A value of 1.0 indicates maximum confidence. The values are
    implementation-dependent, but are intended to align with the
    value interpretation for the confidence MRCPv2 header field
    defined in Section 9.4.1. This attribute is optional.

```
<instance>
  <nameAddress>
      <street confidence="0.75">123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
  </nameAddress>
</instance>
<input>
  My address is 123 Maple Street,
  Mill Valley, California, 90952
</input>


<instance>
    I would like to buy a coca-cola
</instance>
<input>
  I would like to buy a coke
</input>
```

                        Figure 2: NSLML Example

9.6.3.4. INPUT Element

   The <input> element is the text representation of a user's input. It
   includes an optional "confidence" attribute which indicates the
   recognizer's confidence in the recognition result (as opposed to the
   confidence in the interpretation, which is indicated by the
   "confidence" attribute of <interpretation>). Optional "timestamp-
   start" and "timestamp-end" attributes indicate the start and end
   times of a spoken utterance, in ISO 8601 format [ISO.8601.1988].

   Attributes:
   1. timestamp-start: The time at which the input began. (optional)
   2. timestamp-end: The time at which the input ended. (optional)
   3. mode: The modality of the input, for example, speech, dtmf, etc.
       (optional)
   4. confidence: the confidence of the recognizer in the correctness
       of the input in the range 0.0 to 1.0 (optional)
   Note that it may not make sense for temporally overlapping inputs to
   have the same mode; however, this constraint is not expected to be
   enforced by implementations.

   When there is no time zone designator, ISO 8601 time representations
   default to local time.

   There are three possible formats for the <input> element.

   1. The <input> element can contain simple text:
   <input>onions</input>
       A future possibility is for <input> to contain not only text but
       additional markup that represents prosodic information that was
       contained in the original utterance and extracted by the speech
       recognizer. This depends on the availability of ASR's that are
       capable of producing prosodic information. MRCPv2 clients MUST
       be prepared to receive such markup and MAY make use of it.
   2. An <input> tag can also contain additional <input> tags. Having
       additional input elements allows the representation to support
       future multi-modal inputs as well as finer-grained speech
       information, such as timestamps for individual words and word-
       level confidences.
   <input>
         <input mode="speech" confidence="0.5"
            timestamp-start="2000-04-03T0:00:00"
            timestamp-end="2000-04-03T0:00:00.2">fried</input>
          <input mode="speech" confidence="1.0"
            timestamp-start="2000-04-03T0:00:00.25"
            timestamp-end="2000-04-03T0:00:00.6">onions</input>
   </input>
   3. Finally, the <input> element can contain <nomatch> and <noinput>
       elements, which describe situations in which the speech
       recognizer received input that it was unable to process, or did
       not receive any input at all, respectively.

9.6.3.5. NOMATCH Element

   The <nomatch> element under <input> is used to indicate that the
   semantic interpreter was unable to successfully match any input with
   confidence above the threshold. It can optionally contain the text
   of the best of the (rejected) matches.

   <interpretation>
         <input confidence="0.1">
         </input>
   </interpretation>
   <interpretation>
      <input mode="speech" confidence="0.1">
        <nomatch>I want to go to New York</nomatch>
      </input>
   </interpretation>

9.6.3.6. NOINPUT Element

   <noinput> indicates that there was no input - a timeout occurred in
   the speech recognizer due to silence.
   <interpretation>
      <instance/>
      <input>
         <noinput/>
      </input>
   </interpretation>

   If there are multiple levels of inputs, the most natural place for
   <nomatch> and <noinput> elements to appear is under the highest level
   of <input> for <noinput>, and under the appropriate level of
   <interpretation> for <nomatch>. So <noinput> means "no input at all"
   and <nomatch> means "no match in speech modality" or "no match in
   dtmf modality". For example, to represent garbled speech combined
   with dtmf "1 2 3 4", the markup would be:
   <input>
      <input mode="speech"><nomatch/></input>
      <input mode="dtmf">1 2 3 4</input>
   </input>

   Note: while <noinput> could be represented as an attribute of input,
   <nomatch> cannot, since it could potentially include PCDATA content
   with the best match. For parallelism, <noinput> is also an element.

9.7. Enrollment Results

   All enrollment elements are contained within a single <enrollment-
   result> element under <result>. The elements are described below and
   have the schema defined in Section 16.2. The following elements are
   defined:

   1. num-clashes
   2. num-good-repetitions
   3. num-repetitions-still-needed
   4. consistency-status
   5. clash-phrase-ids
   6. transcriptions
   7. confusable-phrases

9.7.1. NUM-CLASHES Element

   The <num-clashes> element contains the number of clashes that this
   pronunciation has with other pronunciations in an active enrollment
   session. The associated Clash-Threshold header field determines the
   sensitivity of the clash measurement. Note that clash testing can be

turned off completely by setting the Clash-Threshold header field
value to 0.

9.7.2. NUM-GOOD-REPETITIONS Element

   The <num-good-repetitions> element contains the number of consistent
   pronunciations obtained so far in an active enrollment session.

9.7.3. NUM-REPETITIONS-STILL-NEEDED Element

   The <num-repetitions-still-needed> element contains the number of
   consistent pronunciations that must still be obtained before the new
   phrase can be added to the enrollment grammar. The number of
   consistent pronunciations required is specified by the client in the
   request header field Num-Min-Consistent-Pronunciations. The returned
   value must be 0 before the client can successfully commit a phrase to
   the grammar by ending the enrollment session.

9.7.4. CONSISTENCY-STATUS Element

   The <consistency-status> element is used to indicate how consistent
   the repetitions are when learning a new phrase. It can have the
   values of consistent, inconsistent, and undecided.

9.7.5. CLASH-PHRASE-IDS Element

   The <clash-phrase-ids> element contains the phrase ids of clashing
   pronunciation(s), if any. This element is absent if there are no
   clashes.

9.7.6. TRANSCRIPTIONS Element

   The <transcriptions> element contains the transcriptions returned in
   the last repetition of the phrase being enrolled.

9.7.7. CONFUSABLE-PHRASES Element

   The <confusable-phrases> element contains a list of phrases from a
   command grammar that are confusable with the phrase being added to
   the personal grammar. This element may be absent if there are no
   confusable phrases.

9.8. DEFINE-GRAMMAR

   The DEFINE-GRAMMAR method, from the client to the server, provides
   one or more grammars and requests the server to access, fetch, and
   compile the grammars as needed. The DEFINE-GRAMMAR method
   implementation MUST do a fetch of all external URIs that are part of

that operation. If caching is implemented, this URI fetching MUST
conform to the cache control hints and parameter header fields
associated with the method in deciding whether it should be fetched
from cache or from the external server. If these hints/parameters
are not specified in the method, the values set for the session using
SET-PARAMS/GET-PARAMS apply. If it was not set for the session their
default values apply.

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
process the supplied grammars, the server MUST return a success code
status and the request-state MUST be COMPLETE.

If the recognizer resource could not define the grammar for some
reason, for example if the download failed, the grammar failed to
compile, or the grammar was in an unsupported form, the MRCPv2
response for the DEFINE-GRAMMAR method MUST contain a failure status-
code of 407, and contain a Completion-Cause header field describing
the failure reason.

```
C->S:MRCP/2.0 ... DEFINE-GRAMMAR 543257
Channel-Identifier:32AECB23433801@speechrecog
Content-Type:application/srgs+xml
Content-ID:<request1@form-level.store>
Content-Length:...

<?xml version="1.0"?>

<!-- the default grammar language is US English -->
<grammar xmlns="http://www.w3.org/2001/06/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/2.0 ... 543257 200 COMPLETE
    Channel-Identifier:32AECB23433801@speechrecog
            Completion-Cause:000 success


    C->S:MRCP/2.0 ... DEFINE-GRAMMAR 543258
    Channel-Identifier:32AECB23433801@speechrecog
    Content-Type:application/srgs+xml
    Content-ID:<helpgrammar@root-level.store>
    Content-Length:...

    <?xml version="1.0"?>

    <!-- the default grammar language is US English -->
    <grammar xmlns="http://www.w3.org/2001/06/grammar"
            xml:lang="en-US" version="1.0">

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

    S->C:MRCP/2.0 ... 543258 200 COMPLETE
    Channel-Identifier:32AECB23433801@speechrecog
            Completion-Cause:000 success

    C->S:MRCP/2.0 ... DEFINE-GRAMMAR 543259
    Channel-Identifier:32AECB23433801@speechrecog
    Content-Type:application/srgs+xml
    Content-ID:<request2@field-level.store>
    Content-Length:...

    <?xml version="1.0" encoding="UTF-8"?>

    <!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
                      "http://www.w3.org/TR/speech-grammar/grammar.dtd">

    <grammar xmlns="http://www.w3.org/2001/06/grammar" xml:lang="en"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.w3.org/2001/06/grammar
            http://www.w3.org/TR/speech-grammar/grammar.xsd"
            version="1.0" mode="voice" root="basicCmd">

    <meta name="author" content="Stephanie Williams"/>
```

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

      <ruleref
        uri="http://grammar.example.com/politeness.grxml#startPolite"/>

      <ruleref uri="#command"/>
      <ruleref
        uri="http://grammar.example.com/politeness.grxml#endPolite"/>
    </rule>

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

    <rule id="action">
       <one-of>
          <item weight="10"> open   <tag>open</tag>   </item>
          <item weight="2">  close  <tag>close</tag>  </item>
          <item weight="1">  delete <tag>delete</tag> </item>
          <item weight="1">  move   <tag>move</tag>   </item>
       </one-of>
    </rule>

    <rule id="object">
      <item repeat="0-1">
        <one-of>
          <item> the </item>
          <item> a </item>
        </one-of>
      </item>

      <one-of>
          <item> window </item>
          <item> file </item>
          <item> menu </item>
      </one-of>
    </rule>

    </grammar>


    S->C:MRCP/2.0 ... 543259 200 COMPLETE
    Channel-Identifier:32AECB23433801@speechrecog
            Completion-Cause:000 success

    C->S:MRCP/2.0 ... RECOGNIZE 543260
```

```
      Channel-Identifier:32AECB23433801@speechrecog
              N-Best-List-Length:2
      Content-Type:text/uri-list
      Content-Length:...

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

      S->C:MRCP/2.0 ... 543260 200 IN-PROGRESS
      Channel-Identifier:32AECB23433801@speechrecog

      S->C:MRCP/2.0 ... START-OF-INPUT 543260 IN-PROGRESS
      Channel-Identifier:32AECB23433801@speechrecog

      S->C:MRCP/2.0 ... RECOGNITION-COMPLETE 543260 COMPLETE
      Channel-Identifier:32AECB23433801@speechrecog
      Completion-Cause:000 success
      Waveform-URI:<http://web.media.com/session123/audio.wav>;
                  size=124535;duration=2340
      Content-Type:application/x-nlsml
      Content-Length:...

      <?xml version="1.0"?>
      <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
              xmlns:ex="http://www.example.com/example"
              grammar="session:request1@form-level.store">
              <interpretation>
                  <instance name="Person">
                  <ex:Person>
                      <ex:Name> Andre Roy </ex:Name>
                  </ex:Person>
               </instance>
               <input>   may I speak to Andre Roy </input>
          </interpretation>
      </result>
```

                        Define Grammar Example

9.9. RECOGNIZE

   The RECOGNIZE method from the client to the server requests the
   recognizer to start recognition and provides it with one or more
   grammar references for grammars to match against the input media.
   The RECOGNIZE method can carry header fields to control the
   sensitivity, confidence level and the level of detail in results
   provided by the recognizer. These header field values override the
   current values set by a previous "SET-PARAMS" method.

The RECOGNIZE method can request the recognizer resource to operate in normal or hotword mode as specified by the Recognition-Mode header field. The default value is "normal". If the resource could not start a recognition, the server MUST respond with a failure status-code of 407 and a Completion-Cause header field in the response describing the cause of failure.

The RECOGNIZE request uses the message body to specify the grammars applicable to the request. The active grammar(s) for the request can be specified in one of 3 ways. If the client needs to explicitly control grammar weights for the recognition operation, it MUST employ method 3 below. The order of these grammars specifies the precedence of the grammars which is used when more than one grammar in the list matches the speech; in this case, the grammar with the higher precedence is returned as a match. This precedence capability is useful in applications like VoiceXML browsers to order grammars specified at the dialog, document and root level of a VoiceXML application.

1. The grammar may be placed directly in the message body as typed content. If more than one grammar is included in the body, the order of inclusion controls the corresponding precedence for the grammars during recognition, with earlier grammars in the body having a higher precedence than later ones.

2. The body may contain a list of grammar URIs specified in content of Media Type text/uri-list RFC2483 [RFC2483]. The order of the URIs determines the corresponding precedence for the grammars during recognition, with highest-precedence first and decreasing for each URI thereafter.

3. The body may contain a list of grammar URIs specified in content of Media Type text/grammar-ref-list. This type defines a list of grammar URIs and allows each grammar URI to be assigned a weight in the list. This weight has the same meaning as the weights described in section 2.4.1 of the Speech Grammar Markup Format (SRGS) [W3C.REC-speech-grammar-20040316].

In addition to performing recognition on the input, the recognizer MUST also enroll the collected utterance in a personal grammar if the Enroll-Utterance header field is set to true and an Enrollment is active (via an earlier execution of the START-PHRASE-ENROLLMENT method). If so, and if the RECOGNIZE request contains a Content-ID header field, then the resulting grammar (which includes the personal grammar as a sub-grammar) can be referenced through the "session" URI scheme (see Section 13.6).

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

If the resource was able to queue the request the server MUST return
a success code and request-state of PENDING. This means that the
recognizer is currently active with another request and that this
request has been queued for processing.

If the resource could not start a recognition, the server MUST
respond with a failure status-code of 407 and a Completion-Cause
header field in the response describing the cause of failure.

For the recognizer resource, RECOGNIZE and INTERPRET are the only
requests that returns a 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 RECOGNITION-COMPLETE event (or INTERPRETATION-COMPLETE, if
INTERPRET was the request) with the result of the recognition and a
request-state of COMPLETE.

Large grammars can take a long time for the server to compile. For
grammars which are used repeatedly, the client can improve server
performance by issuing 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:" URI scheme
(see Section 13.6). This also applies in general if the client wants
to repeat recognition with a previous inline grammar.

The RECOGNIZE method implementation MUST do a fetch of all external
URIs that are part of that operation. If caching is implemented,
this URI fetching MUST conform to the cache control hints and
parameter header fields associated with the method in deciding
whether it should be fetched from cache or from the external server.
If these hints/parameters are not specified in the method, the values
set for the session using SET-PARAMS/GET-PARAMS apply. If it was not
set for the session their default values apply.

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 an audio flow is started by the client at the same time
as the RECOGNIZE method is sent, either the audio or the RECOGNIZE
can arrive at the recognizer first. As another example, the client
may choose to continuously send audio to the Server and signal the
Server to recognize using the RECOGNIZE method. Mechanisms to
resolve this condition are outside the scope of this specification.
The recognizer can expect the media to start flowing when it receives
the recognize request, but MUST NOT buffer anything it receives
beforehand in order to preserve the semantics that application
authors expect with respect to the input timers.

When a RECOGNIZE method has been received the recognition is
initiated on the stream. The No-Input-Timer MUST be started at this
time if the Start-Input-Timers header field is specified as "true".
If this header field is set to "false", the No-Input-Timer MUST be
started when it receives the START-INPUT-TIMERS method from the
client. The Recognition-Timer MUST be started when the recognition
resource detects speech or a DTMF digit in the media stream.

Non-Hotword mode recognition:

When the recognition resource detects speech or a DTMF digit in the
media stream it MUST send the START-OF-INPUT event. When enough
speech has been collected for the server to process, the recognizer
can try to match the collected speech with the active grammars. If
the speech collected at this point fully matches with any of the
active grammars, the Speech-Complete-Timer is started. If it matches
partially with one or more of the active grammars, with more speech
needed before a full match is achieved, then the Speech-Incomplete-
Timer is started.

1. When the No-Input-Timer expires, the recognizer MUST complete
with a Completion-Cause code of "no-input-timeout".

2. The recognizer MUST support detecting a no-match condition upon
detecting end of speech. The recognizer MAY support detecting a no-
match condition before waiting for end-of-speech. If this is
supported, this capability is enabled by setting the "Early-No-Match"
header field to "true". Upon detecting a no-match condition the
RECOGNIZE MUST return with "no-match".

3. When the Speech-Incomplete-Timer expires the recognizer SHOULD
complete with a Completion-Cause code of "partial-match", unless the
recognizer cannot differentiate a partial-match in which case it MUST
return a Completion-Cause code of "no-match". The recognizer MAY
return results for the partially matched grammar.

4. When the Speech-Complete-Timer expires the recognizer MUST
complete with a Completion-Cause code of "success".

5. When the Recognition-Timer expires one of the following MUST
happen:

5.1 If there was a partial-match the recognizer SHOULD complete with
a Completion-Cause code of "partial-match-maxtime", unless the
recognizer cannot differentiate a partial-match in which case it MUST
complete with a Completion-Cause code of "no-match-maxtime". The
recognizer MAY return results for the partially matched grammar.

5.2 If there was a full-match the recognizer MUST complete with a
Completion-Cause code of "success-maxtime".

5.3 If there was a no match the recognizer MUST complete with a
Completion-Cause code of "no-match-maxtime".

For the Hotword mode recognition:

Note that for Hotword mode recognition the START-OF-INPUT event is
not generated when speech or a DTMF digit is detected.

1. When the No-Input-Timer expires, the recognizer MUST complete
with a Completion-Cause code of "no-input-timeout".

2. If at any point a match occurs, the RECOGNIZE MUST complete with
a Completion-Cause code of "success".

3. When the Recognition-Timer expires and there is not a match, the
RECOGNIZE MUST complete with a Completion-Cause code of "hotword-
maxtime".

4. When the Recognition-Timer expires and there is a match, the
RECOGNIZE MUST complete with a Completion-Cause code of "success-
maxtime".

5. When the Recognition-Timer is running but the detected speech/
DTMF has not resulted in a match, the Recognition-Timer MUST be
stopped and reset. It MUST then be restarted when speech/DTMF is
again detected.

```
C->S:MRCP/2.0 ... RECOGNIZE 543257
Channel-Identifier:32AECB23433801@speechrecog
       Confidence-Threshold:0.9
Content-Type:application/srgs+xml
Content-ID:<request1@form-level.store>
Content-Length:...

<?xml version="1.0"?>

<!-- the default grammar language is US English -->
<grammar xmlns="http://www.w3.org/2001/06/grammar"
        xml:lang="en-US" version="1.0" root="request">

<!-- 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/2.0 ... 543257 200 IN-PROGRESS
Channel-Identifier:32AECB23433801@speechrecog

S->C:MRCP/2.0 ... START-OF-INPUT 543257 IN-PROGRESS
Channel-Identifier:32AECB23433801@speechrecog

S->C:MRCP/2.0 ... RECOGNITION-COMPLETE 543257 COMPLETE
Channel-Identifier:32AECB23433801@speechrecog
Completion-Cause:000 success
Waveform-URI:<http://web.media.com/session123/audio.wav>;
             size=424252;duration=2543
Content-Type:application/nlsml+xml
Content-Length:...

<?xml version="1.0"?>
<result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
       xmlns:ex="http://www.example.com/example"
       grammar="session:request1@form-level.store">
    <interpretation>
        <instance name="Person">
            <ex:Person>
                <ex:Name> Andre Roy </ex:Name>
            </ex:Person>
        </instance>
            <input>   may I speak to Andre Roy </input>
    </interpretation>
</result>
```

                          RECOGNIZE Example

```
C->S:     MRCP/2.0 ... RECOGNIZE 543257
          Channel-Identifier:32AECB23433801@speechrecog
          Confidence-Threshold:0.9
          Fetch-Timeout:20
          Content-Type:application/srgs+xml
          Content-Length:...

          <?xml version="1.0"? Version="1.0" mode="voice"
              root="Basic md">
           <rule id="rule_list" scope="public">
              <one-of>
                  <item weight=10>
                      <ruleref uri=
            "http://grammar.example.com/world-cities.grxml#canada"/>
                  </item>
                  <item weight=1.5>
                      <ruleref uri=
            "http://grammar.example.com/world-cities.grxml#america"/>
                  </item>
                  <item weight=0.5>
                      <ruleref uri=
            "http://grammar.example.com/world-cities.grxml#india"/>
                  </item>
              </one-of>
           </rule>
```

                      Second RECOGNIZE Example

9.10. STOP

   The "STOP" method from the client to the server tells the resource to
   stop recognition if a request is active. If a RECOGNIZE request is
   active and the "STOP" request successfully terminated it, then the
   response header section 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 is 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 the response MUST contain a status-code of 200 (Success).

```
C->S:    MRCP/2.0 ... RECOGNIZE 543257
         Channel-Identifier:32AECB23433801@speechrecog
         Confidence-Threshold:0.9
         Content-Type:application/srgs+xml
         Content-ID:<request1@form-level.store>
         Content-Length:...

         <?xml version="1.0"?>

         <!-- the default grammar language is US English -->
         <grammar xmlns="http://www.w3.org/2001/06/grammar"
                  xml:lang="en-US" version="1.0" root="request">

         <!-- 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/2.0 ... 543257 200 IN-PROGRESS
          Channel-Identifier:32AECB23433801@speechrecog

 C->S:    MRCP/2.0 ... STOP 543258 200
          Channel-Identifier:32AECB23433801@speechrecog

 S->C:    MRCP/2.0 ... 543258 200 COMPLETE
          Channel-Identifier:32AECB23433801@speechrecog
          Active-Request-Id-List:543257
```

9.11. GET-RESULT

   The GET-RESULT method from the client to the server may be issued
   when the recognizer resource 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 server receives this

   request it re-computes and returns the results according to the
   recognition constraints provided in the GET-RESULT request.

   The GET-RESULT request can specify constraints such as a different
   confidence-threshold, or n-best-list-length. This capability is
   optional for MRCPv2 servers and the automatic speech recognition
   engine in the server MAY return a status of unsupported feature.

   C->S: MRCP/2.0 ... GET-RESULT 543257
            Channel-Identifier:32AECB23433801@speechrecog
            Confidence-Threshold:0.9

   S->C: MRCP/2.0 ... 543257 200 COMPLETE
            Channel-Identifier:32AECB23433801@speechrecog
            Content-Type:application/nlsml+xml
            Content-Length:...

            <?xml version="1.0"?>
            <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                    xmlns:ex="http://www.example.com/example"
                    grammar="session:request1@form-level.store">
               <interpretation>
                  <instance name="Person">
                     <ex:Person>
                         <ex:Name> Andre Roy </ex:Name>
                     </ex:Person>
                  </instance>
                  <input>   may I speak to Andre Roy </input>
               </interpretation>
            </result>

9.12. START-OF-INPUT

   This is an event from the server to the client indicating that the
   recognition resource has detected speech or a DTMF digit in the media
   stream. This event is useful in implementing kill-on-barge-in
   scenarios when a synthesizer resource is in a different session from
   the recognizer resource and hence is not aware of an incoming audio
   source (see Section 8.4.2). In these cases, it is up to the client
   to act as a intermediary and respond to this event by issuing a
   BARGE-IN-OCCURRED event to the synthesizer resource. The recognizer
   resource also MUST send a Proxy-Sync-Id header field with a unique
   value for this event.

   This event MUST be generated by the server irrespective of whether
   the synthesizer and recognizer are on the same server or not.

9.13. START-INPUT-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 (see
   Section 8.4.2). This is useful in the scenario when the recognition
   and synthesizer engines are not in the same session. When a kill-on-
   barge-in prompt is being played, the client may want a RECOGNIZE
   request to be simultaneously active so that it can detect and
   implement kill on barge-in. But at the same time the client doesn't
   want the recognizer to start the no-input timers until the prompt is
   finished. The Start-Input-Timers header field in the RECOGNIZE
   request allows the client to say whether the timers should be started
   immediately or not. If not, the recognizer resource MUST NOT start
   the timers until the client sends a START-INPUT-TIMERS method to the
   recognizer.

9.14. RECOGNITION-COMPLETE

   This is an Event from the recognizer resource to the client
   indicating that the recognition completed. The recognition result is
   sent in the body of the MRCPv2 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 server MUST maintain the recognizer context containing
   the results and the audio waveform input of that recognition until
   the next RECOGNIZE request is issued for that resource or the session
   terminates. A URI to the audio waveform MAY be returned to the
   client in a Waveform-URI header field in the RECOGNITION-COMPLETE
   event. The client can use this URI to retrieve or playback the
   audio.

   Note if an enrollment session was active, the RECOGNITION-COMPLETE
   event can contain either recognition or enrollment results depending
   on what was spoken. The following example shows a complete exchange
   with a recognition result.

   C->S: MRCP/2.0 ... RECOGNIZE 543257
           Channel-Identifier:32AECB23433801@speechrecog
           Confidence-Threshold:0.9
           Content-Type:application/srgs+xml
           Content-ID:<request1@form-level.store>
           Content-Length:...

           <?xml version="1.0"?>

           <!-- the default grammar language is US English -->
           <grammar xmlns="http://www.w3.org/2001/06/grammar"
                   xml:lang="en-US" version="1.0" root="request">

```
            <!-- 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/2.0 ... 543257 200 IN-PROGRESS
            Channel-Identifier:32AECB23433801@speechrecog

   S->C:    MRCP/2.0 ... START-OF-INPUT 543257 IN-PROGRESS
            Channel-Identifier:32AECB23433801@speechrecog

   S->C:    MRCP/2.0 ... RECOGNITION-COMPLETE 543257 COMPLETE
            Channel-Identifier:32AECB23433801@speechrecog
            Completion-Cause:000 success
            Waveform-URI:<http://web.media.com/session123/audio.wav>;
                        size=342456;duration=25435
            Content-Type:application/nlsml+xml
            Content-Length:...

            <?xml version="1.0"?>
            <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                  xmlns:ex="http://www.example.com/example"
                  grammar="session:request1@form-level.store">
               <interpretation>
                  <instance name="Person">
                     <ex:Person>
                         <ex:Name> Andre Roy </ex:Name>
                     </ex:Person>
                  </instance>
                  <input>   may I speak to Andre Roy </input>
               </interpretation>
            </result>
```

   If the result were instead an enrollment result, the final message
   from the server above could have instead been:

```
S->C:    MRCP/2.0 ... RECOGNITION-COMPLETE 543257 COMPLETE
         Channel-Identifier:32AECB23433801@speechrecog
         Completion-Cause:000 success
         Content-Type:application/nlsml+xml
         Content-Length:...

         <?xml version= "1.0"?>
         <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                 grammar="Personal-Grammar-URI">
            <enrollment-result>
                <num-clashes> 2 </num-clashes>
                <num-good-repetitions> 1 </num-good-repetitions>
                <num-repetitions-still-needed>
                   1
                </num-repetitions-still-needed>
                <consistency-status> consistent </consistency-status>
                <clash-phrase-ids>
                    <item> Jeff </item> <item> Andre </item>
                </clash-phrase-ids>
                <transcriptions>
                    <item> m ay b r ow k er </item>
                    <item> m ax r aa k ah </item>
                </transcriptions>
                <confusable-phrases>
                    <item>
                        <phrase> call </phrase>
                        <confusion-level> 10 </confusion-level>
                    </item>
                </confusable-phrases>
            </enrollment-result>
         </result>
```

9.15. START-PHRASE-ENROLLMENT

The START-PHRASE-ENROLLMENT method from the client to the server
starts a new phrase enrollment session during which the client may
call RECOGNIZE multiple times to enroll a new utterance in a grammar.
An enrollment session consists of a set of calls to RECOGNIZE in
which the caller speaks a phrase several times so the system can
"learn" it. The phrase is then added to a personal grammar (speaker-
trained grammar), so that the system can recognize it later.

Only one phrase enrollment session may be active at a time for a
resource. The Personal-Grammar-URI identifies the grammar that is
used during enrollment to store the personal list of phrases. Once
RECOGNIZE is called, the result is returned in a RECOGNITION-COMPLETE
event and may contain either an enrollment result OR a recognition
result for a regular recognition.

Calling END-PHRASE-ENROLLMENT ends the ongoing phrase enrollment
session, which is typically done after a sequence of successful calls
to RECOGNIZE. This method can be called to commit the new phrase to
the personal grammar or to abort the phrase enrollment session.

The grammar to contain the new enrolled phrase, specified by
Personal-Grammar-URI, is created if it does not exist. Also, the
personal grammar may ONLY contain phrases added via a phrase
enrollment session.

The Phrase-ID passed to this method is used to identify this phrase
in the grammar and will be returned as the speech input when doing a
RECOGNIZE on the grammar. The Phrase-NL similarly is returned in a
RECOGNITION-COMPLETE event in the same manner as other Natural
Language (NL) in a grammar. The tag-format of this NL is
implementation specific.

If the client has specified Save-Best-Waveform as true, then the
response after ending the phrase enrollment session MUST contain the
location/URI of a recording of the best repetition of the learned
phrase.

```
C->S:    MRCP/2.0 ... START-PHRASE-ENROLLMENT 543258
         Channel-Identifier:32AECB23433801@speechrecog
         Num-Min-Consistent-Pronunciations:2
         Consistency-Threshold:30
         Clash-Threshold:12
         Personal-Grammar-URI:<personal grammar uri>
         Phrase-Id:<phrase id>
         Phrase-NL:<NL phrase>
         Weight:1
         Save-Best-Waveform:true

S->C:    MRCP/2.0 ... 543258 200 COMPLETE
         Channel-Identifier:32AECB23433801@speechrecog
```

9.16. ENROLLMENT-ROLLBACK

The ENROLLMENT-ROLLBACK method discards the last live utterance from
the RECOGNIZE operation. The client can invoke this method when the
caller provides undesirable input such as non-speech noises, side-
speech, commands, utterance from the RECOGNIZE grammar, etc. Note
that this method does not provide a stack of rollback states.
Executing ENROLLMENT-ROLLBACK twice in succession without an
intervening recognition operation has no effect the second time.

```
C->S:    MRCP/2.0 ... ENROLLMENT-ROLLBACK 543261
         Channel-Identifier:32AECB23433801@speechrecog

S->C:    MRCP/2.0 ... 543261 200 COMPLETE
         Channel-Identifier:32AECB23433801@speechrecog
```

9.17. END-PHRASE-ENROLLMENT

The client MAY call the END-PHRASE-ENROLLMENT method ONLY during an active phrase enrollment session. It MUST NOT be called during an ongoing RECOGNIZE operation. To commit the new phrase in the grammar, the client MAY call this method once successive calls to RECOGNIZE have succeeded and Num-Repetitions-Still-Needed has been returned as 0 in the RECOGNITION-COMPLETE event. Alternatively, the client MAY abort the phrase enrollment session by calling this method with the Abort-Phrase-Enrollment header field.

If the client has specified Save-Best-Waveform as true in the START-PHRASE-ENROLLMENT request, then the response MUST contain a Waveform-URI header whose value is the location/URI of a recording of the best repetition of the learned phrase.

```
C->S:    MRCP/2.0 ... END-PHRASE-ENROLLMENT 543262
         Channel-Identifier:32AECB23433801@speechrecog


S->C:    MRCP/2.0 ... 543262 200 COMPLETE
         Channel-Identifier:32AECB23433801@speechrecog
         Waveform-URI:<http://mediaserver.com/recordings/file1324.wav>;
                     size=242453;duration=25432
```

9.18. MODIFY-PHRASE

The MODIFY-PHRASE method sent from the client to the server is used to change the phrase ID, NL phrase and/or weight for a given phrase in a personal grammar.

If no fields are supplied then calling this method has no effect.

```
C->S:    MRCP/2.0 ... MODIFY-PHRASE 543265
         Channel-Identifier:32AECB23433801@speechrecog
         Personal-Grammar-URI:<personal grammar uri>
         Phrase-Id:<phrase id>
         New-Phrase-Id:<new phrase id>
         Phrase-NL:<NL phrase>
         Weight:1

S->C:    MRCP/2.0 ... 543265 200 COMPLETE
```

                 Channel-Identifier:32AECB23433801@speechrecog

9.19. DELETE-PHRASE

   The DELETE-PHRASE method sent from the client to the server is used
   to delete a phase in a personal grammar added through voice
   enrollment or text enrollment. If the specified phrase does not
   exist, this method has no effect.

   C->S: MRCP/2.0 ... DELETE-PHRASE 543266
            Channel-Identifier:32AECB23433801@speechrecog
            Personal-Grammar-URI:<personal grammar uri>
            Phrase-Id:<phrase id>

   S->C: MRCP/2.0 ... 543266 200 COMPLETE
            Channel-Identifier:32AECB23433801@speechrecog

9.20. INTERPRET

   The INTERPRET method from the client to the server takes as input an
   interpret-text header field containing the text for which the
   semantic interpretation is desired, and returns, via the
   INTERPRETATION-COMPLETE event, an interpretation result which is very
   similar to the one returned from a RECOGNIZE method invocation. Only
   portions of the result relevant to acoustic matching are excluded
   from the result. The interpret-text header field MUST be included in
   the INTERPRET request.

   Recognizer grammar data is treated in the same way as it is when
   issuing a RECOGNIZE method call.

   If a RECOGNIZE, RECORD or another INTERPRET operation is already in
   progress for the resource, the server MUST reject the request with a
   response having a status-code of 402 "Method not valid in this
   state", and a COMPLETE request state.

   C->S: MRCP/2.0 ... INTERPRET 543266
            Channel-Identifier:32AECB23433801@speechrecog
            Interpret-Text:may I speak to Andre Roy
            Content-Type:application/srgs+xml
            Content-ID:<request1@form-level.store>
            Content-Length:...

            <?xml version="1.0"?>
            <!-- the default grammar language is US English -->
            <grammar xmlns="http://www.w3.org/2001/06/grammar"
                    xml:lang="en-US" version="1.0" root="request">
            <!-- 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/2.0 ... 543266 200 IN-PROGRESS
            Channel-Identifier:32AECB23433801@speechrecog

   S->C:    MRCP/2.0 ... INTERPRETATION-COMPLETE 543266 200 COMPLETE
            Channel-Identifier:32AECB23433801@speechrecog
            Completion-Cause:000 success
            Content-Type:application/nlsml+xml
            Content-Length:...

            <?xml version="1.0"?>
            <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                    xmlns:ex="http://www.example.com/example"
                    grammar="session:request1@form-level.store">
                <interpretation>
                    <instance name="Person">
                        <ex:Person>
                            <ex:Name> Andre Roy </ex:Name>
                        </ex:Person>
                    </instance>
                    <input>   may I speak to Andre Roy </input>
                </interpretation>
            </result>
```

9.21. INTERPRETATION-COMPLETE

   This event from the recognition resource to the client indicates that
   the INTERPRET operation is complete. The interpretation result is
   sent in the body of the MRCP message. The request state MUST be set
   to COMPLETE.

   The Completion-Cause header field MUST be included in this event and

MUST be set to an appropriate value from the list of cause codes.

```
C->S:    MRCP/2.0 ... INTERPRET 543266
         Channel-Identifier:32AECB23433801@speechrecog
         Interpret-Text:may I speak to Andre Roy
         Content-Type:application/srgs+xml
         Content-ID:<request1@form-level.store>
         Content-Length:...

         <?xml version="1.0"?>
         <!-- the default grammar language is US English -->
         <grammar xmlns="http://www.w3.org/2001/06/grammar"
                 xml:lang="en-US" version="1.0" root="request">
         <!-- 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/2.0 ... 543266 200 IN-PROGRESS
         Channel-Identifier:32AECB23433801@speechrecog

S->C:     MRCP/2.0 ... INTERPRETATION-COMPLETE 543266 200 COMPLETE
         Channel-Identifier:32AECB23433801@speechrecog
         Completion-Cause:000 success
         Content-Type:application/nlsml+xml
         Content-Length:...

         <?xml version="1.0"?>
         <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                 xmlns:ex="http://www.example.com/example"
                 grammar="session:request1@form-level.store">
             <interpretation>
                 <instance name="Person">
                     <ex:Person>
                         <ex:Name> Andre Roy </ex:Name>
```

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

9.22. DTMF Detection

   Digits received as DTMF tones are delivered to the recognition
   resource in the MRCPv2 server in the RTP stream according to RFC4733
   [RFC4733]. The automatic speech recognizer (ASR) MUST support
   RFC4733 to recognize digits and it MAY support recognizing DTMF tones
   [Q.23] in the audio.

10. Recorder Resource

   This resource captures received audio and video and stores it as
   content pointed to by a URI. The main usages of recorders are
   1. to capture speech audio that may be submitted for recognition at
       a later time, and
   2. recording voice or video mails.
   Both these applications require functionality above and beyond those
   specified by protocols such as RTSP [RFC2326]. This includes Audio
   endpointing (i.e. detecting speech or silence). The support for
   video is optional and is mainly capturing video mails that may
   require the speech or audio processing mentioned above.

   A recorder MUST provide some endpointing capabilities for suppressing
   silence at the beginning and end of a recording, and MAY also
   suppress silence in the middle of a recording. If such suppression
   is done, the recorder MUST maintain timing metadata to indicate the
   actual time stamps of the recorded media.

   See the discussion on the sensitivity of saved waveforms in
   Section 12.

10.1. Recorder State Machine

```
   Idle                      Recording
   State                     State
     |                         |
     |---------RECORD------->|
     |                         |
     |<------STOP-----------|
     |                         |
     |<--RECORD-COMPLETE-----|
     |                         |
     |              |--------|
     |        START-OF-INPUT  |
     |              |------->|
     |                         |
     |              |--------|
     |     START-INPUT-TIMERS |
     |              |------->|
     |                         |
```
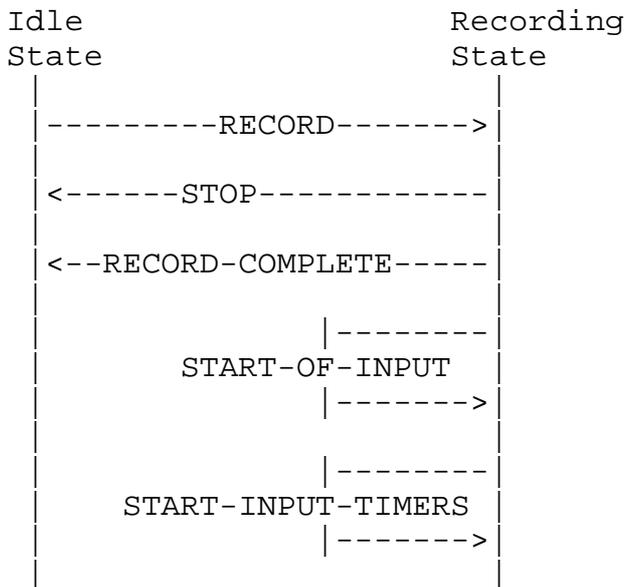
                    Recorder State Machine

10.2. Recorder Methods

   The recorder resource supports the following methods.

   recorder-method   =   "RECORD"
                        /   "STOP"
                        /   "START-INPUT-TIMERS"

10.3. Recorder Events

   The recorder resource may generate the following events.

   recorder-event   =   "START-OF-INPUT"
                        /   "RECORD-COMPLETE"

10.4. Recorder Header Fields

   Method invocations for the recorder resource may contain resource-
   specific header fields containing request options and information to
   augment the Method, Response or Event message it is associated with.

```
recorder-header         =   sensitivity-level
                        /   no-input-timeout
                        /   completion-cause
                        /   completion-reason
                        /   failed-uri
                        /   failed-uri-cause
                        /   record-uri
                        /   media-type
                        /   max-time
                        /   trim-length
                        /   final-silence
                        /   capture-on-speech
                        /   ver-buffer-utterance
                        /   start-input-timers
                        /   new-audio-channel
```

10.4.1. Sensitivity Level

To filter out background noise and not mistake it for speech, the
recorder may support a variable level of sound sensitivity. The
sensitivity-level header field is a float value between 0.0 and 1.0
and allows the client to set the sensitivity level for the recorder.
This header field MAY occur in RECORD, "SET-PARAMS" or "GET-PARAMS".
A higher value for this header field means higher sensitivity. The
default value for this header field is implementation specific.

```
sensitivity-level    =      "Sensitivity-Level" ":" FLOAT CRLF
```

10.4.2. No Input Timeout

When recording is started and there is no speech detected for a
certain period of time, the recorder can send a RECORD-COMPLETE event
to the client and terminate the record operation. The no-input-
timeout header field can set this timeout value. The value is in
milliseconds. This header field MAY occur in RECORD, "SET-PARAMS" or
"GET-PARAMS". The value for this header field ranges from 0 to an
implementation specific maximum value. The default value for this
header field is implementation specific.

```
no-input-timeout     =      "No-Input-Timeout" ":" 1*19DIGIT CRLF
```

10.4.3. Completion Cause

This header field MUST be part of a RECORD-COMPLETE event from the
recorder resource to the client. This indicates the reason behind
the RECORD method completion. This header field MUST be sent in the
RECORD responses if they return with a failure status and a COMPLETE
state. In the ABNF below, the 'cause-code' contains a numerical

value selected from the Cause-Code column of the following table.
The 'cause-name' contains the corresponding token selected from the
Cause-Name column.

```
completion-cause          =  "Completion-Cause" ":" cause-code SP
                             cause-name CRLF
cause-code                =  3DIGIT
cause-name                =  *VCHAR
```

+-----------+----------------------+------------------------------+
| Cause-Code | Cause-Name | Description |
+-----------+----------------------+------------------------------+
000	success-silence	RECORD completed with a
		silence at the end
001	success-maxtime	RECORD completed after
		reaching maximum recording
		time specified in record
		method.
002	noinput-timeout	RECORD failed due to no
		input
003	uri-failure	Failure accessing the record
		URI.
004	error	RECORD request terminated
		prematurely due to a
		recorder error.
+-----------+----------------------+------------------------------+

10.4.4. Completion Reason

This header field MAY be present in a RECORD-COMPLETE event coming
from the recorder resource to the client. It contains the reason
text behind the RECORD request completion. This header field
communicates text describing the reason for the failure.

The completion reason text is provided for client use in logs and for
debugging and instrumentation purposes. Clients MUST NOT interpret
the completion reason text.

```
completion-reason         =  "Completion-Reason" ":"
                             quoted-string CRLF
```

10.4.5. Failed URI

When a recorder method needs to post the audio to a URI and access to
the URI fails, the server MUST provide the failed URI in this header
field in the method response.

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

10.4.6. Failed URI Cause

   When a recorder method needs to post the audio to a URI and access to
   the URI fails, the server MAY provide the URI specific or protocol
   specific response code through this header field in the method
   response. The value encoding is UTF-8 (RFC3629 [RFC3629]) to
   accommodate any access protocol, some of which might have a response
   string instead of a numeric response code.

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

10.4.7. Record URI

   When a recorder method contains this header field the server MUST
   capture the audio and store it. If the header field is present but
   specified with no value, the server MUST store the content locally
   and generate a URI that points to it. This URI is then returned in
   either the "STOP" response or the RECORD-COMPLETE event. If the
   header field in the RECORD method specifies a URI, the server MUST
   attempt to capture and store the audio at that location. If this
   header field is not specified in the RECORD request, the server MUST
   capture the audio, MUST encode it, and MUST send it in the "STOP"
   response or the RECORD-COMPLETE event as a message body. In this
   case, the response carrying the audio content MUST include a Content
   ID (cid) [RFC2392] value in this header pointing to the Content-ID in
   the message body.

   The server MUST also return the size in octets and the duration in
   milliseconds of the recorded audio wave-form as parameters associated
   with the header field.

   Implementations MUST support 'http', 'https' [RFC2616], 'file'
   [RFC3986], and 'cid' [RFC2392] schemes in the URI. Note that
   implementations already exist that support other schemes.

   record-uri   = "Record-URI" ":" ["<" uri ">"
                                 ";" "size" "=" 1*19DIGIT
                                 ";" "duration" "=" 1*19DIGIT] CRLF

10.4.8. Media Type

   A RECORD method MUST contain this header field, which specifies to
   the server the Media Type of the captured audio or video.

   media-type   = "Media-Type" ":" media-type-value
                                 CRLF

10.4.9. Max Time

   When recording is started this specifies the maximum length of the
   recording in milliseconds, calculated from the time the actual
   capture and store begins and is not necessarily the time the RECORD
   method is received. It specifies the duration before silence
   suppression, if any, has been applied by the recorder resource.
   After this time, the recording stops and the server MUST return a
   RECORD-COMPLETE event to the client having a request-state of
   "COMPLETE". This header field MAY occur in RECORD, "SET-PARAMS" or
   "GET-PARAMS". The value for this header field ranges from 0 to an
   implementation specific maximum value. A value of zero means
   infinity and hence the recording continues until one or more of the
   other stop conditions are met. The default value for this header
   field is 0.

   max-time   = "Max-Time" ":" 1*19DIGIT CRLF

10.4.10. Trim-Length

   This header field MAY be sent on a STOP method and specifies the
   length of audio to be trimmed from the end of the recording after the
   stop. The length is interpreted to be in milliseconds. The default
   value for this header field is 0.

   trim-length   = "Trim-Length" ":" 1*19DIGIT CRLF

10.4.11. Final Silence

   When recorder is started and the actual capture begins, this header
   field specifies the length of silence in the audio that is to be
   interpreted as the end of the recording. This header field MAY occur
   in RECORD, "SET-PARAMS" or "GET-PARAMS". The value for this header
   field ranges from 0 to an implementation specific maximum value and
   is interpreted to be in milliseconds. A value of zero means infinity
   and hence the recording will continue until one of the other stop
   conditions are met. The default value for this header field is
   implementation specific.

   final-silence   = "Final-Silence" ":" 1*19DIGIT CRLF

10.4.12. Capture On Speech

   If false, the recorder MUST start capturing immediately when started.
   If true, the recorder MUST wait for the endpointing functionality to
   detect speech before it starts capturing. This header field MAY
   occur in the RECORD, "SET-PARAMS" or "GET-PARAMS". The value for
   this header field is a Boolean. The default value for this header

   field is false.

   capture-on-speech   = "Capture-On-Speech " ":" BOOLEAN CRLF

10.4.13. Ver-Buffer-Utterance

   This header field is the same as the one described for the verifier
   resource (see Section 11.4.14). This tells the server to buffer the
   utterance associated with this recording request into the
   verification buffer. Sending this header field is permitted only if
   the verification buffer is for the session. This buffer is shared
   across resources within a session. It gets instantiated when a
   verifier resource is added to this session and is released when the
   verifier resource is released from the session.

10.4.14. Start Input Timers

   This header field MAY be sent as part of the RECORD request. A value
   of false tells the recorder resource to start the operation, but not
   to start the no-input timer until the client sends a START-INPUT-
   TIMERS request to the recorder resource. This is useful in the
   scenario when the recorder and synthesizer resources are not part of
   the same session. When a kill-on-barge-in prompt is being played,
   the client may want the RECORD request to be simultaneously active so
   that it can detect and implement kill-on-barge-in (see
   Section 8.4.2). But at the same time the client doesn't want the
   recorder resource to start the no-input timers until the prompt is
   finished. The default value is "true".

   start-input-timers   = "Start-Input-Timers" ":"
                                BOOLEAN CRLF

10.4.15. New Audio Channel

   This header field is the same as the one described for the Recognizer
   resource (see Section 9.4.23).

10.5. Recorder Message Body

   If the RECORD request did not have a Record-Uri header field, the
   "STOP" response or the RECORD-COMPLETE event MUST contain a message
   body carrying the captured audio. In this case, the message carrying
   the audio content has a Record-Uri header field with a Content ID
   value pointing to the message body entity that contains the recorded
   audio. See Section 10.4.7 for details.

10.6. RECORD

   The RECORD request places the recorder resource in the Recording
   state. Depending on the header fields specified in the RECORD
   method, the resource may start recording the audio immediately or
   wait for the end pointing functionality to detect speech in the
   audio. The audio is then made available to the client either in the
   message body or as specified by Record-URI.

   The server MUST support the HTTPS URI scheme and MAY support other
   schemes. Note that due to the sensitive nature of voice recordings,
   any protocols used for dereferencing SHOULD employ integrity and
   confidentiality, unless other means, such as physical security, are
   employed.

   If a RECORD operation is already in progress, invoking this method
   causes the server to issue a response having a status code of 402,
   "Method not valid in this state", and a COMPLETE request state.

   If the recording-uri is not valid, a status code of 404, "Illegal
   Value for Header Field", is returned in the response. If it is
   impossible for the server to create the requested stored content, a
   status code of 407, "Method or Operation Failed", is returned.

   If the type specified in the Media-Type header field is not
   supported, the server MUST respond with a status code of 409,
   "Unsupported Header Field Value", with the Media-Type header field in
   its response.

   When the recording operation is initiated, the response indicates an
   IN-PROGRESS request state. The server MAY generate a subsequent
   START-OF-INPUT event when speech is detected. Upon completion of the
   recording operation, the server generates a RECORD-COMPLETE event.

```
C->S:   MRCP/2.0 ... RECORD 543257
        Channel-Identifier:32AECB23433802@recorder
        Record-URI:<file://mediaserver/recordings/myfile.wav>
        Media-Type:audio/wav
        Capture-On-Speech:true
        Final-Silence:300
        Max-Time:6000

S->C:   MRCP/2.0 ... 543257 200 IN-PROGRESS
        Channel-Identifier:32AECB23433802@recorder

S->C:   MRCP/2.0 ... START-OF-INPUT 543257 IN-PROGRESS
        Channel-Identifier:32AECB23433802@recorder

S->C:   MRCP/2.0 ... RECORD-COMPLETE 543257 COMPLETE
        Channel-Identifier:32AECB23433802@recorder
        Completion-Cause:000 success-silence
        Record-URI:<file://mediaserver/recordings/myfile.wav>;
                   size=242552;duration=25645
```

                          RECORD Example

10.7. STOP

   The "STOP" method moves the recorder from the recording state back to
   the idle state. If a RECORD request is active and the "STOP" request
   successfully terminated it, then the STOP response MUST contain an
   active-request-id-list header field containing the "RECORD"
   request-id that was terminated. In this case, no RECORD-COMPLETE
   event is sent for the terminated request. If there was no recording
   active, then the response MUST NOT contain an active-request-id-list
   header field. If the recording was a success the "STOP" response
   MUST contain a Record-URI header field pointing to the recorded audio
   content or to an typed entity in the body of the "STOP" response
   containing the recorded audio. The "STOP" method may have a Trim-
   Length header field, in which case the specified length of audio is
   trimmed from the end of the recording after the stop. In any case,
   the response MUST contain a status-code of 200 (Success).

```
C->S:   MRCP/2.0 ... RECORD 543257
        Channel-Identifier:32AECB23433802@recorder
        Record-URI:<file://mediaserver/recordings/myfile.wav>
        Capture-On-Speech:true
        Final-Silence:300
        Max-Time:6000

S->C:   MRCP/2.0 ... 543257 200 IN-PROGRESS
        Channel-Identifier:32AECB23433802@recorder

S->C:   MRCP/2.0 ... START-OF-INPUT 543257 IN-PROGRESS
        Channel-Identifier:32AECB23433802@recorder

C->S:   MRCP/2.0 ... STOP 543257
        Channel-Identifier:32AECB23433802@recorder
        Trim-Length:200

S->C:   MRCP/2.0 ... 543257 200 COMPLETE
        Channel-Identifier:32AECB23433802@recorder
        Record-URI:<file://mediaserver/recordings/myfile.wav>;
                   size=324253;duration=24561
        Active-Request-Id-List:543257
```

                         STOP Example

10.8. RECORD-COMPLETE

   If the recording completes due to no-input, silence after speech, or
   max-time, the server MUST generate the RECORD-COMPLETE event to the
   client with a request-state of "COMPLETE". If the recording was a
   success the RECORD-COMPLETE event contains a Record-URI header field
   pointing to the recorded audio file on the server or to a typed
   entity in the message body containing the recorded audio.

```
C->S:   MRCP/2.0 ... RECORD 543257
        Channel-Identifier:32AECB23433802@recorder
        Record-URI:<file://mediaserver/recordings/myfile.wav>
        Capture-On-Speech:true
        Final-Silence:300
        Max-Time:6000

S->C:   MRCP/2.0 ... 543257 200 IN-PROGRESS
        Channel-Identifier:32AECB23433802@recorder

S->C:   MRCP/2.0 ... START-OF-INPUT 543257 IN-PROGRESS
        Channel-Identifier:32AECB23433802@recorder

S->C:   MRCP/2.0 ... RECORD-COMPLETE 543257 COMPLETE
        Channel-Identifier:32AECB23433802@recorder
        Completion-Cause:000 success
        Record-URI:<file://mediaserver/recordings/myfile.wav>;
                  size=325325;duration=24652
```

                   RECORD-COMPLETE Example

10.9. START-INPUT-TIMERS

This request is sent from the client to the recorder resource when it
discovers that a kill-on-barge-in prompt has finished playing (see
Section 8.4.2). This is useful in the scenario when the recorder and
synthesizer resources are not in the same MRCPv2 session. When a
kill-on-barge-in prompt is being played, the client wants the RECORD
request to be simultaneously active so that it can detect and
implement kill on barge-in. But at the same time the client doesn't
want the recorder resource to start the no-input timers until the
prompt is finished. The Start-Input-Timers header field in the
RECORD request allows the client to say if the timers should be
started or not. In the above case the recorder resource does not
start the timers until the client sends a START-INPUT-TIMERS method
to the recorder.

10.10. START-OF-INPUT

The START-OF-INPUT event is returned from the server to the client
once the server has detected speech. This event is always returned
by the recording resource when speech has been detected. The
recorder resource also MUST send a Proxy-Sync-Id header field with a
unique value for this event.

```
S->C:   MRCP/2.0 ... START-OF-INPUT 543259 IN-PROGRESS
        Channel-Identifier:32AECB23433801@recorder
        Proxy-Sync-Id:987654321
```

11. Speaker Verification and Identification

   This section describes the methods, responses and events employed by
   MRCPv2 for doing Speaker Verification / Identification.

   Speaker verification is a voice authentication methodology that can
   be used to identify the speaker in order to grant the user access to
   sensitive information and transactions. Because speech is a
   biometric, a number of essential security considerations related to
   biometric authentication technologies apply to its implementation and
   usage. Implementers should carefully read Section 12 in this
   document and the corresponding section of Speechsc Requirements
   [RFC4313].

   In speaker verification, a recorded utterance is compared to a
   previously stored voiceprint which is in turn associated with a
   claimed identity for that user. Verification typically consists of
   two phases: a designation phase to establish the claimed identity of
   the caller and an execution phase in which a voiceprint is either
   created (training) or used to authenticate the claimed identity
   (verification).

   Speaker identification is the process of associating an unknown
   speaker with a member in a population. It does not employ a claim of
   identity. When an individual claims to belong to a group (e.g., one
   of the owners of a joint bank account) a group authentication is
   performed. This is generally implemented as a kind of verification
   involving comparison with more than one voice model. It is sometimes
   called 'multi-verification'. If the individual speaker can be
   identified from the group, this may be useful for applications where
   multiple users share the same access privileges to some data or
   application. Speaker identification and group authentication are
   also done in two phases, a designation phase and an execution phase.
   Note that from a functionality standpoint identification can be
   thought of as a special case of group authentication (if the
   individual is identified) where the group is the entire population,
   although the implementation of speaker identification may be
   different from the way group authentication is performed. To
   accommodate single-voiceprint verification, verification against
   multiple voiceprints, group authentication, and identification, this
   specification provides a single set of methods that can take a list
   of identifiers, called "voiceprint identifiers", and return a list of
   identifiers, with a score for each representing how well the input
   speech matched each identifier. The input and output lists of
   identifiers do not have to match, allowing a vendor-specific group
   identifier to be used as input to indicate that identification is to
   be performed. In this specification, the terms "Identification" and
   "Multi-verification" are used to indicate that the input represents a

group (potentially the entire population) and that results for
multiple voiceprints may be returned.

It is possible for a verifier resource to share the same session with
a recognizer resource or to operate independently. In order to share
the same session, the verifier and recognizer resources MUST be
allocated from within the same SIP dialog. Otherwise, an independent
verifier resource, running on the same physical server or a separate
one, will be set up. Note that in addition to allowing both
resources to be allocated in the same INVITE, it is possible to
allocate one initially and the other later via a re-INVITE.

Some of the speaker verification methods, described below, apply only
to a specific mode of operation.

The verifier resource has a verification buffer associated with it
(see Section 11.4.14). This allows the storage of speech utterances
for the purposes of verification, identification or training from the
buffered speech. This buffer is owned by the verifier resource but
other input resources such as the recognition resource or recorder
resource may write to it. This allows the speech received as part of
a recognition or recording operation to be later used for
verification, identification or training. Access to the buffer is
limited to one operation at time. Hence when the resource is doing
read, write or delete operation such as a RECOGNIZE with ver-buffer-
utterance turned on, another operation involving the buffer fails
with a status-code of 402. The verification buffer can be cleared by
a CLEAR-BUFFER request from the client and is freed when the verifier
resource is deallocated or the session with the server terminates.

The verification buffer is different from collecting waveforms and
processing them using either the real time audio stream or stored
audio, because this buffering mechanism does not simply accumulate
speech to a buffer. The verification buffer may contain additional
information gathered by the recognition resource that serves to
improve verification performance.

11.1. Speaker Verification State Machine

Speaker verification may operate in a training or a verification
session. Starting one of these sessions does not change the state of
the verifier resource, i.e. it remains idle. Once a verification or
training session is started, then utterances are trained or verified
by calling the VERIFY or VERIFY-FROM-BUFFER method. The state of the
verifier resources goes from IDLE to VERIFYING state each time VERIFY
or VERIFY-FROM-BUFFER is called.

    Idle   Session Opened   Verifying/Training

```
         State                 State                  State
          |                     |                      |
          |--START-SESSION--->  |                      |
          |                     |                      |
          |                     |---------|            |
          |                     |   START-SESSION       |
          |                     |<--------|            |
          |                     |                      |
          |<--END-SESSION-----  |                      |
          |                     |                      |
          |                     |--------VERIFY-------->|
          |                     |                      |
          |                     |---VERIFY-FROM-BUFFER--->|
          |                     |                      |
          |                     |---------|            |
          |                     |  VERIFY-ROLLBACK      |
          |                     |<--------|            |
          |                     |                      |
          |                     |             |--------|
          |                     |  GET-INTERMEDIATE-RESULT |
          |                     |             |------->|
          |                     |                      |
          |                     |             |--------|
          |                     |   START-INPUT-TIMERS  |
          |                     |             |------->|
          |                     |                      |
          |                     |             |--------|
          |                     |   START-OF-INPUT      |
          |                     |             |------->|
          |                     |                      |
          |                     |<-VERIFICATION-COMPLETE--|
          |                     |                      |
          |                     |<-------STOP-----------|
          |                     |                      |
          |                     |---------|            |
          |                     |    STOP              |
          |                     |<--------|            |
          |                     |                      |
          |---------|           |                      |
          |    STOP             |                      |
          |<--------|           |                      |
          |                     |                      |
          |                     |---------|            |
          |                     |   CLEAR-BUFFER        |
          |                     |<--------|            |
          |                     |                      |
          |---------|           |                      |
          |   CLEAR-BUFFER       |                      |
          |<--------|           |                      |
```

```
      |                       |                             |
      |                       |----------|                  |
      |                       |    QUERY-VOICEPRINT          |
      |                       |<---------|                  |
      |                       |                             |
      |----------|            |                             |
      |  QUERY-VOICEPRINT     |                             |
      |<---------|            |                             |
      |                       |                             |
      |                       |----------|                  |
      |                       |    DELETE-VOICEPRINT         |
      |                       |<---------|                  |
      |                       |                             |
      |----------|            |                             |
      |  DELETE-VOICEPRINT    |                             |
      |<---------|            |                             |
```
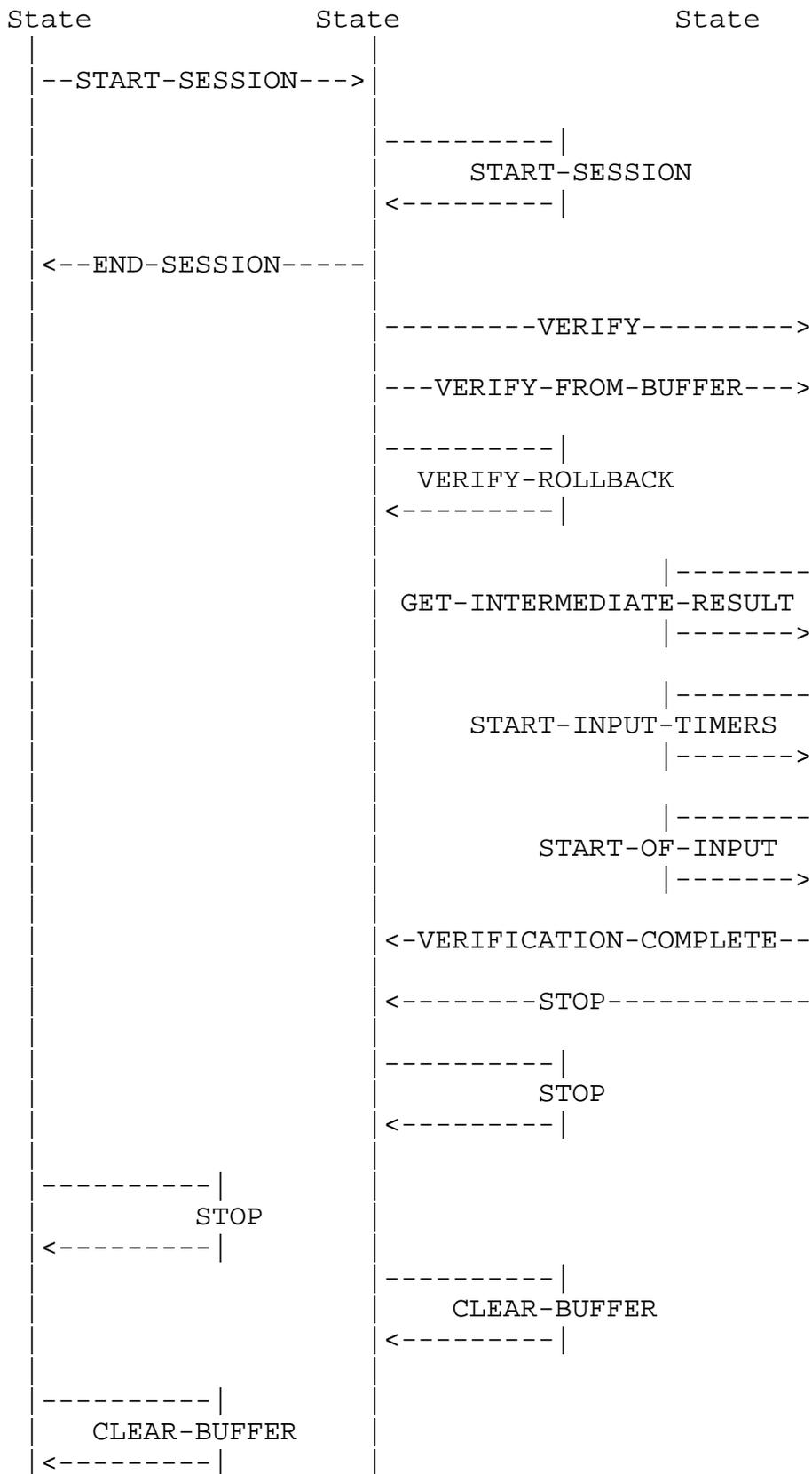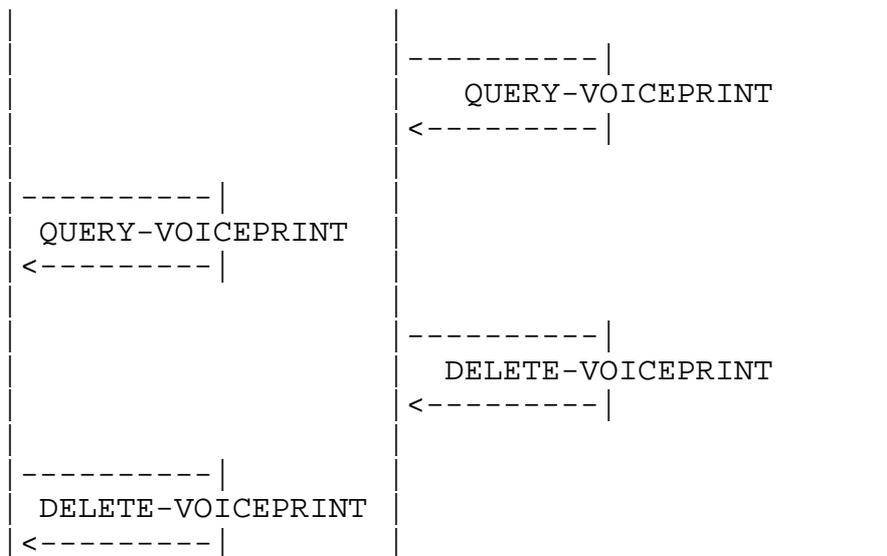
                     Verifier Resource State Machine

11.2. Speaker Verification Methods

   The verifier resource supports the following methods.

   verifier-method   = "START-SESSION"
                          / "END-SESSION"
                          / "QUERY-VOICEPRINT"
                          / "DELETE-VOICEPRINT"
                          / "VERIFY"
                          / "VERIFY-FROM-BUFFER"
                          / "VERIFY-ROLLBACK"
                          / "STOP"
                          / "CLEAR-BUFFER"
                          / "START-INPUT-TIMERS"
                          / "GET-INTERMEDIATE-RESULT"

   These methods allow the client to control the mode and target of
   verification or identification operations within the context of a
   session. All the verification input operations that occur within a
   session may be used to create, update, or validate against the
   voiceprint specified during the session. At the beginning of each
   session the verifier resource is reset to the state it had prior to
   any previous verification session.

   Verification/identification operations can be executed against live
   or buffered audio. The verifier resource provides methods for
   collecting and evaluating live audio data, and methods for
   controlling the verifier resource and adjusting its configured
   behavior.

There are no dedicated methods for collecting buffered audio data.
This is accomplished by calling VERIFY, RECOGNIZE or RECORD as
appropriate for the resource, with the header field Ver-Buffer-
Utterance. Then, when the following method is called verification is
performed using the set of buffered audio.
1.   VERIFY-FROM-BUFFER

The following methods are used for verification of live audio
utterances :
1.   VERIFY
2.   START-INPUT-TIMERS

The following methods are used for configuring the verifier resource
and for establishing resource states :
1.   START-SESSION
2.   END-SESSION
3.   QUERY-VOICEPRINT
4.   DELETE-VOICEPRINT
5.   VERIFY-ROLLBACK
6.   "STOP"
7.   CLEAR-BUFFER

The following method allows the polling a Verification in progress
for intermediate results.
1.   GET-INTERMEDIATE-RESULT

11.3. Verification Events

   The verifier resource generates the following events.

   verifier-event   =   "VERIFICATION-COMPLETE"
                        /   "START-OF-INPUT"

11.4. Verification Header Fields

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

```
verification-header       =   repository-uri
                          /   voiceprint-identifier
                          /   verification-mode
                          /   adapt-model
                          /   abort-model
                          /   min-verification-score
                          /   num-min-verification-phrases
                          /   num-max-verification-phrases
                          /   no-input-timeout
                          /   save-waveform
                          /   media-type
                          /   waveform-uri
                          /   voiceprint-exists
                          /   ver-buffer-utterance
                          /   input-waveform-uri
                          /   completion-cause
                          /   completion-reason
                          /   speech-complete-timeout
                          /   new-audio-channel
                          /   abort-verification
                          /   start-input-timers
```

11.4.1. Repository-URI

This header field specifies the voiceprint repository to be used or
referenced during speaker verification or identification operations.
This header field is required in the START-SESSION, QUERY-VOICEPRINT
and DELETE-VOICEPRINT methods.

```
repository-uri            =   "Repository-URI" ":" uri CRLF
```

11.4.2. Voiceprint-Identifier

This header field specifies the claimed identity for verification
applications. The claimed identity MAY be used to specify an
existing voiceprint or to establish a new voiceprint. This header
field MUST be present in the QUERY-VOICEPRINT and DELETE-VOICEPRINT
methods. The Voiceprint-Identifier MUST be present in the START-
SESSION method for verification operations. For Identification or
Multi-Verification operations this header field MAY contain a list of
voiceprint identifiers separated by semi-colons. For identification
operations the client MAY also specify a voiceprint group identifier
instead of a list of voiceprint identifiers.

```
voiceprint-identifier     =   "Voiceprint-Identifier" ":"
                              vid *[";" vid] CRLF
vid                       =   1*VCHAR ["." 1*VCHAR]
```

11.4.3. Verification-Mode

   This header field specifies the mode of the verifier resource and is
   set by the START-SESSION method. Acceptable values indicate whether
   the verification session will train a voiceprint ("train") or verify/
   identify using an existing voiceprint ("verify").

   Training and verification sessions both require the voiceprint
   Repository-URI to be specified in the START-SESSION. In many usage
   scenarios, however, the system does not know the speaker's claimed
   identity until a recognition operation has, for example, recognized
   an account number to which the user desires access. In order to
   allow the first few utterances of a dialog to be both recognized and
   verified, the verifier resource on the MRCPv2 server retains a
   buffer. In this buffer, the MRCPv2 server accumulates recognized
   utterances. The client can later execute a verification method and
   apply the buffered utterances to the current verification session.

   Some voice user interfaces may require additional user input that
   should not be subject to verification. For example, the user's input
   may have been recognized with low confidence and thus require a
   confirmation cycle. In such cases, the client should not execute the
   VERIFY or VERIFY-FROM-BUFFER methods to collect and analyze the
   caller's input. A separate recognizer resource can analyze the
   caller's response without any participation by the verifier resource.

   Once the following conditions have been met:
   1. Voiceprint identity has been successfully established through the
       voiceprint identifier header fields of the START-SESSION method,
       and
   2. the verification mode has been set to one of "train" or "verify",
   the verifier resource may begin providing verification information
   during verification operations. If the verifier resource does not
   reach one of the two major states ("train" or "verify") , it MUST
   report an error condition in the MRCPv2 status code to indicate why
   the verifier resource is not ready for the corresponding usage.

   The value of verification-mode is persistent within a verification
   session. If the client attempts to change the mode during a
   verification session, the verifier resource reports an error and the
   mode retains its current value.

   verification-mode   = "Verification-Mode" ":"
                                   verification-mode-string

   verification-mode-string   = "train"
                                / "verify"

11.4.4. Adapt-Model

   This header field indicates the desired behavior of the verifier
   resource after a successful verification operation. If the value of
   this header field is "true", the sever SHOULD use audio collected
   during the verification session to update the voiceprint to account
   for ongoing changes in a speaker's incoming speech characteristics,
   unless local policy prohibits updating the voiceprint. If the value
   is "false" (the default), the server MUST NOT update the voiceprint.
   This header field MAY occur in the START-SESSION method.

   adapt-model   = "Adapt-Model" ":" BOOLEAN CRLF

11.4.5. Abort-Model

   The Abort-Model header field indicates the desired behavior of the
   verifier resource upon session termination. If the value of this
   header field is "true", the server MUST discard any pending changes
   to a voiceprint due to verification training or verification
   adaptation. If the value is "false" (the default), the server MUST
   commit any pending changes for a training session or a successful
   verification session to the voiceprint repository. A value of "true"
   for Abort-Model overrides a value of "true" for the Adapt-Model
   header field. This header field MAY occur in the END-SESSION method.

   abort-model   = "Abort-Model" ":" BOOLEAN CRLF

11.4.6. Min-Verification-Score

   The Min-Verification-Score header field, when used with a verifier
   resource through a "SET-PARAMS", "GET-PARAMS" or START-SESSION
   method, determines the minimum verification score for which a
   verification decision of "accepted" may be declared by the server.
   This is a float value between -1.0 and 1.0 that determines the
   minimum verification score for which a verification decision of
   "accepted" may be declared by the server. The default value for this
   header field is implementation specific.

   min-verification-score = "Min-Verification-Score" ":"
                                [%x2D] FLOAT CRLF

11.4.7. Num-Min-Verification-Phrases

   The Num-Min-Verification-Phrases header field is used to specify the
   minimum number of valid utterances before a positive decision is
   given for verification. The value for this header field is an
   integer and the default value is 1. The verifier resource MUST NOT
   declare a verification 'accepted' unless Num-Min-Verification-Phrases

valid utterances have been received. The minimum value is 1. This
header field MAY occur in START-SESSION, "SET-PARAMS" or
"GET-PARAMS".

```
num-min-verification-phrases =  "Num-Min-Verification-Phrases" ":"
                                1*19DIGIT CRLF
```

11.4.8. Num-Max-Verification-Phrases

The Num-Max-Verification-Phrases header field is used to specify the
number of valid utterances required before a decision is forced for
verification. The verifier resource MUST NOT return a decision of
'undecided' once Num-Max-Verification-Phrases have been collected and
used to determine a verification score. The value for this header
field is an integer and the minimum value is 1. The default value is
implementation-specific. This header field MAY occur in START-
SESSION, "SET-PARAMS" or "GET-PARAMS".

```
num-max-verification-phrases =  "Num-Max-Verification-Phrases" ":"
                                1*19DIGIT CRLF
```

11.4.9. No-Input-Timeout

The No-Input-Timeout header field sets the length of time from the
start of the verification timers (see START-INPUT-TIMERS) until the
declaration of a no-input event in the VERIFICATION-COMPLETE server
event message. The value is in milliseconds. This header field MAY
occur in VERIFY, "SET-PARAMS" or "GET-PARAMS". The value for this
header field ranges from 0 to an implementation specific maximum
value. The default value for this header field is implementation
specific.

```
no-input-timeout          = "No-Input-Timeout" ":" 1*19DIGIT CRLF
```

11.4.10. Save-Waveform

This header field allows the client to request that the verifier
resource save the audio stream that was used for verification/
identification. The verifier resource MUST attempt to record the
audio and make it available to the client in the form of a URI
returned in the Waveform-URI header field in the VERIFICATION-
COMPLETE event. If there was an error in recording the stream or the
audio content is otherwise not available, the verifier resource MUST
return an empty Waveform-URI header field. The default value for
this header field is "false". This header field MAY appear in the
VERIFY method. Note that this header field does not appear in the
VERIFY-FROM-BUFFER method since it only controls whether or not to
save the waveform for live verification / identification operations.

```
    save-waveform               =  "Save-Waveform" ":" BOOLEAN CRLF
```

11.4.11. Media Type

   This header field MAY be specified in the SET-PARAMS, GET-PARAMS or
   the VERIFY methods and tells the server resource the Media Type of
   the captured audio or video such as the one captured and returned by
   the Waveform-URI header field.

```
    media-type                  =  "Media-Type" ":" media-type-value
                                   CRLF
```

11.4.12. Waveform-URI

   If the Save-Waveform header field is set to true, the verifier
   resource MUST attempt to record the incoming audio stream of the
   verification into a file and provide a URI for the client to access
   it. This header field MUST be present in the VERIFICATION-COMPLETE
   event if the Save-Waveform header field was set to true by the
   client. The value of the header field MUST be empty if there was
   some error condition preventing the server from recording.
   Otherwise, the URI generated by the server MUST be globally unique
   across the server and all its verification sessions. The content
   MUST be available via the URI until the verification session ends.
   Since the Save-Waveform header field applies only to live
   verification / identification operations, the server can return the
   Waveform-URI only in the VERIFICATION-COMPLETE event for live
   verification / identification operations.

   The server MUST also return the size in octets and the duration in
   milliseconds of the recorded audio wave-form as parameters associated
   with the header field.

```
    waveform-uri                =  "Waveform-URI" ":" ["<" uri ">"
                                   ";" "size" "=" 1*19DIGIT
                                   ";" "duration" "=" 1*19DIGIT] CRLF
```

11.4.13. Voiceprint-Exists

   This header field MUST be returned in QUERY-VOICEPRINT and DELETE-
   VOICEPRINT responses. This is the status of the voiceprint specified
   in the QUERY-VOICEPRINT method. For the DELETE-VOICEPRINT method
   this header field indicates the status of the voiceprint at the
   moment the method execution started.

```
    voiceprint-exists    =  "Voiceprint-Exists" ":" BOOLEAN CRLF
```

11.4.14. Ver-Buffer-Utterance

   This header field is used to indicate that this utterance could be
   later considered for Speaker Verification. This way, a client can
   request the server to buffer utterances while doing regular
   recognition or verification activities and speaker verification can
   later be requested on the buffered utterances. This header field is
   OPTIONAL in the RECOGNIZE, VERIFY and RECORD methods. The default
   value for this header field is "false".

   ver-buffer-utterance   = "Ver-Buffer-Utterance" ":" BOOLEAN
                               CRLF

11.4.15. Input-Waveform-Uri

   This header field specifies stored audio content that the client
   requests the server to fetch and process according to the current
   verification mode, either to train the voiceprint or verify a claimed
   identity. This header field enables the client to implement the
   buffering use case where the recognizer and verifier resources are in
   different sessions and the verification buffer technique cannot be
   used. It MAY be specified on the VERIFY request.

   input-waveform-uri   =   "Input-Waveform-URI" ":" uri CRLF

11.4.16. Completion-Cause

   This header field MUST be part of a VERIFICATION-COMPLETE event from
   the verifier resource to the client. This indicates the cause of
   VERIFY or VERIFY-FROM-BUFFER method completion. This header field
   MUST be sent in the VERIFY, VERIFY-FROM-BUFFER, and QUERY-VOICEPRINT
   responses, if they return with a failure status and a COMPLETE state.
   In the ABNF below, the 'cause-code' contains a numerical value
   selected from the Cause-Code column of the following table. The
   'cause-name' contains the corresponding token selected from the
   Cause-Name column.

   completion-cause   =   "Completion-Cause" ":" cause-code SP
                                 cause-name CRLF
   cause-code   =   3DIGIT
   cause-name   =   *VCHAR

+------------+------------------------+-------------------------+
| Cause-Code | Cause-Name | Description |
+------------+------------------------+-------------------------+
000	success	VERIFY or
		VERIFY-FROM-BUFFER
		request completed
		successfully. The verify
		decision can be
		"accepted", "rejected",
		or "undecided".
001	error	VERIFY or
		VERIFY-FROM-BUFFER
		request terminated
		prematurely due to a
		verifier resource or
		system error.
002	no-input-timeout	VERIFY request completed
		with no result due to a
		no-input-timeout.
003	too-much-speech-timeout	VERIFY request completed
		with no result due to too
		much speech.
004	speech-too-early	VERIFY request completed
		with no result due to
		spoke too soon.
005	buffer-empty	VERIFY-FROM-BUFFER
		request completed with no
		result due to empty
		buffer.
006	out-of-sequence	Verification operation
		failed due to
		out-of-sequence method
		invocations. For example
		calling VERIFY before
		QUERY-VOICEPRINT.
007	repository-uri-failure	Failure accessing
		Repository URI.
008	repository-uri-missing	Repository-uri is not
		specified.
009	voiceprint-id-missing	Voiceprint-identification
		is not specified.
010	voiceprint-id-not-exist	Voiceprint-identification
		does not exist in the
		voiceprint repository.

011	speech-not-usable	VERIFY request completed
		with no result because
		the speech was not usable
		(too noisy, too short,
		etc.)
+-----------+-----------------------+--------------------------+

11.4.17. Completion Reason

This header field MAY be specified in a VERIFICATION-COMPLETE event
coming from the verifier resource to the client. It contains the
reason text behind the VERIFY request completion. This header field
communicates text describing the reason for the failure.

The completion reason text is provided for client use in logs and for
debugging and instrumentation purposes. Clients MUST NOT interpret
the completion reason text.

```
completion-reason        =  "Completion-Reason" ":"
                            quoted-string CRLF
```

11.4.18. Speech Complete Timeout

This header field is the same as the one described for the Recognizer
resource. See Section 9.4.15. This header field MAY occur in
VERIFY, SET-PARAMS, or GET-PARAMS.

11.4.19. New Audio Channel

This header field is the same as the one described for the Recognizer
resource. See Section 9.4.23. This header field MAY be specified in
a VERIFY request.

11.4.20. Abort-Verification

This header field MUST be sent in a "STOP" request to indicate
whether or not to abort a VERIFY method in progress. A value of
"true" requests the server to discard the results. A value of
"false" requests the server to return in the "STOP" response the
verification results obtained up to the point it received the "STOP"
request.

```
Abort-verification   =  "Abort-Verification " ":" BOOLEAN CRLF
```

11.4.21. Start Input Timers

   This header field MAY be sent as part of a VERIFY request. A value
   of false tells the verifier resource to start the VERIFY operation,
   but not to start the no-input timer yet. The verifier resource MUST
   NOT start the timers until the client sends a START-INPUT-TIMERS
   request to the resource. This is useful in the scenario when the
   verifier and synthesizer resources are not part of the same session.
   In this scenario, when a kill-on-barge-in prompt is being played, the
   client may want the VERIFY request to be simultaneously active so
   that it can detect and implement kill-on-barge-in (see
   Section 8.4.2). But at the same time the client doesn't want the
   verifier resource to start the no-input timers until the prompt is
   finished. The default value is "true".

   start-input-timers   = "Start-Input-Timers" ":"
                                BOOLEAN CRLF

11.5. Verification Message Body

   A verification response or event message may carry additional data as
   described in the following subsection.

11.5.1. Verification Result Data

   Verification results are returned to the client in the message body
   of the VERIFICATION-COMPLETE event or the GET-INTERMEDIATE-RESULT
   response message as described in Section 6.3. Element and attribute
   descriptions for the verification portion of the NLSML format are
   provided in Section 11.5.2 with a normative definition of the schema
   in Section 16.3.

11.5.2. Verification Result Elements

   All verification elements are contained within a single
   <verification-result> element under <result>. The elements are
   described below and have the schema defined in Section 16.2. The
   following elements are defined:

   1.   Voiceprint
   2.   Incremental
   3.   Cumulative
   4.   Decision
   5.   Utterance-Length
   6.   Device
   7.   Gender

8.   Adapted
9.   Verification-Score
10.   Vendor-Specific-Results

11.5.2.1. Voiceprint

This element in the verification results provides information on how
the speech data matched a single voiceprint. The result data
returned may have more than one such entity in the case of
Identification or Multi-Verification. Each "<voiceprint>" element
and the XML data within the element describe verification result
information for how well the speech data matched that particular
voiceprint. The list of voiceprint element data are ordered
according to their cumulative verification match scores, with the
highest score first.

11.5.2.2. Cumulative

Within each "<voiceprint>" element there MUST be a "<cumulative>"
element with the cumulative scores of how well multiple utterances
matched the voiceprint.

11.5.2.3. Incremental

The first "<voiceprint>" element MAY contain an "<incremental>"
element with the incremental scores of how well the last utterance
matched the voiceprint.

11.5.2.4. Decision

This element is found within the "<incremental>" or "<cumulative>"
element within the verification results. Its value indicates the
verification decision. It can have the values of "accepted",
"rejected" or "undecided".

11.5.2.5. Utterance-Length

This element MAY occur within either the "<incremental>" or
"<cumulative>" elements within the first "<voiceprint>" element. Its
value indicates the size in milliseconds, respectively, of the last
utterance or the cumulated set of utterances.

11.5.2.6. Device

This element is found within the incremental or cumulative element
within the verification results. Its value indicates the apparent
type of device used by the caller as determined by the verifier
resource. It can have the values of "cellular-phone", "electret-

phone", "carbon-button-phone", or "unknown".

11.5.2.7. Gender

This element is found within the incremental or cumulative element
within the verification results. Its value indicates the apparent
gender of the speaker as determined by the verifier resource. It can
have the values of "male", "female" or "unknown".

11.5.2.8. Adapted

This element is found within the first "<voiceprint>" element within
the verification results. When verification is trying to confirm the
voiceprint, this indicates if the voiceprint has been adapted as a
consequence of analyzing the source utterances. It is not returned
during verification training. The value can be "true" or "false".

11.5.2.9. Verification-Score

This element is found within the incremental or cumulative element
within the verification results. Its value indicates the score of
the last utterance as determined by verification.

During verification, the higher the score the more likely it is that
the speaker is the same one as the one who spoke the voiceprint
utterances. During training, the higher the score the more likely
the speaker is to have spoken all of the analyzed utterances. The
value is a floating point between -1.0 and 1.0. If there are no such
utterances the score is -1. Note that the verification score is not
a probability value.

11.5.2.10. Vendor-Specific-Results

MRCPv2 servers MAY send verification results that contain
implementation specific data which augment the information provided
by the MRCPv2-defined elements. Such data might be useful to clients
who have private knowledge of how to interpret these schema
extensions. Implementation specific additions to the verification
results schema MUST belong to the vendor's own namespace. In the
result structure, either they MUST be indicated by a namespace prefix
declared within the result, or they MUST be children of an element
identified as belonging to the respective namespace.

The following example shows the results of three voiceprints. Note
that the first one has crossed the verification score threshold, and
the speaker has been accepted. The voiceprint was also adapted with
the most recent utterance.

```
<?xml version="1.0"?>
<result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
        grammar="What-Grammar-URI">
  <verification-result>
    <voiceprint id="johnsmith">
      <adapted> true </adapted>
      <incremental>
        <utterance-length> 500 </utterance-length>
        <device> cellular-phone </device>
        <gender> male </gender>
        <decision> accepted </decision>
        <verification-score> 0.98514 </verification-score>
      </incremental>
      <cumulative>
        <utterance-length> 10000 </utterance-length>
        <device> cellular-phone </device>
        <gender> male </gender>
        <decision> accepted </decision>
        <verification-score> 0.96725</verification-score>
      </cumulative>
    </voiceprint>
    <voiceprint id="marysmith">
      <cumulative>
        <verification-score> 0.93410 </verification-score>
      </cumulative>
    </voiceprint>
    <voiceprint uri="juniorsmith">
      <cumulative>
        <verification-score> 0.74209 </verification-score>
      </cumulative>
    </voiceprint>
  </verification-result>
</result>
```

                    Verification Results Example 1

   In this next example, the verifier has enough information to decide
   to reject the speaker.

```
<?xml version="1.0"?>
<result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
        xmlns:xmpl="http://www.example.org/2003/12/mrcpv2"
        grammar="What-Grammar-URI">
  <verification-result>
    <voiceprint id="johnsmith">
      <incremental>
        <utterance-length> 500 </utterance-length>
        <device> cellular-phone </device>
        <gender> male </gender>
        <verification-score> 0.88514 </verification-score>
        <xmpl:raspiness> high </xmpl:raspiness>
        <xmpl:emotion> sadness </xmpl:emotion>
      </incremental>
      <cumulative>
        <utterance-length> 10000 </utterance-length>
        <device> cellular-phone </device>
        <gender> male </gender>
        <decision> rejected </decision>
        <verification-score> 0.9345 </verification-score>
      </cumulative>
    </voiceprint>
  </verification-result>
</result>
```

                    Verification Results Example 2

11.6. START-SESSION

   The START-SESSION method starts a Speaker Verification or
   Identification session. Execution of this method places the verifier
   resource into its initial state. If this method is called during an
   ongoing verification session, the previous session is implicitly
   aborted. If this method is invoked when VERIFY or VERIFY-FROM-BUFFER
   is active, the method fails and the server returns a status code of
   402.

   Upon completion of the START-SESSION method, the verifier resource
   MUST have terminated any ongoing verification session, and cleared
   any voiceprint designation.

   A verification session is associated with the voiceprint repository
   to be used during the session. This is specified through the
   "Repository-URI" header field (see Section 11.4.1).

   The START-SESSION method also establishes, through the Voiceprint-
   Identifier header field, which voiceprints are to be matched or
   trained during the verification session. If this is an

Identification session or if the client wants to do Multi-
Verification, the Voiceprint-Identifier header field contains a list
of semi-colon separated voiceprint identifiers.

The Adapt-Model header field may also be present in the START-SESSION
request to indicate whether or not to adapt a voiceprint based on
data collected during the session (if the voiceprint verification
phase succeeds). By default, the voiceprint model MUST NOT be
adapted with data from a verification session.

The START-SESSION also determines whether the session is for a train
or verify of a voiceprint. Hence the Verification-Mode header field
MUST be sent in every START-SESSION request. The value of the
Verification-Mode header field MUST be one of either "train" or
"verify".

Before a verification/identification session is started, the client
MAY only request that VERIFY-ROLLBACK and generic "SET-PARAMS" and
"GET-PARAMS" operations be performed on the verifier resource. The
server MUST return status-code 402 "Method not valid in this state"
for all other verification operations.

A verifier resource MUST NOT have more than a single session active
at one time.

```
C->S:   MRCP/2.0 ... START-SESSION 314161
        Channel-Identifier:32AECB23433801@speakverify
        Repository-URI:http://www.example.com/voiceprintdbase/
        Voiceprint-Mode:verify
        Voiceprint-Identifier:johnsmith.voiceprint
        Adapt-Model:true

S->C:   MRCP/2.0 ... 314161 200 COMPLETE
        Channel-Identifier:32AECB23433801@speakverify
```

11.7. END-SESSION

The END-SESSION method terminates an ongoing verification session and
releases the verification voiceprint resources. The session may
terminate in one of three ways:
1. abort - the voiceprint adaptation or creation may be aborted so
    that the voiceprint remains unchanged (or is not created).
2. commit - when terminating a voiceprint training session, the new
    voiceprint is committed to the repository.
3. adapt - an existing voiceprint is modified using a successful
    verification.

The Abort-Model header field MAY be included in the END-SESSION to

control whether or not to abort any pending changes to the
voiceprint. The default behavior is to commit (not abort) any
pending changes to the designated voiceprint.

The END-SESSION method may be safely executed multiple times without
first executing the START-SESSION method. Any additional executions
of this method without an intervening use of the START-SESSION method
have no effect on the verifier resource.

The following example assumes there is either a training session or a
verification session in progress.

```
C->S:  MRCP/2.0 ... END-SESSION 314174
       Channel-Identifier:32AECB23433801@speakverify
       Abort-Model:true

S->C:  MRCP/2.0 ... 314174 200 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify
```

11.8. QUERY-VOICEPRINT

The QUERY-VOICEPRINT method is used to get status information on a
particular voiceprint and can be used by the client to ascertain if a
voiceprint or repository exists and if it contains trained
voiceprints.

The response to the QUERY-VOICEPRINT request contains an indication
of the status of the designated voiceprint in the Voiceprint-Exists
header field, allowing the client to determine whether to use the
current voiceprint for verification, train a new voiceprint, or
choose a different voiceprint.

A voiceprint is completely specified by providing a repository
location and a voiceprint identifier. The particular voiceprint or
identity within the repository is specified by a string identifier
that is unique within the repository. The Voiceprint-Identifier
header field carries this unique voiceprint identifier within a given
repository.

The following example assumes a verification session is in progress
and the voiceprint exists in the voiceprint repository.

```
C->S:  MRCP/2.0 ... QUERY-VOICEPRINT 314168
       Channel-Identifier:32AECB23433801@speakverify
       Repository-URI:http://www.example.com/voiceprints/
       Voiceprint-Identifier:johnsmith.voiceprint

S->C:  MRCP/2.0 ... 314168 200 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify
       Repository-URI:http://www.example.com/voiceprints/
       Voiceprint-Identifier:johnsmith.voiceprint
       Voiceprint-Exists:true
```

The following example assumes that the URI provided in the
Repository-URI header field is a bad URI.

```
C->S:  MRCP/2.0 ... QUERY-VOICEPRINT 314168
       Channel-Identifier:32AECB23433801@speakverify
       Repository-URI:http://www.example.com/bad-uri/
       Voiceprint-Identifier:johnsmith.voiceprint

S->C:  MRCP/2.0 ... 314168 405 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify
       Repository-URI:http://www.example.com/bad-uri/
       Voiceprint-Identifier:johnsmith.voiceprint
       Completion-Cause:007 repository-uri-failure
```

11.9. DELETE-VOICEPRINT

   The DELETE-VOICEPRINT method removes a voiceprint from a repository.
   This method MUST carry the Repository-URI and Voiceprint-Identifier
   header fields.

   If the corresponding voiceprint does not exist, the DELETE-VOICEPRINT
   method MUST return a 200 status code.

   The following example demonstrates a DELETE-VOICEPRINT operation to
   remove a specific voiceprint.

```
C->S:  MRCP/2.0 ... DELETE-VOICEPRINT 314168
       Channel-Identifier:32AECB23433801@speakverify
       Repository-URI:http://www.example.com/bad-uri/
       Voiceprint-Identifier:johnsmith.voiceprint

S->C:  MRCP/2.0 ... 314168 200 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify
```

11.10. VERIFY

   The VERIFY method is used to request that the verifier resource
   either train/adapt the voiceprint or verify/identify a claimed
   identity. If the voiceprint is new or was deleted by a previous
   DELETE-VOICEPRINT method, the VERIFY method trains the voiceprint.
   If the voiceprint already exits, it is adapted and not retrained by
   the VERIFY command.

   C->S: MRCP/2.0 ... VERIFY 543260
          Channel-Identifier:32AECB23433801@speakverify

   S->C: MRCP/2.0 ... 543260 200 IN-PROGRESS
          Channel-Identifier:32AECB23433801@speakverify

   When the VERIFY request is completes, the MRCPv2 server MUST send a
   VERIFICATION-COMPLETE event to the client.

11.11. VERIFY-FROM-BUFFER

   The VERIFY-FROM-BUFFER method directs the verifier resource to verify
   buffered audio against a voiceprint. Only one VERIFY or VERIFY-FROM-
   BUFFER method may be active for a verifier resource at a time.

   The buffered audio is not consumed by this method and thus VERIFY-
   FROM-BUFFER may be invoked multiple times by the client to attempt
   verification against different voiceprints.

   For the VERIFY-FROM-BUFFER method, the server MAY optionally return
   an IN-PROGRESS response before the VERIFICATION-COMPLETE event.

   When the VERIFY-FROM-BUFFER method is invoked and the verification
   buffer is in use by another resource sharing it, the server MUST
   return an IN-PROGRESS response and wait until the buffer is available
   to it. The verification buffer is owned by the verifier resource but
   is shared with write access from other input resources on the same
   session. Hence, it is considered to be in use if there is a read or
   write operation such as a RECORD or RECOGNIZE with the Ver-Buffer-
   Utterance header field set to "true" on a resource that shares this
   buffer. Note that if a RECORD or RECOGNIZE method returns with a
   failure cause code, the VERIFY-FROM-BUFFER request waiting to process
   that buffer MUST also fail with a Completion-Cause of 005 (buffer-
   empty).

   The following example illustrates the usage of some buffering
   methods. In this scenario the client first performed a live
   verification, but the utterance had been rejected. In the meantime,
   the utterance is also saved to the audio buffer. Then, another

voiceprint is used to do verification against the audio buffer and
the utterance is accepted. For the example, we assume both Num-Min-
Verification-Phrases and Num-Max-Verification-Phrases are 1.

```
C->S:  MRCP/2.0 ... START-SESSION 314161
       Channel-Identifier:32AECB23433801@speakverify
       Verification-Mode:verify
       Adapt-Model:true
       Repository-URI:http://www.example.com/voiceprints
       Voiceprint-Identifier:johnsmith.voiceprint

S->C:  MRCP/2.0 ... 314161 200 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify

C->S:  MRCP/2.0 ... VERIFY 314162
       Channel-Identifier:32AECB23433801@speakverify
       Ver-buffer-utterance:true

S->C:  MRCP/2.0 ... 314162 200 IN-PROGRESS
       Channel-Identifier:32AECB23433801@speakverify

S->C:  MRCP/2.0 ... VERIFICATION-COMPLETE 314162 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify
       Completion-Cause:000 success
       Content-Type:application/nlsml+xml
       Content-Length:...

       <?xml version="1.0"?>
       <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
               grammar="What-Grammar-URI">
         <verification-result>
           <voiceprint id="johnsmith">
             <incremental>
               <utterance-length> 500 </utterance-length>
               <device> cellular-phone </device>
               <gender> female </gender>
               <decision> rejected </decision>
               <verification-score> 0.05465 </verification-score>
             </incremental>
             <cumulative>
               <utterance-length> 500 </utterance-length>
               <device> cellular-phone </device>
               <gender> female </gender>
               <decision> rejected </decision>
               <verification-score> 0.05465 </verification-score>
             </cumulative>
           </voiceprint>
         </verification-result>
```

```
            </result>

   C->S:   MRCP/2.0 ... QUERY-VOICEPRINT 314163
           Channel-Identifier:32AECB23433801@speakverify
           Repository-URI:http://www.example.com/voiceprints/
           Voiceprint-Identifier:johnsmith

   S->C:   MRCP/2.0 ... 314163 200 COMPLETE
           Channel-Identifier:32AECB23433801@speakverify
           Repository-URI:http://www.example.com/voiceprints/
           Voiceprint-Identifier:johnsmith.voiceprint
           Voiceprint-Exists:true

   C->S:   MRCP/2.0 ... START-SESSION 314164
           Channel-Identifier:32AECB23433801@speakverify
           Verification-Mode:verify
           Adapt-Model:true
           Repository-URI:http://www.example.com/voiceprints
           Voiceprint-Identifier:marysmith.voiceprint

   S->C:   MRCP/2.0 ... 314164 200 COMPLETE
           Channel-Identifier:32AECB23433801@speakverify

   C->S:   MRCP/2.0 ... VERIFY-FROM-BUFFER 314165
           Channel-Identifier:32AECB23433801@speakverify

   S->C:   MRCP/2.0 ... 314165 200 IN-PROGRESS
           Channel-Identifier:32AECB23433801@speakverify

   S->C:   MRCP/2.0 ... VERIFICATION-COMPLETE 314165 COMPLETE
           Channel-Identifier:32AECB23433801@speakverify
           Completion-Cause:000 success
           Content-Type:application/nlsml+xml
           Content-Length:...

           <?xml version="1.0"?>
           <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                   grammar="What-Grammar-URI">
             <verification-result>
               <voiceprint id="marysmith">
                 <incremental>
                   <utterance-length> 1000 </utterance-length>
                   <device> cellular-phone </device>
                   <gender> female </gender>
                   <decision> accepted </decision>
                   <verification-score> 0.98 </verification-score>
                 </incremental>
                 <cumulative>
```

```
                    <utterance-length> 1000 </utterance-length>
                    <device> cellular-phone </device>
                    <gender> female </gender>
                    <decision> accepted </decision>
                    <verification-score> 0.98 </verification-score>
                 </cumulative>
             </voiceprint>
          </verification-result>
       </result>
```

```
   C->S:  MRCP/2.0 ... END-SESSION 314166
          Channel-Identifier:32AECB23433801@speakverify
```

```
   S->C:  MRCP/2.0 ... 314166 200 COMPLETE
          Channel-Identifier:32AECB23433801@speakverify
```

                        VERIFY-FROM-BUFFER example

11.12. VERIFY-ROLLBACK

   The VERIFY-ROLLBACK method discards the last buffered utterance or
   discards the last live utterances (when the mode is "train" or
   "verify"). The client should invoke this method when the user
   provides undesirable input such as non-speech noises, side-speech,
   out-of-grammar utterances, commands, etc. Note that this method does
   not provide a stack of rollback states. Executing VERIFY-ROLLBACK
   twice in succession without an intervening recognition operation has
   no effect on the second attempt.

```
   C->S:  MRCP/2.0 ... VERIFY-ROLLBACK 314165
          Channel-Identifier:32AECB23433801@speakverify
```

```
   S->C:  MRCP/2.0 ... 314165 200 COMPLETE
          Channel-Identifier:32AECB23433801@speakverify
```

                        VERFY-ROLLBACK Example

11.13. STOP

   The "STOP" method from the client to the server tells the verifier
   resource to stop the VERIFY or VERIFY-FROM-BUFFER request if one is
   active. If such a request is active and the "STOP" request
   successfully terminated it, then the response header section contains
   an active-request-id-list header field containing the request-id of
   the VERIFY or VERIFY-FROM-BUFFER request that was terminated. In
   this case, no VERIFICATION-COMPLETE event is sent for the terminated
   request. If there was no verify request active, then the response

MUST NOT contain an active-request-id-list header field. Either way
the response MUST contain a status-code of 200 (Success).

The "STOP" method can carry a Abort-Verification header field which
specifies if the verification result until that point should be
discarded or returned. If this header field is not present or if the
value is "true", the verification result is discarded and the "STOP"
response does not contain any result data. If the header field is
present and its value is "false", the "STOP" response MUST contain a
Completion-Cause header field and carry the Verification result data
in its body.

An aborted VERIFY request does an automatic roll-back and hence does
not affect the cumulative score. A VERIFY request that was stopped
with no Abort-Verification header field or with the Abort-
Verification header field set to "false" does affect cumulative
scores and would need to be explicitly rolled-back if the client does
not want the verification result considered in the cumulative scores.

The following example assumes a voiceprint identity has already been
established.

```
C->S:  MRCP/2.0 ... VERIFY 314177
       Channel-Identifier:32AECB23433801@speakverify

S->C:  MRCP/2.0 ... 314177 200 IN-PROGRESS
       Channel-Identifier:32AECB23433801@speakverify

C->S:  MRCP/2.0 ... STOP 314178
       Channel-Identifier:32AECB23433801@speakverify

S->C:  MRCP/2.0 ... 314178 200 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify
       Active-Request-Id-List:314177
```

                        STOP verification Example

11.14. START-INPUT-TIMERS

This request is sent from the client to the verifier resource to
start the no-input timer, usually once the client has ascertained
that any audio prompts to the user have played to completion.

```
C->S:  MRCP/2.0 ... START-INPUT-TIMERS 543260
       Channel-Identifier:32AECB23433801@speakverify

S->C:  MRCP/2.0 ... 543260 200 COMPLETE
       Channel-Identifier:32AECB23433801@speakverify
```

11.15. VERIFICATION-COMPLETE

   The VERIFICATION-COMPLETE event follows a call to VERIFY or VERIFY-
   FROM-BUFFER and is used to communicate the verification results to
   the client. The event message body contains only verification
   results.

   S->C: MRCP/2.0 ... VERIFICATION-COMPLETE 543259 COMPLETE
          Completion-Cause:000 success
          Content-Type:application/nlsml+xml
          Content-Length:...

          <?xml version="1.0"?>
          <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                  grammar="What-Grammar-URI">
            <verification-result>
              <voiceprint id="johnsmith">
                <incremental>
                  <utterance-length> 500 </utterance-length>
                  <device> cellular-phone </device>
                  <gender> male </gender>
                  <decision> accepted </decision>
                  <verification-score> 0.85 </verification-score>
                </incremental>
                <cumulative>
                  <utterance-length> 1500 </utterance-length>
                  <device> cellular-phone </device>
                  <gender> male </gender>
                  <decision> accepted </decision>
                  <verification-score> 0.75 </verification-score>
                </cumulative>
              </voiceprint>
            </verification-result>
          </result>

11.16. START-OF-INPUT

   The START-OF-INPUT event is returned from the server to the client
   once the server has detected speech. This event is always returned
   by the verifier resource when speech has been detected, irrespective
   of whether the recognizer and verifier resources share the same
   session or not.

   S->C: MRCP/2.0 ... START-OF-INPUT 543259 IN-PROGRESS
          Channel-Identifier:32AECB23433801@speakverify

11.17. CLEAR-BUFFER

   The CLEAR-BUFFER method can be used to clear the verification buffer.
   This buffer is used to buffer speech during a recognition, record or
   verification operations that may later be used by VERIFY-FROM-BUFFER.
   As noted before, the buffer associated with the verifier resource is
   shared by other input resources like recognizers and recorders.
   Hence, a CLEAR-BUFFER request fails if the verification buffer is in
   use. This can happen when any one of the input resources that shares
   this buffer has an active read or write operation such as RECORD,
   RECOGNIZE or VERIFY with the Ver-Buffer-Utterance header field set to
   "true".

   C->S: MRCP/2.0 ... CLEAR-BUFFER 543260
          Channel-Identifier:32AECB23433801@speakverify

   S->C: MRCP/2.0 ... 543260 200 COMPLETE
          Channel-Identifier:32AECB23433801@speakverify

11.18. GET-INTERMEDIATE-RESULT

   A client can use the GET-INTERMEDIATE-RESULT method to poll for
   intermediate results of a verification request that is in progress.
   Invoking this method does not change the state of the resource. The
   verifier resource collects the accumulated verification results and
   returns the information in the method response. The message body in
   the response to a GET-INTERMEDIATE-RESULT REQUEST contains only
   verification results. The method response MUST NOT contain a
   Completion-Cause header field as the request is not yet complete. If
   the resource does not have a verification in progress the response
   has a 402 failure status-code and no result in the body.

```
C->S:   MRCP/2.0 ... GET-INTERMEDIATE-RESULT 543260
        Channel-Identifier:32AECB23433801@speakverify

S->C:   MRCP/2.0 ... 543260 200 COMPLETE
        Channel-Identifier:32AECB23433801@speakverify
        Content-Type:application/nlsml+xml
        Content-Length:...

        <?xml version="1.0"?>
        <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                grammar="What-Grammar-URI">
          <verification-result>
            <voiceprint id="marysmith">
              <incremental>
                <utterance-length> 50 </utterance-length>
                <device> cellular-phone </device>
                <gender> female </gender>
                <decision> undecided </decision>
                <verification-score> 0.85 </verification-score>
              </incremental>
              <cumulative>
                <utterance-length> 150 </utterance-length>
                <device> cellular-phone </device>
                <gender> female </gender>
                <decision> undecided </decision>
                <verification-score> 0.65 </verification-score>
              </cumulative>
            </voiceprint>
          </verification-result>
        </result>
```

12. Security Considerations

MRCPv2 is designed to comply with the security-related requirements
documented in SpeechSC Requirements [RFC4313]. Implementers and
users of MRCPv2 are strongly encouraged to read the Security
Considerations section of [RFC4313], because that document contains
discussion of a number of important security issues associated with
the utilization of speech as biometric authentication technology, and
on the threats against systems which store recorded speech, contain
large corpora of voiceprints, and send and receive sensitive
information based on voice input to a recognizer or speech output
from a synthesizer. Specific security measures employed by MRCPv2
are summarized in the following subsections. See the corresponding
sections of this specification for how the security-related machinery
is invoked by individual protocol operations.

12.1. Rendezvous and Session Establishment

   MRCPv2 control sessions are established as media sessions described
   by SDP within the context of a SIP dialog. In order to ensure secure
   rendezvous between MRCPv2 clients and servers, the following are
   required:

   1.   The SIP implementation in MRCPv2 clients and servers MUST support
        SIP digest authentication [RFC3261] and SHOULD employ it.
   2.   The SIP implementation in MRCPv2 clients and servers SHOULD
        employ 'sips' URIs, including that clients and servers SHOULD set
        up TLS [RFC5246] connections.
   3.   If media stream cryptographic keying is done through SDP (e.g.
        using [RFC4568]), the MRCPv2 clients and servers MUST employ
        SIPS:.
   4.   When TLS is used for SIP, the client MUST verify the identity of
        the server to which it connects, following the rules and
        guidelines defined in [RFC5922].

12.2. Control channel protection

   Sensitive data is carried over the MRCPv2 control channel. This
   includes things like the output of speech recognition operations,
   speaker verification results, input to text-to-speech conversion,
   personally-identifying grammars, etc. For this reason MRCPv2 servers
   must be properly authenticated and the control channel must permit
   the use of both confidentiality and integrity for the data. To
   ensure control channel protection, MRCPv2 clients and servers MUST
   support TLS and SHOULD utilize it by default unless alternative
   control channel protection is used. When TLS is used, the client
   MUST verify the identity of the server to which it connects,
   following the rules and guidelines defined in [RFC4572]. If there
   are multiple TLS-protected channels between the client and the
   server, the server MUST NOT send a response to the client over a
   channel for which the TLS identities of the server or client differ
   from the channel over which the server received the corresponding
   request. Alternative control channel protection MAY be used if
   desired (e.g. Security Architecture for the Internet Protocol
   (IPsec) [RFC4301]).

12.3. Media session protection

   Sensitive data is also carried on media sessions terminating on
   MRCPv2 servers (the other end of a media channel may or may not be on
   the MRCPv2 client). This data includes the user's spoken utterances
   and the output of text-to-speech operations. MRCPv2 servers MUST
   support a security mechanism for protection of audio media sessions.
   MRCPv2 clients that originate or consume audio similarly MUST support

a security mechanism for protection of the audio. If appropriate, usage of the Secure Real-time Transport Protocol (SRTP) [RFC3711] is RECOMMENDED.

12.4. Indirect Content Access

MCRPv2 employs content indirection extensively. Content may be fetched and/or stored based on URI-addressing on systems other than the MRCPv2 client or server. Not all of the stored content is necessarily sensitive (e.g. XML schemas), but the majority generally needs protection, and some indirect content, such as voice recordings and voiceprints, are extremely sensitive and must always be protected. MRCPv2 clients and servers MUST implement HTTPS for indirect content access, and SHOULD employ secure access for all sensitive indirect content. Other secure URI-schemes such as Secure FTP (FTPS) [RFC4217] MAY also be used. See Section 6.2.15 for the header fields used to transfer cookie information between the MRCPv2 client and server if needed for authentication.

MRCPv2 makes no inherent assumptions about the lifetime and access controls associated with a URI. For example, if neither authentication nor scheme-specific access controls are used, a leak of the URI is equivalent to a leak of the content. Moreover, MRCPv2 makes no specific demands on the lifetime of a URI. If a server offers a URI and the client takes a long, long time to access that URI, the server may have removed the resource in the interim time period. MRCPv2 deals with this case by using the URI access scheme's resource not found error, such as 404 for HTTPS. How long a server should keep a dynamic resource available is highly application and context dependent. However, the server SHOULD keep the resource available for a reasonable amount of time to make it likely the client will have the resource available when the client needs the resource. Conversely, to mitigate state exhaustion attacks, MRCPv2 servers are not obligated to keep resources and resource state in perpetuity. The server SHOULD delete dynamically-generated resources associated with an MRCPv2 session when the session ends.

One method to avoid resource leakage is for the server to use one-time resource URIs. In this instance, there can be only a single access to the underlying resource using the given URI. A downside to this approach is if an attacker uses the URI before the client uses the URI, then the client is denied the resource. Other methods would be to adopt a mechanism similar to the URLAUTH IMAP extension [RFC4467], where the server sets cryptographic checks on URI usage, as well as capabilities for expiration, revocation, and so on. Specifying such a mechanism is beyond the scope of this document.

12.5. Protection of stored media

   MRCPv2 applications often require the use of stored media. Voice
   recordings are both stored (e.g. for diagnosis and system tuning),
   and fetched (for replaying utterances into multiple MRCPv2
   resources). Voiceprints are fundamental to the speaker
   identification and verification functions. This data can be
   extremely sensitive and can present substantial privacy and
   impersonation risks if stolen. Systems employing MRCPv2 should be
   deployed in ways that minimize these risks. The SpeechSC
   Requirements [RFC4313] contains a more extensive discussion of these
   risks and ways they may be mitigated.

12.6. DTMF and recognition buffers

   DTMF buffers and recognition buffers may grow large enough to exceed
   the capabilities of a server, and the server MUST be prepared to
   gracefully handle resource consumption. A server MAY respond with
   the appropriate recognition incomplete if the server is in danger of
   running out of resources.

13. IANA Considerations

13.1. New registries

   This section describes the name spaces (registries) for MRCPv2 that
   IANA is requested to create and maintain. Assignment/registration
   policies are described in RFC5226 [RFC5226].

13.1.1. MRCPv2 resource types

   IANA SHALL create a new name space of "MRCPv2 resource types". All
   maintenance within and additions to the contents of this name space
   MUST be according to the "Standards Action" registration policy. The
   initial contents of the registry, defined in Section 4.2, are given
   below:

Resource type	Resource description	Reference
speechrecog	Speech Recognizer	[RFCXXXX]
dtmfrecog	DTMF Recognizer	[RFCXXXX]
speechsynth	Speech Synthesizer	[RFCXXXX]
basicsynth	Basic Synthesizer	[RFCXXXX]
speakverify	Speaker Verifier	[RFCXXXX]
recorder	Speech Recorder	[RFCXXXX]

13.1.2. MRCPv2 methods and events

   IANA SHALL create a new name space of "MRCPv2 methods and events".
   All maintenance within and additions to the contents of this name
   space MUST be according to the "Standards Action" registration
   policy. The initial contents of the registry, defined by the
   "method-name" and "event-name" BNF in Section 15 and explained in
   Section 5.2 and Section 5.5, are given below.

```
   Name                        Resource type   Method/Event    Reference
   ----                        -------------    ------------    ---------
   SET-PARAMS                  Generic          Method          [RFCXXXX]
   GET-PARAMS                  Generic          Method          [RFCXXXX]
   SPEAK                       Synthesizer      Method          [RFCXXXX]
   STOP                        Synthesizer      Method          [RFCXXXX]
   PAUSE                       Synthesizer      Method          [RFCXXXX]
   RESUME                      Synthesizer      Method          [RFCXXXX]
   BARGE-IN-OCCURRED           Synthesizer      Method          [RFCXXXX]
   CONTROL                     Synthesizer      Method          [RFCXXXX]
   DEFINE-LEXICON              Synthesizer      Method          [RFCXXXX]
   DEFINE-GRAMMAR              Recognizer       Method          [RFCXXXX]
   RECOGNIZE                   Recognizer       Method          [RFCXXXX]
   INTERPRET                   Recognizer       Method          [RFCXXXX]
   GET-RESULT                  Recognizer       Method          [RFCXXXX]
   START-INPUT-TIMERS          Recognizer       Method          [RFCXXXX]
   STOP                        Recognizer       Method          [RFCXXXX]
   START-PHRASE-ENROLLMENT     Recognizer       Method          [RFCXXXX]
   ENROLLMENT-ROLLBACK         Recognizer       Method          [RFCXXXX]
   END-PHRASE-ENROLLMENT       Recognizer       Method          [RFCXXXX]
   MODIFY-PHRASE               Recognizer       Method          [RFCXXXX]
   DELETE-PHRASE               Recognizer       Method          [RFCXXXX]
   RECORD                      Recorder         Method          [RFCXXXX]
   STOP                        Recorder         Method          [RFCXXXX]
   START-INPUT-TIMERS          Recorder         Method          [RFCXXXX]
   START-SESSION               Verifier         Method          [RFCXXXX]
   END-SESSION                 Verifier         Method          [RFCXXXX]
   QUERY-VOICEPRINT            Verifier         Method          [RFCXXXX]
   DELETE-VOICEPRINT           Verifier         Method          [RFCXXXX]
   VERIFY                      Verifier         Method          [RFCXXXX]
   VERIFY-FROM-BUFFER          Verifier         Method          [RFCXXXX]
   VERIFY-ROLLBACK             Verifier         Method          [RFCXXXX]
   STOP                        Verifier         Method          [RFCXXXX]
   START-INPUT-TIMERS          Verifier         Method          [RFCXXXX]
   GET-INTERMEDIATE-RESULT     Verifier         Method          [RFCXXXX]
   SPEECH-MARKER               Synthesizer      Event           [RFCXXXX]
   SPEAK-COMPLETE              Synthesizer      Event           [RFCXXXX]
   START-OF-INPUT              Recognizer       Event           [RFCXXXX]
   RECOGNITION-COMPLETE        Recognizer       Event           [RFCXXXX]
   INTERPRETATION-COMPLETE     Recognizer       Event           [RFCXXXX]
   START-OF-INPUT              Recorder         Event           [RFCXXXX]
   RECORD-COMPLETE             Recorder         Event           [RFCXXXX]
   VERIFICATION-COMPLETE       Verifier         Event           [RFCXXXX]
   START-OF-INPUT              Verifier         Event           [RFCXXXX]
```

13.1.3. MRCPv2 header fields

   IANA SHALL create a new name space of "MRCPv2 header fields". All
   maintenance within and additions to the contents of this name space
   MUST be according to the "Standards Action" registration policy. The
   initial contents of the registry, defined by the "message-header" BNF
   in Section 15 and explained in Section 5.1, are given below. Note
   that the values permitted for the "Vendor-Specific-Parameters"
   parameter are managed according to a different policy. See
   Section 13.1.6.

Name	Resource type	Reference
channel-identifier	Generic	[RFCXXXX]
accept	Generic	[RFC2616]
active-request-id-list	Generic	[RFCXXXX]
proxy-sync-id	Generic	[RFCXXXX]
accept-charset	Generic	[RFC2616]
content-type	Generic	[RFCXXXX]
content-id	Generic	[RFC2392, RFC2046, and RFC5322]
content-base	Generic	[RFCXXXX]
content-encoding	Generic	[RFCXXXX]
content-location	Generic	[RFCXXXX]
content-length	Generic	[RFCXXXX]
fetch-timeout	Generic	[RFCXXXX]
cache-control	Generic	[RFCXXXX]
logging-tag	Generic	[RFCXXXX]
set-cookie	Generic	[RFCXXXX]
set-cookie2	Generic	[RFCXXXX]
vendor-specific	Generic	[RFCXXXX]
jump-size	Synthesizer	[RFCXXXX]
kill-on-barge-in	Synthesizer	[RFCXXXX]
speaker-profile	Synthesizer	[RFCXXXX]
completion-cause	Synthesizer	[RFCXXXX]
completion-reason	Synthesizer	[RFCXXXX]
voice-parameter	Synthesizer	[RFCXXXX]
prosody-parameter	Synthesizer	[RFCXXXX]
speech-marker	Synthesizer	[RFCXXXX]
speech-language	Synthesizer	[RFCXXXX]
fetch-hint	Synthesizer	[RFCXXXX]
audio-fetch-hint	Synthesizer	[RFCXXXX]
failed-uri	Synthesizer	[RFCXXXX]
failed-uri-cause	Synthesizer	[RFCXXXX]
speak-restart	Synthesizer	[RFCXXXX]
speak-length	Synthesizer	[RFCXXXX]
load-lexicon	Synthesizer	[RFCXXXX]
lexicon-search-order	Synthesizer	[RFCXXXX]
confidence-threshold	Recognizer	[RFCXXXX]
sensitivity-level	Recognizer	[RFCXXXX]

```
speed-vs-accuracy                   Recognizer      [RFCXXXX]
n-best-list-length                  Recognizer      [RFCXXXX]
input-type                          Recognizer      [RFCXXXX]
no-input-timeout                    Recognizer      [RFCXXXX]
recognition-timeout                 Recognizer      [RFCXXXX]
waveform-uri                        Recognizer      [RFCXXXX]
input-waveform-uri                  Recognizer      [RFCXXXX]
completion-cause                    Recognizer      [RFCXXXX]
completion-reason                   Recognizer      [RFCXXXX]
recognizer-context-block            Recognizer      [RFCXXXX]
start-input-timers                  Recognizer      [RFCXXXX]
speech-complete-timeout             Recognizer      [RFCXXXX]
speech-incomplete-timeout           Recognizer      [RFCXXXX]
dtmf-interdigit-timeout             Recognizer      [RFCXXXX]
dtmf-term-timeout                   Recognizer      [RFCXXXX]
dtmf-term-char                      Recognizer      [RFCXXXX]
failed-uri                          Recognizer      [RFCXXXX]
failed-uri-cause                    Recognizer      [RFCXXXX]
save-waveform                       Recognizer      [RFCXXXX]
media-type                          Recognizer      [RFCXXXX]
new-audio-channel                   Recognizer      [RFCXXXX]
speech-language                     Recognizer      [RFCXXXX]
ver-buffer-utterance                Recognizer      [RFCXXXX]
recognition-mode                    Recognizer      [RFCXXXX]
cancel-if-queue                     Recognizer      [RFCXXXX]
hotword-max-duration                Recognizer      [RFCXXXX]
hotword-min-duration                Recognizer      [RFCXXXX]
interpret-text                      Recognizer      [RFCXXXX]
dtmf-buffer-time                    Recognizer      [RFCXXXX]
clear-dtmf-buffer                   Recognizer      [RFCXXXX]
early-no-match                      Recognizer      [RFCXXXX]
num-min-consistent-pronunciations   Recognizer      [RFCXXXX]
consistency-threshold               Recognizer      [RFCXXXX]
clash-threshold                     Recognizer      [RFCXXXX]
personal-grammar-uri                Recognizer      [RFCXXXX]
enroll-utterance                    Recognizer      [RFCXXXX]
phrase-id                           Recognizer      [RFCXXXX]
phrase-nl                           Recognizer      [RFCXXXX]
weight                              Recognizer      [RFCXXXX]
save-best-waveform                  Recognizer      [RFCXXXX]
new-phrase-id                       Recognizer      [RFCXXXX]
confusable-phrases-uri              Recognizer      [RFCXXXX]
abort-phrase-enrollment             Recognizer      [RFCXXXX]
sensitivity-level                   Recorder        [RFCXXXX]
no-input-timeout                    Recorder        [RFCXXXX]
completion-cause                    Recorder        [RFCXXXX]
completion-reason                   Recorder        [RFCXXXX]
failed-uri                          Recorder        [RFCXXXX]
```

failed-uri-cause	Recorder	[RFCXXXX]
record-uri	Recorder	[RFCXXXX]
media-type	Recorder	[RFCXXXX]
max-time	Recorder	[RFCXXXX]
trim-length	Recorder	[RFCXXXX]
final-silence	Recorder	[RFCXXXX]
capture-on-speech	Recorder	[RFCXXXX]
ver-buffer-utterance	Recorder	[RFCXXXX]
start-input-timers	Recorder	[RFCXXXX]
new-audio-channel	Recorder	[RFCXXXX]
repository-uri	Verifier	[RFCXXXX]
voiceprint-identifier	Verifier	[RFCXXXX]
verification-mode	Verifier	[RFCXXXX]
adapt-model	Verifier	[RFCXXXX]
abort-model	Verifier	[RFCXXXX]
min-verification-score	Verifier	[RFCXXXX]
num-min-verification-phrases	Verifier	[RFCXXXX]
num-max-verification-phrases	Verifier	[RFCXXXX]
no-input-timeout	Verifier	[RFCXXXX]
save-waveform	Verifier	[RFCXXXX]
media-type	Verifier	[RFCXXXX]
waveform-uri	Verifier	[RFCXXXX]
voiceprint-exists	Verifier	[RFCXXXX]
ver-buffer-utterance	Verifier	[RFCXXXX]
input-waveform-uri	Verifier	[RFCXXXX]
completion-cause	Verifier	[RFCXXXX]
completion-reason	Verifier	[RFCXXXX]
speech-complete-timeout	Verifier	[RFCXXXX]
new-audio-channel	Verifier	[RFCXXXX]
abort-verification	Verifier	[RFCXXXX]
start-input-timers	Verifier	[RFCXXXX]
input-type	Verifier	[RFCXXXX]

13.1.4. MRCPv2 status codes

   IANA SHALL create a new name space of "MRCPv2 status codes" with the
   initial values that are defined in Section 5.4 All maintenance within
   and additions to the contents of this name space MUST be according to
   the "Specification Required with Expert Review" registration policy.

13.1.5. Grammar Reference List Parameters

   IANA SHALL create a new name space of "Grammar Reference List
   Parameters". All maintenance within and additions to the contents of
   this name space MUST be according to the "Specification Required with
   Expert Review" registration policy. There is only one initial
   parameter, "weight", which is defined in Section 13.5.1 and
   Section 9.9.

13.1.6. MRCPv2 vendor-specific parameters

   IANA SHALL create a new name space of "MRCPv2 vendor-specific
   parameters". All maintenance within and additions to the contents of
   this name space MUST be according to the "Hierarchical Allocation"
   registration policy as follows. Each name (corresponding to the
   "vendor-av-pair-name" ABNF production) MUST satisfy the syntax
   requirements of Internet Domain Names as described in section 2.3.1
   of RFC 1035 [RFC1035] (and as updated or obsoleted by successive
   RFCs), with one exception, the order of the domain names is reversed.
   For example, a vendor-specific parameter "foo" by example.com would
   have the form "com.example.foo". The first, or top-level domain, is
   restricted to exactly the set of Top-Level Internet Domains defined
   by IANA and will be updated by IANA when and only when that set
   changes. The second-level and all subdomains within the parameter
   name MUST be allocated according to the "Expert Review" policy. The
   Designated Expert MAY advise IANA to allow delegation of subdomains
   to the requester. As a general guideline, the Designated Expert is
   encouraged to manage the allocation of corporate, organizational, or
   institutional names and delegate all subdomains accordingly. For
   example, the Designated Expert MAY allocate "com.example" and
   delegate all subdomains of that name to the organization represented
   by the Internet domain name "example.com". For simplicity, the
   Designated Expert is encouraged to perform allocations according to
   the existing allocations of Internet domain names to organizations,
   institutions, corporations, etc.

   The registry contains a list of vendor-registered parameters, where
   each defined parameter is associated with a reference to an RFC
   defining it. The registry is initially empty.

13.2. NLSML-related registrations

13.2.1. application/nlsml+xml Media Type registration

   IANA is requested to register the following Media Type according to
   the process defined in RFC 4288 [RFC4288].
   To: ietf-types@iana.org
   Subject: Registration of media type application/nlsml+xml
   MIME media type name: application
   MIME subtype name: nlsml+xml
   Required parameters: none
   Optional parameters:
      charset: All of the considerations described in RFC 3023
         [RFC3023] also apply to the application/nlsml+xml media type.

Encoding considerations: All of the considerations described in RFC
   3023 also apply to the application/nlsml+xml media type.
Security considerations: As with HTML, NLSML documents contain links
   to other data stores (grammars, verifier resources, etc.). Unlike
   HTML, however, the data stores are not treated as media to be
   rendered. Nevertheless, linked files may themselves have security
   considerations, which would be those of the individual registered
   types. Additionally, this media type has all of the security
   considerations described in RFC 3023.
Interoperability considerations: Although an NLSML document is
   itself a complete XML document, for a fuller interpretation of the
   content a receiver of an NLSML document may wish to access
   resources linked to by the document. The inability of an NLSML
   processor to access or process such linked resources could result
   in different behavior by the ultimate consumer of the data.
Published specification: RFCXXXX
Applications which use this media type: MRCPv2 clients and servers
Additional information: none
Magic number(s): There is no single initial octet sequence that is
   always present for NLSML files.
Person & email address to contact for further information: Sarvi
   Shanmugham, sarvi@cisco.com
Intended usage: This media type is expected to be used only in
   conjunction with MRCPv2.

13.3. NLSML XML Schema registration

   IANA is requested to register and maintain the following XML Schema.
   Information provided follows the template in RFC 3688 [RFC3688].
   XML element type: schema
   URI: http://www.ietf.org/xml/schema/mrcpv2
   Registrant Contact: IESG
   XML: See Section 16.1.

13.4. MRCPv2 XML Namespace registration

   IANA is requested to register and maintain the following XML Name
   space. Information provided follows the template in RFC 3688
   [RFC3688].
   XML element type: ns
   URI: http://www.ietf.org/xml/ns/mrcpv2
   Registrant Contact: IESG
   XML: RFCXXXX

13.5. text Media Type Registrations

   IANA is requested to register the following text Media Types
   according to the process defined in RFC 4288 [RFC4288].

13.5.1. text/grammar-ref-list

   To: ietf-types@iana.org
   Subject: Registration of media type text/grammar-ref-list
   MIME media type name: application
   MIME subtype name: text/grammar-ref-list
   Required parameters: none
   Optional parameters: none
   Encoding considerations: Depending on the transfer protocol, a
      transfer encoding may be necessary to deal with very long lines.
   Security considerations: This media type contains URIs which may
      represent references to external resources. As these resources
      are assumed to be speech recognition grammars, similar
      considerations as for the media types "application/srgs" and
      "application/srgs+xml" apply.
   Interoperability considerations; '>' must be percent encoded in URIs
      according to RFC 3986 [RFC3986].
   Published specification: The RECOGNIZE method of the MRCP protocol
      performs a recognition operation that matches input against a set
      of grammars. When matching against more than one grammar, it is
      sometimes necessary to use different weights for the individual
      grammars. These weights are not a property of the grammar
      resource itself but qualify the reference to that grammar for the
      particular recognition operation initiated by the RECOGNIZE
      method. The format of the proposed text/grammar-ref-list media
      type is as follows: body = *reference where reference = "<" uri
      ">" [parameters] CRLF parameters = ";" parameter *(";" parameter)
      and parameter = attribute "=" value. This specification currently
      only defines a 'weight' parameter, but new parameters may be added
      through the "Grammar Reference List Parameters" IANA registry
      established through this specification. Example:
      <http://example.com/grammars/field1.gram>
      <http://example.com/grammars/field2.gram>;weight="0.85"
      <session:field3@form-level.store>;weight="0.9"
      <http://example.com/grammars/universals.gram>;weight="0.75"
   Applications which use this media type: MRCPv2 clients and servers
   Additional information: none
   Magic number(s): none
   Person & email address to contact for further information: Sarvi
      Shanmugham, sarvi@cisco.com
   Intended usage: This media type is expected to be used only in
      conjunction with MRCPv2.

13.6. session URI scheme registration

   IANA is requested to register the following new URI scheme. The
   information below follows the template given in RFC 4395 [RFC4395].

        URI scheme name: "session"
        Status: "Permanent"
        URI scheme syntax: The syntax of this scheme is identical to that
           defined for the "cid" scheme in section 2 of RFC 2392 [RFC2392].
        URI scheme semantics: The URI is intended to identify a data
           resource previously given to the network computing resource. The
           purpose of this scheme is to permit access to the specific
           resource for the lifetime of the session with the entity storing
           the resource. The media type of the resource CAN vary. There is
           no explicit mechanism for communication of the media type. This
           scheme is currently widely used internally by existing
           implementations, and the registration is intended to provide
           information in the rare (and unfortunate) case that the scheme is
           used elsewhere. The scheme SHOULD NOT be used for open internet
           protocols.
        Encoding considerations: There are no other encoding considerations
           for the 'session' URIs not described in RFC 3986 [RFC3986]
        Applications/protocols that use this URI scheme name: This scheme
           name is used by MRCPv2 clients and servers.
        Interoperability considerations: Note that none of the resources are
           accessible after the MCRPv2 session ends, hence the name of the
           scheme. For clients who establish one MRCPv2 session only for the
           entire speech application being implemented this is sufficient,
           but clients who create, terminate, and recreate MRCP sessions for
           performance or scalability reasons will lose access to resources
           established in the earlier session(s).
        Security considerations: Generic security considerations for URIs
           described in RFC 3986 [RFC3986] apply to this scheme as well. The
           URIs defined here provide an identification mechanism only. Given
           that the communication channel between client and server is
           secure, that the server correctly accesses the resource associated
           with the URI, and that the server ensures session-only lifetime
           and access for each URI, the only additional security issues are
           those of the types of media referred to by the URI.
        Contact: Sarvi Shanmugham, sarvi@cisco.com
        Author/Change controller: IESG, iesg@ietf.org
        References: This specification, particularly sections Section 6.2.7,
           Section 8.5.2, Section 9.5.1, and Section 9.9.

13.7. SDP parameter registrations

        IANA is requested to register the following SDP parameter values.
        The information for each follows the template given in RFC 4566
        [RFC4566], Appendix B.

13.7.1. sub-registry "proto"

   "TCP/MRCPv2" value of the "proto" parameter
   Contact name, email address and telephone number: Sarvi Shanmugham,
      sarvi@cisco.com, +1.408.902.3875
   Name being registered (as it will appear in SDP): TCP/MRCPv2
   Long-form name in English: MCRPv2 over TCP
   Type of name: proto
   Explanation of name: This name represents the MCRPv2 protocol
      carried over TCP.
   Reference to specification of name: RFCXXXX

   "TCP/TLS/MRCPv2" value of the "proto" parameter
   Contact name, email address and telephone number: Sarvi Shanmugham,
      sarvi@cisco.com, +1.408.902.3875
   Name being registered (as it will appear in SDP): TCP/TLS/MRCPv2
   Long-form name in English: MCRPv2 over TLS over TCP
   Type of name: proto
   Explanation of name: This name represents the MCRPv2 protocol
      carried over TLS over TCP.
   Reference to specification of name: RFCXXXX

13.7.2. sub-registry "att-field (media-level)"

   "resource" value of the "att-field" parameter
   Contact name, email address and telephone number: Sarvi Shanmugham,
      sarvi@cisco.com, +1.408.902.3875
   Attribute name (as it will appear in SDP): resource
   Long-form attribute name in English: MRCPv2 resource type
   Type of attribute: media-level
   Subject to charset attribute? no
   Explanation of attribute: See Section 4.2 of RFCXXXX for description
      and examples.
   Specification of appropriate attribute values: See section
      Section 13.1.1 of RFCXXXX.

   "channel" value of the "att-field" parameter
   Contact name, email address and telephone number: Sarvi Shanmugham,
      sarvi@cisco.com, +1.408.902.3875
   Attribute name (as it will appear in SDP): channel
   Long-form attribute name in English: MRCPv2 resource channel
      identifier
   Type of attribute: media-level
   Subject to charset attribute? no

      Explanation of attribute: See Section 4.2 of RFCXXXX for description
         and examples.
      Specification of appropriate attribute values See Section 4.2 and
         the "channel-id" ABNF production rules of RFCXXXX.

      "cmid" value of the "att-field" parameter
      Contact name, email address and telephone number: Sarvi Shanmugham,
         sarvi@cisco.com, +1.408.902.3875
      Attribute name (as it will appear in SDP): cmid
      Long-form attribute name in English: MRCPv2 resource channel media
         identifier
      Type of attribute: media-level
      Subject to charset attribute? no
      Explanation of attribute: See Section 4.4 of RFCXXXX for description
         and examples.
      Specification of appropriate attribute values See Section 4.4 and
         the "cmid-attribute" ABNF production rules of RFCXXXX.

14. Examples

14.1. Message Flow

   The following is an example of a typical MRCPv2 session of speech
   synthesis and recognition between a client and a server. Although
   the SDP "s" attribute in these examples has a text description value
   to assist in understanding the examples, please keep in mind that RFC
   3264 [RFC3264] recommends that messages actually put on the wire use
   a space or a dash.

   The figure below illustrates opening a session to the MRCPv2 server.
   This exchange does not allocate a resource or setup media. It simply
   establishes a SIP session with the MRCPv2 server.

   C->S:
           INVITE sip:mresources@example.com SIP/2.0
           Via:SIP/2.0/TCP client.atlanta.example.com:5060;
            branch=z9hG4bK74bg1
           Max-Forwards:6
           To:MediaServer <sip:mresources@example.com>
           From:sarvi <sip:sarvi@example.com>;tag=1928301774
           Call-ID:a84b4c76e66710
           CSeq:323123 INVITE
           Contact:<sip:sarvi@client.example.com>
           Content-Type:application/sdp
           Content-Length:...

           v=0

```
        o=sarvi 2614933546 2614933546 IN IP4 192.0.2.12
        s=Set up MRCPv2 control and audio
        i=Initial contact
        c=IN IP4 192.0.2.12
```


    S->C:
```
        SIP/2.0 200 OK
        Via:SIP/2.0/TCP client.atlanta.example.com:5060;
         branch=z9hG4bK74bg1;received=192.0.32.10
        To:MediaServer <sip:mresources@example.com>;tag=62784
        From:sarvi <sip:sarvi@example.com>;tag=1928301774
        Call-ID:a84b4c76e66710
        CSeq:323123 INVITE
        Contact:<sip:mresources@server.example.com>
        Content-Type:application/sdp
        Content-Length:...

        v=0
        o=- 3000000001 3000000001 IN IP4 192.0.2.11
        s=Set up MRCPv2 control and audio
        i=Initial contact
        c=IN IP4 192.0.2.11
```


    C->S:
```
        ACK sip:mresources@server.example.com SIP/2.0
        Via:SIP/2.0/TCP client.atlanta.example.com:5060;
         branch=z9hG4bK74bg2
        Max-Forwards:6
        To:MediaServer <sip:mresources@example.com>;tag=62784
        From:Sarvi <sip:sarvi@example.com>;tag=1928301774
        Call-ID:a84b4c76e66710
        CSeq:323123 ACK
        Content-Length:0
```

The client requests the server to create a synthesizer resource
control channel to do speech synthesis. This also adds a media
stream to send the generated speech. Note that in this example, the
client requests a new MRCPv2 TCP stream between the client and the
server. In the following requests, the client will ask to use the
existing connection.

    C->S:
```
        INVITE sip:mresources@server.example.com SIP/2.0
        Via:SIP/2.0/TCP client.atlanta.example.com:5060;
         branch=z9hG4bK74bg3
        Max-Forwards:6
```

```
          To:MediaServer <sip:mresources@example.com>;tag=62784
          From:sarvi <sip:sarvi@example.com>;tag=1928301774
          Call-ID:a84b4c76e66710
          CSeq:323124 INVITE
          Contact:<sip:sarvi@client.example.com>
          Content-Type:application/sdp
          Content-Length:...

          v=0
          o=sarvi 2614933546 2614933547 IN IP4 192.0.2.12
          s=Set up MRCPv2 control and audio
          i=Add TCP channel, synthesizer and one-way audio
          c=IN IP4 192.0.2.12
          t=0 0
          m=application 9  TCP/MRCPv2 1
          a=setup:active
          a=connection:new
          a=resource:speechsynth
          a=cmid:1
          m=audio 49170 RTP/AVP 0 96
          a=rtpmap:0 pcmu/8000
          a=rtpmap:96 telephone-event/8000
          a=fmtp:96 0-15
          a=recvonly
          a=mid:1


    S->C:
          SIP/2.0 200 OK
          Via:SIP/2.0/TCP client.atlanta.example.com:5060;
           branch=z9hG4bK74bg3;received=192.0.32.10
          To:MediaServer <sip:mresources@example.com>;tag=62784
          From:sarvi <sip:sarvi@example.com>;tag=1928301774
          Call-ID:a84b4c76e66710
          CSeq:323124 INVITE
          Contact:<sip:mresources@server.example.com>
          Content-Type:application/sdp
          Content-Length:...

          v=0
          o=- 3000000001 3000000002 IN IP4 192.0.2.11
          s=Set up MRCPv2 control and audio
          i=Add TCP channel, synthesizer and one-way audio
          c=IN IP4 192.0.2.11
          t=0 0
          m=application 32416  TCP/MRCPv2 1
          a=setup:passive
          a=connection:new
```

```
            a=channel:32AECB23433801@speechsynth
            a=cmid:1
            m=audio 48260 RTP/AVP 0
            a=rtpmap:0 pcmu/8000
            a=sendonly
            a=mid:1
```

   C->S:

```
            ACK sip:mresources@server.example.com SIP/2.0
            Via:SIP/2.0/TCP client.atlanta.example.com:5060;
             branch=z9hG4bK74bg4
            Max-Forwards:6
            To:MediaServer <sip:mresources@example.com>;tag=62784
            From:Sarvi <sip:sarvi@example.com>;tag=1928301774
            Call-ID:a84b4c76e66710
            CSeq:323124 ACK
            Content-Length:0
```

   This exchange allocates an additional resource control channel for a
   recognizer. Since a recognizer would need to receive an audio stream
   for recognition, this interaction also updates the audio stream to
   sendrecv, making it a 2-way audio stream.

   C->S:

```
            INVITE sip:mresources@server.example.com SIP/2.0
            Via:SIP/2.0/TCP client.atlanta.example.com:5060;
             branch=z9hG4bK74bg5
            Max-Forwards:6
            To:MediaServer <sip:mresources@example.com>;tag=62784
            From:sarvi <sip:sarvi@example.com>;tag=1928301774
            Call-ID:a84b4c76e66710
            CSeq:323125 INVITE
            Contact:<sip:sarvi@client.example.com>
            Content-Type:application/sdp
            Content-Length:...

            v=0
            o=sarvi 2614933546 2614933548 IN IP4 192.0.2.12
            s=Set up MRCPv2 control and audio
            i=Add recognizer and duplex the audio
            c=IN IP4 192.0.2.12
            t=0 0
            m=application 9  TCP/MRCPv2 1
            a=setup:active
            a=connection:existing
            a=resource:speechsynth
            a=cmid:1
```

```
        m=audio 49170 RTP/AVP 0 96
        a=rtpmap:0 pcmu/8000
        a=rtpmap:96 telephone-event/8000
        a=fmtp:96 0-15
        a=recvonly
        a=mid:1
        m=application 9  TCP/MRCPv2 1
        a=setup:active
        a=connection:existing
        a=resource:speechrecog
        a=cmid:2
        m=audio 49180 RTP/AVP 0 96
        a=rtpmap:0 pcmu/8000
        a=rtpmap:96 telephone-event/8000
        a=fmtp:96 0-15
        a=sendonly
        a=mid:2


   S->C:
        SIP/2.0 200 OK
        Via:SIP/2.0/TCP client.atlanta.example.com:5060;
         branch=z9hG4bK74bg5;received=192.0.32.10
        To:MediaServer <sip:mresources@example.com>;tag=62784
        From:sarvi <sip:sarvi@example.com>;tag=1928301774
        Call-ID:a84b4c76e66710
        CSeq:323125 INVITE
        Contact:<sip:mresources@server.example.com>
        Content-Type:application/sdp
        Content-Length:...

        v=0
        o=- 3000000001 3000000003 IN IP4 192.0.2.11
        s=Set up MRCPv2 control and audio
        i=Add recognizer and duplex the audio
        c=IN IP4 192.0.2.11
        t=0 0
        m=application 32416  TCP/MRCPv2 1
        a=channel:32AECB23433801@speechsynth
        a=cmid:1
        m=audio 48260 RTP/AVP 0
        a=rtpmap:0 pcmu/8000
        a=sendonly
        a=mid:1
        m=application 32416  TCP/MRCPv2 1
        a=channel:32AECB23433801@speechrecog
        a=cmid:2
        m=audio 48260 RTP/AVP 0
```

```
        a=rtpmap:0 pcmu/8000
        a=rtpmap:96 telephone-event/8000
        a=fmtp:96 0-15
        a=recvonly
        a=mid:2


C->S:

        ACK sip:mresources@server.example.com SIP/2.0
        Via:SIP/2.0/TCP client.atlanta.example.com:5060;
         branch=z9hG4bK74bg6
        Max-Forwards:6
        To:MediaServer <sip:mresources@example.com>;tag=62784
        From:Sarvi <sip:sarvi@example.com>;tag=1928301774
        Call-ID:a84b4c76e66710
        CSeq:323125 ACK
        Content-Length:0
```

A MRCPv2 "SPEAK" request initiates speech.

```
C->S:
        MRCP/2.0 ... SPEAK 543257
        Channel-Identifier:32AECB23433801@speechsynth
        Kill-On-Barge-In:false
        Voice-gender:neutral
        Voice-age:25
        Prosody-volume:medium
        Content-Type:application/ssml+xml
        Content-Length:...

        <?xml version="1.0"?>
        <speak version="1.0"
               xmlns="http://www.w3.org/2001/10/synthesis"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
               http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
               xml:lang="en-US">
          <p>
            <s>You have 4 new messages.</s>
            <s>The first is from Stephanie Williams
              <mark name="Stephanie"/>
              and arrived at <break/>
              <say-as interpret-as="vxml:time">0345p</say-as>.</s>
            <s>The subject is <prosody
              rate="-20%">ski trip</prosody></s>
          </p>
        </speak>


S->C:
        MRCP/2.0 ... 543257 200 IN-PROGRESS
        Channel-Identifier:32AECB23433801@speechsynth
        Speech-Marker:timestamp=857205015059
```

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

```
S->C:   MRCP/2.0 ... SPEECH-MARKER 543257 IN-PROGRESS
        Channel-Identifier:32AECB23433801@speechsynth
        Speech-Marker:timestamp=857206027059;Stephanie
```

The synthesizer finishes with the "SPEAK" request.

```
S->C:   MRCP/2.0 ... SPEAK-COMPLETE 543257 COMPLETE
        Channel-Identifier:32AECB23433801@speechsynth
        Speech-Marker:timestamp=857207685213;Stephanie
```

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

```
C->S:   MRCP/2.0 ... RECOGNIZE 543258
        Channel-Identifier:32AECB23433801@speechrecog
        Content-Type:application/srgs+xml
        Content-Length:...

        <?xml version="1.0"?>
        <!-- the default grammar language is US English -->
        <grammar xmlns="http://www.w3.org/2001/06/grammar"
                 xml:lang="en-US" version="1.0" root="request">
        <!-- 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:   MRCP/2.0 ... 543258 200 IN-PROGRESS
        Channel-Identifier:32AECB23433801@speechrecog
```

The client issues the next MRCPv2 "SPEAK" method.

```
C->S:   MRCP/2.0 ... SPEAK 543259
        Channel-Identifier:32AECB23433801@speechsynth
        Kill-On-Barge-In:true
        Content-Type:application/ssml+xml
        Content-Length:...

        <?xml version="1.0"?>
        <speak version="1.0"
               xmlns="http://www.w3.org/2001/10/synthesis"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
               http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
               xml:lang="en-US">
          <p>
            <s>Welcome to ABC corporation.</s>
            <s>Who would you like Talk to.</s>
          </p>
        </speak>


S->C:   MRCP/2.0 ... 543259 200 IN-PROGRESS
        Channel-Identifier:32AECB23433801@speechsynth
        Speech-Marker:timestamp=857207696314
```

This next section of this ongoing example demonstrates how kill-on-
barge-in support works. Since this last "SPEAK" request had Kill-On-
Barge-In set to "true", when the recognizer (the server) generated
the START-OF-INPUT event while a SPEAK was active the client
immediately issued a BARGE-IN-OCCURRED method to the synthesizer
resource. The speech synthesizer then terminated playback and
notified the client. The completion-cause code provided the
indication that this was a kill-on-barge-in interruption rather than
a normal completion.

Note that since the recognition and synthesizer resources are in the
same session on the same server, to obtain a faster response the
server might have internally relayed the start-of-input condition to
the synthesizer directly, before receiving the expected BARGE-IN-
OCCURRED event. However, any such communication is outside the scope
of the MRCPv2 protocol.

```
S->C:   MRCP/2.0 ... START-OF-INPUT 543258 IN-PROGRESS
        Channel-Identifier:32AECB23433801@speechrecog
        Proxy-Sync-Id:987654321


C->S:   MRCP/2.0 ... BARGE-IN-OCCURRED 543259
        Channel-Identifier:32AECB23433801@speechsynth
        Proxy-Sync-Id:987654321


S->C:   MRCP/2.0 ... 543259 200 COMPLETE
        Channel-Identifier:32AECB23433801@speechsynth
        Active-Request-Id-List:543258
        Speech-Marker:timestamp=857206096314

S->C:   MRCP/2.0 ... SPEAK-COMPLETE 543259 COMPLETE
        Channel-Identifier:32AECB23433801@speechsynth
        Completion-Cause:001 barge-in
        Speech-Marker:timestamp=857207685213
```

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:   MRCP/2.0 ... RECOGNITION-COMPLETE 543258 COMPLETE
        Channel-Identifier:32AECB23433801@speechrecog
        Completion-Cause:000 success
        Waveform-URI:<http://web.media.com/session123/audio.wav>;
                     size=423523;duration=25432
        Content-Type:application/nlsml+xml
        Content-Length:...

        <?xml version="1.0"?>
        <result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
                xmlns:ex="http://www.example.com/example"
                grammar="session:request1@form-level.store">
           <interpretation>
              <instance name="Person">
                  <ex:Person>
                      <ex:Name> Andre Roy </ex:Name>
                  </ex:Person>
              </instance>
              <input>   may I speak to Andre Roy </input>
           </interpretation>
        </result>
```

Since the client was now finished with the session, including all

resources, it issued a SIP BYE request to close the SIP session.
This caused all control channels and resources allocated under the
session to be de-allocated.

```
C->S:   BYE sip:mresources@server.example.com SIP/2.0
        Via:SIP/2.0/TCP client.atlanta.example.com:5060;
         branch=z9hG4bK74bg7
        Max-Forwards:6
        From:Sarvi <sip:sarvi@example.com>;tag=1928301774
        To:MediaServer <sip:mresources@example.com>;tag=62784
        Call-ID:a84b4c76e66710
        CSeq:323126 BYE
        Content-Length:0
```

14.2. Recognition Result Examples

14.2.1. Simple ASR Ambiguity

```
System: To which city will you be traveling?
User:   I want to go to Pittsburgh.

<?xml version="1.0"?>
<result xmlns="http://www.ietf.org/xml/ns/mrcpv2"
        xmlns:ex="http://www.example.com/example"
        grammar="http://www.example.com/flight">
  <interpretation confidence="0.6">
     <instance>
        <ex:airline>
           <ex:to_city>Pittsburgh</ex:to_city>
        <ex:airline>
     <instance>
     <input mode="speech">
        I want to go to Pittsburgh
     </input>
  </interpretation>
  <interpretation confidence="0.4"
     <instance>
        <ex:airline>
           <ex:to_city>Stockholm</ex:to_city>
        </ex:airline>
     </instance>
     <input>I want to go to Stockholm</input>
  </interpretation>
</result>
```

14.2.2. Mixed Initiative

   System: What would you like?
   User: I would like 2 pizzas, one with pepperoni and cheese,
           one with sausage and a bottle of coke, to go.

   This example includes an order object which in turn contains objects
   named "food_item", "drink_item" and "delivery_method". The
   representation assumes there are no ambiguities in the speech or
   natural language processing. Note that this representation also
   assumes some level of intra-sentential anaphora resolution, i.e., to
   resolve the two "one's" as "pizza".

   ```
   <?xml version="1.0"?>
   <nl:result xmlns:nl="http://www.ietf.org/xml/ns/mrcpv2"
              xmlns="http://www.example.com/example"
              grammar="http://www.example.com/foodorder">
     <nl:interpretation confidence="1.0" >
        <nl:instance>
         <order>
           <food_item confidence="1.0">
             <pizza>
               <ingredients confidence="1.0">
                 pepperoni
               </ingredients>
               <ingredients confidence="1.0">
                 cheese
               </ingredients>
             </pizza>
             <pizza>
               <ingredients>sausage</ingredients>
             </pizza>
           </food_item>
           <drink_item confidence="1.0">
             <size>2-liter</size>
           </drink_item>
           <delivery_method>to go</delivery_method>
         </order>
       </nl:instance>
       <nl:input mode="speech">I would like 2 pizzas,
            one with pepperoni and cheese, one with sausage
            and a bottle of coke, to go.
       </nl:input>
     </nl:interpretation>
   </nl:result>
   ```

14.2.3. DTMF Input

   A combination of DTMF input and speech is represented using nested
   input elements. For example:
   User: My pin is (dtmf 1 2 3 4)

   <input>
     <input mode="speech" confidence ="1.0"
        timestamp-start="2000-04-03T0:00:00"
        timestamp-end="2000-04-03T0:00:01.5">My pin is
     </input>
     <input mode="dtmf" confidence ="1.0"
        timestamp-start="2000-04-03T0:00:01.5"
        timestamp-end="2000-04-03T0:00:02.0">1 2 3 4
     </input>
   </input>

   Note that grammars that recognize mixtures of speech and DTMF are not
   currently possible in SRGS; however, this representation might be
   needed for other applications of NLSML, and this mixture capability
   might be introduced in future versions of SRGS.

14.2.4. Interpreting Meta-Dialog and Meta-Task Utterances

   Natural language communication makes use of meta-dialog and meta-task
   utterances. This specification is flexible enough so that meta
   utterances can be represented on an application-specific basis
   without requiring other standard markup.

   Here are two examples of how meta-task and meta-dialog utterances
   might be represented.

System: What toppings do you want on your pizza?
User: What toppings do you have?

```
<interpretation grammar="http://www.example.com/toppings">
   <instance>
      <question>
         <questioned_item>toppings<questioned_item>
         <questioned_property>
          availability
         </questioned_property>
      </question>
   </instance>
   <input mode="speech">
     what toppings do you have?
   </input>
</interpretation>
```


User: slow down.

```
<interpretation grammar="http://www.example.com/generalCommandsGrammar">
   <instance>
    <command>
       <action>reduce speech rate</action>
       <doer>system</doer>
    </command>
   </instance>
  <input mode="speech">slow down</input>
</interpretation>
```

14.2.5. Anaphora and Deixis

   This specification can be used on an application-specific basis to
   represent utterances that contain unresolved anaphoric and deictic
   references. Anaphoric references, which include pronouns and
   definite noun phrases that refer to something that was mentioned in
   the preceding linguistic context, and deictic references, which refer
   to something that is present in the non-linguistic context, present
   similar problems in that there may not be sufficient unambiguous
   linguistic context to determine what their exact role in the
   interpretation should be. In order to represent unresolved anaphora
   and deixis using this specification, one strategy would be for the
   developer to define a more surface-oriented representation that
   leaves the specific details of the interpretation of the reference
   open. (This assumes that a later component is responsible for
   actually resolving the reference).

     Example: (ignoring the issue of representing the input from the
              pointing gesture.)

     System: What do you want to drink?
     User: I want this (clicks on picture of large root beer.)

     <?xml version="1.0"?>
     <nl:result xmlns:nl="http://www.ietf.org/xml/ns/mrcpv2"
             xmlns="http://www.example.com/example"
             grammar="http://www.example.com/beverages.grxml">
        <nl:interpretation>
           <nl:instance>
            <doer>I</doer>
            <action>want</action>
            <object>this</object>
           </nl:instance>
           <nl:input mode="speech">I want this</nl:input>
        </nl:interpretation>
     </nl:result>

14.2.6. Distinguishing Individual Items from Sets with One Member

   For programming convenience, it is useful to be able to distinguish
   between individual items and sets containing one item in the XML
   representation of semantic results. For example, a pizza order might
   consist of exactly one pizza, but a pizza might contain zero or more
   toppings. Since there is no standard way of marking this distinction
   directly in XML, in the current framework, the developer is free to
   adopt any conventions that would convey this information in the XML
   markup. One strategy would be for the developer to wrap the set of
   items in a grouping element, as in the following example.

     <order>
        <pizza>
           <topping-group>
              <topping>mushrooms</topping>
           </topping-group>
        </pizza>
        <drink>coke</drink>
     </order>

   In this example, the programmer can assume that there is supposed to
   be exactly one pizza and one drink in the order, but the fact that
   there is only one topping is an accident of this particular pizza
   order.

   Note that the client controls both the grammar and the semantics to
   be returned upon grammar matches, so the user of the MRCPv2 protocol

   is fully empowered to cause results to be returned in NLSML in such a
   way that the interpretation is clear to that user.

14.2.7. Extensibility

   Extensibility in NLSML is provided via result content flexibility, as
   discussed in the discussions of meta utterances and anaphora. NLSML
   can easily be used in sophisticated systems to convey application-
   specific information that more basic systems would not make use of,
   for example defining speech acts.

15. ABNF Normative Definition

   The following productions make use of the core rules defined in
   Section 6.1 of RFC 5234 [RFC5234].

```
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
UTFCHAR           =     %x21-7E
                  /     UTF8-NONASCII
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

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

alphanum         =    ALPHA / DIGIT

BOOLEAN          =    "true" / "false"

FLOAT            =    *DIGIT ["." *DIGIT]

escaped          =    "%" HEXDIG HEXDIG

fragment         =    *uric

uri              =    [ absoluteURI / relativeURI ]
                      [ "#" fragment ]

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*19DIGIT


; generic-message is the top-level rule
```

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

message-body     =      *OCTET

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

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

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

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

method-name      =      generic-method
                 /      synthesizer-method
                 /      recognizer-method
                 /      recorder-method
                 /      verifier-method

generic-method   =      "SET-PARAMS"
                 /      "GET-PARAMS"

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

event-name       =      synthesizer-event
                 /      recognizer-event
                 /      recorder-event
                 /      verifier-event

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

generic-field    =      field-name ":" [ field-value ]
field-name       =      token
field-value      =      *LWS field-content *( CRLF 1*LWS field-content)
field-content    =      <the OCTETs making up the field-value
                        and consisting of either *TEXT or combinations
                        of token, separators, and quoted-string>

resource-header  =      synthesizer-header
                 /      recognizer-header
                 /      recorder-header
                 /      verifier-header
```

```
generic-header    =     channel-identifier
                  /     accept
                  /     active-request-id-list
                  /     proxy-sync-id
                  /     accept-charset
                  /     content-type
                  /     content-id
                  /     content-base
                  /     content-encoding
                  /     content-location
                  /     content-length
                  /     fetch-timeout
                  /     cache-control
                  /     logging-tag
                  /     set-cookie
                  /     set-cookie2
                  /     vendor-specific

; -- content-id is as defined in RFC2392, RFC2046 and RFC5322
; -- accept and accept-charset are as defined in RFC2616

mrcp-version      =     "MRCP" "/" 1*2DIGIT "." 1*2DIGIT

message-length    =     1*19DIGIT

request-id        =     1*10DIGIT

status-code       =     3DIGIT

channel-identifier =    "Channel-Identifier" ":"
                        channel-id CRLF

channel-id        =     1*alphanum "@" 1*alphanum

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

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

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

content-length    =     "Content-Length" ":" 1*19DIGIT CRLF

content-type      =     "Content-Type" ":" media-type-value CRLF

media-type-value  =     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

fetch-timeout    =      "Fetch-Timeout" ":" 1*19DIGIT CRLF

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

delta-seconds        =    1*19DIGIT

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

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

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

vendor-av-pair-name    = 1*UTFCHAR

set-cookie       =      "Set-Cookie:" cookies CRLF

cookies          =      cookie *("," *LWS cookie)

cookie           =      attribute "=" value *(";" cookie-av)
```

```
cookie-av          =      "Comment" "=" value
                   /      "Domain" "=" value
                   /      "Max-Age" "=" value
                   /      "Path" "=" value
                   /      "Secure"
                   /      "Version" "=" 1*19DIGIT
                   /      "Age" "=" delta-seconds

set-cookie2        =      "Set-Cookie2:" cookies2 CRLF

cookies2           =      cookie2 *("," *LWS cookie2)

cookie2            =      attribute "=" value *(";" cookie-av2)

cookie-av2         =      "Comment" "=" value
                   /      "CommentURL" "=" DQUOTE uri DQUOTE
                   /      "Discard"
                   /      "Domain" "=" value
                   /      "Max-Age" "=" value
                   /      "Path" "=" value
                   /      "Port" [ "=" DQUOTE portlist DQUOTE ]
                   /      "Secure"
                   /      "Version" "=" 1*19DIGIT
                   /      "Age" "=" delta-seconds

portlist           =      portnum *("," *LWS portnum)

portnum            =      1*19DIGIT

; Synthesizer ABNF

synthesizer-method     =      "SPEAK"
                       /      "STOP"
                       /      "PAUSE"
                       /      "RESUME"
                       /      "BARGE-IN-OCCURRED"
                       /      "CONTROL"
                       /      "DEFINE-LEXICON"

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

synthesizer-header     =      jump-size
                       /      kill-on-barge-in
                       /      speaker-profile
                       /      completion-cause
                       /      completion-reason
                       /      voice-parameter
```

```
                             /      prosody-parameter
                             /      speech-marker
                             /      speech-language
                             /      fetch-hint
                             /      audio-fetch-hint
                             /      failed-uri
                             /      failed-uri-cause
                             /      speak-restart
                             /      speak-length
                             /      load-lexicon
                             /      lexicon-search-order


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

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

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

numeric-speech-length =      ("+" / "-") positive-speech-length

positive-speech-length =    1*19DIGIT SP numeric-speech-unit

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

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

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

completion-cause           =    "Completion-Cause" ":" cause-code SP
                                 cause-name CRLF
cause-code                 =    3DIGIT
cause-name                 =    *VCHAR

completion-reason        =      "Completion-Reason" ":"
                                quoted-string CRLF

voice-parameter          =      voice-gender
                         /      voice-age
                         /      voice-variant
                         /      voice-name

voice-gender             =      "Voice-Gender:" voice-gender-value CRLF
```

```
voice-gender-value    =    "male"
                      /    "female"
                      /    "neutral"

voice-age             =    "Voice-Age:" 1*3DIGIT CRLF

voice-variant         =    "Voice-Variant:" 1*19DIGIT CRLF

voice-name            =    "Voice-Name:"
                           1*UTFCHAR *(1*WSP 1*UTFCHAR) CRLF

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

prosody-param-name    =    1*VCHAR

prosody-param-value   =    1*VCHAR

timestamp             =    "timestamp" "=" time-stamp-value

time-stamp-value      =    1*20DIGIT

speech-marker         =    "Speech-Marker" ":"
                           timestamp
                           [";" 1*(UTFCHAR / %x20)] CRLF

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

fetch-hint            =    "Fetch-Hint" ":" ("prefetch" / "safe") CRLF

audio-fetch-hint      =    "Audio-Fetch-Hint" ":"
                          ("prefetch" / "safe" / "stream") CRLF

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

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

speak-restart         =    "Speak-Restart" ":" BOOLEAN CRLF

speak-length          =    "Speak-Length" ":" positive-length-value
                           CRLF

positive-length-value  =  positive-speech-length
                       /  text-speech-length

load-lexicon          =    "Load-Lexicon" ":" BOOLEAN CRLF

lexicon-search-order  =    "Lexicon-Search-Order" ":"
```

```
          "<" absoluteURI ">" *(" " "<" absoluteURI ">") CRLF

; Recognizer ABNF

recognizer-method      =      recog-only-method
                       /      enrollment-method

recog-only-method      =      "DEFINE-GRAMMAR"
                       /      "RECOGNIZE"
                       /      "INTERPRET"
                       /      "GET-RESULT"
                       /      "START-INPUT-TIMERS"
                       /      "STOP"

enrollment-method      =      "START-PHRASE-ENROLLMENT"
                       /      "ENROLLMENT-ROLLBACK"
                       /      "END-PHRASE-ENROLLMENT"
                       /      "MODIFY-PHRASE"
                       /      "DELETE-PHRASE"

recognizer-event       =      "START-OF-INPUT"
                       /      "RECOGNITION-COMPLETE"
                       /      "INTERPRETATION-COMPLETE"

recognizer-header      =      recog-only-header
                       /      enrollment-header


recog-only-header      =      confidence-threshold
                       /      sensitivity-level
                       /      speed-vs-accuracy
                       /      n-best-list-length
                       /      input-type
                       /      no-input-timeout
                       /      recognition-timeout
                       /      waveform-uri
                       /      input-waveform-uri
                       /      completion-cause
                       /      completion-reason
                       /      recognizer-context-block
                       /      start-input-timers
                       /      speech-complete-timeout
                       /      speech-incomplete-timeout
                       /      dtmf-interdigit-timeout
                       /      dtmf-term-timeout
                       /      dtmf-term-char
                       /      failed-uri
                       /      failed-uri-cause
```

```
                        /       save-waveform
                        /       media-type
                        /       new-audio-channel
                        /       speech-language
                        /       ver-buffer-utterance
                        /       recognition-mode
                        /       cancel-if-queue
                        /       hotword-max-duration
                        /       hotword-min-duration
                        /       interpret-text
                        /       dtmf-buffer-time
                        /       clear-dtmf-buffer
                        /       early-no-match


enrollment-header     =       num-min-consistent-pronunciations
                        /       consistency-threshold
                        /       clash-threshold
                        /       personal-grammar-uri
                        /       enroll-utterance
                        /       phrase-id
                        /       phrase-nl
                        /       weight
                        /       save-best-waveform
                        /       new-phrase-id
                        /       confusable-phrases-uri
                        /       abort-phrase-enrollment

confidence-threshold  =       "Confidence-Threshold" ":"
                              FLOAT CRLF

sensitivity-level     =       "Sensitivity-Level" ":" FLOAT
                              CRLF

speed-vs-accuracy     =       "Speed-Vs-Accuracy" ":" FLOAT
                              CRLF

n-best-list-length    =       "N-Best-List-Length" ":" 1*19DIGIT
                              CRLF

input-type            =       "Input-Type" ":"  inputs CRLF
inputs                =       "speech" / "dtmf"

no-input-timeout      =       "No-Input-Timeout" ":" 1*19DIGIT
                              CRLF

recognition-timeout   =       "Recognition-Timeout" ":" 1*19DIGIT
                              CRLF
```

```
waveform-uri            =       "Waveform-URI" ":" ["<" uri ">"
                                ";" "size" "=" 1*19DIGIT
                                ";" "duration" "=" 1*19DIGIT] CRLF

recognizer-context-block = "Recognizer-Context-Block" ":"
                                [1*VCHAR] CRLF

start-input-timers      =       "Start-Input-Timers" ":"
                                BOOLEAN CRLF

speech-complete-timeout =       "Speech-Complete-Timeout" ":"
                                1*19DIGIT CRLF

speech-incomplete-timeout = "Speech-Incomplete-Timeout" ":"
                                 1*19DIGIT CRLF

dtmf-interdigit-timeout = "DTMF-Interdigit-Timeout" ":"
                                1*19DIGIT CRLF

dtmf-term-timeout       =       "DTMF-Term-Timeout" ":" 1*19DIGIT
                                CRLF

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

save-waveform           =       "Save-Waveform" ":" BOOLEAN CRLF

new-audio-channel       =       "New-Audio-Channel" ":"
                                BOOLEAN CRLF

recognition-mode            =   "Recognition-Mode" ":"
                                 normal-value / hotword-value CRLF
normal-value                =   "normal"
hotword-value               =   "hotword"

cancel-if-queue         =       "Cancel-If-Queue" ":" BOOLEAN CRLF

hotword-max-duration    =       "Hotword-Max-Duration" ":"
                                1*19DIGIT CRLF

hotword-min-duration    =       "Hotword-Min-Duration" ":"
                                1*19DIGIT CRLF

interpret-text              =   "Interpret-Text" ":" 1*VCHAR CRLF

dtmf-buffer-time        =       "DTMF-Buffer-Time" ":" 1*19DIGIT CRLF

clear-dtmf-buffer       =       "Clear-DTMF-Buffer" ":" BOOLEAN CRLF
```

```
early-no-match          =      "Early-No-Match" ":" BOOLEAN CRLF

num-min-consistent-pronunciations    =
     "Num-Min-Consistent-Pronunciations" ":" 1*19DIGIT CRLF


consistency-threshold =        "Consistency-Threshold" ":" FLOAT
                               CRLF

clash-threshold         =      "Clash-Threshold" ":" FLOAT CRLF

personal-grammar-uri    =      "Personal-Grammar-URI" ":" uri CRLF

enroll-utterance        =      "Enroll-Utterance" ":" BOOLEAN CRLF

phrase-id               =      "Phrase-ID" ":" 1*VCHAR CRLF

phrase-nl               =      "Phrase-NL" ":" 1*UTFCHAR CRLF

weight                  =      "Weight" ":" weight-value CRLF
weight-value            =      FLOAT

save-best-waveform      =      "Save-Best-Waveform" ":"
                               BOOLEAN CRLF

new-phrase-id           =      "New-Phrase-ID" ":" 1*VCHAR CRLF

confusable-phrases-uri =       "Confusable-Phrases-URI" ":"
                               uri CRLF

abort-phrase-enrollment =      "Abort-Phrase-Enrollment" ":"
                               BOOLEAN CRLF


; Recorder ABNF

recorder-method         =      "RECORD"
                        /      "STOP"
                        /      "START-INPUT-TIMERS"

recorder-event          =      "START-OF-INPUT"
                        /      "RECORD-COMPLETE"

recorder-header         =      sensitivity-level
                        /      no-input-timeout
                        /      completion-cause
                        /      completion-reason
                        /      failed-uri
```

```
                            /    failed-uri-cause
                            /    record-uri
                            /    media-type
                            /    max-time
                            /    trim-length
                            /    final-silence
                            /    capture-on-speech
                            /    ver-buffer-utterance
                            /    start-input-timers
                            /    new-audio-channel

record-uri             =    "Record-URI" ":" [ "<" uri ">"
                            ";" "size" "=" 1*19DIGIT
                            ";" "duration" "=" 1*19DIGIT] CRLF

media-type             =    "Media-Type" ":" media-type-value CRLF

max-time               =    "Max-Time" ":" 1*19DIGIT CRLF

trim-length            =    "Trim-Length" ":" 1*19DIGIT CRLF

final-silence          =    "Final-Silence" ":" 1*19DIGIT CRLF

capture-on-speech      =    "Capture-On-Speech " ":"
                            BOOLEAN CRLF


; Verifier ABNF

verifier-method        =    "START-SESSION"
                            /    "END-SESSION"
                            /    "QUERY-VOICEPRINT"
                            /    "DELETE-VOICEPRINT"
                            /    "VERIFY"
                            /    "VERIFY-FROM-BUFFER"
                            /    "VERIFY-ROLLBACK"
                            /    "STOP"
                            /    "CLEAR-BUFFER"
                            /    "START-INPUT-TIMERS"
                            /    "GET-INTERMEDIATE-RESULT"


verifier-event         =    "VERIFICATION-COMPLETE"
                            /    "START-OF-INPUT"


verifier-header        =    repository-uri
                            /    voiceprint-identifier
```

```
                          /     verification-mode
                          /     adapt-model
                          /     abort-model
                          /     min-verification-score
                          /     num-min-verification-phrases
                          /     num-max-verification-phrases
                          /     no-input-timeout
                          /     save-waveform
                          /     media-type
                          /     waveform-uri
                          /     voiceprint-exists
                          /     ver-buffer-utterance
                          /     input-waveform-uri
                          /     completion-cause
                          /     completion-reason
                          /     speech-complete-timeout
                          /     new-audio-channel
                          /     abort-verification
                          /     start-input-timers
                          /     input-type


repository-uri         =      "Repository-URI" ":" uri CRLF

voiceprint-identifier        =    "Voiceprint-Identifier" ":"
                                  vid *[";" vid] CRLF
vid                          =    1*VCHAR ["." 1*VCHAR]

verification-mode      =      "Verification-Mode" ":"
                              verification-mode-string

verification-mode-string = "train" / "verify"

adapt-model            =      "Adapt-Model" ":" BOOLEAN CRLF

abort-model            =      "Abort-Model" ":" BOOLEAN CRLF

min-verification-score  =  "Min-Verification-Score" ":"
                               [ %x2D ] FLOAT CRLF

num-min-verification-phrases = "Num-Min-Verification-Phrases"
                                 ":" 1*19DIGIT CRLF

num-max-verification-phrases = "Num-Max-Verification-Phrases"
                                 ":" 1*19DIGIT CRLF

voiceprint-exists      =      "Voiceprint-Exists" ":"
```

```
                            BOOLEAN CRLF

ver-buffer-utterance   =     "Ver-Buffer-Utterance" ":"
                             BOOLEAN CRLF

input-waveform-uri     =     "Input-Waveform-URI" ":" uri CRLF

abort-verification     =     "Abort-Verification " ":"
                             BOOLEAN CRLF
```

   The following productions add a new SDP session-level attribute. See
   Paragraph 5.
```
   cmid-attribute   =     "a=cmid:" identification-tag

   identification-tag =     token
```

16. XML Schemas

16.1. NLSML Schema Definition

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.ietf.org/xml/ns/mrcpv2"
            xmlns="http://www.ietf.org/xml/ns/mrcpv2"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified" >
   <xs:annotation>
     <xs:documentation> Natural Language Semantic Markup Schema
     </xs:documentation>
   </xs:annotation>
   <xs:include schemaLocation="enrollment-schema.rng"/>
   <xs:include schemaLocation="verification-schema.rng"/>
   <xs:element name="result">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="interpretation" maxOccurs="unbounded">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="instance" minOccurs="0">
                 <xs:complexType mixed="true">
                   <xs:sequence minOccurs="0">
                     <xs:any namespace="##other" processContents="lax"/>
                   </xs:sequence>
                 </xs:complexType>
               </xs:element>
               <xs:element name="input">
```

```
                        <xs:complexType mixed="true">
                          <xs:choice>
                            <xs:element name="noinput" minOccurs="0"/>
                            <xs:element name="nomatch" minOccurs="0"/>
                            <xs:element name="input" minOccurs="0"/>
                          </xs:choice>
                          <xs:attribute name="mode"
                                        type="xs:string"
                                        default="speech"/>
                          <xs:attribute name="confidence"
                                        type="confidenceinfo"
                                        default="1.0"/>
                          <xs:attribute name="timestamp-start"
                                        type="xs:string"/>
                          <xs:attribute name="timestamp-end"
                                        type="xs:string"/>
                        </xs:complexType>
                      </xs:element>
                  </xs:sequence>
                  <xs:attribute name="confidence" type="confidenceinfo"
                                default="1.0"/>
                  <xs:attribute name="grammar" type="xs:anyURI"
                                use="optional"/>
               </xs:complexType>
             </xs:element>
             <xs:element name="enrollment-result"
                         type="enrollment-contents"/>
             <xs:element name="verification-result"
                         type="verification-contents"/>
        </xs:sequence>
        <xs:attribute name="grammar" type="xs:anyURI"
                      use="optional"/>
     </xs:complexType>
   </xs:element>

   <xs:simpleType name="confidenceinfo">
     <xs:restriction base="xs:float">
        <xs:minInclusive value="0.0"/>
        <xs:maxInclusive value="1.0"/>
     </xs:restriction>
   </xs:simpleType>
 </xs:schema>
```

16.2. Enrollment Results Schema Definition
   <?xml version="1.0" encoding="UTF-8"?>

   <!-- MRCP Enrollment Schema
   (See http://www.oasis-open.org/committees/relax-ng/spec.html)

```
   -->

   <grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
           ns="http://www.ietf.org/xml/ns/mrcpv2"
           xmlns="http://relaxng.org/ns/structure/1.0">

      <start>
        <element name="enrollment-result">
          <ref name="enrollment-content"/>
        </element>
      </start>

      <define name="enrollment-content">
        <interleave>
          <element name="num-clashes">
            <data type="nonNegativeInteger"/>
          </element>
          <element name="num-good-repetitions">
            <data type="nonNegativeInteger"/>
          </element>
          <element name="num-repetitions-still-needed">
            <data type="nonNegativeInteger"/>
          </element>
          <element name="consistency-status">
            <choice>
              <value>consistent</value>
              <value>inconsistent</value>
              <value>undecided</value>
            </choice>
          </element>
          <optional>
            <element name="clash-phrase-ids">
              <oneOrMore>
                <element name="item">
                  <data type="token"/>
                </element>
              </oneOrMore>
            </element>
          </optional>
          <optional>
            <element name="transcriptions">
              <oneOrMore>
                <element name="item">
                  <text/>
                </element>
              </oneOrMore>
            </element>
          </optional>
```

```
          <optional>
            <element name="confusable-phrases">
              <oneOrMore>
                <element name="item">
                  <text/>
                </element>
              </oneOrMore>
            </element>
          </optional>
        </interleave>
    </define>

  </grammar>
```

16.3. Verification Results Schema Definition

```
    <?xml version="1.0" encoding="UTF-8"?>

    <!--    MRCP Verification Results Schema
            (See http://www.oasis-open.org/committees/relax-ng/spec.html)
      -->

    <grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
             ns="http://www.ietf.org/xml/ns/mrcpv2"
             xmlns="http://relaxng.org/ns/structure/1.0">

      <start>
        <element name="verification-result">
          <ref name="verification-contents"/>
        </element>
      </start>

      <define name="verification-contents">
        <element name="voiceprint">
          <ref name="firstVoiceprintContent"/>
        </element>
        <zeroOrMore>
          <element name="voiceprint">
            <ref name="restVoiceprintContent"/>
          </element>
        </zeroOrMore>
      </define>

      <define name="firstVoiceprintContent">
        <attribute name="id">
          <data type="string"/>
        </attribute>
        <interleave>
          <optional>
```

```
              <element name="adapted">
                <data type="boolean"/>
              </element>
            </optional>
            <optional>
              <element name="needmoredata">
                <ref name="needmoredataContent"/>
              </element>
            </optional>
            <optional>
              <element name="incremental">
                <ref name="firstCommonContent"/>
              </element>
            </optional>
            <element name="cumulative">
              <ref name="firstCommonContent"/>
            </element>
          </interleave>
        </define>

        <define name="restVoiceprintContent">
          <attribute name="id">
            <data type="string"/>
          </attribute>
          <element name="cumulative">
            <ref name="restCommonContent"/>
          </element>
        </define>

        <define name="firstCommonContent">
          <interleave>
            <element name="decision">
              <ref name="decisionContent"/>
            </element>
            <optional>
              <element name="utterance-length">
                <ref name="utterance-lengthContent"/>
              </element>
            </optional>
            <optional>
              <element name="device">
                <ref name="deviceContent"/>
              </element>
            </optional>
            <optional>
              <element name="gender">
                <ref name="genderContent"/>
              </element>
```

```
          </optional>
          <zeroOrMore>
            <element name="verification-score">
              <ref name="verification-scoreContent"/>
            </element>
          </zeroOrMore>
        </interleave>
      </define>

      <define name="restCommonContent">
        <interleave>
          <optional>
            <element name="decision">
              <ref name="decisionContent"/>
            </element>
          </optional>
          <optional>
            <element name="device">
              <ref name="deviceContent"/>
            </element>
          </optional>
          <optional>
            <element name="gender">
              <ref name="genderContent"/>
            </element>
          </optional>
        <zeroOrMore>
            <element name="verification-score">
              <ref name="verification-scoreContent"/>
            </element>
        </zeroOrMore>
        </interleave>
      </define>

      <define name="decisionContent">
        <choice>
          <value>accepted</value>
          <value>rejected</value>
          <value>undecided</value>
        </choice>
      </define>

      <define name="needmoredataContent">
        <data type="boolean"/>
      </define>

      <define name="utterance-lengthContent">
        <data type="nonNegativeInteger"/>
```

```
      </define>

      <define name="deviceContent">
        <choice>
          <value>cellular-phone</value>
          <value>electret-phone</value>
          <value>carbon-button-phone</value>
          <value>unknown</value>
        </choice>
      </define>

      <define name="genderContent">
        <choice>
          <value>male</value>
          <value>female</value>
          <value>unknown</value>
        </choice>
      </define>

      <define name="verification-scoreContent">
        <data type="float">
          <param name="minInclusive">-1</param>
          <param name="maxInclusive">1</param>
        </data>
      </define>

   </grammar>
```

17. References

17.1. Normative References

   [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V.
              Jacobson, "RTP: A Transport Protocol for Real-Time
              Applications", STD 64, RFC 3550, July 2003.

   [RFC3261] 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.

   [RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time
              Streaming Protocol (RTSP)", RFC 2326, April 1998.

   [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session
              Description Protocol", RFC 4566, July 2006.

   [RFC0793]   Postel, J., "Transmission Control Protocol", STD 7,
               RFC 793, September 1981.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
               Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC2616]   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.

   [RFC3264]   Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model
               with Session Description Protocol (SDP)", RFC 3264,
               June 2002.

   [RFC3629]   Yergeau, F., "UTF-8, a transformation format of ISO
               10646", STD 63, RFC 3629, November 2003.

   [RFC5234]   Crocker, D. and P. Overell, "Augmented BNF for Syntax
               Specifications: ABNF", STD 68, RFC 5234, January 2008.

   [RFC4145]   Yon, D. and G. Camarillo, "TCP-Based Media Transport in
               the Session Description Protocol (SDP)", RFC 4145,
               September 2005.

   [RFC4572]   Lennox, J., "Connection-Oriented Media Transport over the
               Transport Layer Security (TLS) Protocol in the Session
               Description Protocol (SDP)", RFC 4572, July 2006.

   [RFC5888]   Camarillo, G. and H. Schulzrinne, "The Session Description
               Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

   [RFC5322]   Resnick, P., Ed., "Internet Message Format", RFC 5322,
               October 2008.

   [RFC2392]   Levinson, E., "Content-ID and Message-ID Uniform Resource
               Locators", RFC 2392, August 1998.

   [RFC2109]   Kristol, D. and L. Montulli, "HTTP State Management
               Mechanism", RFC 2109, February 1997.

   [RFC2965]   Kristol, D. and L. Montulli, "HTTP State Management
               Mechanism", RFC 2965, October 2000.

   [RFC5646]   Phillips, A. and M. Davis, "Tags for Identifying
               Languages", BCP 47, RFC 5646, September 2009.

   [RFC5226]   Narten, T. and H. Alvestrand, "Guidelines for Writing an
               IANA Considerations Section in RFCs", BCP 26, RFC 5226,

            May 2008.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246, August 2008.

   [RFC1035] Mockapetris, P., "Domain names - implementation and
              specification", STD 13, RFC 1035, November 1987.

   [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
              Resource Identifier (URI): Generic Syntax", STD 66,
              RFC 3986, January 2005.

   [RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and
              Registration Procedures", BCP 13, RFC 4288, December 2005.

   [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
              January 2004.

   [RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session
              Description Protocol (SDP) Security Descriptions for Media
              Streams", RFC 4568, July 2006.

   [W3C.REC-speech-synthesis-20040907]
              Burnett, D., Walker, M., and A. Hunt, "Speech Synthesis
              Markup Language (SSML) Version 1.0", World Wide Web
              Consortium Recommendation REC-speech-synthesis-20040907,
              September 2004,
              <http://www.w3.org/TR/2004/REC-speech-synthesis-20040907>.

   [RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network
              Time Protocol Version 4: Protocol and Algorithms
              Specification", RFC 5905, June 2010.

   [RFC2483] Mealling, M. and R. Daniel, "URI Resolution Services
              Necessary for URN Resolution", RFC 2483, January 1999.

   [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K.
              Norrman, "The Secure Real-time Transport Protocol (SRTP)",
              RFC 3711, March 2004.

   [RFC5922] Gurbani, V., Lawrence, S., and A. Jeffrey, "Domain
              Certificates in the Session Initiation Protocol (SIP)",
              RFC 5922, June 2010.

   [W3C.REC-speech-grammar-20040316]
              Hunt, A. and S. McGlashan, "Speech Recognition Grammar
              Specification Version 1.0", World Wide Web Consortium
              Recommendation REC-speech-grammar-20040316, March 2004,

                 <http://www.w3.org/TR/2004/REC-speech-grammar-20040316>.

   [W3C.REC-semantic-interpretation-20070405]
            Tichelen, L. and D. Burke, "Semantic Interpretation for
            Speech Recognition (SISR) Version 1.0", World Wide Web
            Consortium REC REC-semantic-interpretation-20070405,
            April 2007, <http://www.w3.org/TR/2007/
            REC-semantic-interpretation-20070405>.

   [W3C.REC-xml-names11-20040204]
            Tobin, R., Layman, A., Bray, T., and D. Hollander,
            "Namespaces in XML 1.1", World Wide Web Consortium
            FirstEdition REC-xml-names11-20040204, February 2004,
            <http://www.w3.org/TR/2004/REC-xml-names11-20040204>.

   [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media
            Types", RFC 3023, January 2001.

   [ISO.8859-1.1987]
            International Organization for Standardization,
            "Information technology - 8-bit single byte coded graphic
            - character sets - Part 1: Latin alphabet No. 1, JTC1/
            SC2", ISO Standard 8859-1, 1987.

17.2. Informative References

   [RFC4960] Stewart, R., "Stream Control Transmission Protocol",
            RFC 4960, September 2007.

   [RFC4313] Oran, D., "Requirements for Distributed Control of
            Automatic Speech Recognition (ASR), Speaker
            Identification/Speaker Verification (SI/SV), and Text-to-
            Speech (TTS) Resources", RFC 4313, December 2005.

   [Q.23]   International Telecommunications Union, "Technical
            Features of Push-Button Telephone Sets", ITU-T Q.23, 1993.

   [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and
            Registration Procedures for New URI Schemes", BCP 35,
            RFC 4395, February 2006.

   [RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF
            Digits, Telephony Tones, and Telephony Signals", RFC 4733,
            December 2006.

   [W3C.REC-voicexml20-20040316]
            Danielsen, P., Porter, B., Carter, J., Rehor, K., Hunt,
            A., McGlashan, S., Ferrans, J., Lucas, B., Burnett, D.,

and S. Tryphonas, "Voice Extensible Markup Language
(VoiceXML) Version 2.0", World Wide Web Consortium
Recommendation REC-voicexml20-20040316, March 2004,
<http://www.w3.org/TR/2004/REC-voicexml20-20040316>.

[RFC4463]   Shanmugham, S., Monaco, P., and B. Eberman, "A Media
Resource Control Protocol (MRCP) Developed by Cisco,
Nuance, and Speechworks", RFC 4463, April 2006.

[refs.javaSpeechGrammarFormat]
Sun Microsystems, "Java Speech Grammar Format Version
1.0", October 1998.

[W3C.REC-emma-20090210]
Johnston, M., Baggia, P., Burnett, D., Carter, J., Dahl,
D., McCobb, G., and D. Raggett, "EMMA: Extensible
MultiModal Annotation markup language", World Wide Web
Consortium Recommendation REC-emma-20090210,
February 2009,
<http://www.w3.org/TR/2009/REC-emma-20090210>.

[RFC4467]   Crispin, M., "Internet Message Access Protocol (IMAP) -
URLAUTH Extension", RFC 4467, May 2006.

[W3C.REC-pronunciation-lexicon-20081014]
Baggia, P., Bagshaw, P., Burnett, D., Carter, J., and F.
Scahill, "Pronunciation Lexicon Specification (PLS)",
World Wide Web Consortium Recommendation REC-
pronunciation-lexicon-20081014, October 2008, <http://
www.w3.org/TR/2008/REC-pronunciation-lexicon-20081014>.

[RFC2818]   Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[ISO.8601.1988]
International Organization for Standardization, "Data
elements and interchange formats - Information interchange
- Representation of dates and times", ISO Standard 8601,
June 1988.

[RFC4301]   Kent, S. and K. Seo, "Security Architecture for the
Internet Protocol", RFC 4301, December 2005.

[RFC4217]   Ford-Hutchinson, P., "Securing FTP with TLS", RFC 4217,
October 2005.

Appendix A. Contributors

    Pierre Forgues
    Nuance Communications Ltd.
    1500 University Street
    Suite 935
    Montreal, Quebec
    Canada H3A 3S7

    Email: forgues@nuance.com

    Charles Galles
    Intervoice, Inc.
    17811 Waterview Parkway
    Dallas, Texas 75252

    Email: charles.galles@intervoice.com

    Klaus Reifenrath
    Scansoft, Inc
    Guldensporenpark 32
    Building D
    9820 Merelbeke
    Belgium

    Email: klaus.reifenrath@scansoft.com

Appendix B. Acknowledgements

    Andre Gillet (Nuance Communications)
    Andrew Hunt (ScanSoft)
    Andrew Wahbe (Genesys)
    Aaron Kneiss (ScanSoft)
    Brian Eberman (ScanSoft)
    Corey Stohs (Cisco Systems Inc)
    Dave Burke (VoxPilot)
    Jeff Kusnitz (IBM Corp)
    Ganesh N Ramaswamy (IBM Corp)
    Klaus Reifenrath (ScanSoft)
    Kristian Finlator (ScanSoft)
    Magnus Westerlund (Ericsson)
    Martin Dragomirecky (Cisco Systems Inc)
    Paolo Baggia (Loquendo)
    Peter Monaco (Nuance Communications)
    Pierre Forgues (Nuance Communications)
    Ran Zilca (IBM Corp)
    Suresh Kaliannan (Cisco Systems Inc.)
    Skip Cave (Intervoice Inc)
    Thomas Gal (LumenVox)

    The chairs of the speechsc work group are Eric Burger (Georgetown
    University) and Dave Oran (Cisco Systems, Inc.).

    Many thanks go in particular to Robert Sparks, Alex Agranovsky, and
    Henry Phan, who were there at the end to dot all the i's and cross
    all the t's.

Authors' Addresses

    Daniel C. Burnett
    Voxeo
    189 South Orange Avenue #2050
    Orlando, FL 32801
    USA

    Email: dburnett@voxeo.com

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

Email: sarvi@cisco.com