W3C Incubator Group Report

HTML Speech Incubator Group Final Report

W3C Incubator Group Report 06 December 2011

This version:
http://www.w3.org/2005/Incubator/htmlspeech/XGR-htmlspeech-20111206/
Latest published version:
http://www.w3.org/2005/Incubator/htmlspeech/XGR-htmlspeech/
Previous version:
none
Editors:
Michael Bodell, Microsoft
Björn Bringert, Google
Robert Brown, Microsoft
Daniel C. Burnett, Voxeo
Deborah Dahl, W3C Invited Expert
Dan Druta, AT&T
Patrick Ehlen, AT&T
Charles Hemphill, EverSpeech
Michael Johnston, AT&T
Olli Pettay, Mozilla
Satish Sampath, Google
Marc Schröder, German Research Center for Artificial Intelligence (DFKI) GmbH
Glen Shires, Google
Raj Tumuluri, Openstream
Milan Young, Nuance

Abstract

This document is the Final Report of the HTML Speech Incubator Group. It presents the deliverables of the group, including use cases, requirements, design decisions, and preliminary proposals for html elements, a JavaScript API, and a protocol for implementing both. It is expected that this work will feed directly into new standards-track working groups in W3C and the IETF to bring this work to completion.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of Final Incubator Group Reports is available. See also the W3C technical reports index at http://www.w3.org/TR/.

This document was produced by the HTML Speech Incubator Group. It represents the Final Report of the Incubator Group. It is expected that the contents of this Report will be used as input to existing or new Recommendation-track Working Groups of W3C and/or other Standards Development Organizations.

Publication of this document by W3C as part of the W3C Incubator Activity indicates no endorsement of its content by W3C, nor that W3C has, is, or will be allocating any resources to the issues addressed by it. Participation in Incubator Groups and publication of Incubator Group Reports at the W3C site are benefits of W3C Membership.

Incubator Groups have as a goal to produce work that can be implemented on a Royalty Free basis, as defined in the W3C Patent Policy. Participants in this Incubator Group have made no statements about whether they will offer licenses according to the licensing requirements of the W3C Patent Policy for portions of this Incubator Group Report that are subsequently incorporated in a W3C Recommendation.

Table of Contents

1. Overview

This document presents the deliverables of the HTML Speech Incubator Group. First, it covers the use cases developed by the group. Next, it presents the requirements developed by the group, ordered by priority of interest of the group members. It then presents design decisions on important topics that go beyond the requirements and into the realm of implementation, with a focus on satisfying the high-interest requirements. Finally, the document contains a preliminary proposal for a JavaScript API and associated HTML bindings, followed by a preliminary proposal for a WebSockets-based protocol for communication between a User Agent, such as a web browser, and URI-addressed ASR and TTS services.

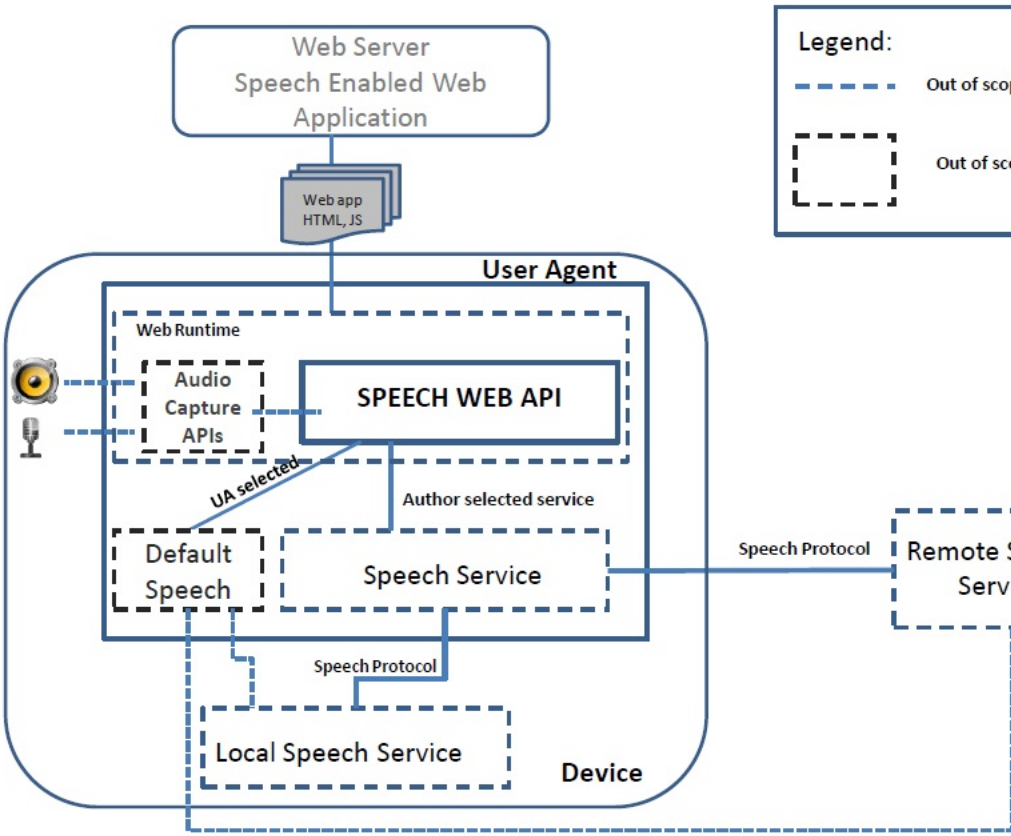
The major steps the group took in working towards API recommendations, rather than just the final decisions, are recorded to act as an aid to any future standards-track efforts in understanding the motivations that drove the recommendations. Thus, even if a final standards-track document differs from any API recommendations in this document, the final standard should address the use cases, requirements, and design decisions laid out by this Incubator Group.

The following diagram outlines what items would be in and out of scope for the final solution to the task this group has begun. The in-scope items are:

- The Speech Web API
- The Speech Protocol

- The ability for the app author to choose between using the default (User Agent-selected) or an author-selected service

The other items are out of scope.



2. Conformance

Everything in this proposal is informative since this is not a standards track document. However, RFC2119 normative language is used where appropriate to aid in the future should this proposal be moved into a standards track process.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in [*RFC2119*].

3. Deliverables

According to the charter, the group is to produce one deliverable, this document. It goes on to state that the document may include

- Requirements

- Use cases
- Change requests to HTML5 [*HTML5*] and, as appropriate, other specifications, e.g., capture API, CSS, Audio XG, EMMA, SRGS, VoiceXML 3

The group has developed requirements, some with use cases, and has made progress towards one or more API proposals. These subdeliverables follow.

4. Use Cases

Throughout this process the group has developed a lot of different use cases covering a variety of scenarios. These use cases were developed as part of the requirements process, through the proposals that were submitted, and in our group discussion. It is important that it be possible to support as many of these use cases as easily as possible by our proposed solutions.

4.1 Voice Web Search

The user can speak a query and get a result.

4.2 Speech Command Interface

A Speech Command and Control Shell that allows multiple commands, many of which may take arguments, such as "call [number]", "call [person]", "calculate [math expression]", "play [song]", or "search for [query]".

4.3 Domain Specific Grammars Contingent on Earlier Inputs

A use case exists around collecting multiple domain specific inputs sequentially where the later inputs depend on the results of the earlier inputs. For instance, changing which cities are in a grammar of cities in response to the user saying in which state they are located.

4.4 Continuous Recognition of Open Dialog

This use case is to collect free form spoken input from the user. This might be particularly relevant to an email system, for instance. When dictating an email, the user will continue to utter sentences until they're done composing their email. The application will provide continuous feedback to the user by displaying words within a brief period of the user uttering them. The application continues listening and updating the screen until the user is done. Sophisticated applications will also listen for command words used to add formatting, perform edits, or correct errors.

4.5 Domain Specific Grammars Filling Multiple Input Fields

Many web applications incorporate a collection of input fields, generally expressed as forms, with some text boxes to type into and lists to select from, with a "submit"

button at the bottom. For example, "find a flight from New York to San Francisco on Monday morning returning Friday afternoon" might fill in a web form with two input elements for origin (place & date), two for destination (place & time), one for mode of transport (flight/bus/train), and a command (find) for the "submit" button. The results of the recognition would end up filling all of these multiple input elements with just one user utterance. This application is valuable because the user just has to initiate speech recognition once to complete the entire screen.

4.6 Speech UI present when no visible UI need be present

Some speech applications are oriented around determining the user's intent before gathering any specific input, and hence their first interaction may have no visible input fields whatsoever, or may accept speech input that is far less constrained than the fields on the screen. For example, the user may simply be presented with the text "how may I help you?" (maybe with some speech synthesis or an earcon), and then utter their request, which the application analyzes in order to route the user to an appropriate part of the application. This isn't simply selection from a menu, because the list of options may be huge, and the number of ways each option could be expressed by the user is also huge. In any case, the speech UI (grammar) is very different from whatever input elements may or may not be displayed on the screen. In fact, there may not even be any visible non-speech input elements displayed on the page.

4.7 Rerecognition

Some sophisticated applications will re-use the same utterance in two or more recognitions turns in what appears to the user as one turn. For example, an application may ask "how may I help you?", to which the user responds "find me a round trip from New York to San Francisco on Monday morning, returning Friday afternoon". An initial recognition against a broad language model may be sufficient to understand that the user wants the "flight search" portion of the app. Rather than get the user to repeat themselves, the application will just re-use the existing utterance for the recognition on the flight search recognition.

4.8 Voice Activity Detection

Automatic detection of speech/non-speech boundaries is needed for a number of valuable user experiences such as "Push once to talk" or "hands-free dialog". In press-once to talk the user manually interacts with the app to indicate that the app should start listening. For example, they raise the device to their ear, press a button on the keypad, or touch a part of the screen. When they're done talking, the app automatically performs the speech recognition without the user needing to touch the device again. In hands-free dialog, where the user can start and stop talking without any manual input to indicate when the application should be listening. The application and/or browser needs to automatically detect when the user has started

talking, so it can initiate speech recognition. This is particularly useful for in-car, or 10-foot usage (e.g. living room), or for people with disabilities.

4.9 Temporal Structure of Synthesis to Provide Visual Feedback

The application may wish to visually highlight the word or phrase that the application is synthesizing. Or, alternatively, the visual application may wish to coordinate the synthesis with animations of an avatar speaking or with appropriately timed slide transitions and thus need to know where in the reading of the synthesized text the application currently is. In addition, the application may wish to know where in a piece of synthesized text an interruption occurred and use the temporal feedback to tell.

4.10 Hello World

The web page when loaded may wish to say a simple phrase of synthesized text such as "hello world".

4.11 Speech Translation

The application can act as a translator between two individuals fluent in different languages. The application can listen to one speaker and understand the utterances in one language, and can then translate the spoken phrases to a different language, and then can speak the translation to the other individual.

4.12 Speech Enabled Email Client

The application reads out subjects and contents of email and also listens for commands, for instance, "archive", "reply: ok, let's meet at 2 pm", "forward to bob", "read message". Some commands may relate to VCR like controls of the message being read back, for instance, "pause", "skip forwards", "skip back", or "faster". Some of those controls may include controls related to parts of speech, such as, "repeat last sentence" or "next paragraph".

One other important email scenario is that when an email message is received, a summary notification may be raised that displays a small amount of content (for instance the person the email is from and a couple of words of the subject). It is desirable that a speech API be present and listening for the duration of this notification, allowing a user experience of being able to say "Reply to that" or "Read that email message". Note that this recognition UI could not be contingent on the user clicking a button, as that would defeat much of the benefit of this scenario (being able to reply and control the email without using the keyboard or mouse).

4.13 Dialog Systems

The type of dialogs that allow for collecting multiple pieces of information in either one turn or sequential turns in response to frequently synthesized prompts. Types of dialogs might be around ordering a pizza or booking a flight route complete with the system repeating back the choices the user said. This dialog system may well be represented by a VXML form or application that allows for control of the dialog. The VXML dialog may be fetched using XMLHttpRequest.

4.14 Multimodal Interaction

The ability to mix and integrate input from multiple modalities such as by saying "I want to go from here to there" while tapping two points on a touch screen map.

4.15 Speech Driving Directions

A direction service that speaks turn-by-turn directions. Accepts hands-free spoken instructions like "navigate to [address]" or "navigate to [business listing]" or "reroute using [road name]". Input from the location of the user may help the service know when to play the next direction. It is possible that user is not able to see any output so the service needs to regularly synthesize phrases like "turn left on [road] in [distance]".

4.16 Multimodal Video Game

The user combines speech input and output with tactile input and visual output to enable scenarios such as tapping a location on the screen while issuing an in game command like "Freeze Spell". Speech could be used either to initiate the action or as an inventory changing system, all while the normal action of the video game is continuing.

4.17 Multimodal Search

The user points their cell phone camera at an object, and uses voice to issue some query about that object (such as "what is this", "where can I buy one", or "do these also come in pink").

5. Requirements

5.1 Prioritized Requirements

The HTML Speech Incubator Group measured industry interest in the importance of the various requirements by surveying the membership. The complete results are available here.

A summary of the results is presented below with requirements listed in priority order, and segmented into those with strong interest, those with moderate interest,

and those with mild interest. Each requirement is linked to its description in section A.2 First Pass Requirements.

5.1.1 Strong Interest

A requirement was classified as having "strong interest" if at least 80% of the group believed it needs to be addressed by any specification developed based on the work of this group. These requirements are:

- FPR40. Web applications must be able to use barge-in (interrupting audio and TTS output when the user starts speaking).
- FPR4. It should be possible for the web application to get the recognition results in a standard format such as EMMA.
- FPR24. The web app should be notified when recognition results are available.
- FPR50. Web applications must not be prevented from integrating input from multiple modalities.
- FPR59. While capture is happening, there must be a way for the web application to abort the capture and recognition process.
- FPR52. The web app should be notified when TTS playback finishes.
- FPR60. Web application must be able to programatically abort tts output.
- FPR38. Web application must be able to specify language of recognition.
- FPR45. Applications should be able to specify the grammars (or lack thereof) separately for each recognition.
- FPR1. Web applications must not capture audio without the user's consent.
- FPR19. User-initiated speech input should be possible.
- FPR21. The web app should be notified that capture starts.
- FPR22. The web app should be notified that speech is considered to have started for the purposes of recognition.
- FPR23. The web app should be notified that speech is considered to have ended for the purposes of recognition.
- FPR25. Implementations should be allowed to start processing captured audio before the capture completes.
- FPR26. The API to do recognition should not introduce unneeded latency.
- FPR34. Web application must be able to specify domain specific custom grammars.
- FPR35. Web application must be notified when speech recognition errors or non-matches occur.
- FPR42. It should be possible for user agents to allow hands-free speech input.
- FPR48. Web application author must be able to specify a domain specific statistical language model.
- FPR54. Web apps should be able to customize all aspects of the user interface for speech recognition, except where such customizations conflict with security and privacy requirements in this document, or where they cause other security or privacy problems.
- FPR51. The web app should be notified when TTS playback starts.

- FPR53. The web app should be notified when the audio corresponding to a TTS element is played back.
- FPR5. It should be easy for the web appls to get access to the most common pieces of recognition results such as utterance, confidence, and nbests.
- FPR39. Web application must be able to be notified when the selected language is not available.
- FPR13. It should be easy to assign recognition results to a single input field.
- FPR14. It should not be required to fill an input field every time there is a recognition result.
- FPR15. It should be possible to use recognition results to multiple input fields.
- FPR16. User consent should be informed consent.
- FPR18. It must be possible for the user to revoke consent.
- FPR11. If the web apps specify speech services, it should be possible to specify parameters.
- FPR12. Speech services that can be specified by web apps must include network speech services.
- FPR2. Implementations must support the XML format of SRGS and must support SISR.
- FPR27. Speech recognition implementations should be allowed to add implementation specific information to speech recognition results.
- FPR3. Implementation must support SSML.
- FPR46. Web apps should be able to specify which voice is used for TTS.
- FPR7. Web apps should be able to request speech service different from default.
- FPR9. If browser refuses to use the web application requested speech service, it must inform the web app.
- FPR17. While capture is happening, there must be an obvious way for the user to abort the capture and recognition process.
- FPR37. Web application should be given captured audio access only after explicit consent from the user.
- FPR49. End users need a clear indication whenever microphone is listening to the user.

5.1.2 Moderate Interest

A requirement was classified as having "moderate interest" if less than 80% but at least 50% of the group believed it needs to be addressed by any specification developed based on the work of this group. These requirements are:

- FPR33. There should be at least one mandatory-to-support codec that isn't encumbered with IP issues and has sufficient fidelity & low bandwidth requirements.
- FPR28. Speech recognition implementations should be allowed to fire implementation specific events.
- FPR41. It should be easy to extend the standard without affecting existing speech applications.

- FPR36. User agents must provide a default interface to control speech recognition.
- FPR44. Recognition without specifying a grammar should be possible.
- FPR61. Aborting the TTS output should be efficient.
- FPR32. Speech services that can be specified by web apps must include aal speech services.
- FPR47. When speech input is used to provide input to a web app, it should be possible for the user to select alternative input methods.
- FPR56. Web applications must be able to request NL interpretation based only on text input (no audio sent).
- FPR30. Web applications must be allowed at least one form of communication with a particular speech service that is supported in all UAs.
- FPR55. Web application must be able to encrypt communications to remote speech service.
- FPR58. Web application and speech services must have a means of binding session information to communications.
- FPR6. Browser must provide default speech resource.
- FPR20. The spec should not unnecessarily restrict the UA's choice in privacy policy.

5.1.3 Mild Interest

A requirement was classified as having "mild interest" if less than 50% of the group believed it needs to be addressed by any specification developed based on the work of this group. These requirements are:

- FPR29. Speech synthesis implementations should be allowed to fire implementation specific events.
- FPR31. User agents and speech services may agree to use alternate protocols for communication.
- FPR43. User agents should not be required to allow hands-free speech input.
- FPR10. If browser uses speech services other than the default one, it must inform the user which one(s) it is using.
- FPR8. User agent (browser) can refuse to use requested speech service.
- FPR57. Web applications must be able to request recognition based on previously sent audio.

5.2 New Requirements

While discussing some of the use cases, proposals, design decisions, and possible solution a few more requirements were discovered and agreed to. These requirements are:

1. The web application must be able to perform continuous recognition (I.e., dictation).

2. The web application must be able to perform an open mic scenario (I.e., always listening for keywords).
3. The web application must be able to get interim recognition results when it is performing continuous recognition.
4. It must be possible for the application author to provide feedback on the recognition result

6. Solution Design Agreements and Alternatives

This section attempts to capture the major design decisions the group made. In cases where substantial disagreements existed, the relevant alternatives are presented rather than a decision. Note that text only went into this section if it either represented group consensus or an accurate description of the specific alternative, as appropriate.

Also note that the current text reflects our most recent understanding. Where this differs from the original understanding/wording, the original wording is provided as well.

1. There are three aspects to the solution which must be addressed: communication with and control of speech services, a script-level API, and markup-level hooks and capabilities.
2. The script API will be JavaScript.
3. The scripting API is the primary focus, with all key functionality available via scripting. Any HTML markup capabilities, if present, will be based completely on the scripting capabilities.
4. Notifications from the user agent to the web application should be in the form of JavaScript events/callbacks.
5. For ASR, there must at least be these three logical functions:
    1. start speech input and start processing
    2. stop speech input and get result
    3. cancel (stop speech input and ignore result)
6. For TTS, there must be at least these two logical functions:
    1. play
    2. pause

    There is agreement that it should be possible to stop playback, but there is not agreement on the need for an explicit stop function.
7. It must be possible for a web application to specify the speech engine.
8. Speech service implementations must be referenceable by URI.
9. It must be possible to reference ASR grammars by URI.
10. It must be possible to select the ASR language using language tags.
11. It must be possible to leave the ASR grammar unspecified. Behavior in this case is not yet defined.
12. The XML format of SRGS 1.0 is mandatory to support, and it is the only mandated grammar format. Note in particular that this means we do not have any requirement for SLM support or SRGS ABNF support.

13. For TTS, SSML 1.1 is mandatory to support, as is UTF-8 plain text. These are the only mandated formats.
14. SISR 1.0 support is mandatory, and it is the only mandated semantic interpretation format.
15. There must be no technical restriction that would prevent using only TTS or only ASR.
16. There must be no technical restriction that would prevent implementing only TTS or only ASR. There is *mostly* agreement on this.
17. There will be a mandatory set of capabilities with stated limitations on interoperability.
18. For reco results, both the DOM representation of EMMA and the XML text representation must be provided.
19. For reco results, a simple JavaScript representation of a list of results must be provided, with each result containing the recognized utterance, confidence score, and semantic interpretation. Note that this may need to be adjusted based on any decision regarding support for continuous recognition.
20. For grammar URIs, the "HTTP" and "data" protocol schemes must be supported.
21. A standard set of common-task grammars must be supported. The details of what those are is TBD.
22. The API should be able to start speech reco without having to select a microphone, i.e., there must be a notion of a "default" microphone.
23. There should be a default user interface.
24. The user agent must notify the user when audio is being captured. Web applications must not be able to override this notification.
25. It must be possible to customize the user interface to control how recognition start is indicated.
26. If the HTML standard has an audio capture API, we should be able to use it for ASR. If not, we should not create one, and we will not block waiting for one to be created.
27. We will collect requirements on audio capture APIs and relay them to relevant groups.
28. A low-latency endpoint detector must be available. It should be possible for a web app to enable and disable it, although the default setting (enabled/disabled) is TBD. The detector detects both start of speech and end of speech and fires an event in each case.
29. The API will provide control over which portions of the captured audio are sent to the recognizer.
30. We expect to have the following six audio/speech events: onaudiostart/onaudioend, onsoundstart/onsoundend, onspeechstart/onspeechend. The onsound* events represent a "probably speech but not sure" condition, while the onspeech* events represent the recognizer being sure there's speech. The former are low latency. An end event can only occur after at least one start event of the same type has occurred. Only the user agent can generate onaudio* events, the energy detector can only

generate onsound* events, and the speech service can only generate onspeech* events.

31. There are 3 classes of codecs: audio to the web-app specified ASR engine, recognition from existing audio (e.g., local file), and audio from the TTS engine. We need to specify a mandatory-to-support codec for each.
32. It must be possible to specify and use other codecs in addition to those that are mandatory-to-implement.
33. Support for streaming audio is required -- in particular, that ASR may begin processing before the user has finished speaking.
34. It must be possible for the recognizer to return a final result before the user is done speaking.

35. We will require support for WebSockets for all communication between the user agent and any selected engine, including chunked audio for media streaming.

    Original wording: We will require support for http for all communication between the user agent and any selected engine, including chunked http for media streaming, and support negotiation of other protocols (such as WebSockets or whatever RTCWeb/WebRTC comes up with).

36. Maxresults should be an ASR parameter representing the maximum number of results to return.

37. The user agent will use the URI for the ASR or TTS service exactly as specified by the web application, including all parameters, and will not modify it to add, remove, or change parameters.

    Original wording: The user agent will use the URI for the ASR engine exactly as specified by the web application, including all parameters, and will not modify it to add, remove, or change parameters.

38. The scripting API communicates its parameter settings by sending them as typed content in the protocol.

    Original wording: The scripting API communicates its parameter settings by sending them in the body of a POST request as Media Type "multipart". The subtype(s) accepted (e.g., mixed, formdata) are TBD.

39. If an ASR or TTS service allows parameters to be specified in the URI in addition to being transported as content, when a parameter is specified in both places the one in the content takes precedence. This has the effect of making parameters set in the URI be treated as default values.

    Original wording: If an ASR engine allows parameters to be specified in the URI in addition to in the POST body, when a parameter is specified in both places the one in the body takes precedence. This has the effect of making parameters set in the URI be treated as default values.

40. We cannot expect consistency in language support and performance/quality.
41. We agree that there must be API-level consistency regardless of user agent and engine.
42. We agree on having the same level of consistency across all four of the following categories:
    1. consistency between different UAs using their default engine
    2. consistency between different UAs using web app specified engine
    3. consistency between different UAs using different web specified engines
    4. consistency between default engine and specified engines

    With exception that #4 may have limitations due to privacy issues.
43. From this point on we will use "service" rather than "engine" because a service may be a proxy for more than one engine.
44. We will not support selection of service by characteristics.
45. Add to list of expected inconsistency (change from existing wording of interoperability): reco performance including maximum size on parameters, microphone characteristics, semantics and exact values of sensitivity and confidence, time need to perform ASR/TTS, latencies, endpoint sensitivity and latency, result contents, presence/absence of optional events, recorded waveform
46. For continuous recognition, we must support the ability to change grammars and parameters for each chunk/frame/result
47. If the user's device is emitting other sounds than those produced by the current HTML page, there is no particular requirement that the User Agent be required to detect/reduce/eliminate it.
48. If a web app specifies a speech service and it is not available, an error is thrown. No automatic fallback to another service or the default service takes place.
49. The API should provide a way to determine if a service is available before trying to use the service; this applies to the default service as well.
50. The API must provide a way to query the availability of a specific configuration of a service.
51. The API must provide a way to ask the user agent for the capabilities of a service. In the case of private information that the user agent may have when the default service is selected, the user agent may choose to answer with "no comment" (or equivalent).
52. Informed user consent is required for all use of private information. This includes list of languages for ASR and voices for TTS. When such information is requested by the web app or speech service and permission is refused, the API must return "no comment" (or equivalent).
53. It must be possible for user permission to be granted at the level of specific web apps and/or speech services.
54. User agents, acting on behalf of the user, may deny the use of specific web apps and/or speech services.
55. The API will support multiple simultaneous grammars, any combination of allowed grammar formats. It will also support a weight on each grammar.

56. The API will support multiple simultaneous requests to speech services (same or different, ASR and TTS).
57. We disagree about whether there needs to be direct API support for a single ASR request and single TTS request that are tied together.
58. It must be possible to individually control ASR and TTS.
59. It must be possible for the web app author to get timely information about recognition event timing and about TTS playback timing. It must be possible for the web app author to determine, for any specific UA local time, what the previous TTS mark was and the offset from that mark.
60. It must be possible for the web app to stop/pause/silence audio output directly at the client/user agent.
61. When audio corresponding to TTS mark location begins to play, a JavaScript event must be fired, and the event must contain the name of the mark and the UA timestamp for when it was played.
62. It must be possible to specify service-specific parameters in both the URI and the message body. It must be clear in the API that these parameters are service-specific, i.e., not standard.

63. Every request from UA to recognition service should send the UA-local timestamp.

    Original wording: Every message from UA to speech service should send the UA-local timestamp.

64. API must have ability to set service-specific parameters using names that clearly identify that they are service-specific, e.g., using an "x-" prefix. Parameter values can be arbitrary JavaScript objects.
65. EMMA already permits app-specific result info, so there is no need to provide other ways for service-specific information to be returned in the result.
66. The API must support DOM 3 extension events as defined (which basically require vendor prefixes). See http://www.w3.org/TR/2009/WD-DOM-Level-3-Events-20090908/#extending_events-Vendor_Extensions. It must allow the speech service to fire these events.

67. The UA must send its current timestamp to the speech service when it sends its first audio data.

    Original wording: The protocol must send its current timestamp to the speech service when it sends its first audio data.

68. It must be possible for the speech service to instruct the UA to fire a vendor-specific event when a specific offset to audio playback start is reached by the UA. What to do if audio is canceled, paused, etc. is TBD.

69. It MUST be possible to use an encrypted protocol.

    Original wording: HTTPS must also be supported.

70. Using web app in secure communication channel should be treated just as when working with all secured sites (e.g., with respect to non-secured channel for speech data).
71. Default speech service implementations are encouraged not to use unsecured network communication when started by a web app in a secure communication channel
72. In JavaScript, speech reco requests should have an attribute for a sequence of grammars, each of which can have properties, including weight (and possibly language, but that is TBD).
73. In JavaScript will be able to set parameters as dot properties and also via a getParameters method. Browser should also allow service-specific parameters to be set this way.
74. This email on continuous recognition represents our decisions regarding continuous recognition, except that there needs to be a feedback mechanism which could result in the service sending replaces. We may refer to "intermediate" as "partial", but naming changes such as this are TBD.
75. There will be an API method for sending text input rather than audio. There must also be a parameter to indicate how text matching should be done, including at least "strict" and "fuzzy". Other possible ways could be defined as vendor-specific additions.
76. It must be possible to do one or more re-recognitions with any request that you have indicated before first use that it can be re-recognized later. This will be indicated in the API by setting a parameter to indicate re-recognition. Any parameter can be changed, including the speech service.

77. In the protocol, the client must store the audio for re-recognition. It may be possible for the server to indicate that it also has stored the audio so it doesn't have to be resent.

    Original wording: In the protocol, the client must store the audio for re-recognition. It may be possible for the server to indicate that it also has stored the audio so it doesn't have to be resent.

78. Once there is a way (defined by another group) to get access to some blob of stored audio, we will support re-recognition of it.
79. No explicit need for JSON format of EMMA, but we might use it if it existed.
80. Candidate codecs to consider are Speex, FLAC, and Ogg Vorbis, in addition to plain old mu-law/a-law/linear PCM.
81. Protocol design should not prevent implementability of low-latency event delivery.
82. Protocol should support the client to begin TTS playback before receipt of all of the audio.
83. We will not require support for video codecs. However, protocol design must not prohibit transmission of codecs that have the same interface requirements as audio codecs.

84. Every event from speech service to the user agent must include timing information that the UA can convert into a UA-local timestamp. This timing info must be for the occurrence represented by the event, not the event time itself. For example, an end-of-speech event would contain timing for the actual end of speech, not the time when the speech service realizes end of speech occurred or when the event is sent.
85. This group will not define an explicit recording capability at this time. Existing use cases can be satisfied either via a recognizer's recording capability or via protocols defined outside this group.
86. There are use cases for separately preparing grammars before recognition. There must be a way for author to request grammar prep and get a notification back when prepared.
87. HTMLMediaElement has properties such as preload and buffer that we would like to expose for synthesis to inherit from if possible.
88. Must be a mechanism for client to send feedback to recognizer about recognition, even while recognition is ongoing.
89. Nomatch and noinput are not errors but other conditions
90. Top-level weight values are monotonically non-decreasing, even relative to each other. For example, if grammar A has weight 1.8 and grammar B has weight 1.9, grammar B cannot be weighted less by the recognizer than grammar A.
91. It must be possible to define at least the following handlers (names TBD):
    - *onspeechstart* (not yet clear precisely what start of speech means)
    - *onspeechend* (not yet clear precisely what end of speech means)
    - *onerror* (one or more handlers for errors)
    - a handler for when the recognition result is available

    Note: significant work is needed to get interoperability here.

7. Proposed Solutions

The following sections cover proposed solutions that this Incubator Group recommends. The proposed solutions represent the consensus of the group, except where clearly indicated that an impasse was reached.

7.1 Speech Web API proposal

This proposed API represents the web API for doing speech in HTML. This proposal is the HTML bindings and JS functions that sits on top of the protocol proposal [section 7.2 HTML Speech Protocol Proposal]. This includes:

- A declarative <reco> tag which can optionally bind to recoable elements.
- A JS API that allows programmatic control and parameterization of recognition.
- The ability to do recognition either as one-shot turns or as continuous dictation.
- The ability of the web application author to use the recognition service of their choice.
- A declarative <tts> tag which can be used to output synthesis.
- A JS API that allows programmatic control of the synthesis.

- The ability of the web application author to choose the synthesis service of their choice.

The section on Design Decisions [section 6. Solution Design Agreements and Alternatives] covers the design decisions the group agreed to that helped direct this API proposal.

The sections on Use Cases [section 4. Use Cases] and Requirements [section 5. Requirements] cover the motivation behind this proposal.

This API is designed to be used in conjunction with other APIs and elements on the web platform, including APIs to capture input and APIs to do bidirectional communications with a server (WebSockets [*WEBSOCKETS-API*]).

7.1.1 Introduction

Web applications should have the ability to use speech to interact with users. That speech could be for output through synthesized speech, or could be for input through the user speaking to fill form items, the user speaking to control page navigation or many other collected use cases. A web application author should be able to add speech to a web application using methods familiar to web developers and should not require extensive specialized speech expertise. The web application should build on existing W3C web standards and support a wide variety of use cases. The web application author should have the flexibility to control the recognition service the web application uses, but should not have the obligation of needing to support a service. This proposal defines the basic representations for how to use grammars, parameters, and recognition results and how to process them. The interfaces and API defined in this proposal can be used with other interfaces and APIs exposed to the web platform.

Note that privacy and security concerns exist around allowing web applications to do speech recognition. User agents should make sure that end users are aware that speech recognition is occurring, and that the end users have given informed consent for this to occur. The exact mechanism of consent is user agent specific, but the privacy and security concerns have shaped many aspects of the proposal.

7.1.2 Reco Element

The reco element is the way to do speech recognition using markup bindings. The reco element is legal where ever phrasing content is expected, and can contain any phrasing content, except with no descendant recoable elements unless it is the element's reco control, and no descendant reco elements.

IDL

```
[NamedConstructor=Reco(),
NamedConstructor=Reco(in DOMString for)]
  interface HTMLRecoElement : HTMLElement {
```

```
        // Attributes
        readonly attribute HTMLFormElement? form;
        attribute DOMString htmlFor;
        readonly attribute HTMLElement? control;
        attribute SpeechReco request;

        attribute DOMString grammar;

        // From the SpeechReco
        integer maxNBest;
        DOMString lang;
        boolean saveForRereco;
        boolean endpointDetection;
        boolean finalizeBeforeEnd;
        boolean interimResults;
        float confidenceThreshold;
        float sensitivity;
        float speedVsAccuracy;
        integer completeTimeout;
        integer incompleteTimeout;
        integer maxSpeechTimeout;
        DOMString inputWaveformURI;
        attribute DOMString serviceURI;
        attribute boolean continuous;

        // event handlers
        attribute Function onaudiostart;
        attribute Function onsoundstart;
        attribute Function onspeechstart;
        attribute Function onspeechend;
        attribute Function onsoundend;
        attribute Function onaudioend;
        attribute Function onresult;
        attribute Function onnomatch;
        attribute Function onresultdeleted;
        attribute Function onerror;
        attribute Function onauthorizationchange;
        attribute Function onopen;
        attribute Function onstart;
        attribute Function onend;
    };
```

The reco represents a speech input in a user interface. The speech input can be associated with a specific form control, known as the reco element's reco control, either using for attribute, or by putting the form control inside the reco element itself.

Except where otherwise specified by the following rules, a reco element has no reco control.

Some elements are catagorized as ***recoable elements***. These are elements that can be associated with a reco element:

button input keygen select textarea

The reco element's exact default presentation and behavior, in particular what its activation behavior might be is unspecified and user agent specific. When the reco element is bound to a recoable element if no grammar attribute is specified, then by

default the default builtin uri is used. The activation behavior of a reco element for events targeted at interactive content descendants of a reco element, and any descendants of those interactive content descendants, MUST be to do nothing. When a reco element with a reco control is activated and gets a reco result, the default action of the recognition event MUST be to use the value of the top n-best interpretation of the current result event. The exact binding depends on the recoable element in question and is covered in the binding results section.

Warning:
Not all implementers see value in linking the recognition behavior to markup, versus an all scripting API. Some user agents like the possibility of good defaults based on the associations. Some user agents like the idea of different consent bars based on the user clicking a markup button, rather then just relying on scripting. User agents are cautioned to remember click-jacking and SHOULD NOT automatically assume that when a reco element is activated it means the user meant to start recognition in all situations.

7.1.2.1 Reco Attributes

form

The form attribute is used to explicitly associate the reco element with its form owner.

The form IDL attribute is part of the element's forms API.

htmlFor

The htmlFor IDL attribute MUST reflect the ***for content attribute***.

The for attribute MAY be specified to indicate a form control with which a speech input is to be associated. If the attribute is specified, the attribute's value MUST be the ID of a recoable element in the same Document as the reco element. If the attribute is specified and there is an element in the Document whose ID is equal to the value of the for attribute, and the first such element is a recoable element, then that element is the reco element's reco control.

If the for attribute is not specified, but the reco element has a recoable element descendant, then the first such descendant in tree order is the reco element's reco control.

control

The control attribute returns the form control that is associated with this element. The control IDL attribute MUST return the reco element's reco control, if any, or null if there isn't one.

control . ***recos*** returns a NodeList of all the reco elements that the form control is associated with.

Recoable elements have a NodeList object associated with them that represents the list of reco elements, in tree order, whose reco control is the element in question. The reco IDL attribute of recoable elements, on getting, MUST return that NodeList object.

request
: The request attribute represents the SpeechReco associated with this reco element. By default the User Agent sets up the speech service specified by serviceURI and the default speech reco object associated with this reco. The author MAY set this attribute to associate a markup reco element with an author created speech reco object. In this way the author has control over the reco involved. When the reco request object is set then the object's speech parameters take priority over the corresponding parameters on the reco attributes.

grammar
: The uri of a grammar associated with this reco. If unset, this defaults to the default builtin uri. Note that to use multiple grammars or different weights the user must use the scripted SpeechReco API.

The other attributes are all defined identically to how they appear in the SpeechInputResult section.

7.1.2.2 Reco Constructors

Two constructors are provided for creating HTMLRecoElement objects (in addition to the factory methods from DOM Core such as createElement()): Reco() and Reco(for). When invoked as constructors, these MUST return a new HTMLRecoElement object (a new reco element). If the for argument is present, the object created MUST have its for content attribute set to the provided value. The element's document MUST be the active document of the browsing context of the Window object on which the interface object of the invoked constructor is found.

7.1.2.3 Builtin Default Grammars

When the user agent needs to create a default grammar from a recoable element it builds a uri using the builtin scheme. The format of the uri is of the form: `builtin:<tag-name>?<tag-attributes>` where the tag-name is just the name of the recoable element, and the tag-attributes are the content attributes in the form name=value&name=value. Since this a uri, both name and value must be properly uri escaped. Note the ? character may be omitted when there are no tag-attributes. For example:

- A simple textarea like `<textarea />` would produce: `builtin:textarea`
- A simple number like `<input type="number" max="3" />` would produce: `builtin:input?type=number&max=3`
- A zip code like `<input type="text" pattern="[0-9]{5}" />` would produce: `builtin:input?type=text&pattern=%5B0-9%5D%7B5%7D`

Speech services may define other builtin grammars as well. It is recommended that speech services allow a `builtin:dictation` to represents a "say anything" grammar and `builtin:websearch` to represent a speech web search and `builtin:default` to represent whatever default grammar, if any, the recognition service defines.

builtin uri for grammars can be used even when the reco element is not bound to any particular element and may also be used by the SpeechReco object and as a rule reference in an SRGS grammar.

In addition to the content attribute other parameters may be specified. It is recommended that speech services support a filter parameter that can be set to the value noOffensiveWords to represent a desire to not recognize offensive words. Speech services may define other extension parameters.

Note the exact grammar that is generated from any builtin uri is specific to the recognition service and the content attributes are best thought of as hints for the service.

7.1.2.4 Default Binding of Results

When a reco element is bound to a recoable element and does not have an onresult attribute set then the default binding is used. The default binding can also be used if the SpeechInputResult's outputElement attribute is set. In both cases the exact binding depends on the recoable element in question. In general, the binding will use the value associated with the interpretation of the top n-best element.

When the recoable element is a button then if the button is not disabled, then the result of a speech recognition is to activate the button.

When the recoable element is a input element then the exact binding depends on the control type. For basic text fields (input elements with a type attribute of text, search, tel, url, email, password, and number) the value of the result should be assigned to the input (inserted at the cursor and replacing any selected text). For button controls (submit, image, reset, button) the act of recognition just activates the input. For type checkbox, the input should be set to a checkedness of true. For type radiobutton, the input should be set to a checkedness of true, and all other inputs in the radio button group must be set to a checkedness of false. For date and time types (datetime, date, month, week, time, datetime-local) then the value should be assigned unless the value represents an non-empty string that is not valid for that type as described here. For type of color then the value should be assigned unless

the value does not represent a valid lowercase simple color. For type of range the assignment is only allowed if it is a valid floating point number, and before being assigned, it must undergo the value sanitization algorithm as described here.

When the recoable element is a keygen element then the element should regenerate the key.

When the recoable element is a select element then the recognition result will be used to select any options that are named the same as the interpretations value (that is any that are returned by namedItem(value)).

When the recoable element is a textarea element then the recognized value is inserted in to the textarea where the text cursor is if it is in the textarea. If text in the textarea is selected then the new value replaces the highlighted text. If the text cursor is not in the textarea then the value is appended to the end of the textarea.

Note:

More work is needed on making sure that this binding behavior is well specified and matches the input validation described in HTML 5. The general desire from the group is for binding to do a reasonable default behavior, for the additional attributes (type, pattern, max, etc.) to be available to possibly improve recognition, but for as much as possible those additional attributes to merely be hints - other then where the HTML 5 specification forbids user agents to assign certain elements certain values. For example, an input with type of Date can't take arbitrary non-empty strings, they can only take non-empty string that are valid date string, at an recognized string of "brown" wouldn't work). In addition, some would like the ability to both check selectors as well as uncheck them, but the exact semantics of how that work (does a default binding toggle the selected item? check it unless a magic phrase like "uncheck" is recognized before it? something else?) is not well understood, so no further effort to explain that is made here in this section.

7.1.3 TTS Element

The TTS element is the way to do speech synthesis using markup bindings. The TTS element is legal where embedded content is expected. If the TTS element has a src attribute, then its content model is zero or more track elements, then transparent, but with no media element descendants. If the element does not have a src attribute, then one or more source elements, then zero or more track elements, then transparent, but with no media element descendants.

IDL

```
[NamedConstructor=TTS(),
```

```
NamedConstructor=TTS(in DOMString src)]
  interface HTMLTTSElement : HTMLMediaElement {
      attribute DOMString serviceURI;
      attribute DOMString text;
      attribute DOMString lastMark;
  };
```

A TTS element represents a synthesized audio stream. A TTS element is a media element whose media data is ostensibly synthesized audio data.

When a TTS element is potentially playing, it must have its TTS data played synchronized with the current playback position, at the element's effective media volume.

When a TTS element is not potentially playing, TTS must not play for the element.

Content MAY be provided inside the TTS element. User agents SHOULD NOT show this content to the user; it is intended for older Web browsers which do not support TTS.

In particular, this content is not intended to address accessibility concerns. To make TTS content accessible to those with physical or cognitive disabilities, authors are expected to provide alternative media streams and/or to embed accessibility aids (such as transcriptions) into their media streams.

Implementations SHOULD support at least UTF-8 encoded text/plain and application/ssml+xml (both SSML 1.0 and 1.1 SHOULD be supported).

User Agents MUST pass the xml:lang values to synthesizers and synthesizers MUST use the passed in xml:lang to determine the language of text/plain.

The existing timeupdate event is dispatched to report progress through the synthesized speech. If the synthesis is of type application/ssml+xml, timeupdate events should be fired for each mark element that is encountered.

7.1.3.1 Attributes

The ***src***, preload, autoplay, mediagroup, loop, muted, and controls attributes are the attributes common to all media elements.

serviceURI

The serviceURI attribute specifies the speech service to use in the constructed default request. If the serviceURI is unset then the User Agent MUST use the User Agent default service. Note that the serviceURI is a generic URI and can thus point to local services either through use of a URN with meaning to the User Agent or by specifying a URL that the User Agent recognizes as a local

service. Additionally, the User Agent default can of course be local or remote and can incorporate end user choices via interfaces provided by the User Agent such as browser configuration parameters.

text

The text is an optional attribute to specify plain text to be synthesized. The text attribute, on setting MUST construct a data: URI that encodes the text to be played and assign that to the src property of the TTS element. If there are no encoding issues the URI for any given string would be `data:text/plain,string`, but if there are problematic characters the UserAgent should use base64 encoding.

lastMark

The new lastMark attribute must, on getting, return the name of the last SSML mark element that was encountered during playback. If no mark has been encountered yet, the attribute must return null.

There has been some interest in allowing lastMark to be set to a value as a way to control where playback would start/resume. This feature is simple in principle but may have subtle implementation consequences. It is not a feature that has been seriously studied in other similar languages such as SSML. Thus, the group decided not to consider this feature at this time.

7.1.3.2 Constructors

Two constructors are provided for creating HTMLTTSElement objects (in addition to the factory methods from DOM Core such as createElement()): TTS() and TTS(src). When invoked as constructors, these MUST return a new HTMLTTSElement object (a new tts element). The element MUST have its preload attribute set to the literal value "auto". If the src argument is present, the object created MUST have its src content attribute set to the provided value, and the user agent MUST invoke the object's resource selection algorithm before returning. The element's document MUST be the active document of the browsing context of the Window object on which the interface object of the invoked constructor is found.

7.1.4 The Speech Reco Interface

The speech reco interface is the scripted web API for controlling a given recognition.

IDL

```
[Constructor]
interface SpeechReco {
    // recognition parameters
    SpeechGrammarList grammars;

    // misc parameter attributes
    integer maxNBest;
```

```
    DOMString lang;
    boolean saveForRereco;
    boolean endpointDetection;
    boolean finalizeBeforeEnd;
    boolean interimResults;
    float confidenceThreshold;
    float sensitivity;
    float speedVsAccuracy;
    integer completeTimeout;
    integer incompleteTimeout;
    integer maxSpeechTimeout;
    DOMString inputWaveformURI;

    // the generic set of parameters
    SpeechParameterList parameters;
    void setCustomParameter(in DOMString name, in DOMString value);

    // other attributes
    attribute DOMString serviceURI;
    attribute MediaStream input;
    const unsigned short UNKNOWN = 0;
    const unsigned short AUTHROIZED = 1;
    const unsigned short NOT_AUTHORIZED = 2;
    readonly attribute unsigned short authorizationState;
    attribute boolean continuous;
    attribute Element outputElement;

    // the generic send info method
    void sendInfo(in DOMString type, in DOMString value);
    // methods to drive the speech interaction
    void open();
    void start();
    void stop();
    void abort();
    void interpret(in DOMString text);

    // event methods
    attribute Function onaudiostart;
    attribute Function onsoundstart;
    attribute Function onspeechstart;
    attribute Function onspeechend;
    attribute Function onsoundend;
    attribute Function onaudioend;
    attribute Function onresult;
    attribute Function onnomatch;
    attribute Function onresultdeleted;
    attribute Function onerror;
    attribute Function onauthorizationchange;
    attribute Function onopen;
    attribute Function onstart;
    attribute Function onend;
};
SpeechReco implements EventTarget;

interface SpeechInputNomatchEvent : Event {
    readonly attribute SpeechInputResult result;
};

interface SpeechInputErrorEvent : Event {
    readonly attribute SpeechInputError error;
};

interface SpeechInputError {
    const unsigned short OTHER = 0;
    const unsigned short NO_SPEECH = 1;
    const unsigned short ABORTED = 2;
```

```
    const unsigned short AUDIO_CAPTURE = 3;
    const unsigned short NETWORK = 4;
    const unsigned short NOT_ALLOWED = 5;
    const unsigned short SERVICE_NOT_ALLOWED = 6;
    const unsigned short BAD_GRAMMAR = 7;
    const unsigned short LANGUAGE_NOT_SUPPORTED = 8;

    readonly attribute unsigned short code;
    readonly attribute DOMString message;
};

// Item in N-best list
interface SpeechInputAlternative {
    readonly attribute DOMString transcript;
    readonly attribute float confidence;
    readonly attribute any interpretation;
};

// A complete one-shot simple response
interface SpeechInputResult {
    readonly attribute Document EMMAXML;
    readonly attribute DOMString EMMAText;
    readonly attribute unsigned long length;
    getter SpeechInputAlternative item(in unsigned long index);
    readonly attribute boolean final;
};

// A collection of responses (used in continuous mode)
interface SpeechInputResultList {
    readonly attribute unsigned long length;
    getter SpeechInputResult item(in unsigned long index);
};

// A full response, which could be interim or final, part of a continuous respo
interface SpeechInputResultEvent : Event {
    readonly attribute SpeechInputResult result;
    readonly attribute DOMString transcript;
    readonly attribute float confidence;
    readonly attribute any interpretation;
    readonly attribute short resultIndex;
    readonly attribute SpeechInputResultList resultHistory;
    readonly attribute DOMString sessionId;
};

// A full result deleted event, containing the updated history
interface SpeechInputResultDeletedEvent : Event {
    readonly attribute short resultIndex;
    readonly attribute SpeechInputResultList resultHistory;
    readonly attribute DOMString sessionId;
};

// The object representing a speech grammar
[Constructor]
interface SpeechGrammar {
    attribute DOMString src;
    attribute float weight;
};

// The object representing a speech grammar collection
[Constructor]
interface SpeechGrammarList {
    readonly attribute unsigned long length;
    getter SpeechGrammar item(in unsigned long index);

    // Default markup binding methods
    void addFromElement(in Element inputElement, optional float weight);
```

```
        // grammar methods
        void addFromUri(in DOMString src,
                        optional float weight);
        void addFromString(in DOMString string,
                        optional float weight);
    };

    // The object representing a speech parameter
    [Constructor]
    interface SpeechParameter {
        attribute DOMString name;
        attribute DOMString value;
    };

    // The object representing a speech parameter collection
    [Constructor]
    interface SpeechParameterList {
        readonly attribute unsigned long length;
        getter SpeechParameter item(in unsigned long index);
    };
```

7.1.4.1 Speech Reco Attributes

***grammars* attribute**

The grammars attribute stores the collection of SpeechGrammar objects which represent the grammars that are active for this recognition.

***maxNBest* attribute**

This attribute will set the maximum number of recognition results that should be returned. The default value is 1.

***lang* attribute**

This attribute will set the language of the recognition for the request, using a valid BCP 47 language tag. If unset it remains unset for getting in script, but will default to use the lang of its recoable element, if tied to an html element, and the lang of the html document root element and associated hierachy are used when the SpeechReco is not associated with a recoable element. This default value is computed and used when the input request opens a connection to the recognition service.

***saveForRereco* attribute**

This attribute instructs the speech recognition service if the utterance should be saved for later use in a rerecognition (true means save). The default value is false.

***endpointDetection* attribute**

This attribute instructs the user agent if it should do a low latency endpoint detection (true means do endpointing). The user agent default SHOULD be true.

***finalizeBeforeEnd* attribute**

This attribute instructs the recognition service if it should send final results when it gets them, even if the user is not done talking (true means yes it should send the results early). The user agent default SHOULD be true.

***interimResults* attribute**

If interimResults is set to false, that instructs the recognition service that it MUST NOT send any interim results. A value of true represents a hint to the service that the web application would like interim results. The service MAY not follow the hint, as the ability to send interim results depends on a combination of the recognition service, the grammars in use, and the utterance being recognized. The user agent default value SHOULD be false.

***confidenceThreshold* attribute**

This attribute represents the degree of confidence the recognition system needs in order to return a recognition match instead of a nomatch. The confidence threshold is a value between 0.0 (least confidence needed) and 1.0 (most confidence) with 0.5 as the default.

***sensitivity* attribute**

This attribute represents the sensitivity to quiet input. The sensitivity is a value between 0.0 (least sensitive) and 1.0 (most sensitivity) with 0.5 as the default.

***speedVsAccuracy* attribute**

This attribute instructs the recognition service on the desired trade off between low latency and high speed. The speedVsAccuracy is a value between 0.0 (least accurate) and 1.0 (most accurate) with 0.5 as the default.

***completeTimeout* attribute**

This attribute represents the amount of silence, in milliseconds, needed to match a grammar when a hypothesis is at a complete match of the grammar (that is the hypothesis matches a grammar, and no larger input can possibly match a grammar).

***incompleteTimeout* attribute**

This attribute represents the amount of silence, in milliseconds, needed to match a grammar when a hypothesis is not at a complete match of the grammar (that is the hypothesis does not match a grammar, or it does match a grammar but so could a larger input).

***maxSpeechTimeout* attribute**

This attribute represents how much speech, in milliseconds, the recognition service should process before an end of speech or an error occurs.

***inputWaveformURI* attribute**

This attribute, if set, instructs the speech recognition service to recognize from this URI instead of from the input MediaStream attribute.

***parameters* attribute**

This attribute holds an array of arbitrary extension parameters. These parameters could set user specific information (such as profile, gender, or age information) or could be used to set recognition parameters specific to the recognition service in use.

***serviceURI* attribute**

The serviceURI attribute specifies the location of the speech service the web application wishes to connect to. If this attribute is unset at the time of the open call, then the user agent MUST use the user agent default speech service.

***input* attribute**

The input attribute is the MediaStream that we are recognizing against. If input is not set, the speech reco object uses the default UA provided capture (which MAY be nothing), in which case the value of input will be null. In cases where the MediaStream is set but the SpeechReco hasn't yet called start the User Agent SHOULD NOT buffer the audio, the semantics are that the web application wants to start listening to the Media Stream at the moment it calls Start, and not earlier than that.

***authorizationState* attribute**

The authorizationState variable tracks if the web application is authorized to do speech recognition. The UA SHOULD start in UNKNOWN if the user agent can not determine if the web application is able to be authorized. The state variable may change values in response to policies of the user agent and possibly security interactions with the end user. If the web application is authorized then the user agent MUST set this variable to AUTHORIZED. If the web application is not authorized then the user agent MUST set this variable to NOT_AUTHORIZED. Any time this state variable changes in value the user agent MUST raise a authorizationchange event.

***continuous* attribute**

When the continuous attribute is set to false the service MUST only return a single simple recognition response as a result of starting recognition. This represents a request/response single turn pattern of interaction. When the continuous attribute is set to true the service MUST return a set of recognitions representing more a dictation of multiple recognitions in response to a single starting of recognition. The user agent default value SHOULD be false.

***outputElement* attribute**

This attribute, if set, causes default binding of recognition matches to the recoable element it is set to. This is the same default binding that occurs when a <reco> element is bound as described at Default Binding of Results section.

7.1.4.2 Speech Reco Methods

The *setCustomParameter* method

This method appends an arbitrary recognition service parameter to the parameters array. The name of the parameter is given by the name parameter and the value by the value parameter. This arbitrary parameter mechanism allows services that want to have extensions or to set user specific information (such as profile, gender, or age information) to accomplish the task.

The *sendInfo* method

The method allows one to pass arbitrary information to the recognition service, even while recognition is on going. Each set info call gets transmitted immediately to the recognition service. The ***type*** attribute specifies the content-type of the info message and the ***value*** attribute specifies the payload of the info method.

The *open* method

When the open method is called the user agent MUST connect to the speech service. All of the attributes and parameters of the SpeechInputResult (I.e., languages, grammars, service uri, etc.) MUST be set before this method is called, because they will be fixed with the values they have at the time open is called, at least until open is called again. Note that the user agent MAY need to have a permissions dialog at this point to ensure that the end user has given informed consent for the web application to listen to the user and recognize. Errors MAY be raised at this point for a variety of reasons including: not authorized to do recognition, failure to connect to the service, the service can not handle the languages or grammars needed for this turn, etc. When the service is successfully completed the open with no errors the user agent MUST raise an open event.

The *start* method

When the start method is called it represents the moment in time the web application wishes to begin recognition. When the speech input is streaming live through the input media stream, then this start call represents the moment in time that the service MUST begin to listen and try to match the grammars associated with this request. If the SpeechReco has not yet called open before the start call is made, a call to open is made by the start call (complete with the open event being raised). Once the system is successfully listening to the recognition the user agent MUST raise a start event.

The *stop* method

The stop method represents an instruction to the recognition service to stop listening to more audio, and to try and return a result using just the audio that it has received to date. A typical use of the stop method might be for a web application where the end user is doing the end pointing, similar to a walkie-talkie. The end user might press and hold the space bar to talk to the system and on the space down press the start call would have occurred and when the space bar is released the stop method is called to ensure that the system is no

longer listening to the user. Once the stop method is called the speech service MUST NOT collect additional audio and MUST NOT continue to listen to the user. The speech service MUST attempt to return a recognition result (or a nomatch) based on the audio that it has collected to date.

The *abort* method

The abort method is a request to immediately stop listening and stop recognizing and do not return any information but that the system is done. When the stop method is called the speech service MUST stop recognizing. The user agent MUST raise a end event once the speech service is no longer connected.

The *interpret* method

The interpret method provides a mechanism to request recognition using text, rather than audio. The ***text*** parameter is the string of text to recognize against. When bypassing audio recognition a number of the normal parameters MAY be ignored and the sound and audio events SHOULD NOT be generated. Other normal SpeechReco events SHOULD be generated.

7.1.4.3 Speech Reco Events

The DOM Level 2 Event Model is used for speech recognition events. The methods in the EventTarget interface should be used for registering event listeners. The SpeechReco interface also contains convenience attributes for registering a single event handler for each event type.

For all these events, the timeStamp attribute defined in the DOM Level 2 Event interface must be set to the best possible estimate of when the real-world event which the event object represents occurred. This timestamp must be represented in the User Agent's view of time, even for events where the timestamps in question could be raised on a different machine like a remote recognition service (I.e., in a speechend event with a remote speech endpointer).

Unless specified below, the ordering of the different events is undefined. For example, some implementations may fire audioend before speechstart or speechend if the audio detector is client-side and the speech detector is server-side.

***audiostart* event**

Fired when the user agent has started to capture audio.

***soundstart* event**

Some sound, possibly speech, has been detected. This MUST be fired with low latency, e.g. by using a client-side energy detector.

***speechstart* event**

The speech that will be used for speech recognition has started.

***speechend* event**

The speech that will be used for speech recognition has ended. speechstart MUST always have been fire before speechend.

***soundend* event**

Some sound is no longer detected. This MUST be fired with low latency, e.g. by using a client-side energy detector. soundstart MUST always have been fired before soundend.

***audioend* event**

Fired when the user agent has finished capturing audio. audiostart MUST always have been fired before audioend.

***result* event**

Fired when the speech recognizer returns a result. See here for more information.

***nomatch* event**

Fired when the speech recognizer returns a final result with no recognition hypothesis that meet or exceed the confidence threshold. The result field in the event MAY contain speech recognition results that are below the confidence threshold or MAY be null.

***resultdeleted* event**

Fired when the recognizer needs to delete one of the previously returned interim results in a continuous recognition. In the protocol, this would be represented by an empty recognition complete. A simplified example of this might be the recognizer gives an interim result for "hot" as the zeroth index of the continuous result and then gives an interim result of "dog" as the first index. Later the recognize wants to give a final result that is just one word "hotdog". In order to do that it needs to change the zeroth index to "hotdog" and delete the first index. When the first element is deleted the response is the raising of a resultdeleted event. The resultIndex of this event will be the element that was deleted and the resultHistory will have the updated value.

***error* event**

Fired when a speech recognition error occurs. The error attribute MUST be set to a SpeechInputError object.

***authorizationchange* event**

Fired whenever the state variable tracking if the web application is authorized to listen to the user and do speech recognition changes its value.

***open* event**

Fired whenever the SpeechReco has successfully connected to the speech service and the various parameters of the request can be satisfied with the service.

***start* event**

Fired when the recognition service has begun to listen to the audio with the intention of recognizing.

***end* event**

Fired when the service has disconnected. The event MUST always be generated when the session ends no matter the reason for the end.

7.1.4.4 Speech Input Error

The speech input error object has two attributes `code` and `message`.

code

The code is a numeric error code for whatever has gone wrong. The values are:

***OTHER* (numeric code 0)**

This is the catch all error code.

***NO_SPEECH* (numeric code 1)**

No speech was detected.

***ABORTED* (numeric code 2)**

Speech input was aborted somehow, maybe by some UA-specific behavior such as UI that lets the user cancel speech input.

***AUDIO_CAPTURE* (numeric code 3)**

Audio capture failed.

***NETWORK* (numeric code 4)**

Some network communication that was required to complete the recognition failed.

***NOT_ALLOWED* (numeric code 5)**

The user agent is not allowing any speech input to occur for reasons of security, privacy or user preference.

***SERVICE_NOT_ALLOWED* (numeric code 6)**

The user agent is not allowing the web application requested speech service, but would allow some speech service, to be used either because the user agent doesn't support the selected one or because of reasons of security, privacy or user preference.

***BAD_GRAMMAR* (numeric code 7)**

There was an error in the speech recognition grammar.

***LANGUAGE_NOT_SUPPORTED* (numeric code 8)**

The language was not supported.

message
The message content is implementation specific. This attribute is primarily intended for debugging and developers should not use it directly in their application user interface.

7.1.4.5 Speech Input Alternative

The SpeechInputAlternative represents a simple view of the response that gets used in a n-best list. To see a more full view of how these simple view are derived from the raw emma see the Mapping EMMA to Speech Input Alternatives section.

transcript
The transcript string represents the raw words that the user spoke.

confidence
The confidence represents a numeric estimate between 0 and 1 of how confident the recognition system is that the recognition is correct. A higher number means the system is more confident.

interpretation
The interpretation represents the semantic meaning from what the user said. This might be determined, for instance, through the SISR specification of semantics in a grammar.

7.1.4.6 Speech Input Result

The SpeechInputResult object represents a single one-shot recognition match, either as one small part of a continuous recognition or as the complete return result of a non-continuous recognition.

EMMAXML
The EMMAXML is a Document that contains the complete EMMA [*EMMA*] document the recognition service returned from the recognition. The Document has all the normal XML DOM processing to inspect the content.

EMMAText
The EMMAText is a text representation of the EMMAXML.

length
The long attribute represents how many n-best alternatives are represented in the item array. The user agent MUST not return more SpeechInputAlternatives than the value of the maxNBest attribute on the recognition request.

item
The item getter returns a SpeechInputAlternative from the index into an array of n-best values. The user agent MUST ensure that there are not more elements in

the array then the maxNBest attribute was set. The user agent MUST ensure that the length attribute is set to the number of elements in the array. The user agent MUST ensure that the n-best list is sorted in non-increasing confidence order (each element must be less than or equal to the confidence of the preceding elements).

final

The final boolean MUST be set to true if this is the final time the speech service will return this particular index value. If the value is false, then this represents an interim result that could still be changed.

7.1.4.7 Speech Input Result List

The SpeechInputResultList object holds a sequence of recognition results representing the complete return result of a continuous recognition. For a non-continuous recognition it will hold only a single value.

length

The length attribute indicates how many results are represented in the item array.

item

The item getter returns a SpeechInputResult from the index into an array of result values. The user agent MUST ensure that the length attribute is set to the number of elements in the array.

7.1.4.8 Speech Input Result Event

The Speech Input Result event is the event that is raised each time there is an interim or final result. The event contains both the current most recent recognized bit (in the result object) as well as a history of the complete recognition session so far (in the results object).

result

The result element is the one single SpeechInputResult that is new as of this request.

transcript

This attribute has the same value as result.item(0).transcript and is provided as a convenience for the developer.

confidence

This attribute has the same value as result.item(0).confidence and is provided as a convenience for the developer.

interpretation

This attribute has the same value as result.item(0).interpretation and is provided as a convenience for the developer.

resultIndex

The resultIndex MUST be set to the place in the results array that this particular new result goes. The resultIndex MAY refer to a previous occupied array index from a previous SpeechInputResultEvent. When this is the case this new result overwrites the earlier result and is a more accurate result; however, when this is the case the previous value MUST NOT have been a final result. When continuous was false, the resultIndex MUST always be 0.

resultHistory

The array of all of the recognition results that have been returned as part of this session. This array MUST be identical to the array that was present when the last SpeechInputResultEvent was raised, with the exception of the new result value.

sessionId

The sessionId is a unique identifier of this SpeechReco object that identifies the session. This id MAY be used to correlate logging and also as part of rerecognition.

7.1.4.9 Speech Input Result Deleted Event

The Speech Input Result Deleted event is raised each time an interim result is removed from the result history, without a new result to take its place.

resultIndex

The resultIndex MUST be set to the place in the results array that this particular new result goes. The resultIndex MAY refer to a previous occupied array index from a previous SpeechInputResultEvent. When this is the case this new result overwrites the earlier result and is a more accurate result; however, when this is the case the previous value MUST NOT have been a final result. When continuous was false, the resultIndex MUST always be 0.

resultHistory

The array of all of the recognition results that have been returned as part of this session. This array MUST be identical to the array that was present when the last SpeechInputResultEvent was raised, with the exception of the new result value.

sessionId

The sessionId is a unique identifier of this SpeechReco object that identifies the session. This id MAY be used to correlate logging and also as part of rerecognition.

7.1.4.10 Speech Grammar Interface

The SpeechGrammar object represents a container for a grammar. This structure has the following attributes:

***src* attribute**

The required src attribute is the URI for the grammar. Note some services may support builtin grammars that can be specified using a builtin URI scheme.

***weight* attribute**

The optional weight attribute controls the weight that the speech recognition service should use with this grammar. By default, a grammar has a weight of 1. Larger weight values positively weight the grammar while smaller weight values make the grammar weighted less strongly.

7.1.4.11 Speech Grammar List Interface

The SpeechGrammarList object represents a collection of SpeechGrammar objects. This structure has the following attributes:

length

The length attribute represents how many grammars are currently in the array.

item

The item getter returns a SpeechGrammar from the index into an array of grammars. The user agent MUST ensure that the length attribute is set to the number of elements in the array. The user agent MUST ensure that the index order from smallest to largest matches the order in which grammars were added to the array.

The *addFromElement* method

This method allows one to create a builtin grammar uri from a recoable element as outlined in the builtin uri description. This grammar is then appended to the grammars array parameter. The element in question is provided by the ***inputElement*** attribute. The optional ***weight*** attribute sets the corresponding attribute value in the created SpeechGrammar object.

The *addFromURI* method

This method appends a grammar to the grammars array parameter based on URI. The URI for the grammar is specified by the src parameter, which represents the URI for the grammar. Note, some services may support builtin grammars that can be specified by URI. If the weight parameter is present it represents this grammar's weight relative to the other grammar. If the weight parameter is not present, the default value of 1.0 is used.

The *addFromString* method

This method appends a grammar to the grammars array parameter based on text. The content of the grammar is specified by the ***string*** parameter, which represents the content for the grammar. This content should be encoded into a data: URI when the SpeechGrammar object is created. If the weight parameter is present it represents this grammar's weight relative to the other grammar. If the weight parameter is not present, the default value of 1.0 is used.

7.1.4.12 Speech Parameter Interface

The SpeechParameter object represents the container for arbitrary name/value parameters. This extensible mechanism allows developers to take advantage of extensions that recognition services may allow.

***name* attribute**
The required name attribute is the name of the custom parameter.

***value* attribute**
The required value attribute is the value of the custom parameter.

7.1.4.13 Speech Parameter List Interface

The SpeechParameterList object represents a collection of SpeechParameter objects. This structure has the following attributes:

length
The length attribute represents how many parameters are currently in the array.

item
The item getter returns a SpeechParameter from the index into an array of parameters. The user agent MUST ensure that the length attribute is set to the number of elements in the array.

7.1.5 Mapping EMMA to Speech Input Alternatives

An EMMA document represents the users input and may contain a number of possible hypothesis of what the user said. Each individual hypothesis is represented by a emma:interpretation. In the case where there is only one hypothesis than the emma:interpretation may be a direct child of the emma:emma root element. In cases where there are one or more hypothesis, the emma:interpretation tags may be included as children of a emma:one-of tag which is a child of the root emma:emma element.

Each emma:interpretation in turn contains information about the transcript, confidence, and interpretation for that one hypothesis. The interpretation information is given in EMMA by the instance data representing the semantics of what the user meant. In the case that the instance data is in an emma:literal then the interpretation

variable of the Speech Input Alternative MUST be set to the contents of the emma:literal tag. In the case where the instance data is given by an XML structure then the interpretation MUST be set to the XML document created when you wrap the instance data in a root result element. This Document has all the normal XML DOM processing to inspect the content.

Each emma:interpretation also contains the transcript of what the user said, represented by the emma:tokens attribute. The transcript variable of the Speech Input Alternative MUST be set to the contents of this attribute.

Each emma:interpreation also contains the confidence the recognizer has in this hypothesis, represented by the emma:confidence attribute. The confidence variable of the Speech Input Alternative MUST be set to this value.

Note that while these mapping cover the common usecases, the full EMMA document is available for inspection when needed through the XML and text properties on the SpeechInputResult object.

7.1.5.1 EMMA Mapping Example

Here is an example that shows how this mapping works.

Example

In the example below the different recognition results are mapped to a result set.

EMMA Document

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns="http://www.example.com/example">
  <emma:one-of id="r1" emma:medium="acoustic" emma:mode="voice">
    <emma:interpretation id="int1" emma:confidence="0.8" emma:token
      <emma:literal>BOS</emma:literal>
    </emma:interpretation>

    <emma:interpretation id="int2" emma:confidence="0.5" emma:token
      <origin>AUS</origin>
      <destination>DEN</destination>
      <date>03112003</date>
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

As a result of getting the above EMMA a SpeechInputResultEvent will be generated. The complete EMMA and n-best list will be accessible

from the result property in the form of the SpeechInputResult object. But the top level mapping is as follows.

SpeechInputResultEvent top level mappings

```
{transcript:"Boston Flight 311",
 confidence:0.8,
 interpretation:"BOS",
 ...}
```

The SpeechInputResult object that is generated has two SpeechInputAlternative in the item collection. The one at the 0th index (the best result) has the same transcript, confidence, and interpretation as the above top level mappings. The SpeechInputAlternative at the 1st index (the 2nd best result) is associated with the second emma:interpretation from the EMMA and thus has a transcript of "Austin to Denver Flight on 3 11" with a confidence of 0.5 and the interpretation is the XML Document, complete with all the normal DOM parsing, that is represented below.

2nd best SpeechInputAlternative Interpretation XML Document

```
<result>
  <origin>AUS</origin>
  <destination>DEN</destination>
  <date>03112003</date>
</result>
```

7.1.6 Extension Events

Some speech services may want to raise custom extension interim events either while doing speech recognition or while synthesizing audio. An example of this kind of event might be viseme events that encode lip and mouth positions while speech is being synthesized that can help with the creation of avatars and animation. These extension events MUST begin with "speech-x", so the hypothetical viseme event might be something like "speech-x-viseme".

7.1.7 Examples

Example

In the example below the various speech APIs are used to do basic speech web search.

Speech Web Search Markup Only

Speech Web Search Markup Only

```
<!DOCTYPE html>
  <html>
    <head>
      <title>Example Speech Web Search Markup Only</title>
    </head>
    <body>
      <form id="f" action="/search" method="GET">
        <label for="q">Search</label>
        <reco for="q"/>
        <input id="q" name="q" type="text"/>
        <input type="submit" value="Example Search"/>
      </form>
    </body>
  </html>
```

Speech Web Search JS API With Functional Binding

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example Speech Web Search JS API and Bindings</title>
  </head>
  <body>
    <script type="text/javascript">
      function speechClick() {
        var inputField = document.getElementById('q');
        var sr = new SpeechReco();
        sr.addGrammarFrom(inputField);
        sr.outputToElement(inputField);
        // Set what ever other parameters you want on the sr
        sr.serviceURI = "https://example.org/recoService";
        sr.speedVsAccuracy = 0.75;
        sr.start();
      }
    </script>
    <form id="f" action="/search" method="GET">
      <label for="q">Search</label>
      <input id="q" name="q" type="text" />
      <button name="mic" onclick="speechClick()">
        <img src="http://www.openclipart.org/image/15px/svg_to_png/
      </button>
      <br />
      <input type="submit" value="Example Search" />
    </form>
  </body>
</html>
```

Speech Web Search JS API Only

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example Speech Web Search</title>
  </head>
  <body>
    <script type="text/javascript">
      function speechClick() {
```

```
        var sr = new SpeechReco();

        // Build grammars from scratch
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("http://example.org/topChoices.srgs"
        sr.grammars.addFromUri("builtin:input?type=text", 0.5);
        sr.grammars.addFromUri("builtin:websearch", 1.1);
        // This 3rd grammar is an inline version of the http://www.
        sr.grammars.addFromUri("data:application/srgs+xml;base64,PG
                 "hlbWEtaW5zdGFuY2UiIA0KICAgICAgICAgeHNpOnNjaGVtYUx
                 "3JnL1RSL3NwZWVjaC1ncmFtbWFyL2dyYW1tYXIueHNkIg0KIC
                 "IHNjb3BlPSJwdWJsaWMiPg0KICAgICA8b25lLW9mPg0KICAgI
                 "gIDwvb25lLW9mPg0KICAgPC9ydWxlPg0KDQogICA8cnVsZSBp
                 "ggRGFrb3RhPC9pdGVtPg0KICAgICAgIDxpdGVtPk5ldyBZb3J
                 "nVsZXJlZiB1cmk9IiNjaXR5Ii8+IDxydWxlcmVmIHVyaT0iI3
```

```
        // Say what happens on a match
        sr.onresult = function(event) {
          var q = document.getElementById('q');
          q.value = event.result.item(0).interpretation;
          var f = document.getElementById('f');
          f.submit();
        };

        // Also do something on a nomatch
        sr.onnomatch() = function(event) {
          // even though it is a no match we might have a result
          alert("no match: " + event.result.item(0).interpretation)
        }

        // Set what ever other parameters you want on the sr
        sr.serviceURI = "https://example.org/recoService";
        sr.speedVsAccuracy = 0.75;

        // Start will call open for us, if we wanted to open the sr
        sr.start();
      }
    </script>
    <form id="f" action="/search" method="GET">
      <label for="q">Search</label>
      <input id="q" name="q" type="text" />
      <button name="mic" onclick="speechClick()">
        <img src="http://www.openclipart.org/image/15px/svg_to_png/
      </button>
      <br />
      <input type="submit" value="Example Search" />
    </form>
  </body>
</html>
```

Web search by voice, with auto-submit

```
<script type="text/javascript">
     function startSpeech(event) {
       var sr = new SpeechReco();
       sr.onresult = handleSpeechInput;
       sr.start();
     }
     function handleSpeechInput(event) {
       var q = document.getElementById("q");
       q.value = event.result.item(0).interpretation;
       q.form.submit();
```

```
    }
  </script>

  <form action="http://www.example.com/search">
  <input type="search" id="q" name="q">
  <input type="button" value="Speak" onclick="startSpeech">
  </form>
```

Behavior

1. User clicks button.
2. Audio is captured and the speech recognizer runs.
3. If some speech was recognized, the first hypothesis is put in the text field and the form is submitted.
4. Search results are loaded.

Web search by voice, with "Did you say..."

This example uses the second best result. The search results page will display a link with the text "Did you say $second_best?".

```
<script type="text/javascript">
      function startSpeech(event) {
        var sr = new SpeechReco();
        sr.onresult = handleSpeechInput;
        sr.start();
      }
      function handleSpeechInput(event) {
        var q = document.getElementById("q");
        q.value = event.result.item(0).interpretation;

        if (event.result.length > 1) {
          var second = event.result[1].interpretation;
          document.getElementById("second_best").value = second;
        }

        q.form.submit();
      }
  </script>

  <form action="http://www.example.com/search">
  <input type="search" id="q" name="q">
  <input type="button" value="Speak" onclick="startSpeech">
  <input type="hidden" name="second_best" id="second_best">
  </form>
```

Speech translator

```
<script type="text/javascript" src="http://www.example.com/jsapi"><
  <script type="text/javascript">
    library.load("language", "1");  // Load the translator JS lib

    // These will most likely be set in a UI.
    var fromLang = "en";
```

```
      var toLang = "es";

      function startSpeech(event) {
        var sr = new SpeechReco();
        sr.onresult = handleSpeechInput;
        sr.start();
      }
      function handleSpeechInput(event) {
        var text = event.result.item(0).interpretation;
        var callback = function(translationResult) {
          if (translationResult.translation)
            speak(translationResult.translation, toLang);
        };
        library.language.translate(text, fromLang, toLang, callback
      }

      function speak(output, lang) {
        var tts = new TTS();
        // NOTE: these attributes don't seem to be in the proposal
        tts.text = output;
        tts.lang = lang;
        tts.play();
      }
    </script>

    <form>
    <input type="button" value="Speak" onclick="startSpeech">
    </form>
```

Behavior

1. User clicks button and speaks in English.
2. System recognizes the text in English.
3. A web service translates the text from English to Spanish.
4. System synthesizes and speaks the translated text in Spanish.

Speech shell

HTML:

```
<script type="text/javascript">
      function startSpeech(event) {
        var sr = new SpeechReco();
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("commands.grxml");
        sr.onresult = handleSpeechInput;
        sr.start();
      }

      function handleSpeechInput(event) {
        var command = event.result.item(0).interpretation;
        if (command.action == "call_contact") {
          var number = getContactNumber(command.contact);
          callNumber(number);
        } else if (command.action == "call_number") {
```

```
        } else if (command.action == "call_number") {
          callNumber(command.number);
        } else if (command.action == "calculate") {
          say(evaluate(command.expression));
        } else if {command.action == "search") {
          search(command.query);
        }
      }
      function callNumber(number) {
        window.location = "tel:" + number;
      }
      function search(query) {
        // Start web search for query.
      }
      function getContactNumber(contact) {
        // Get the phone number of the contact.
      }
      function say(text) {
        // Speak text.
      }
    </script>

    <form>
    <input type="button" value="Speak" onclick="startSpeech">
    </form>
```

English SRGS XML Grammar (commands.grxml):

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
                      "http://www.w3.org/TR/speech-grammar/grammar.
    <grammar xmlns="http://www.w3.org/2001/06/grammar" xml:lang="en
             version="1.0" mode="voice" root="command"
             tag-format="semantics/1.0">

    <rule id="command" scope="public">
      <example> call Bob </example>
      <example> calculate 4 plus 3 </example>
      <example> search for pictures of the Golden Gate bridge </exa

      <one-of>
        <item>
          call <ruleref uri="contacts.grxml">
          <tag>out.action="call_contact"; out.contact=rules.latest(
        </item>
        <item>
          call <ruleref uri="phonenumber.grxml">
          <tag>out.action="call_number"; out.number=rules.latest()<
        </item>
        <item>
          calculate <ruleref uri="#expression">
          <tag>out.action="calculate"; out.expression=rules.latest(
        </item>
        <item>
          search for <ruleref uri="http://grammar.example.com/searc
          <tag>out.action="search"; out.query=rules.latest()</tag>
        </item>
      </one-of>
    </rule>
    </grammar>
```

Turn-by-turn navigation

Turn-by-turn navigation

HTML:

```
<script type="text/javascript">
      var directions;

      function startSpeech(event) {
        var sr = new SpeechReco();
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("grammar-nav-en.grxml");
        sr.onresult = handleSpeechInput;
        sr.start();
      }

      function handleSpeechInput(event) {
        var command = event.result.item(0).interpretation;
        getDirections(command.destination, handleDirections);
        speakNextInstruction();
      }

      function getDirections(query, handleDirections) {
        // Get location, then get directions from server, pass to h
      }

      function handleDirections(newDirections) {
        directions = newDirections;

        // List for location changes and call speakNextInstruction(
        // when appropriate
      }

      function speakNextInstruction() {
        var instruction = directions.pop();
        var tts = new TTS();
        tts.text = instruction;
        tts.play();
     }
    </script>

    <form>
    <input type="button" value="Speak" onclick="startSpeech">
    </form>
```

English SRGS XML Grammar (grammar-nav-en.grxml):

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
                        "http://www.w3.org/TR/speech-grammar/grammar.
    <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-gramma
             version="1.0" mode="voice" root="nav_cmd"
             tag-format="semantics/1.0">

    <rule id="nav_cmd" scope="public">
      <example> navigate to 76 Buckingham Palace Road, London </exa
      <example> go to Phantom of the Opera </example>
      <item>
```

```
      <ruleref uri="#nav_action" />
      <ruleref uri="builtin:search" />
      <tag>out.action="navigate_to"; out.destination=rules.latest
    </item>
  </rule>

  <rule id="nav_action">
    <one-of>
      <item>navigate to</item>
      <item>go to</item>
    </one-of>
  </rule>

  </grammar>
```

Example

The examples below show a few ways to handle permissions using Speech API.

Hide possible graphical UI related to reco element if permission is denied

```
<!-- Anywhere in the page -->
<script>
  /** When the document has been loaded, iterate through all the
   *  reco elements and remove the ones which have request.authoriz
   *  NOT_AUTHORIZED. For the rest of the reco elements
   *  add authorizationchange event listener, which will remove the
   *  the event from DOM if its request.authorizationState changes
   */ NOT_AUTHORIZED.
  window.addEventListener("load",
    function() {
      var recos = document.getElementsByTagName("reco");
      for (var i = recos.length - 1; i >= 0; --i) {
        if (!removeNotAuthorizedReco(recos[i])) {
          recos[i].addEventListener("authorizationchange",
            function(evt) { removeNotAuthorizedReco(evt.target); })
        }
      }
    });

  function removeNotAuthorizedReco(reco) {
    if (reco.request.authorizationState ==
        SpeechReco.NOT_AUTHORIZED) {
      reco.parentNode.removeChild(reco);
      return true;
    }
    return false;
  }
</script>
```

Ask permission as soon as the page has loaded and if permission is granted activate SpeechRequest object whenever user clicks the page

```
<!-- Anywhere in the page -->
<script>
  var request; // Make sure the request object is kept alive.

  window.addEventListener("load",
    function () {
      request = new SpeechRequest();
      request.onresult =
        function(evt) {
          // Do something with the speech recognition result.
        }
      request.open(); // Trigger permission handling UI in the user

      document.addEventListener("click",
        function() {
          if (request.authorizationState ==
              SpeechReco.AUTHORIZED) {
            request.start();
          }
        });
    });

</script>
```

Example

Domain Specific Grammars Contingent on Earlier Inputs

A use case exists around collecting multiple domain specific inputs sequentially where the later inputs depend on the results of the earlier inputs. For instance, changing which cities are in a grammar of cities in response to the user saying in which state they are located. This seems trivial, assuming there are a variety of suitable grammars to choose from.

```
<input id="txtState" type="text"/>
    <button id="btnStateMic" type="button" onclick="stateMicClicked
      <img src="microphone.png"/>
    </button>

    <br/>

    <input id="txtCity" type="text"/>
    <button id="btnCityMic" type="button" onclick="cityMicClicked()
      <img src="microphone.png"/>
    </button>


    <script type="text/javascript">

    function stateMicClicked() {
        var sr = new SpeechReco();
```

```
        // add the grammar that contains all the states:
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("states.grxml");

        sr.onmatch = function (e) {
            // assume the grammar returns a standard string for eac
            // in which case we just need to get the interpretation
            document.getElementById("txtState").value =
            e.result.item(0).interpretation;
          }
        sr.start();
      }

      function cityMicClicked() {
        var sr = new SpeechReco();

        // The cities grammar depends on what state has been select
        // If state is blank, use major US cities
        // otherwise use cities in that state

        var state = document.getElementById("txtState").value;
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri(state ? "majorUScities.grxml" : "cit

        sr.onresult = function(e) {
          document.getElementById("txtState").value =
          e.result.item(0).interpretation;
          }
        sr.start();
      }
</script>
```

Rerecognition

Some sophisticated applications will re-use the same utterance in two or more recognitions turns in what appears to the user as one turn. For example, an application may ask "how may I help you?", to which the user responds "find me a round trip from New York to San Francisco on Monday morning, returning Friday afternoon". An initial recognition against a broad language model may be sufficient to understand that the user wants the "flight search" portion of the app. Rather than get the user to repeat themselves, the application will just re-use the existing utterance for the recognition on the flight search recognition.

```
<script type="text/javascript">
    function listenAndClassifyThenReco() {

        var sr = new SpeechReco();
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("broadClassifier.grxml");
```

```
        sr.onresult = function (e) {
            if ("broadClassifier.grxml" == sr.grammars[0].uri) {
                // Also, this switch is a little contrived. no reas
                // wouldn't just be returned in the interpretation
                switch (getClassification(e.interpretation)) {
                    case "flightsearch":
                        sr.grammars = new SpeechGrammarList();
                        sr.grammars.addFromUri("flightsearch.grxml"
                        break;
                    case "hotelbooking":
                        sr.grammars = new SpeechGrammarList();
                        sr.grammars.addFromUri("hotelbooking.grxml"
                        break;
                    case "carrental":
                        sr.grammars = new SpeechGrammarList();
                        sr.grammars.addFromUri("carrental.grxml");
                        break;
                    default:
                        throw ("cannot interpret:" + e.result.item(
                }
                sr.inputWaveformUri = getUri(e); // This probably c
                // re-initialize with new settings:
                sr.open();
                // start listening:
                sr.start();
            }
            else if ("flightsearch.grxml" == sr.grammars[0].uri) {
                processFlightSearch(e.result.item(0).interpretation
            }
            else if ("hotelbooking.grxml" == sr.grammars[0].uri) {
                processHotelBooking(e.result.item(0).interpretation
            }
            else if ("carrental.grxml" == sr.grammars[0].uri) {
                processCarRental(e.result.item(0).interpretation);
            }
        }

        sr.saveForRereco = true;
        sr.start();
    }
</script>
```

Speech Enabled Email Client

The application reads out subjects and contents of email and also listens for commands, for instance, "archive", "reply: ok, let's meet at 2 pm", "forward to bob", "read message". Some commands may relate to VCR like controls of the message being read back, for instance, "pause", "skip forwards", "skip back", or "faster". Some of those controls may include controls related to parts of speech, such as, "repeat last sentence" or "next paragraph".

This is the other end of the spectrum. It's a huge app, if it works as described. But here's a simplistic version of some

of the logic:

```
<script type="text/javascript">
    var sr;
    var tts;

    // initialize the speech object when the page loads:
    function initializeSpeech() {
        sr = new SpeechReco();
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("globalcommands.grxml");
        sr.onresult = doSRResult;
        sr.onerror = doSRError;

        tts = new TTS();
    }

    function onMicClicked() {
        // stop TTS if there is any...
        if (tts.paused == false && tts.ended == false) {
            tts.controller.pause();
        }
        sr.start();
    }

    function doSRError(e) {
        // do something informative...
    }

    function doSRResult(e) {
        // I don't want to write this line. How do I avoid it?
        if (!e.result.final) {
            return;
        }

        // Assume the interpretation is an object with a
        // pile of properties that have been glommed together
        // by the grammar
        var semantics = e.result.item(0).interpretation;
        switch (semantics.command) {
            case "reply":
                composeReply(currentMessage, semantics.message)
                break;
            case "compose":
                composeNewMessage(semantics.subject, semantics.
                break;
            case "send":
                currentComposition.Send();
                break;
            case "readout":
                readout(currentMessage);
                break;
            default:
                throw ("cannot interpret:" + semantics.command)
        }
    }

    function readout(message) {
        tts.src = "data:,message from " + message.sendername +
        tts.play();
    }
```

From the multimodal use case "The ability to mix and integrate input from multiple modalities such as by saying 'I want to go from here to there' while tapping two points on a touch screen map."

Example

In the example below the various speech APIs are used to do a simple multimodal example where the user is presented with a series of buttons and may click one while they are speaking. The click event is sent to the speech service using the sendInfo method on the SpeechReco. The result coming back from the speech service is the integration of the voice command and gesture. The combined multimodal result is represented in the EMMA document along with the EMMA interpretations for the individual modalities.

Simple Multimodal Example JS API Only

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Multimodal Example</title>
  </head>
  <body>
    <script type="text/javascript">
      // Only send gestures if sr is active, start() has been calle
      sr_started = false;

      function speechClick() {
        var sr = new SpeechReco();

        // Indicate the reco service to be used
        sr.serviceURI = "https://example.org/recoService/multimodal

  // Set any parameters
        sr.speedVsAccuracy = 0.75;
        sr.completeTimeout = 2000;
        sr.incompleteTimeout = 3000;

        // Specifying the grammar
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("http://example.org/commands.srgs");

        // Say what happens on a match
        sr.onresult = function(event) {
          sr_started=false;
          var af = document.getElementById('asr_feedback');
          af.value = event.result.item(0).transcript;
          // code to pull out interpretation and execute;
          interp = event.result.item(0).interpretation;
          //  ....
        };

        // Also do something on a nomatch
        sr.onnomatch() = function(event) {
          // even though it is a no match we might have a result
          alert("no match: " + event.result.item(0).interpretation)
        }
```

```
        //

        // Start will call open for us, if we wanted to open the
        // sr on page start we could have to do initial permission
        sr.start();
        sr_started=true;
      };
    // What do if the user clicks a button during speech capture
    // Uses sendInfo to add the click to the stream sent to the ser
    function sendClick(payload) {
if (sr_started) {
sr.sendInfo('text/xml',payload);
      }
    };

    </script>

    <div>
    <input id="asr_feedback" name="asr_feedback" type="text"/>
    <button name="mic" onclick="speechClick()">SPEAK</button>
    <br/>
    <br/>
    <button name="item1" onclick="sendClick('<click>joe_bloggs</cli
    <button name="item2" onclick="sendClick('<click>pam_brown</clic
    <button name="item3" onclick="sendClick('<click>peter_smith</cl
    </div>

  </body>
</html>
```

SRGS Grammar (http://example.org/commands.srgs)

```
<?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 version="1.0" mode="voice" root="top" tag-format="semantic

<rule id="top" scope="public">
<one-of>
  <item>call this person<tag>out._cmd="CALL";out._obj="DEICTIC";</t
  <item>email this person<tag>out._cmd="CALL";out._obj="DEICTIC";</
  <item>call<tag>out._cmd="CALL";out._obj="DEICTIC";</tag></item>
  <item>email<tag>out._cmd="CALL";out._obj="DEICTIC";</tag></item>
</one-of>
</rule>
</grammar>
```

EMMA Returned

```
<emma:emma version="1.1"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
  <emma:grammar id="gram1" grammar-type="application/srgs-xml" ref=
  <emma:interpretation id="multimodal1"
      emma:confidence="0.6"
```

```
      emma:start="1087995961500"
      emma:end="1087995962542"
      emma:medium="acoustic tactile"
      emma:mode="voice gui"
      emma:function="dialog"
      emma:verbal="true"
      emma:lang="en-US"
      emma:result-format="application/json">
    <emma:derived-from resource="#voice1" composite="true"/>
    <emma:derived-from resource="#gui1" composite="true"/>
    <emma:literal><![CDATA[
              {_cmd:"CALL",
               _obj:"joe_bloggs"}]]></emma:literal>
  </emma:interpretation>
  <emma:derivation>
    <emma:interpretation id="voice1"
        emma:start="1087995961500"
        emma:end="1087995962542"
        emma:process="https://example.org/recoService/multimodal"
        emma:grammar-ref="gram1"
        emma:confidence="0.6"
        emma:medium="acoustic"
        emma:mode="voice"
        emma:function="dialog"
        emma:verbal="true"
        emma:lang="en-US"
        emma:tokens="call this person"
        emma:result-format="application/json">
       <emma:literal><![CDATA[
              {_cmd:"CALL",
               _obj:"DEICTIC"}]]></emma:literal>
    </emma:interpretation>
    <emma:interpretation id="gui1"
        emma:medium="tactile"
        emma:mode="gui"
        emma:function="dialog"
        emma:verbal="false">
        <click>joe_bloggs</click>
    </emma:interpretation>
  </emma:derivation>
</emma:emma>
```

Example

Speech XG Translating Example

This is a complete example that can be saved as an HTML document.

```
<html>
  <head>
    <title>Speech XG Translating Example</title>
  </head>
  <body>
    <script type="text/javascript" src="https://example.org/transla
    <script type="text/javascript">
      function speechClick() {
```

```
        var sr = new SpeechReco();
        sr.lang = document.getElementsById("inlang").value;
        sr.grammars = new SpeechGrammarList();
        sr.grammars.addFromUri("builtin:dictation");
        sr.serviceURI = "ws://example.org/reco";
        // Haven't had a good link with the capture stuff in exampl
        sr.input = document.getElementById("speak").capture;
        sr.onopen = function (event) { document.getElementById("p")
        sr.onstart = function (event) { document.getElementById("p"
        sr.onaudiostart = function (event) { document.getElementByI
        sr.onsoundstart = function (event) { document.getElementByI
        sr.onspeechstart = function (event) { document.getElementBy
        sr.onspeechend = function (event) { document.getElementById
        sr.onsoundend = function (event) { document.getElementById(
        sr.onaudioend = function (event) { document.getElementById(
        sr.onresult = readResult();
        sr.start();
      }
      function readResult(event) {
        document.getElementsById("p").value = 100;
        var tts = new TTS();
        tts.serviceURI = "ws://example.org/speak";
        tts.lang = document.getElementById("outlang").value;
        tts.src = "data:text/plain;" + translate(event.result.item(

      }
    </script>
    <h1>Speech XG Translating Example</h1>
    <form id="foo">
      <p>
        <label for="inlang">Spoken Language to Translate: </label>
        <select id="inlang" name="inlang">
          <option value="en-US">American English</option>
          <option value="en-GB">UK English</option>
          <option value="fr-CA">French Canadian</option>
          <option value="es-ES">Spanish</option>
        </select>
      </p>
      <p>
        <label>Translation Output Language:
          <select id="outlang" name="outlang">
            <option value="en-US">American English</option>
            <option value="en-GB">UK English</option>
            <option value="fr-CA" selected="true">French Canadian</
            <option value="es-ES">Spanish</option>
          </select>
        </label>
      </p>
      <p>
        <label for="speak">Click to speak: </label>
        <input type="button" name="speak" value="speak" capture="mi
      </p>
      <p>
        <progress id="p" max="100" value="0"/>
      </p>
    </form>
  </body>
</html>
```

Example

Multi-slot Filling Example

Multi-slot Filling Example

This is a complete example that can be saved as an HTML document.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="content-type"
              content="text/html; charset=ISO-8859-1">
        <title>Mult-slot filling</title>
        <script type="text/javascript">
            function speechClick() {
                var sr = new SpeechReco();
                //assume some multislot grammar coming from a URL
                sr.grammars = new SpeechGrammarList();
                sr.grammars.addFromUri("http://bookhotelspeechtools
                sr.start();
            }

            // on a match, extract the appropriate EMMA slots and p
            sr.onresult = function(event) {
                //get EMMA
                var emma = event.EMMAXML;
                //the author of this application will know the name
                //there is a root node called "booking" in the appl
                //there are four slots called "arrival", "departure
                var booking = emma.getElementsByTagName("booking").
                //get arrival
                var arrivalDate = getEMMAValue(booking, "arrival");
                arrival.value = arrivalDate;
                // get departure
                var departureDate = getEMMAValue(booking, "departur
                departure.value = departureDate;
                //get num adults (default is 1)
                var numAdults = getEMMAValue(booking, "adults");
                adults.value = numAdults;
                //get num children (default is 0)
                var numChildren = getEMMAValue(booking, "children")
                children.value = numChildren;
                //In a real application, before submitting, you wou
                //check that arrival and departure are filled, if e
                //missing, prompt for user to fill. This could be d
                //by initiating a directed dialog with slot-specifi
                The user should also be able to type values.
                    f.submit();
            };

            getEMMAValue(Node node String name){
                return node.getElementsByTagName(name).item(0).node'
            }

        }
        </script>
    </head>
    <body>
        <h1>Multi-slot Filling Example</h1>
        <h2>Welcome to the Speech Hotel!<br>
        </h2>
        <br>
        <form id="f" action="/search" method="GET"> Please tell us
            and departure dates, and how many adults and children w
            be staying with us. <br>
```

```
            For example "I want a room from December 1 to December
            adults" <br>
            Click the microphone to start speaking. <button name="m
                                                              onclick
                    src="http://www.openclipart.org/image/15px/svg_
                    alt="microphone picture"> </button> <br>
            <input id="arrival" name="arrival" type="text"> <br>
            <label for="arrival">Arrival Date</label> <br>
            <br>
            <input id="departure" name="departure" type="text"> <br
            <label for="departure">Departure Date</label> <br>
            <br>
            <input id="adults" name="adults" value="1" type="text">
            <label for="adults">Adults</label> <br>
            <br>
            <input id="children" name="children" value="0" type="te
            <label for="children">Children (under age 12)</label> <
            <br>
            <input value="Check Availability" type="submit"></form>
    </body>
</html>
```

Example

These are inline grammar examples

Referencing external grammar

We could write the grammar to a file and then reference the grammar via URI. Then we end up with file overhead and temporary files.

```
<form>
 <reco grammar="http://example.com/rosternames.xml">
   <input type="text"/>
 </reco>
</form>
```

Inline using data URI

Note that whitespace is only allowed with base64, so we must have a really long string.

```
<form>
 <reco grammar='data:text/html;charset=utf-8, <?xml version="1.0"
encoding="UTF-8"?>\r\n<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR
1.0//EN"\r\n
"http://www.w3.org/TR/speech-grammar/grammar.dtd">\r\n<grammar
xmlns="http://www.w3.org/2001/06/grammar"
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" xml:lang="en-US"
version="1.0" root="roster"><meta name="help-hint" content="player
scope="public"><example>Axel</example><example>Axel Eric and
Ondrej</example><item repeat="1-"><ruleref uri="#players"/></item><
uri="#players"/></item></rule><rule id="players"
scope="private"><one-of><item>David<tag>David
</tag></item><item>Ondrej<tag>Ondrej </tag></item><item>Eric<tag>Er
</tag></item><item>Kasraa<tag>Kasraa </tag></item><item>Axel<tag>Ax
</tag></item><item>Marcus<tag>Marcus </tag></item><!-- and so on up
names --></one-of></rule></grammar>'

   <input type="text"/>
 </reco>
</form>
```

Inline grammar

Note the following example is not supported by the proposal since we do not support inline grammars in the html content. However, some in the group wished this to be considered in the future. This examlpes shows what this would be like.

Much easier to read. More easily supports server-side scripting to plug in names (at least for humans while developing). Note that the <reco> element is the parent of both the <input> and <grammar> elements.

```
<form>
 <reco>
   <input type="text"/>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
        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"
        xml:lang="en-US" version="1.0"
        root="roster">
 <meta name="help-hint" content="player names"/>
 <rule id="roster" scope="public">
   <example>Axel</example>

   <example>Axel Eric and Ondrej</example>
   <item repeat="1-"><ruleref uri="#players"/></item>
   <item repeat="0-1">
     and
     <ruleref uri="#players"/>
   </item>
 </rule>
 <rule id="players" scope="private">
   <one-of>
     <item>David<tag>David </tag></item>
     <item>Ondrej<tag>Ondrej </tag></item>
     <item>Eric<tag>Eric </tag></item>
     <item>Kasraa<tag>Kasraa </tag></item>
     <item>Axel<tag>Axel </tag></item>
     <item>Marcus<tag>Marcus </tag></item>
```

```
      <!-- and so on up to 18 names -->
    </one-of>
  </rule>
</grammar>
 </reco>
</form>
```

7.2 HTML Speech Protocol Proposal

Multimodal interfaces enable users to interact with web applications using multiple different modalities. The HTML Speech protocol, and associated HTML Speech API, are designed to enable speech modalities as part of a common multimodal user experience combining spoken and graphical interaction across browsers. The specific goal of the HTML Speech protocol is to enable a web application to utilize the same network-based speech resources regardless of the browser used to render the application. The HTML Speech protocol is defined as a sub-protocol of WebSockets [*WEBSOCKETS-PROTOCOL*], and enables HTML user agents and applications to make interoperable use of network-based speech service providers, such that applications can use the service providers of their choice, regardless of the particular user agent the application is running in. The protocol bears some similarity to [*MRCPv2*] where it makes sense to borrow from that prior art. However, since the use cases for HTML Speech applications are in many cases considerably different from those around which MRCPv2 was designed, the HTML Speech protocol is not a direct transcript of MRCP. Similarly, because the HTML Speech protocol builds on WebSockets, its session negotiation and media transport needs are quite different from those of MRCP.

7.2.1 Architecture

7.2.2 Definitions

Recognizer

Because continuous recognition plays an important role in HTML Speech scenarios, a Recognizer is a resource that essentially acts as a filter on its input streams. Its grammars/language models can be specified and changed, as needed by the application, and the recognizer adapts its processing accordingly.

Single-shot recognition (e.g. a user on a web search page presses a button and utters a single web-search query) is a special case of this general pattern, where the application specifies its model once, and is only interested in one match event, after which it stops sending audio (if it hasn't already).

A Recognizer performs speech recognition, with the following characteristics:

1. Support for one or more spoken languages and acoustic scenarios.
2. Processing of one or more input streams. The typical scenario consists of a single stream of encoded audio. But some scenarios will involve multiple audio streams, such as multiple beams from an array microphone picking up different speakers in a room; or streams of multimodal input such as gesture or motion, in addition to speech.
3. Support for multiple simultaneous grammars/language models, including but not limited to application/srgs+xml [*SPEECH-GRAMMAR*]. Implementations MAY support additional formats, such as ABNF SRGS or an SLM format.
4. Support for continuous recognition, generating events as appropriate such as match/no-match, detection of the start/end of speech, etc.
5. Support for at least one "dictation" language model, enabling essentially unconstrained spoken input by the user.
6. Support for "hotword" recognition, where the recognizer ignores speech that is out of grammar. This is particularly useful for open-mic scenarios.
7. Support for recognition of media delivered slower than real-time, since network conditions can and will introduce delays in the delivery of media.

"Recognizers" are not strictly required to perform speech recognition, and may perform additional or alternative functions, such as speaker verification, emotion detection, or audio recording.

Synthesizer

A Synthesizer generates audio streams from textual input. It essentially produces a media stream with additional events, which the user agent buffers and plays back as required by the application. A Synthesizer service has the following characteristics:

1. Rendering of application/ssml+xml [*SPEECH-SYNTHESIS*] or text/plain input to an output audio stream. Implementations MAY support additional input text formats.
2. Each synthesis request results in a separate output stream that is terminated once rendering is complete, or if it has been canceled by the client.
3. Rendering must be performed and transmitted at least as rapidly as would be needed to support real-time playback, and preferably faster. In some cases, network conditions between service and UA may result in slower-

than-real-time delivery of the stream, and the UA or application will need to cope with this appropriately.

4. Generation of interim events, such as those corresponding to SSML marks, with precise timing. Events are transmitted by the service as closely as possible to the corresponding audio packet, to enable real-time playback by the client if required by the application.
5. Multiple Synthesis requests may be issued concurrently rather than queued serially. This is because HTML pages will need to be able to simultaneously prepare multiple synthesis objects in anticipation of a variety of user-generated events. A server MAY service simultaneous requests in parallel or in series, as deemed appropriate by the implementation, but it MUST accept them in parallel (i.e. support having multiple outstanding requests).
6. Because a Synthesizer resource only renders a stream, and is not responsible for playback of that stream to a user, it does NOT provide any form of shuttle control (pausing or skipping), since this is performed by the client; nor does it provide any control over volume, rate, pitch, etc, other than as specified in the SSML input document.

7.2.3 Protocol Basics

In the HTML speech protocol, the control signals and the media itself are transported over the same WebSocket connection. Earlier implementations utilized a simple HTTP connection for speech recognition and synthesis. Use cases involving continuous recognition motivated the move to WebSockets. This simple design avoids all the normal media problems of session negotiation, packet delivery, port & IP address assignments, NAT-traversal, etc, since the underlying WebSocket already satisfies these requirements. A beneficial side-effect of this design is that by limiting the protocol to WebSockets over HTTP there should be less problems with firewalls compared to having a separate RTP connection of other for the media transport. This design is different from MRCP, which is oriented around telephony/IVR and all its impediments, rather than HTML and WebServices, and is motivated by simplicity and desire to keep the protocol within HTTP.

7.2.3.1 Session Establishment

The WebSockets session is established through the standard WebSockets HTTP handshake, with these specifics:

- The Sec-WebSocket-Protocol header MUST be "web-speech/1.0". The client MAY also request alternative sub-protocols, in which case the server selects whichever it supports or prefers. This follows standard WebSockets handshake logic, and allows for extensibility, either for vendor-specific innovation, or for future protocol versions.
- Service-specific parameters MAY be passed in the URL query string. Any parameters set this way can be overridden by subsequent messages sent in the

web-speech/1.0 protocol after the WebSockets session has been established. The corollary to this is that Service parameters MUST NOT be passed in the HTTP GET message body of the initial WebSockets handshake.
- Prior to sending the 101 Switching Protocols message, the service MAY send other HTTP responses to perform useful functions such as redirection, user authentication, or 4xx errors. This is just standard WebSockets behavior.

For example:

```
C->S: GET /speechservice123?customparam=foo&otherparam=bar HTTP/1.1
      Host: examplespeechservice.com
      Upgrade: websocket
      Connection: Upgrade
      Sec-WebSocket-Key: OIUSDGY67SDGLjkSD&g2 (for example)
      Sec-WebSocket-Version: 9
      Sec-WebSocket-Protocol: web-speech/1.0, x-proprietary-speech

S->C: HTTP/1.1 101 Switching Protocols
      Upgrade: websocket
      Connection: Upgrade
      Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
      Sec-WebSocket-Protocol: web-speech/1.0
```

Once the WebSockets session is established, the UA can begin sending requests and media to the service, which can respond with events, responses or media.

A session MAY have a maximum of one synthesizer resource and one recognizer resource. If an application requires multiple resources of the same type (such as, for example, two synthesizers from different vendors), it MUST use separate WebSocket sessions.

There is no association of state between sessions. If a service wishes to provide a special association between separate sessions, it may do so behind the scenes (for example, to re-use audio input from one session in another session without resending it, or to cause service-side barge-in of TTS in one session by recognition in another session, would be service-specific extensions).

7.2.3.2 Control Messages

The signaling design borrows its basic pattern from [*MRCPv2*], where there are three classes of control messages:

Requests
: C->S requests from the UA to the service. The client requests a method (SPEAK, LISTEN, STOP, etc) from a particular remote speech resource.

Status Notifications
: S->C general status notification messages from the service to the UA, marked as either PENDING, IN-PROGRESS or COMPLETE.

Named Events

S->C named events from the service to the UA, that are essentially special cases of 'IN-PROGRESS' request-state notifications.

```
control-message =   start-line ; i.e. use the typical MIME message format
                  *(header CRLF)
                    CRLF
                   [body]
start-line      =   request-line | status-line | event-line
header          =  <Standard MIME header format> ; case-insensitive. Actual headers depend on
body            =  *OCTET                        ; depends on the type of message
```

The interaction is full-duplex and asymmetrical: service activity is instigated by requests the UA, which may be multiple and overlapping, and where each request results in one or more messages from the service back to the UA.

For example:

```
C->S: web-speech/1.0 SPEAK 3257                    ; request synthesis of string
        Resource-ID:synthesizer
        Audio-codec:audio/basic
        Content-Type:text/plain

        Hello world! I speak therefore I am.

S->C: web-speech/1.0 3257 200 IN-PROGRESS         ; server confirms it will start synthesizing

S->C: media for 3257                               ; receive synthesized media

S->C: web-speech/1.0 SPEAK-COMPLETE 3257 COMPLETE ; done!
```

Request Messages

Request messages are sent from the client to the server, usually to request an action or modify a setting. Each request has its own request-id, which is unique within a given WebSockets web-speech session. Any status or event messages related to a request use the same request-id. All request-ids MUST have a non-negative integer value of 1-10 decimal digits.

```
request-line    = version SP method-name SP request-id SP CRLF
version         = "web-speech/" 1*DIGIT "." 1*DIGIT ; web-speech/1.0
method-name     = general-method | synth-method | reco-method | proprietary-method
request-id      = 1*10DIGIT
```

NOTE: In some other protocols, messages also include their message length, so that they can be framed in what is otherwise an open stream of data. In web-

speech/1.0, framing is already provided by WebSockets, and message length is not needed, and therefore not included.

For example, to request the recognizer to interpret text as if it were spoken:

```
C->S: web-speech/1.0 INTERPRET 8322
      Resource-Identifier: recognizer
      Active-Grammars: <http://myserver.example.com/mygrammar.grxml>
      Interpret-Text: Send a dozen yellow roses and some expensive chocolates to my mother
```

Status Messages

Status messages are sent from the server to the client, to indicate the state of a request.

```
status-line    =  version SP request-id SP status-code SP request-state CRLF
status-code    =  3DIGIT       ; Specific codes TBD, but probably similar to those used in MRCP

; All communication from the server is labeled with a request state.
request-state = "COMPLETE"    ; Processing of the request has completed.
              | "IN-PROGRESS" ; The request is being fulfilled.
              | "PENDING"     ; Processing of the request has not begun.
```

Specific status code values follow a pattern similar to [*MRCPv2*]:

2xx Success Codes

- 200: Success
- 201: Success with some optional header fields ignored.

4xx Client Failure Codes

- 401: Method not allowed.
- 402: Method not valid in this state.
- 403: Unsupported header field.
- 404: Illegal header field value.
- 405: Resource does not exist.
- 406: Mandatory header field missing.
- 407: Method or operation failed (e.g. grammar compilation failed).
- 408: Unrecognized or unsupported message entity.
- 409: Unsupported header field value.
- 480: No input stream.
- 481: Unsupported content language.

5xx Server Failure

- 501: Server internal error
- 502: Protocol version not supported

Event messages are sent by the server, to indicate specific data, such as synthesis marks, speech detection, and recognition results. They are essentially specialized status messages.

```
event-line    =  version SP event-name SP request-id SP request-state CRLF
event-name    =  synth-event | reco-event | proprietary-event
```

For example, an event indicating that the recognizer has detected the start of speech:

```
S->C: web-speech/2.0 START-OF-SPEECH 8322 IN-PROGRESS
      Resource-ID: recognizer
      Source-time: 2011-09-06T21:47:31.981+1:30 (when speech was detected)
```

7.2.3.3 Media Transmission

HTML Speech applications feature a wide variety of media transmission scenarios. The number of media streams at any given time is not fixed. For example:

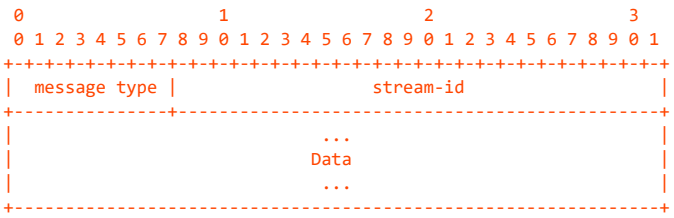
- A recognizer may accept one or more input streams, which may start and end at any time as microphones or other input devices are activated/deactivated by the application or the user.
- Applications may, and often will, request the synthesis of multiple SSML documents at the same time, which are buffered by the UA for playback at the application's discretion.
- The synthesizer needs to return rendered data to the client rapidly (generally faster than real time), and MAY render multiple requests in parallel if it has the capacity to do so.

Whereas a human listener will tolerate the clicks and pops of missing packets so they can continue listening in real time, recognizers do not require their data in real-time, and will generally prefer to wait for delayed packets in order to maintain accuracy.

Advanced implementations of HTML Speech may incorporate multiple channels of audio in a single transmission. For example, a living-room device with a microphone array may send separate streams capturing the speech of multiple individuals within the room. Or, for example, a device may send parallel streams with alternative encodings that may not be human-consumable but contain information that is of particular value to a recognition service.

In web-speech/1.0, audio (or other media) is packetized and transmitted as a series of WebSockets binary messages, on the same WebSockets session used for the control messages.

```
media-packet        =  binary-message-type
                       binary-stream-id
                       binary-data
binary-message-type =  OCTET ; Values > 0x03 are reserved. 0x00 is undefined.
binary-stream-id    = 3OCTET ; Unique identifier for the stream 0..2^24-1
binary-data         = *OCTET
```



The **binary-stream-id** field is used to identify the messages for a particular stream. It is a 24-bit unsigned integer. Its value for any given stream is assigned by the sender (client or server) in the initial message of the stream, and must be unique to the sender within the WebSockets session.

A sequence of media messages with the same stream-ID represents an in-order contiguous stream of data. Because the messages are sent in-order and audio packets cannot be lost (WebSockets uses TCP), there is no need for sequence numbering or timestamps. The sender just packetizes audio from the encoder and sends it, while the receiver just un-packs the messages and feeds them to the consumer (e.g. the recognizer's decoder, or the TTS playback buffer). Timing of coordinated events is calculated by decoded offset from the beginning of the stream.

The WebSockets stack de-multiplexes text and binary messages, thus separating signaling from media, while the stream-ID on each media message is used to de-multiplex the messages into separate media streams.

The **binary-message-type** field has these defined values:

0x01: Start of Stream

The message indicates the start of a new stream. This MUST be the first message in any stream. The stream-ID must be new and unique to the session. This same stream-ID is used in all future messages in the stream. The message body contains a 64-bit NTP timestamp [*NTP*] containing the local time at the stream originator, which is used as the base time for calculating event timing. The timestamp is followed by the ASCII-encoded MIME media type (see [*RFC3555*]) describing the format that the stream's media is encoded in (usually an audio encoding). All implementations MUST support 8kHz single-channel mulaw (audio/basic) and 8kHz single-channel 16-bit linear PCM (audio/L16;rate=8000). Implementations MAY support other content types, for example: to recognize from a video stream; to provide a stream of recognition preprocessing coefficients; to provide textual metadata streams; or to provide

auxiliary multimodal input streams such as touches/strokes, gestures, clicks, compass bearings, etc.

```
message type = 0x01; stream-id = 112233; media-type = audio/amr-wb
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                  stream-id                    |
        |1 0 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
        |1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0| } NTP Timestamp
        |0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1| }
        |      61               75              64              69      | a u d i
        |      6F               2F              61              6D      | o / a m
        |      72               2D              77              62      | r - w b
        +---------------------------------------------------------------+
```

0x02: Media

The message is a media packet, and contains encoded media data.

```
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                  stream-id                    |
        |0 1 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
        |                       encoded audio data                      |
        |                              ...                              |
        |                              ...                              |
        |                              ...                              |
        +---------------------------------------------------------------+
```

The encoding format is specified in the start of stream message (0x01).

There is no strict constraint on the size and frequency of audio messages. Nor is there a requirement for all audio packets to encode the same duration of sound. However, implementations SHOULD seek to minimize interference with the flow of other messages on the same socket, by sending messages that encode between 20 and 80 milliseconds of media. Since a WebSockets frame header is typically only 4 bytes, overhead is minimal and implementations SHOULD err on the side of sending smaller packets more frequently.

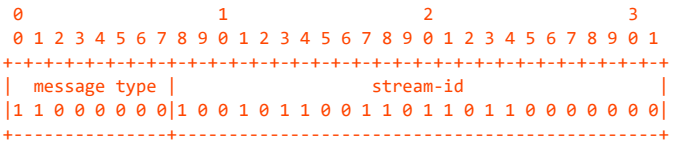
A synthesis service MAY (and typically will) send audio faster than real-time, and the client MUST be able to handle this.

A recognition service MUST be prepared to receive slower-than-real-time audio due to practical throughput limitations of the network.

The design does not permit the transmission of binary media as base-64 text messages, since WebSockets already provides native support for binary messages. Base-64 encoding would incur an unnecessary 33% transmission overhead.

0x03: End-of-stream

The 0x03 end-of-stream message indicates the end of the media stream, and MUST be used to terminate a media stream. Any future media stream messages with the same stream-id are invalid.



Although most services will be strictly either recognition or synthesis services, some services may support both in the same session. While this is a more advanced scenario, the design does not introduce any constraints to prevent it. Indeed, both the client and server MAY send audio streams in the same session.

7.2.3.4 Security

Both the signaling and media transmission aspects of the web-speech/1.0 protocol inherit a number of security features from the underlying WebSockets protocol [[!WEBSOCKETS-PROTOCOL]:

Server Authentication
: Clients may authenticate servers using standard TLS [*TLS*], simply by using the WSS: uri scheme rather than the WS: scheme in the service URI. This is standard WebSockets functionality in much the same way as HTTP specifies TLS by using the HTTPS: scheme.

Encryption
: Similarly, all traffic (media and signaling) is encrypted by TLS, when using the WSS: uri scheme.

: In practice, to prevent man-in-the-middle snooping of a user's voice, user agents SHOULD NOT use the WS: scheme, and SHOULD ONLY use the WSS: scheme. In non-mainstream cases, such as service-to-service mashups, or specialized user agents for secured networks, the unencrypted WS: scheme MAY be used.

User Authentication
: User authentication, when required by a server, will commonly be done using the standard [*HTTP11*] challenge-response mechanism in the initial websocket bootstrap. A server may also choose to use TLS client authentication, and although this will probably be uncommon, WebSockets stacks should support it.

Application Authentication:
: Web services may wish to authenticate the application requesting service. There is no standardized way to do this. However, the Vendor-Specific-

Parameters header can be used to perform proprietary authentication using a key-value-pair scheme defined by the service.

HTML speech network scenarios also have security boundaries outside of signaling and media:

Transitive Access to Resources

A client may require a server to access resources from a third location. Such resources may include SRGS documents, SSML documents, audio files, etc. This may either be a result of the application referring to the resource by URI; or of an already loaded resource containing a URI reference to a separate resource. In these cases the server will need permission to access these resources. There are three ways in which this may be accomplished:

1. **Out-of-band configuration**. The developer, administrator, or a provisioning system may control both the speech server and the server containing the referenced resource. In this case they may configure appropriate accounts and network access permissions beforehand.
2. **Limited-use URIs**. The server containing the resource may issue limited-use URIs, that may be valid for a small finite number of uses, or for a limited period, in order to minimise the exposure of the resource. The developer would obtain these URIs as needed, using a mechanism that is proprietary to the resource server.
3. **Cookies**. The client may have a cookie containing a secret that is used to authorize access to the resource. In this case, the cookie may be passed to the speech server using cookie headers in a request. The speech server would then use this cookie when accessing resources required by that request.

Access to Retained Media

Through the use of certain headers during speech recognition, the client may request the server to retain a recording of the input media, and make this recording available at a URL for retrieval. The server that holds the recording MAY secure this recording by using standard HTTP security mechanisms: it MAY authenticate client using standard HTTP challenge/response; it MAY use TLS to encrypt the recording when transmitting it back to the client; and it MAY use TLS to authenticate the client. The server that holds a recording MAY also discard a recording after a reasonable period, as determined by the server.

7.2.3.5 Time Stamps

Timestamps are used in a variety of headers in the protocol. Binary messages use the 64-bit NTP timestamp format, as defined in [*NTP*]. Text messages use the encoding format defined in [*RFC3339*], and reproduced here:

```
date-time       = full-date "T" full-time
; For example: 2011-09-06T10:33:16.612Z
;          or: 2011-09-06T21:47:31.981+1:30

full-date       = date-fullyear "-" date-month "-" date-mday
full-time       = partial-time time-offset

date-fullyear   = 4DIGIT
date-month      = 2DIGIT  ; 01-12
date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on
                          ; month/year

partial-time    = time-hour ":" time-minute ":" time-second
                  [time-secfrac]
time-hour       = 2DIGIT  ; 00-23
time-minute     = 2DIGIT  ; 00-59
time-second     = 2DIGIT  ; 00-58, 00-59, 00-60 based on leap second
                          ; rules
time-secfrac    = "." 1*DIGIT
time-numoffset  = ("+" / "-") time-hour ":" time-minute
time-offset     = "Z" / time-numoffset
```

7.2.4 General Capabilities

7.2.4.1 Generic Headers

These headers may be used in any control message. All header names are case-insensitive.

```
generic-header  =
                | accept
                | accept-charset
                | content-base
                | logging-tag
                | resource-id
                | vendor-specific
                | content-type
                | content-encoding

resource-id     = "Resource-ID:" ("recognizer" | "synthesizer" | vendor-resource)
vendor-resource = "x-" 1*UTFCHAR

accept           = <indicates the content-types the sender will accept>
accept-charset   = <indicates the character set the sender will accept>
content-base     = <the base for relative URIs>
content-type     = <the type of content contained in the message body>
content-encoding = <the encoding of message body content>
logging-tag      = <a tag to be inserted into server logs>
vendor-specific  = "Vendor-Specific-Parameters:" vendor-specific-av-pair
                   *[";" vendor-specific-av-pair] CRLF
vendor-specific-av-pair = vendor-av-pair-name "=" vendor-av-pair-value
```

Resource-ID

The Resource-ID header is included in all signaling messages. In requests, it indicates the resource to which the request is directed. In status messages and events, it indicates the resource from which the message originated.

Accept

The Accept header MAY be included in any message to indicate the content types that will be accepted by the sender of the message from the receiver of the message. When absent, the following defaults should be assumed: clients will accept "application/emma+xml" from recognizers; recognizers will accept "application/srgs+xml"; synthesizers will accept "application/ssml+xml".

Accept-Charset

When absent, any charset may be used. This header has two general purposes: so the client can indicate the charset it will accept in recognition results; and so the synthesizer can indicate the charset it will accept for SSML documents.

Content-Base

When a message contains an entity that includes relative URIs, Content-Base provides the absolute URI against which they are based.

Logging-Tag

Specifies a tag to be inserted into server logs. It is generally only used in requests, or in response to GET-PARAMS.

Vendor-Specific-Parameters

A catch-all header for vendors to include their own name/value pairs..

7.2.4.2 Capabilities Discovery

The web-speech/1.0 protocol provides a way for the application/UA to determine whether a resource supports the basic capabilities it needs. In most cases applications will know a service's resource capabilities ahead of time. However, some applications may be more adaptable, or may wish to double-check at runtime. To determine resource capabilities, the UA sends a GET-PARAMS request to the resource, containing a set of capabilities, to which the resource responds with the specific subset it actually supports.

The GET-PARAMS request is much more limited in scope than its [*MRCPv2*] counterpart. It is only used to discover the capabilities of a resource. Like all messages, they must always include the Resource-ID header.

```
general-method = "GET-PARAMS"

header         = capability-query-header
               | interim-event-header
               | reco-header
               | synth-header

capability-query-header =
                 "Supported-Content:" mime-type *("," mime-type)
               | "Supported-Languages:" lang-tag *("," lang-tag) ; See [RFC5646]
               | "Builtin-Grammars:" "<" URI ">" *("," "<" URI ">")

interim-event-header =
```

```
                "Interim-Events:" event-name *("," event-name)
event-name = 1*UTFCHAR
```

Supported-Languages

This read-only property is used by the client to discover whether a resource supports a particular set of languages. Unlike most headers, when a blank value is used in GET-PARAMS, the resource will respond with a blank header rather than the full set of languages it supports. This avoids the resource having to respond with a potentially cumbersome and possibly ambiguous list of languages and dialects. Instead, the client must include the set of languages in which it is interested as the value of the Supported-Languages header in the GET-PARAMS request. The service will respond with the subset of these languages that it actually supports.

Supported-Content

This read-only property is used to discover whether a resource supports particular encoding formats for input or output data. Given the broad variety of codecs, and the large set of parameter permutations for each codec, it is impractical for a resource to advertize all media encodings it could possibly support. Hence, when a blank value is used in GET-PARAMS, the resource will respond with a blank value. Instead, the client must supply the of data encodings it is interested in. The resource responds with the subset it actually supports. This is used not only to discover supported media encoding formats, but also, to discover other input and output data formats, such as alternatives to SRGS, EMMA and SSML.

Builtin-Grammars

This read-only property is used by the client to discover whether a recognizer has a particular set of built-in grammars. The client provides a list of builtin: URIs in the GET-PARAMS request, to which the recognizer responds with the subset of URIs it actually supports.

Interim-Events

This read/write property contains the set of interim events the client would like the service to send (see Interim Events below).

For example, discovering whether the recognizer supports the desired CODECs, grammar format, languages/dialects and built-in grammars:

```
C->S: web-speech/1.0 GET-PARAMS 34132
      resource-id: recognizer
      supported-content: audio/basic, audio/amr-wb,
                         audio/x-wav;channels=2;formattag=pcm;samplespersec=44100,
                         audio/dsr-es202212; rate:8000; maxptime:40,
                         application/x-ngram+xml
      supported-languages: en-AU, en-GB, en-US, en (A variety of English dialects are desired)
      builtin-grammars: <builtin:dictation?topic=websearch>,
                        <builtin:dictation?topic=message>,
                        <builtin:ordinals>,
```

```
                        <builtin:datetime>,
                        <builtin:cities?locale=USA>

S->C: web-speech/1.0 34132 200 COMPLETE
      resource-id: recognizer
      supported-content: audio/basic, audio/dsr-es202212; rate:8000; maxptime:40
      supported-languages: en-GB, en (The recognizer supports UK English, but will work with a
      builtin-grammars: <builtin:dictation?topic=websearch>, <builtin:dictation?topic=message>
```

For example, discovering whether the synthesizer supports the desired CODECs, languages/dialects, and content markup format:

```
C->S: web-speech/1.0 GET-PARAMS 48223
      resource-id: synthesizer

      supported-content: audio/ogg, audio/flac, audio/basic, application/ssml+xml
      supported-languages: en-AU, en-GB

S->C: web-speech/1.0 48223 200 COMPLETE
      resource-id: synthesizer
      supported-content: audio/basic, application/ssml+xml
      supported-languages: en-GB
```

7.2.4.3 Interim Events

Speech services may care to send optional vendor-specific interim events during the processing of a request. For example: some recognizers are capable of providing additional information as they process input audio; and some synthesizers are capable of firing progress events on word, phoneme, and viseme boundaries. These are exposed through the HTML Speech API as events that the webapp can listen for if it knows to do so. A service vendor MAY require a vendor-specific value to be set in a LISTEN or SPEAK request before it starts to fire certain events.

```
interim-event =   version request-ID SP INTERIM-EVENT CRLF
                *(header CRLF)
                  CRLF
                 [body]
event-name-header = "Event-Name:" event-name
```

The Event-Name header is required and must contain a value that was previously subscribed to with the Interim-Events header.

The Request-ID and Content-Type headers are required, and any data conveyed by the event must be contained in the body.

For example, a synthesis service might choose to communicate visemes through interim events:

```
C->S: web-speech/1.0 SPEAK 3257
        Resource-ID:synthesizer
        Audio-codec:audio/basic
        Content-Type:text/plain
```

```
        Hello world! I speak therefore I am.

S->C: web-speech/1.0 3257 200 IN-PROGRESS

S->C: media for 3257

S->C: web-speech/1.0 INTERIM-EVENT 3257 IN-PROGRESS
        Resource-ID:synthesizer
        Event-Name:x-viseme-event
        Content-Type:application/x-viseme-list

        "Hello"
        0.500 H
        0.850 A
        1.050 L
        1.125 OW
        1.800 SILENCE

S->C: more media for 3257

S->C: web-speech/1.0 INTERIM-EVENT 3257 IN-PROGRESS
        Resource-ID:synthesizer
        Event-Name:x-viseme-event
        Content-Type:application/x-viseme-list

        "World"
        2.200 W
        2.350 ER
        2.650 L
        2.800 D
        3.100 SILENCE

S->C: etc
```

7.2.4.4 Resource Selection

Applications will generally want to select resources with certain capabilities, such as the ability to recognize certain languages, work well in specific acoustic conditions, work well with specific genders or ages, speak particular languages, speak with a particular style, age or gender, etc.

There are three ways in which resource selection can be achieved, each of which has relevance:

By URI

This is the preferred mechanism.

Any service may enable applications to encode resource requirements as query string parameters in the URI, or by using specific URIs with known resources. The specific URI format and parameter scheme is by necessity not standardized and is defined by the implementer based on their architecture and service offerings.

For example, a German recognizer with a cell-phone acoustic environment model:

```
ws://example1.net:2233/webreco/de-de/cell-phone
```

A UK English recognizer for a two-beam living-room array microphone:

```
ws://example2.net/?reco-lang=en-UK&reco-acoustic=10-foot-open-room&sample-rate=16kHz&chan
```

Spanish recognizer and synthesizer, where the synthesizer uses a female voice provided by AcmeSynth:

```
ws://example3.net/speech?reco-lang=es-es&tts-lang=es-es&tts-gender=female&tts-vendor=Acme
```

A pre-defined profile specified by an ID string:

```
ws://example4.com/profile=af3e-239e-9a01-66c0
```

By Request Header

Request headers may also be used to select specific resource capabilities. Synthesizer parameters are set through the SPEAK request; whereas recognition paramaters are set either through the LISTEN request. There is a small set of standard headers that can be used with each resource: the Speech-Language header may be used with both the recognizer and synthesizer, and the synthesizer may also accept a variety of voice selection parameters as headers. A resource MAY honor these headers, but does not need to where it does not have the ability to so. If a particular header value is unsupported, the request should fail with a status of 409 "Unsupported Header Field Value".

For example, a client requires Canadian French recognition but it isn't available:

```
C->S: web-speech/1.0 LISTEN 8322
      Resource-ID: Recognizer
      Speech-Language: fr-CA

S->C: web-speech/1.0 8322 409 COMPLETE ; 409, since fr-CA isn't supported.
      resource-id: Recognizer
```

A client requres Brazilian Portuguese, and is successful:

```
C->S: web-speech/1.0 LISTEN 8323
      Resource-ID: Recognizer
      Speech-Language: pt-BR

S->C: web-speech/1.0 8323 200 COMPLETE
      resource-id: Recognizer
```

Speak with the voice of a Korean woman in her mid-thirties:

```
C->S: web-speech/1.0 SPEAK 8324
      Resource-ID: Synthesizer
      Speech-Language: ko-KR
      Voice-Age: 35
      Voice-Gender: female
```

Speak with a Swedish voice named "Kiana":

```
C->S: web-speech/1.0 SPEAK 8325
      Resource-ID: Synthesizer
      Speech-Language: sv-SE
      Voice-Name: Kiana
```

This approach is very versatile. However some implementations will be incapable of this kind of versatility in practice.

By Input Document

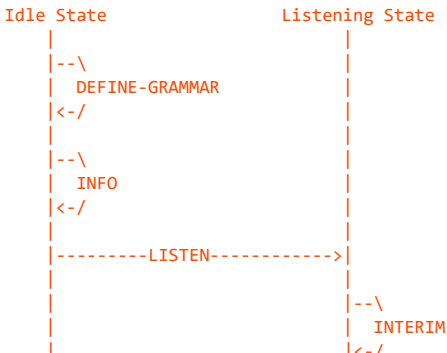
The [*SPEECH-GRAMMAR*] and [*SPEECH-SYNTHESIS*] input documents for the recognizer and synthesizer will specify the language for the overall document, and MAY specify languages for specific subsections of the document. The resource consuming these documents SHOULD honor these language assignments when they occur. If a resource is unable to do so, it should error with a 481 status "Unsupported content language". (It should be noted that at the time of writing, most currently available recognizer and synthesizer implementations will be unable to suppor this capability.)

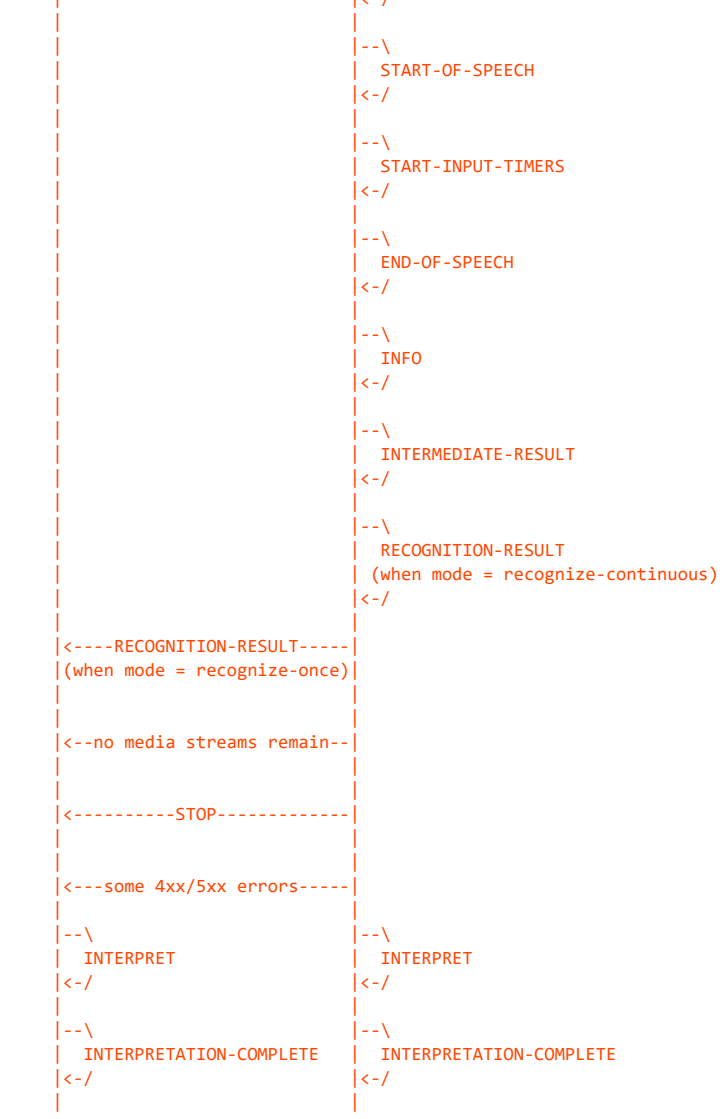
Generally speaking, given the current typical state of speech technology, unless a service is unusually adaptable, applications will be most successful using specific proprietary URLs that encode the abilities they need, so that the appropriate resources can be allocated during session initiation.

7.2.5 Recognition

A recognizer resource is either in the "listening" state, or the "idle" state. Because continuous recognition scenarios often don't have dialog turns or other down-time, all functions are performed in series on the same input stream(s). The key distinction between the idle and listening states is the obvious one: when listening, the recognizer processes incoming media and produces results; whereas when idle, the recognizer SHOULD buffer audio but will not process it.

Recognition is accomplished with a set of messages and events, to a certain extent inspired by those in [*MRCPv2*].





7.2.5.1 Recognition Requests

```
reco-method  = "LISTEN"              ; Transitions Idle -> Listening
             | "START-INPUT-TIMERS"  ; Starts the timer for the various input timeout condition
             | "STOP"                ; Transitions Listening -> Idle
             | "DEFINE-GRAMMAR"      ; Pre-loads & compiles a grammar, assigns a temporary URI
             | "CLEAR-GRAMMARS"      ; Unloads all grammars, whether active or inactive
             | "INTERPRET"           ; Interprets input text as though it was spoken
             | "INFO"                ; Sends metadata to the recognizer
```

LISTEN

The LISTEN method transitions the recognizer from the idle state to the listening state. The recognizer then processes the media input streams against the set of active grammars. The request MUST include the Source-Time header, which is used by the Recognizer to determine the point in the input stream(s) that the recognizer should start processing from (which won't necessarily be the start of the stream). The request MUST also include the Listen-Mode header to indicate whether the recognizer should perform continuous recognition, a single recognition, or vendor-specific processing.

A LISTEN request MAY also activate or deactivate grammars and rules using the Active-Grammars and Inactive-Grammars headers. These grammars/rules are considered to be activated/deactivated from the point specified in the Source-Time header.

When there are no input media streams, and the Input-Waveform-URI header has not been specified, the recognizer cannot enter the listening state, and the listen request will fail (480). When in the listening state, and all input streams have ended, the recognizer automatically transitions to the idle state, and issues a RECOGNITION-RESULT event, with Completion-Cause set to 080 ("no-input-stream").

A LISTEN request that is made while the recognizer is already listening results in a 402 error ("Method not valid in this state", since it is already listening).

START-INPUT-TIMERS

This is used to indicate when the input timeout clock should start. For example, when the application wants to enable voice barge-in during a prompt, but doesn't want to start the time-out clock until after the prompt has completed, it will delay sending this request until it's finished playing the prompt.

STOP

The STOP method transitions the recognizer from the listening state to the idle state. No RECOGNITION-RESULT event is sent. The Source-Time header MUST be used, since the recognizer may still fire a RECOGNITION-RESULT event for any completion state it encounters prior to that time in the input stream.

A STOP request that is sent while the recognizer is idle results in a 402 response (method not valid in this state, since there is nothing to stop).

DEFINE-GRAMMAR

The DEFINE-GRAMMAR method does not activate a grammar. It simply causes the recognizer to pre-load and compile it, and associates it with a temporary URI that can then be used to activate or deactivate the grammar or one of its rules. DEFINE-GRAMMAR is not required in order to use a grammar, since the recognizer can load grammars on demand as needed. However, it is useful when an application wants to ensure a large grammar is pre-loaded and ready

for use prior to the recognizer entering the listening state. DEFINE-GRAMMAR can be used when the recognizer is in either the listening or idle state.

All recognizer services MUST support grammars in the SRGS XML format, and MAY support additional alternative grammar/language-model formats.

The client SHOULD remember the temporary URIs, but if it loses track, it can always re-issue the DEFINE-GRAMMAR request, which MUST not result in a service error as long as the mapping is consistent with the original request. Once in place, the URI MUST be honored by the service for the duration of the session. If the service runs low in resources, it is free to unload the URI's payload, but must always continue to honor the URI even if it means reloading the grammar (performance notwithstanding).

Refer to [*MRCPv2*] for more details on this method.

CLEAR-GRAMMARS

In continuous recognition, a variety of grammars may be loaded over time, potentially resulting in unused grammars consuming memory resources in the recognizer. The CLEAR-GRAMMARS method unloads all grammars, whether active or inactive. Any URIs previously defined in DefineGrammar become invalid.

INTERPRET

The INTERPRET method processes the input text according to the set of grammar rules that are active at the time it is received by the recognizer. It MUST include the Interpret-Text header. The use of INTERPRET is orthogonal to any audio processing the recognizer may be doing, and will not affect any audio processing. The recognizer can be in either the listening or idle state.

An INTERPRET request MAY also activiate or deactivate grammars and rules using the Active-Grammars and Inactive-Grammars headers, but only if the recognizer is in the Idle state. These grammars/rules are considered to be activated/deactivated from the point specified in the Source-Time header.

INFO

In multimodal applications, some recognizers will benefit from additional context. Clients can use the INFO request to send this context. The Content-Type header should specify the type of data, and the data itself is contained in the message body.

7.2.5.2 Recognition Events

Recognition events are associated with 'IN-PROGRESS' request-state notifications from the 'recognizer' resource.

```
reco-event    = "START-OF-SPEECH"       ; Start of speech has been detected
              | "END-OF-SPEECH"         ; End of speech has been detected
              | "INTERIM-EVENT"         ; See Interim Events above
              | "INTERMEDIATE-RESULT"   ; A partial hypothesis
              | "RECOGNITION-RESULT"    ; Similar to MRCP2 RECOGNITION-COMPLETE except that appl
              | "INTERPRETATION-COMPLETE"
```

END-OF-SPEECH

END-OF-INPUT is the logical counterpart to START-OF-INPUT, and indicates that speech has ended. The event MUST include the Source-Time header, which corresponds to the point in the input stream where the recognizer estimates speech to have ended, NOT when the endpointer finally decided that speech ended (which will be a number of milliseconds later).

INTERIM-EVENT

See Interim Events above. For example, a recognition service may send interim events to indicate it's begun to recognize a phrase, or to indicate that noise or cross-talk on the input channel is degrading accuracy.

INTERMEDIATE-RESULT

Continuous speech (aka dictation) often requires feedback about what has been recognized thus far. INTERMEDIATE-RESULT provides this intermediate feedback without having to waiting for a finalized recognition result. As with RECOGNITION-RESULT, contents are assumed to be EMMA unless an alternate Content-Type is provided.

INTERPRETATION-COMPLETE

This event contains the result of an INTERPRET request.

RECOGNITION-RESULT

This method is similar to the [*MRCPv2*] RECOGNITION-COMPLETE method, except that application/emma+xml (EMMA) is the default Content-Type. The Source-Time header must be included, to indicate the point in the input stream when the event occurred. When the Listen-Mode is reco-once, the recognizer will transition from the listening state to the idle state when this message is fired, and the Recognizer-State header in the event is set to "idle". Note that in continuous mode the request-state will indicate whether recognition is ongoing (IN-PROGRESS) or complete (COMPLETE).

Where applicable, the body of the message SHOULD contain an EMMA that is consistent with the Completion-Cause.

START-OF-SPEECH

Indicates that start of speech has been detected. The Source-Time header MUST correspond to the point in the input stream(s) where speech was estimated to begin, NOT when the endpointer finally decided that speech began (a number of milliseconds later).

7.2.5.3 Recognition Headers

The list of valid headers for the recognizer resource include a subset of the [*MRCPv2*] Recognizer Header Fields, where they make sense for HTML Speech requirements, as well as a handful of headers that are required for HTML Speech.

```
reco-header =  ; Headers borrowed from MRCP
               Confidence-Threshold
             | Sensitivity-Level
             | Speed-Vs-Accuracy
             | N-Best-List-Length
             | No-Input-Timeout
             | Recognition-Timeout
             | Media-Type
             | Input-Waveform-URI
             | Completion-Cause
             | Completion-Reason
             | Recognizer-Context-Block
             | Start-Input-Timers
             | Speech-Complete-Timeout
             | Speech-Incomplete-Timeout
             | Failed-URI
             | Failed-URI-Cause
             | Save-Waveform
             | Speech-Language
             | Hotword-Min-Duration
             | Hotword-Max-Duration
             | Interpret-Text
             | Vendor-Specific       ; see Generic Headers

            ; Headers added for web-speech/1.0
             | audio-codec           ; The audio codec used in an input media stream
             | active-grammars       ; Specifies a grammar or specific rule to activate.
             | inactive-grammars     ; Specifies a grammar or specific rule to deactivate.
             | hotword               ; Whether to listen in "hotword" mode (i.e. ignore out-of
             | listen-mode           ; Whether to do continuous or one-shot recognition
             | partial               ; Whether to send partial results
             | partial-interval      ; Suggested interval between partial results, in millisec
             | result-index          ; Indicates which section of continuous result this is
             | recognizer-state      ; Indicates whether the recognizer is listening or idle
             | source-time           ; The UA local time at the request was initiated
             | user-id               ; Unique identifier for the user, so that adaptation can
             | Wave-Start-Time       ; The start point of a recognition in the audio referred
             | Wave-End-Time         ; The end point of a recognition in the audio referred to
             | Waveform-URIs         ; List of URIs to recorded input streams

hotword            = "Hotword:" BOOLEAN
listen-mode        = "Listen-Mode:" ("reco-once" | "reco-continuous" | vendor-listen-mode)
vendor-listen-mode = "x-" 1*UTFCHAR
recognizer-state   = "Recognizer-State:" ("listening" | "idle")
source-time        = "Source-Time:" 1*20DIGIT
audio-codec        = "Audio-Codec:" mime-media-type ; see [RFC3555]
partial            = "Partial:" BOOLEAN
partial-interval   = "Partial-Interval:" 1*5DIGIT
result-index       = "Result-Index:" 1*20DIGIT
active-grammars    = "Active-Grammars:" "<" URI ["#" rule-name] [SP weight] ">" *("," "<" URI
rule-name          = 1*UTFCHAR
weight             = 1*3DIGIT ["." 1*3DIGIT]
inactive-grammars  = "Inactive-Grammars:" "<" URI ["#" rule-name] ">" *("," "<" URI ["#" rule-
user-id            = "User-ID:" 1*UTFCHAR
wave-start-time    = "Wave-Start-Time:" 1*DIGIT ["." 1*DIGIT]
```

```
wave-end-time       = "Wave-End-Time:"  1*DIGIT ["." 1*DIGIT]
waveform-URIs       = "Waveform-URIs:" "<" URI ">" *("," "<" URI ">")
```

TODO: discuss how recognition from file would work.

Headers with the same names as their [*MRCPv2*] counterparts are considered to have the same specification. Other headers are describe as follows:

Audio-Codec

The Audio-Codec header is used in the START-MEDIA-STREAM request, to specify the codec and parameters used to encode the input stream, using the MIME media type encoding scheme specified in [*RFC3555*].

Active-Grammars

The Active-Grammars header specifies a list of grammars, and optionally specific rules within those grammars. The header is used in LISTEN to activate grammars/rules. If no rule is specified for a grammar, the root rule is activated. This header may also specify the relative weight of the rule. If weight is not specified, the default weight is "1".

Inactive-Grammars

The Inactive-Grammars header specifies a list of grammars, and optionally specific rules within those grammars, to be deactivated. If no rule is specified, all rules in the grammar are deactivated, including the root rule. The Grammar-Deactivate header MAY be used in the LISTEN method.

Hotword

The Hotword header is analogous to the [*MRCPv2*] Recognition-Mode header, however it has a different name and boolean type in web-speech/1.0 in order to avoid confusion with the Listen-Mode header. When true, the recognizer functions in "hotword" mode, which essentially means that out-of-grammar speech is ignored.

Listen-Mode

Listen-Mode is used in the LISTEN request to specify whether the recognizer should listen continuously, or return to the idle state after the first RECOGNITION-RESULT event. It MUST NOT be used in any other type of request other than LISTEN. When the recognizer is in the listening state, it should include Listen-Mode in all event and status messages it sends.

Partial

This header is required to support the continuous speech scenario on the recognizer resource. When sent by the client in a LISTEN request, this header controls whether or not the client is interested in partial results from the service. In this context, the term 'partial' is meant to describe mid-utterance results that provide a best guess at the user's speech thus far (e.g. "deer", "dear father", "dear father christmas"). These results should contain all recognized speech

from the point of the last non-partial (ie complete) result, but it may be common for them to omit fully-qualified result attributes like an NBest list, timings, etc. The only guarantee is that the content must be EMMA. Note that this header is valid on both regular command-and-control recognition requests as well as dictation sessions. This is because at the API level, there is no syntactic difference between the recognition types. They are both simply recognition requests over an SRGS grammar or set of URL(s). Additionally partial results can be useful in command-and-control scenarios, for example: open-microphone applications, dictation enrollment applications, and lip-sync. When sent by the server, this header indicates whether the message contents represent a full or partial result. It's valid for a server to send this header on both INTERMEDIATE-RESULT and RECOGNITION-RESULT, and in response to a GET-PARAMS message.

Partial-Interval

A suggestion from the client to the service on the frequency at which partial results should be sent. It is an integer value represents desired interval expressed in milliseconds. The recognizer does not need to precisely honor the requested interval, but SHOULD provide something close, if it is within the operating parameters of the implementation.

Result-Index

During continuous recognition a speech service might send results that are intended to be replacements for earlier results during that recognition. The server MAY include the Result-Index header field in the INTERMEDIATE-RESULT and RECOGNITION-RESULT events to indicate which piece of the overall result is being sent (or re-sent). If this header field is present when in "reco-once" mode, it MUST have the value "0".

Recognizer-State

Indicates whether the recognizer is listening or idle. This MUST NOT be included by the client in any requests, and MUST be included by the recognizer in all status and event messages it sends.

Source-Time

Indicates the timestamp of a message using the client's local time. All requests sent from the client to the recognizer MUST include the Source-Time header, which must faithfully specify the client's local system time at the moment it sends the request. This enables the recognizer to correctly synchronize requests with the precise point in the input stream at which they were actually sent by the client. All event messages sent by the recognizer MUST include the Source-Time, calculated by the recognizer service based on the point in the input stream at which the event occurred, and expressed in the client's local clock time (since the recognizer knows what this was at the start of the input stream). By expressing all times in client-time, the user agent or application is able to correctly sequence events, and implement timing-sensitive scenarios,

that involve other objects outside the knowledge of the recognizer service (for example, media playback objects or videogame states).

User-ID

Recognition results are often more accurate if the recognizer can train itself to the user's speech over time. This is especially the case with dictation as vocabularies are so large. A User-ID field would allow the recognizer to establish the user's identify if the webapp decided to supply this information.

Wave-Start-Time, Wave-End-Time, Input Waveform-URI and Waveform-URIs

Some applications will wish to re-recognize an utterance using different grammars. For example, an application may accept a broad range of input, and use the first round of recognition simply to classify an utterance so that it can use a more focused grammar on the second round. Others will wish to record an utterance for future use. For example, an application transcribes an utterance to text may store a recording so that untranscribed information is not lost (tone, emotion, etc.). While these are not mainstream scenarios, they are both valid and inevitable, and may be achieved using the headers provided for recognition.

If the Save-Waveform header is set to true (with LISTEN), then the recognizer will save the input audio. Consequent RECOGNITION-RESULT events sent by the recognizer will contain a URI in the Waveform-URI header which refers to the stored audio (multiple URIs when multiple input streams are present). In the case of continuous recognition, the Waveform-URI header refers to all of the audio captured so far. The application may fetch the audio from this URI, assuming it has appropriate credentials (the credential policy is determined by the service provider). The application may also use the URI as input to future LISTEN requests by passing the URI in the Input-Waveform-URI header.

When RECOGNITION-RESULT returns a Waveform-URI header, it also returns the time interval within the recorded waveform that the recognition result applies to, in the Wave-Start-Time and Wave-End-Time headers, which indicate the offsets in seconds from the start of the waveform. A client MAY also use the SourceTime header of other events such as START-OF-SPEECH and END-OF-SPEECH to calculate other intervals of interest. When using the Input-Wavefor-URI header, the client may suffix the URI with an "interval" parameter to indicate that the recognizer should only decode that particular interval of the audio:

```
interval = "interval=" start "," end
start    = seconds | "start"
end      = seconds | "end"
seconds  = 1*DIGIT ["." 1*DIGIT]
```

For example:

```
http://example.com/retainedaudio/fe429ac870a?interval=0.3,2.86
http://example.com/temp44235.wav?interval=0.65,end
```

When the Input-Waveform-URI header is used, all other input streams are ignored.

7.2.5.4 Predefined Grammars

Speech services MAY support pre-defined grammars that can be referenced through a 'builtin:' uri. For example:

```
builtin:dictation?context=email&lang=en_US
builtin:date
builtin:search?context=web
```

These can be used as top-level grammars in the Grammar-Activate/Deactivate headers, or in rule references within other grammars. If a speech service does not support the referenced builtin or if it does not specify the builtin in combination with other active grammars, it should return a grammar compilation error.

The specific set of predefined grammars is to be defined later. However, there MUST be a certain small set of predefined grammars that a user agent's default speech recognizer MUST support. For non-default recognizers, support for predefined grammars is optional, and the set that is supported is also defined by the service provider and may include proprietary grammars (e.g. builtin:x-acme-parts-catalog).

7.2.5.5 Recognition Examples

EXAMPLE OF RECO-ONCE

```
Start streaming audio:

C->S: binary message: start of stream (stream-id = 112233)
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                  stream-id                    |
        |1 0 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
        |1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0| } NTP Timestamp
        |0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1| }
        |      61             75              64             69       | a u d i
        |      6F             2F              61             6D       | o / a m
        |      72             2D              77             62       | r - w b
        +---------------------------------------------------------------+

C->S: binary message: media packet (stream-id = 112233)
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                  stream-id                    |
        |0 1 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
        |                       encoded audio data                      |
```

```
          |                              ...                              |
          |                              ...                              |
          |                              ...                              |
          +---------------------------------------------------------------+

C->S: more binary media packets...

Send the LISTEN request:

C->S: web-speech/1.0 LISTEN 8322
      Resource-Identifier: recognizer
      Confidence-Threshold:0.9
      Active-Grammars: <built-in:dictation?context=message>
      Listen-Mode: reco-once
      Source-time: 2011-09-06T21:47:31.981+1:30 (where in the input stream recognition should

S->C: web-speech/2.0 START-OF-SPEECH 8322 IN-PROGRESS

C->S: more binary media packets...

C->S: binary audio packets...
C->S: binary audio packet in which the user stops talking
C->S: binary audio packets...

S->C: web-speech/1.0 END-OF-SPEECH 8322 IN-PROGRESS (i.e. the recognizer has detected the user

C->S: binary audio packet: end of stream (i.e. since the recognizer has signaled end of input,
          0                   1                   2                   3
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          |  message type |                    stream-id                  |
          |1 1 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
          +---------------+-----------------------------------------------+

S->C: web-speech/1.0 RECOGNITION-RESULT 8322 COMPLETE (because mode = reco-once, it the reques
      Resource-Identifier: recognizer

      <emma:emma version="1.0"
      ...etc
```

Example of continuous reco with intermediate results

```
C->S: binary message: start of stream (stream-id = 112233)
          0                   1                   2                   3
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          |  message type |                    stream-id                  |
          |1 0 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
          +---------------+-----------------------------------------------+
          |1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0| } NTP Timestamp
          |0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1| }
          |      61              75              64              69       | a u d i
          |      6F              2F              61              6D       | o / a m
          |      72              2D              77              62       | r - w b
          +---------------------------------------------------------------+

C->S: binary message: media packet (stream-id = 112233)
          0                   1                   2                   3
          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```


```
C->S: more binary media packets...

C->S: web-speech/1.0 LISTEN 8322
      Resource-Identifier: recognizer
      Confidence-Threshold:0.9
      Active-Grammars: <built-in:dictation?context=message>
      Listen-Mode: reco-continuous
      Partial: TRUE
      Source-time: 2011-09-06T21:47:31.981+1:30 (where in the input stream recognition should

C->S: more binary media packets...

S->C: web-speech/2.0 START-OF-SPEECH 8322 IN-PROGRESS
      Source-time: 12753439912 (when speech was detected)

C->S: more binary media packets...

S->C: web-speech/2.0 INTERMEDIATE-RESULT 8322 IN-PROGRESS

C->S: more binary media packets...

S->C: web-speech/1.0 END-OF-SPEECH 8322 IN-PROGRESS (i.e. the recognizer has detected the user

C->S: more binary media packets...

S->C: web-speech/1.0 RECOGNITION-RESULT 8322 IN-PROGRESS (because mode = reco-continuous, the

C->S: more binary media packets...

S->C: web-speech/2.0 START-OF-SPEECH 8322 IN-PROGRESS

S->C: web-speech/2.0 INTERMEDIATE-RESULT 8322 IN-PROGRESS

S->C: web-speech/1.0 RECOGNITION-RESULT 8322 IN-PROGRESS

S->C: web-speech/2.0 INTERMEDIATE-RESULT 8322 IN-PROGRESS

S->C: web-speech/1.0 RECOGNITION-RESULT 8322 IN-PROGRESS (because mode = reco-continuous, the

S->C: web-speech/1.0 END-OF-SPEECH 8322 IN-PROGRESS (i.e. the recognizer has detected the user

C->S: binary audio packet: end of stream (i.e. since the recognizer has signaled end of input,
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |  message type |                  stream-id                    |
       |1 1 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
       +---------------+-----------------------------------------------+

S->C: web-speech/1.0 RECOGNITION-RESULT 8322 COMPLETE
      Recognizer-State:idle
      Completion-Cause: 080
      Completion-Reason: No Input Streams
```

1-Best EMMA document

Example showing 1-best with an XML semantics within emma:interpretation. The 'interpretation' is contained within the emma:interpretation element. The 'transcript' is the value of emma:tokens and 'confidence' is the value of emma:confidence.

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
    <emma:grammar id="gram1"
    grammar-type="application/srgs-xml" <!-- From EMMA 1.1 -->
    ref="http://example.com/flightquery.grxml"/>
  <emma:interpretation id="int1"
        emma:start="1087995961542"
        emma:end="1087995963542"
        emma:medium="acoustic"
        emma:mode="voice"
        emma:confidence="0.75"
   emma:lang="en-US"
   emma:grammar-ref="gram1"
   emma:media-type="audio/x-wav; rate:8000;"
   emma:signal="http://example.com/signals/145.wav"
        emma:tokens="flights from boston to denver"
   emma:process="http://example.com/my_asr.xml">
      <origin>Boston</origin>
      <destination>Denver</destination>

  </emma:interpretation>
</emma:emma>
```

N-Best EMMA Doucment

Example showing multiple recognition results and their associated interpretations.

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
    <emma:grammar id="gram1"
    grammar-type="application/srgs-xml"
    ref="http://example.com/flightquery.grxml"/>
    <emma:grammar id="gram2"
    grammar-type="application/srgs-xml"
    ref="http://example.com/pizzaorder.grxml"/>
  <emma:one-of id="r1"
        emma:start="1087995961542"
        emma:end="1087995963542"
        emma:medium="acoustic"
        emma:mode="voice"
   emma:lang="en-US"
   emma:media-type="audio/x-wav; rate:8000;"
```

```
    emma:signal="http://example.com/signals/789.wav
    emma:process="http://example.com/my_asr.xml">
     <emma:interpretation id="int1"
         emma:confidence="0.75"
         emma:tokens="flights from boston to denver"
        emma:grammar-ref="gram1">
                 <origin>Boston</origin>
                 <destination>Denver</destination>
     </emma:interpretation>
     <emma:interpretation id="int2"
         emma:confidence="0.68"
         emma:tokens="flights from austin to denver"
                 emma:grammar-ref="gram1">
                 <origin>Austin</origin>
                 <destination>Denver</destination>
     </emma:interpretation>
   </emma:one-of>
</emma:emma>
```

No-match EMMA Document

In the case of a no-match the EMMA result returned MUST be annotated as emma:uninterpreted="true".

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
    http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
  <emma:interpretation id="interp1"
        emma:uninterpreted="true"
    emma:medium="acoustic"
    emma:mode="voice"
    emma:process="http://example.com/my_asr.xml"/>
</emma:emma>
```

No-input

In the case of a no-match the EMMA interpretation returned must be annotated as emma:no-input="true" and the <emma:interpretation> element must be empty.

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"

    xmlns="http://www.example.com/example">
  <emma:interpretation id="int1"
        emma:no-input="true"
        emma:medium="acoustic"
        emma:mode="voice"
   emma:process="http://example.com/my_asr.xml"/>
</emma:emma>
```

Multimodal EMMA Document

Example showing a multimodal interpretation resulting from combination of speech input with a mouse event passed in through a control metadata message.

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
  <emma:interpretation
      emma:medium="acoustic tactile"
      emma:mode="voice touch"
      emma:lang="en-US"
      emma:start="1087995963542"
           emma:end="1087995964542"
      emma:process="http://example.com/myintegrator.xml">
    <emma:derived-from resource="voice1" composite="true"/>
    <emma:derived-from resource="touch1" composite="true"/>
    <command>
       <action>zoom</action>
       <location>
         <point>42.1345 -37.128</point>
        </location>
     </command>
  </emma:interpretation>
   <emma:derivation>
               <emma:interpretation id="voice1"
                       emma:medium="acoustic"
                       emma:mode="voice"
          emma:lang="en-US"
          emma:start="1087995963542"
               emma:end="1087995964542"
          emma:media-type="audio/x-wav; rate:8000;"
                       emma:tokens="zoom in here"
          emma:signal="http://example.com/signals/456.wav"
          emma:process="http://example.com/my_asr.xml">
                       <command>
                 <action>zoom</action>
                 <location/>
                </command>
       </emma:interpretation>
       <emma:interpretation id="touch1"
                       emma:medium="tactile"
                       emma:mode="touch"
          emma:start="1087995964000"
               emma:end="1087995964000">
            <point>42.1345 -37.128</point>
       </emma:interpretation>
   </emma:derivation>
</emma:emma>
```

Lattice EMMA Document

As an example of a lattice of semantic interpretations, in a travel application where the source is either "Boston" or "Austin"and the destination is either "Newark" or

"New York", the possibilities might be represented in a lattice as follows:

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
  <emma:grammar id="gram1"
    grammar-type="application/srgs-xml"
    ref="http://example.com/flightquery.grxml"/>
  <emma:interpretation id="interp1"
        emma:medium="acoustic"
        emma:mode="voice"
        emma:start="1087995961542"
        emma:end="1087995963542"
        emma:medium="acoustic"
        emma:mode="voice"
        emma:confidence="0.75"
        emma:lang="en-US"
        emma:grammar-ref="gram1"
   emma:signal="http://example.com/signals/123.wav"
        emma:media-type="audio/x-wav; rate:8000;"
   emma:process="http://example.com/my_asr.xml">
     <emma:lattice initial="1" final="8">
       <emma:arc from="1" to="2">flights</emma:arc>
       <emma:arc from="2" to="3">to</emma:arc>
       <emma:arc from="3" to="4">boston</emma:arc>
       <emma:arc from="3" to="4">austin</emma:arc>
       <emma:arc from="4" to="5">from</emma:arc>
       <emma:arc from="5" to="6">portland</emma:arc>
       <emma:arc from="5" to="6">oakland</emma:arc>
       <emma:arc from="6" to="7">today</emma:arc>
       <emma:arc from="7" to="8">please</emma:arc>
       <emma:arc from="6" to="8">tomorrow</emma:arc>
     </emma:lattice>
  </emma:interpretation>
</emma:emma>
```

7.2.6 Synthesis

In HTML speech applications, the synthesizer service does not participate directly in the user interface. Rather, it simply provides rendered audio upon request, similar to any media server, plus interim events such as marks. The UA buffers the rendered audio, and the application may choose to play it to the user at some point completely unrelated to the synthesizer service. It is the synthesizer's role to render the audio stream in a timely manner, at least rapidly enough to support real-time playback. The synthesizer MAY also render and transmit the stream faster than required for real time playback, or render multiple streams in parallel, in order to reduce latency in the application. This is a stark contrast to the IVR implemented by MRCP, where the synthesizer essentially renders directly to the user's telephone, and is an active part of the user interface.

The synthesizer MUST support [*SPEECH-SYNTHESIS*] AND plain text input. A synthesizer MAY also accept other input formats. In all cases, the client should use the content-type header to indicate the input format.

7.2.6.1 Synthesis Requests

```
synth-method = "SPEAK"
             | "STOP"
             | "DEFINE-LEXICON"
```

The set of synthesizer request methods is a subset of those defined in [*MRCPv2*]

SPEAK

The SPEAK method operates similarly to its [*MRCPv2*] namesake. The primary difference is that SPEAK results in a new audio stream being sent from the server to the client, using the same Request-ID. A SPEAK request MUST include the Audio-Codec header. When the rendering has completed, and the end-of-stream message has been sent, the synthesizer sends a SPEAK-COMPLETE event.

STOP

When the synthesizer receives a STOP request, it ceases rendering the requests specified in the Active-Request-ID header. If the Active-Request-ID header is missing, it ceases rendring all active SPEAK requests. For any SPEAK request that is ceased, the synthesizer sends an end-of-stream message, and a SPEAK-COMPLETE event.

DEFINE-LEXICON

This is used to load or unload a lexicon, and is identical to its namesake in [*MRCPv2*].

7.2.6.2 Synthesis Events

Synthesis events are associated with 'IN-PROGRESS' request-state notifications from the synthesizer resource.

```
synth-event  = "INTERIM-EVENT"   ; See Interim Events above
             | "SPEECH-MARKER"   ; An SSML mark has been rendered
             | "SPEAK-COMPLETE"
```

INTERIM-EVENT

See Interim Events above.

SPEECH-MARKER

This event indicates that an SSML mark has been rendered. It uses the Speech-Marker header, which contains a timestamp indicating where in the stream the mark occurred, and the label associated in the mark.

Implementations should send the SPEECH-MARKER as closely as possible to the corresponding media packet so clients may play the media and fire events in real time if needed.

SPEAK-COMPLETE

Indicates that rendering of the SPEAK request has completed.

7.2.6.3 Synthesis Headers

The synthesis headers used in web-speech/1.0 are mostly a subset of those in [*MRCPv2*], with some minor modification and additions.

```
synth-header = ; headers borrowed from [MRCPv2]
               active-request-id-list
             | Completion-Cause
             | Completion-Reason
             | Voice-Gender
             | Voice-Age
             | Voice-Variant
             | Voice-Name
             | Prosody-parameter ; Actually a collection of headers, see [MRCPv2]
             | Speech-Marker
             | Speech-Language
             | Failed-URI
             | Failed-URI-Cause
             | Load-Lexicon
             | Lexicon-Search-Order
             | Vendor-Specific       ; see Generic Headers
               ; new headers for web-speech/1.0
             | Audio-Codec
             | Stream-ID ; read-only

Speech-Marker = "Speech-Marker:" "timestamp" "=" date-time [";" 1*(UTFCHAR)]
                ; e.g. Speech-Marker:timestamp=2011-09-06T10:33:16.612Z;banana
Audio-Codec   = "Audio-Codec:" mime-media-type ; See [RFC3555]
Stream-ID     = 1*8DIGIT ; decimal representation of 24-bit stream-ID
```

Audio-Codec

Because an audio stream is created in response to a SPEAK request, the audio codec and parameters must be specified in the SPEAK request using the Audio-Codec header. If the synthesizer is unable to encode with this codec, it terminates the request with a 409 (unsupported header field) COMPLETE status message.

Speech-Marker

This event indicates when an SSML mark is rendered. It is similar to its namesake in [*MRCPv2*], except that the clock is defined as the local time at the service, and the timestamp format is as defined in this document. By using the timestamp from the beginning of the stream, and the timestamp of this event, the UA can calculate when to raise the event to the application based on where it is in the playback of the rendered stream.

Stream-ID

Specifies the ID of the stream that contains the rendered audio, so that the UA can associate audio streams it receives with particular SPEAK requests. This is a read-only parameter, returned in responses to the SPEAK request.

Simple rendering of plain text

The most straightforward use case for TTS is the synthesis of one utterance at a time. This is inevitable for just-in-time rendition of speech, for example in dialogue systems or in in-car navigation scenarios. Here, the web application will send a single speech synthesis SPEAK request to the speech service.

```
C->S: web-speech/1.0 SPEAK 3257
        Resource-ID:synthesizer
        Audio-codec:audio/flac
        Speech-Language: de-DE
        Content-Type:text/plain

        Hallo, ich heiße Peter.

S->C: web-speech/1.0 3257 200 IN-PROGRESS
        Resource-ID:synthesizer
        Stream-ID: 112233
        Speech-Marker:timestamp=2011-09-06T10:33:16.612Z

S->C: binary message: start of stream (stream-id = 112233)
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                   stream-id                   |
        |1 0 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
        |1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0| } NTP Timestamp
        |0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1| }
        |      61               75              64              69      | a u d i
        |      6F               2F              66              6C      | o / f l
        |      61               63       +------------------------------+ a c
        +-------------------------------+

S->C: more binary media packets...

S->C: web-speech/1.0 SPEAK-COMPLETE 3257 COMPLETE
        Resource-ID:Synthesizer
        Completion-Cause:000 normal
        Speech-Marker:timestamp=2011-09-06T10:33:26.922Z

S->C: binary audio packets...

S->C: binary audio packet: end of stream ( message type = 0x03 )
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                   stream-id                   |
        |1 1 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
```

Simple Rendering of SSML

For richer markup of the text, it is possible to use the SSML format for sending an annotated request. For example, it is possible to propose an appropriate pronunciation or to indicate where to insert pauses. (SSML example adapted from http://www.w3.org/TR/speech-synthesis11/#edef_break)

```
C->S: web-speech/1.0 SPEAK 3257
        Resource-ID:synthesizer
        Voice-gender:neutral
        Voice-Age:25
        Audio-codec:audio/flac
        Prosody-volume:medium
        Content-Type:application/ssml+xml

        <?xml version="1.0"?>
        <speak version="1.1" xmlns="http://www.w3.org/2001/10/synthesis"
                xml:lang="en-US">
           Please make your choice. <break time="3s"/>
           Click any of the buttons to indicate your preference.
        </speak>

Remainder of example as above
```

Bulk Requests for Synthesis

Some use cases require relatively static speech output which can be known at the time of loading a web page. In these cases, all required speech output can be requested in parallel as multiple concurrent requests. Callback methods in the web api are responsible to relate each speech stream to the appropriate place in the web application.

On the protocol level, the request of multiple speech streams concurrently is realized as follows.

```
C->S: web-speech/1.0 SPEAK 3257
        Resource-ID:synthesizer
        Audio-codec:audio/basic
        Speech-Language: es-ES
        Content-Type:text/plain

        Hola, me llamo Maria.

C->S: web-speech/1.0 SPEAK 3258
        Resource-ID:synthesizer
        Audio-codec:audio/basic
        Speech-Language: en-UK
        Content-Type:text/plain

        Hi, I'm George.

C->S: web-speech/1.0 SPEAK 3259
        Resource-ID:synthesizer
        Audio-codec:audio/basic
        Speech-Language: de-DE
        Content-Type:text/plain
```

```
        Hallo, ich heiße Peter.

S->C: web-speech/1.0 3257 200 IN-PROGRESS

S->C: media for 3257

S->C: web-speech/1.0 3258 200 IN-PROGRESS

S->C: media for 3258

S->C: web-speech/1.0 3259 200 IN-PROGRESS

S->C: media for 3259

S->C: more media for 3257

S->C: web-speech/1.0 SPEAK-COMPLETE 3257 COMPLETE

S->C: more media for 3258

S->C: web-speech/1.0 SPEAK-COMPLETE 3258 COMPLETE

S->C: more media for 3259

S->C: web-speech/1.0 SPEAK-COMPLETE 3259 COMPLETE
```

The service MAY choose to serialize its processing of certain requests (such as only rendering one SPEAK request at a time), but MUST still accept multiple active requests.

Multimodal Coordination with Bookmarks

In order to synchronize the speech content with other events in the web application, it is possible to mark relevant points in time using the SSML <mark> tag. When the speech is played back, a callback method is called for these markers, allowing the web application to present, e.g., visual displays synchronously.

(Example adapted from http://www.w3.org/TR/speech-synthesis11/#S3.3.2)

```
C->S: web-speech/1.0 SPEAK 3257
        Resource-ID:synthesizer
        Voice-gender:neutral
        Voice-Age:25
        Audio-codec:audio/flac
        Prosody-volume:medium
        Content-Type:application/ssml+xml

        <?xml version="1.0"?>
        <speak version="1.1" xmlns="http://www.w3.org/2001/10/synthesis"
                xml:lang="en-US">
        Would you like to sit <mark name="window_seat"/> here at the window, or
        rather <mark name="aisle_seat"/> here at the aisle?
        </speak>

S->C: web-speech/1.0 3257 200 IN-PROGRESS
        Resource-ID:synthesizer
```

```
        Stream-ID: 112233
        Speech-Marker:timestamp=2011-09-06T10:33:16.612Z

S->C: binary message: start of stream (stream-id = 112233)
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                  stream-id                    |
        |1 0 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
        |1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0| } NTP Timestamp
        |0 0 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1| }
        |      61             75              64              69       | a u d i
        |      6F             2F              66              6C       | o / f l
        |      61             63       +------------------------------+ a c
        +------------------------------+

S->C: more binary media packets...

S->C: web-speech/2.0 SPEECH-MARKER 3257 IN-PROGRESS
        Resource-ID:synthesizer
        Stream-ID: 112233
        Speech-Marker:timestamp=2011-09-06T10:33:18.310Z;window_seat

S->C: more binary media packets...

S->C: web-speech/2.0 SPEECH-MARKER 3257 IN-PROGRESS
        Resource-ID:synthesizer
        Stream-ID: 112233
        Speech-Marker:timestamp=2011-09-06T10:33:21.008Z;aisle_seat

S->C: more binary media packets...

S->C: web-speech/1.0 SPEAK-COMPLETE 3257 COMPLETE
        Resource-ID:Synthesizer
        Completion-Cause:000 normal
        Speech-Marker:timestamp=2011-09-06T10:33:23.881Z

S->C: binary audio packets...

S->C: binary audio packet: end of stream ( message type = 0x03 )
         0                   1                   2                   3
         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |  message type |                  stream-id                    |
        |1 1 0 0 0 0 0 0|1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0|
        +---------------+-----------------------------------------------+
```

7.2.7 Examples

Example

This example shows the protocol interactions on the "Testing this example, and launching this missile. < cough>" user utterance.

???

```
// Preload large grammar
```

```
C->S: web-speech/1.0 DEFINE-GRAMMAR 257
  Resource-ID:recognizer
  Content-Type:text/uri-list
  Content-ID:Precompile-123

  builtin:dictation?type=x-acme-military-control



S->C: web-speech/1.0 257 200 COMPLETE
  Completion-Cause: 000 Success



C->S: web-speech/1.0 LISTEN 258
  Resource-ID:recognizer
  Source-Time: 6023
  Listen-Mode: continuous
  Active-Grammars: session:Precompile-123



S->C: web-speech/1.0 258 200 IN-PROGRESS



// User starts speaking

User Speech: "test"



S->C: web-speech/1.0 258 START-OF-SPEECH IN-PROGRESS
  Source-Time: 6035



S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS
  Resource-ID:recognizer
```

```
  Content-Type:application/emma+xml
  Source-Time: 6035
  Result-Index: 0

  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
      <emma:interpretation id="int1" emma:confidence="0.65"
       emma:tokens="text"/>
      <emma:interpretation id="int2" emma:confidence="0.60"
       emma:tokens="test"/>
    </emma:one-of>
  </emma:emma>



API Event1: [0] {"text", "test"}-I
Aggregate result: [{"text", "test"}-I]



// Modifying top choice


S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS
  Resource-ID:recognizer
  Content-Type:application/emma+xml
  Source-Time: 6035
  Result-Index: 0

  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
      <emma:interpretation id="int1" emma:confidence="0.65"
       emma:tokens="test"/>
      <emma:interpretation id="int2" emma:confidence="0.50"
       emma:tokens="text"/>
    </emma:one-of>
```

```
  </emma:emma>


API Event2: [0] {"test", "text"}-I
Aggregate result: [{"test", "text"}-I]


User Speech: "ing"


S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS
  Resource-ID:recognizer
  Content-Type:application/emma+xml
  Source-Time: 6035
  Result-Index: 0

  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
      <emma:interpretation id="int1" emma:confidence="0.70"
       emma:tokens="test sting"/>
      <emma:interpretation id="int2" emma:confidence="0.50"
       emma:tokens="texting"/>
    </emma:one-of>
  </emma:emma>


API Event3: [0] {"test sting", "texting"}-I
Aggregate result: [ {"test sting"], ["texting"}-I ]


User Speech: "this"
```

```
// Combining words within a phrase


S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS
  Resource-ID:recognizer
  Content-Type:application/emma+xml
  Source-Time: 6035
  Result-Index: 0

  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
      <emma:interpretation id="int1" emma:confidence="0.75"
       emma:tokens="testing this"/>
      <emma:interpretation id="int2" emma:confidence="0.50"
       emma:tokens="texting this"/>
    </emma:one-of>
  </emma:emma>



API Event4: [0] {"testing this", "texting this"}-I
Aggregate result: [ {"testing this"], ["texting this"}-I ]



User Speech: "example <small pause> and"



S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS
  Resource-ID:recognizer
  Content-Type:application/emma+xml
  Source-Time: 6035
  Result-Index: 0

  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
```

```
      <emma:interpretation id="int1" emma:confidence="0.75"
       emma:tokens="testing this example and"/>
      <emma:interpretation id="int2" emma:confidence="0.50"
       emma:tokens="texting this sample ant"/>
    </emma:one-of>
  </emma:emma>



API Event5: [0] {"testing this example and", "texting this sample
ant"}-I

Aggregate result: [ {"testing this example and", "texting this samp
ant"}-I ]



User Speech: "launching"



// Moving words across phrase boundaries


S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS
  Resource-ID:recognizer
  Content-Type:application/emma+xml
  Source-Time: 6035
  Result-Index: 0


  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
      <emma:interpretation id="int1" emma:confidence="0.75"
       emma:tokens="testing this example"/>
      <emma:interpretation id="int2" emma:confidence="0.50"
       emma:tokens="texting this sample"/>
    </emma:one-of>
  </emma:emma>
```

```
API Event6: [0] {"testing this example", "texting this sample"}-I

Aggregate result: [ {"testing this example", "texting this sample"}



S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS

  Resource-ID:recognizer

  Content-Type:application/emma+xml

  Source-Time: 10032

  Result-Index: 1


  <emma:emma version="1.0" ...>

    <emma:one-of id="r1" emma:mode="voice">

      <emma:interpretation id="int1" emma:confidence="0.65"

       emma:tokens="ant launching"/>

      <emma:interpretation id="int2" emma:confidence="0.45"

       emma:tokens="ant lunching"/>

    </emma:one-of>

  </emma:emma>



Event7: [1] {"and launching", "ant lunching"}-I

Aggregate result: [ {"testing this example", "texting this sample"}
{"and launching", "ant lunching"}-I ]



User Speech: "this missile"



S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS

  Resource-ID:recognizer

  Content-Type:application/emma+xml

  Source-Time: 10032

  Result-Index: 1
```

```
  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
      <emma:interpretation id="int1" emma:confidence="0.70"
       emma:tokens="and launching this missile"/>
      <emma:interpretation id="int2" emma:confidence="0.40"
       emma:tokens="ant lunching this thistle"/>
    </emma:one-of>
  </emma:emma>
```

```
Event8: [1] {"and launching this missile", "ant lunching this
thistle"}-I

Aggregate result: [ {"testing this example", "texting this sample"}
{"and launching this missile", "ant lunching this thistle"}-I ]
```

```
// First result is finalized
```

```
S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS
  Resource-ID:recognizer
  Content-Type:application/emma+xml
  Source-Time: 6035
  Result-Index: 0
  Result-Status: final

  <emma:emma version="1.0" ...>
    <emma:one-of id="r1" emma:mode="voice">
      <emma:interpretation id="int1" emma:confidence="0.80"
       emma:tokens="testing this example"/>
      <emma:interpretation id="int2" emma:confidence="0.45"
       emma:tokens="texting this sample"/>
    </emma:one-of>
  </emma:emma>
```

```
Event8: [0] {"testing this example", "texting this sample"}-F

Aggregate result: [ {"testing this example", "texting this sample"}
{"and launching this missile", "ant lunching this thistle"}-I ]



User Speech: "<cough>"


S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS

  Resource-ID:recognizer

  Content-Type:application/emma+xml

  Source-Time: 13022

  Result-Index: 2


  <emma:emma version="1.0" ...>

    <emma:one-of id="r1" emma:mode="voice">

      <emma:interpretation id="int1" emma:confidence="0.45"

       emma:tokens="confirm"/>

    </emma:one-of>

  </emma:emma>



Event9: [2] {"cancel"}-I

Aggregate result: [ {"testing this example", "texting this sample"}
{"and launching this missile", "ant lunching this thistle"}-I,
{"confirm"}-I ]



// Retracts spurious result


S->C: web-speech/1.0 INTERMEDIATE-RESULT 258 IN-PROGRESS

  Resource-ID:recognizer

  Result-Index: 2
```

```
  Source-Time: 13022

  Result-Status: final



Event10: [2] {}-F

Aggregate result: [ {"testing this example", "texting this sample"}
{"and launching this missile", "ant lunching this thistle"}-I ]



// Second result finalized, which completes transaction


S->C: web-speech/1.0  RECOGNITION-RESULT 258 COMPLETE

  Resource-ID:recognizer

  Content-Type:application/emma+xml

  Result-Index: 1

  Source-Time: 10032

  Result-Status: final


  <emma:emma version="1.0" ...>

    <emma:one-of id="r1" emma:mode="voice">

      <emma:interpretation id="int1" emma:confidence="0.75"

       emma:tokens="and launching this missile"/>

      <emma:interpretation id="int2" emma:confidence="0.40"

       emma:tokens="ant lunching this thistle"/>

    </emma:one-of>

  </emma:emma>



Event11: [1] {"and launching this missile", "ant lunching this
thistle"}-F

Aggregate result: [ {"testing this example", "texting this sample"}
{"and launching this missile", "ant lunching this thistle"}-F ]
```

A. Requirements development

A.1 Overview

The "initial requirements" were collected on the group's public email alias, collated into one large list, and then refactored and structured as you see at the end of this section of the appendix. These requirements were then refined into what were called the first pass requirements. To judge how important each requirement is and how much support there was for the requirement in the broader community these first pass requirements were then prioritized. These prioritized requirements helped to inform the discussion around the development of the design decisions and proposals in the main body of this report..

A.2 First Pass Requirements

This section covers group consensus of a clearer expansion of the earlier requirements described in the next section. These are not presented in order of prioritization. That order is provided in section 5.1 Prioritized Requirements. Including a requirement in this space certainly does not necessarily mean everyone in the group agrees that it is an important requirement than MUST be addressed. The requirements are not in numeric order since these requirements have been organized conceptually and have evolved over time from the initial requirements found in section A.3 Initial Requirements and any renumbering may be confusing when considering existing and historic discussions about the requirements.

A.2.1 Web Authoring Feature Requirements

This section covers requirements related to fundamental functionality required for scenarios and use cases.

A.2.1.1 Web Authoring Feature Speech System Requirements

This section covers web authoring feature requirements that are related to the whole speech system, that is to both the speech recognition and the speech synthesis system.

FPR8. User agent (browser) can refuse to use requested speech service.

This is part of the expansion of requirements 1, 15, 16, 22, and 31. It is expected that user agents should not refuse in the common case.

FPR11. If the web apps specify speech services, it should be possible to specify parameters.

This is part of the expansion of requirements 1, 15, 16, 22, and 31.

FPR12. Speech services that can be specified by web apps must include network speech services.

This is part of the expansion of requirements 1, 15, 16, 22, and 31.

FPR31. User agents and speech services may agree to use alternate protocols for communication.

This is a part of the expansion of requirement 18.

FPR32. Speech services that can be specified by web apps must include local speech services.

This is a part of the expansion of requirement 18.

FPR33. There should be at least one mandatory-to-support codec that isn't encumbered with IP issues and has sufficient fidelity & low bandwidth requirements.

This is a part of the expansion of requirement 18.

FPR40. Web applications must be able to use barge-in (interrupting audio and TTS output when the user starts speaking).

Notification of barge-in should be delivered in a timely manner after detection occurs. This is a part of the expansion of requirement 14.

FPR58. Web application and speech services must have a means of binding session information to communications.

This is a part of the expansion of mailing list discussions.

A.2.1.2 Web Authoring Feature Recognition Requirements

This section covers web authoring feature requirements that are related primarily to the speech recognition system.

FPR2. Implementations must support the XML format of SRGS and must support SISR.

This is one part of the evolution of requirement 27.

FPR4. It should be possible for the web application to get the recognition results in a standard format such as EMMA.

This is one part of the evolution of requirement 27.

FPR19. User-initiated speech input should be possible.

This is a part of the expansion of requirement 33 and requirement 29.

FPR21. The web app should be notified that capture starts.

Notification should be delivered in a timely manner after detection occurs. This is a part of the expansion of requirement 4.

FPR22. The web app should be notified that speech is considered to have started for the purposes of recognition.

Notification should be delivered in a timely manner after detection occurs. This is a part of the expansion of requirement 4.

FPR23. The web app should be notified that speech is considered to have ended for the purposes of recognition.

Notification should be delivered in a timely manner after detection occurs. This is a part of the expansion of requirement 4.

FPR24. The web app should be notified when recognition results are available.

Notification should be delivered in a timely manner after detection occurs. These results may be partial results and may occur several times. This is a part of the expansion of requirement 4.

FPR25. Implementations should be allowed to start processing captured audio before the capture completes.

This is a part of the expansion of requirement 17.

FPR26. The API to do recognition should not introduce unneeded latency.

This is a part of the expansion of requirement 17.

FPR27. Speech recognition implementations should be allowed to add implementation specific information to speech recognition results.

This is a part of the expansion of requirement 18.

FPR28. Speech recognition implementations should be allowed to fire implementation specific events.

This is a part of the expansion of requirement 18.

FPR34. Web application must be able to specify domain specific custom grammars.

This is a part of the expansion of requirement 7.

FPR35. Web application must be notified when speech recognition errors or non-matches occur.

The intent is that this requirement covers errors like bad format of grammars but also covers no input and no matches. Notification should be delivered in a timely manner after detection occurs. This is a part of the expansion of requirement 5.

FPR42. It should be possible for user agents to allow hands-free speech input.

This is a part of the expansion of requirement 24.

FPR43. User agents should not be required to allow hands-free speech input.

This is a part of the expansion of requirement 24.

FPR47. When speech input is used to provide input to a web app, it should be possible for the user to select alternative input methods.

This is a part of the expansion of requirement 23.

FPR48. Web application author must be able to specify a domain specific statistical language model.

This is a part of the expansion of requirement 12.

FPR50. Web applications must not be prevented from integrating input from multiple modalities.

This is a part of the expansion of requirement 11.

FPR54. Web apps should be able to customize all aspects of the user interface for speech recognition, except where such customizations conflict with security and privacy requirements in this document, or where they cause other security or privacy problems.

Other security and privacy requirements include the following FPR: 1, 10, 16, 17, 18, and 20. This is a part of the expansion of requirement 13.

FPR56. Web applications must be able to request NL interpretation based only on text input (no audio sent).

This is a part of the expansion of mailing list discussions.

FPR57. Web applications must be able to request recognition based on previously sent audio.

This is a part of the expansion of mailing list discussions.

FPR59. While capture is happening, there must be a way for the web application to abort the capture and recognition process.

This is a part of the expansion of mailing list discussions.

A.2.1.3 Web Authoring Feature Synthesis Requirements

This section covers web authoring feature requirements that are related primarily to the speech synthesis system.

FPR3. Implementation must support SSML.

This is one part of the evolution of requirement 27.

FPR29. Speech synthesis implementations should be allowed to fire implementation specific events.

This is a part of the expansion of requirement 18.

FPR41. It should be easy to extend the standard without affecting existing speech applications.

This is a part of the expansion of requirement 25.

FPR46. Web apps should be able to specify which voice is used for TTS.

This is a part of the expansion of requirement 20.

FPR51. The web app should be notified when TTS playback starts.

Notification should be delivered in a timely manner after playback begins. This is a part of the expansion of requirement 9.

FPR52. The web app should be notified when TTS playback finishes.

Notification should be delivered in a timely manner after playback finishes. This is a part of the expansion of requirement 9.

FPR53. The web app should be notified when the audio corresponding to a TTS <mark> element is played back.

Notification should be delivered in a timely manner after the mark is played back. This is a part of the expansion of requirement 9.

FPR60. Web application must be able to programatically abort tts output.

This is a part of the expansion of mailing list discussions.

A.2.2 Web Authoring Convenience Requirements

This section covers requirements related to making web authoring easy and convenient or exposing a recognition feature in a way that is consistent with web technologies.

A.2.2.1 Web Authoring Convenience Speech System Requirements

This section covers web authoring convenience requirements that are related to the whole speech system, that is to both the speech recognition and the speech synthesis system.

FPR7. Web apps should be able to request speech service different from default.

This is part of the expansion of requirements 1, 15, 16, 22, and 31.

FPR9. If browser refuses to use the web application requested speech service, it must inform the web app.

This is part of the expansion of requirements 1, 15, 16, 22, and 31.

FPR10. If browser uses speech services other than the default one, it must inform the user which one(s) it is using.

Note by "the default one" we mean the user agent default speech service. This is part of the expansion of requirements 1, 15, 16, 22, and 31.

FPR30. Web applications must be allowed at least one form of communication with a particular speech service that is supported in all UAs.

This evolved from: "The communication between the user agent and the speech server must require a mandatory-to-support lowest common denominator such as HTTP 1.1, TBD" which is part of the expansion of requirement 18.

A.2.2.2 Web Authoring Convenience Recognition Requirements

This section covers web authoring convenience requirements that are related primarily to the speech recognition system.

FPR5. It should be easy for the web appls to get access to the most common pieces of recognition results such as utterance, confidence, and nbests.

This is one part of the evolution of requirement 27.

FPR6. Browser must provide default speech resource.

This is part of the expansion of requirements 1, 15, 16, 22, and 31.

FPR36. User agents must provide a default interface to control speech recognition.

This is a part of the expansion of requirement 26.

FPR38. Web application must be able to specify language of recognition.

This is a part of the expansion of requirement 8.

FPR39. Web application must be able to be notified when the selected language is not available.

Notification should be delivered in a timely manner after detection occurs. This is a part of the expansion of requirement 8.

FPR44. Recognition without specifying a grammar should be possible.

I.e., free-form recognition. This is a part of the expansion of requirement 2.

FPR45. Applications should be able to specify the grammars (or lack thereof) separately for each recognition.

This is a part of the expansion of requirement 2.

A.2.2.3 Web Authoring Convenience Synthesis Requirements

This section covers web authoring convenience requirements that are related primarily to the speech synthesis system.

FPR13. It should be easy to assign recognition results to a single input field.

This is a part of the expansion of requirement 3.

FPR14. It should not be required to fill an input field every time there is a recognition result.

This is a part of the expansion of requirement 3.

FPR15. It should be possible to use recognition results to multiple input fields.

This is a part of the expansion of requirement 3.

FPR61. Aborting the TTS output should be efficient.

This is a part of the expansion of mailing list discussions.

A.2.3 Security and Privacy Requirements

This section covers requirements related to ensuring speech is allowed in a way that is in line with desired security and privacy requirements.

A.2.3.1 Security and Privacy Speech System Requirements

This section covers requirements related to ensuring speech in a speech system, that is with both the speech recognition and speech synthesis systems, is allowed in a way that is in line with desired security and privacy requirements.

FPR16. User consent should be informed consent.

This is a part of the expansion of requirement 33.

FPR20. The spec should not unnecessarily restrict the UA's choice in privacy policy.

This is a part of the expansion of requirement 33 and requirement 29.

FPR55. Web application must be able to encrypt communications to remote speech service.

This is a part of the expansion of mailing list discussions.

A.2.3.2 Security and Privacy Recognition Requirements

This section covers requirements related to ensuring speech in a recognition system is allowed in a way that is in line with desired security and privacy requirements.

FPR1. Web applications must not capture audio without the user's consent.

This is the evolution of requirement 29 and requirment 33.

FPR17. While capture is happening, there must be an obvious way for the user to abort the capture and recognition process.

This is a part of the expansion of requirement 33. Here the word abort should mean "as soon as you can, stop capturing, stop processing for recognition, and stop processing any recognition results".

FPR18. It must be possible for the user to revoke consent.

This is a part of the expansion of requirement 33.

FPR37. Web application should be given captured audio access only after explicit consent from the user.

This is a part of the expansion of requirement 28.

FPR49. End users need a clear indication whenever microphone is listening to the user

This is a part of the expansion of requirement 32.

A.2.3.3 Security and Privacy Synthesis Requirements

This section covers requirements related to ensuring speech in a synthesis system is allowed in a way that is in line with desired security and privacy requirements.

A.3 Initial Requirements

The use cases motivate a number of requirements for integrating speech into HTML. The Incubator Group has members that initially felt that each of the requirements described below are essential to the language; however, this represents the requirements before any group evaluation, rewording, and prioritization of these requirements. Each requirement should include a short description and should be motivated by one or more use cases from the previous section (not all use cases may be listed). For convenience, the requirements are organized around different high level themes.

A.3.1 Web Authoring Feature Requirements

The following requirements are around features that HTML web authors require to build speech applications.

A.3.1.1 R1. Web author needs full control over specification of speech resources

The HTML web author must have control over specifying both the speech recognizing technology used and the speech parameters that go to the recognizer. In particular, this also means that it must be possible to do recognition on a networked speech recognizer and this should also mean that it is possible to have any user agent work with any vendor's speech services provided the specified open protocols are used. Also, any recognizer parameters or hints must be able to be specified by the web application author.

Relevant Use Cases Include: U1 Voice Web Search, U3 Domain Specific Grammars Contingent on Earlier Inputs, U5 Domain Specific Grammars Filling Multiple Input Fields, U7 Rerecognition, U11 Speech Translation.

A.3.1.2 R2. Application change from directed input to free form input

An application may be required to switch between a grammar-based recognition to free form recognition. For example for simple date, yes/no, quantity etc grammar based reco might work fine. But for filling a comments section etc., we might want to use a free form recognizer.

Relevant Use Cases Include: U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog

A.3.1.3 R3. Ability to bind results to specific input fields

An application wants the results of matching a particular grammar or speech turn to fill a particular input field.

Relevant Use Cases Include: U3 Domain Specific Grammars Contingent on Earlier Inputs, U5 Domain Specific Grammars Filling Multiple Input Fields, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.1.4 R4. Web application must be notified when recognition occurs

When the speech recognition occurs the web application must be notified.

Relevant Use Cases Include: U8 Voice Activity Detection, U9 Temporal Structure of Synthesis to Provide Visual Feedback, U13 Dialog Systems, U14 Multimodal Interaction.

A.3.1.5 R5. Web application must be notified when speech recognition errors and other non-matches occur

When a recognition is attempted and either an error occurs or an utterance doesn't provide a recognition match or else the system doesn't detect speech for a sufficiently long time (I.e., noinput) the web application must be notified.

Relevant Use Cases Include: U1 Voice Web Search, U3 Domain Specific Grammars Contingent on Earlier Inputs, U5 Domain Specific Grammars Filling Multiple Input Fields, U7 Rerecognition, U8 Voice Activity Detection, U13 Dialog Systems, U14 Multimodal Interaction.

A.3.1.6 R6. Web application must be provided with full context of recognition

Because speech recognition is by its nature imperfect and probabilistic a set of additional metadata is frequently generated including n-best list of alternate suggestions, confidences or recognition results, and semantic structure represented by recognition results. All of this data must be provided to the web application.

Relevant Use Cases Include: U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U5 Domain Specific Grammars Filling Multiple Input Fields, U7 Rerecognition, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.1.7 R7. Web application must be able to specify domain specific custom grammars

It is necessary that the HTML author must be able to specify the grammars of their choosing and must not be restricted to only use grammars natively installed in the user-agent.

Relevant Use Cases Include: U1 Voice Web Search, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U5 Domain Specific Grammars Filling Multiple Input Fields, U7 Rerecognition, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.1.8 R8. Web application must be able to specify language of recognition

The HTML author must be able to specify the language of the recognition to be used for any given spoken interaction. This must be the case even if the language is different that that used in the content of the rest of the web page. This also may mean multiple different spoken language input elements are present in the same web page.

Relevant Use Cases Include: U1 Voice Web Search, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U5 Domain Specific Grammars Filling Multiple Input Fields, U11 Speech Translation.

A.3.1.9 R9. Web application author provided synthesis feedback

It is necessary that the web application author receive notification of the temporal and structural feedback of the synthesis of text.

Relevant Use Cases Include: U9 Temporal Structure of Synthesis to Provide Visual Feedback, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.1.10 R10. Web application authors need to be able to use full SSML features

When rendering synthesized speech HTML application authors need to be able to take advantage of features such as gender, language, pronunciations, etc.

Relevant Use Cases Include: U9 Temporal Structure of Synthesis to Provide Visual Feedback, U10 Hello World Use Case, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems.

A.3.1.11 R11. Web application author must integrate input from multiple modalities

The author may have multiple inputs that must be integrated to provide a quality user experience. For instance, the application might combine information from geolocation (to understand "here"), speech recognition, and touch to provide driving directions in response to the spoken phrase "Get me directions to this place" while tapping on a map. Since new modalities are continually becoming available, so it would be difficult to provide for integration on a case by case basis in the user agent, so it must be easy for the web application author to provide the integration.

Relevant Use Cases Include: U9 Temporal Structure of Synthesis to Provide Visual Feedback, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.1.12 R12. Web application author must be able to specify a domain specific statistical language model

A typical approach for open dialog is to provide a statistical language model (or SLM) and use that to anticipate likely user dialog.

Relevant Use Cases Include: U1 Voice Web Search, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog,

U6 Speech UI present when no visible UI need be present, U7 Rerecognition, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems.

A.3.1.13 R13. Web application author should have ability to customize speech recognition graphical user interface

Multimodal speech recognition apps are typically accompanied by a GUI experience to (i) provide a means to invoke SR; and (ii) indicate progress of recognition through various states (listening to the user speak; waiting for the recognition result; displaying errors; displaying alternates; etc). Polished applications generally have their own GUI design for the speech experience. This will usually include a clickable graphic to invoke speech recognition, and graphics to indicate the progress of the recognition through various states.

Relevant Use Cases Include: U1 Voice Web Search, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U6 Speech UI present when no visible UI need be present, U7 Rerecognition, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.1.14 R14. Web application authors need a way to specify and effectively create barge-in (interrupt audio and synthesis)

The ability to stop output (text-to-speech or media) in response to events (user starting to speak, a recognition occurring, other events or selections or browser interactions, etc.) so that the web application user experience is acceptable and the web application doesn't appear confused or deaf to user input. While barge-in aids the usability of an application by allowing the user provide spoken input even while the application is playing media/TTS. However, applications that both speak (or play media) and listen at the same time can potentially interfere with their own speech recognition. In telephony, this is less of a problem due to the design of the handset, and built-in echo-cancelling technology. However, with broad variety of HTML-capable devices, situations that involve open-mic and open-speaker will be potentially more common. To help developers cope with this, it may be useful to either specify a minimum barge-in capability that all browsers should meet, or make it easier for developers to discover when barge-in may be an issue and allow appropriate parameter settings to help mitigate the situation.

Relevant Use Cases Include: U8 Voice Activity Detection, U9 Temporal Structure of Synthesis to Provide Visual Feedback, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction.

A.3.2 Web Author Convenience and Quality

The following requirements do not provide any additional functionality to an HTML web author, but instead make the task of authoring a speech HTML page much easier or else convert the authored application into a much more high quality application.

A.3.2.1 R15. Web application authors must not need run their own speech service

Running a speech service can be difficult, and a default speech interface/service is needed in the user agent so that a web application author can use speech resources without needing to run their own speech service.

Relevant Use Cases Include: U2 Speech Command Interface, U3 Domain Specific Grammars Contingent on Earlier Inputs, U5 Domain Specific Grammars Filling Multiple Input Fields,

A.3.2.2 R16. Web application authors must not be excluded from running their own speech service

Running a speech service can provide fine grained customization of the application for the web application author.

Relevant Use Cases Include: U1 Voice Web Search, U2 Speech Command Interface, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U5 Domain Specific Grammars Filling Multiple Input Fields, U6 Speech UI present when no visible UI need be present, U7 Rerecognition, U8 Voice Activity Detection, U9 Temporal Structure of Synthesis to Provide Visual Feedback, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.2.3 R17. User perceived latency of recognition must be minimized

The time between the user completing their utterance, and an application providing a response, needs to fall below an acceptable threshold to be usable. For example, "find a flight from New York to San Francisco on Monday morning returning Friday afternoon" takes about 6 seconds to say, but the user still expects a response within a couple of seconds (generally somewhere between 500 and 3000 milliseconds, depending on the specific application and audience). In the case of applications/browsers that invoke speech recognition over a network, the platform needs to support (i) using a codec that can be transmitted in real-time on the modest bandwidth of many cell networks and (ii) transmitting the user's utterance in real-time (e.g. in 100ms packets) rather than collect the full utterance before transmitting any of it. For applications where the utterances are non-trivial and the grammars can be

recognized in real-time or better, real-time streaming can all but eliminate user-perceived latency.

Relevant Use Cases Include: U1 Voice Web Search, U2 Speech Command Interface, U4 Continuous Recognition of Open Dialog, U5 Domain Specific Grammars Filling Multiple Input Fields, U6 Speech UI present when no visible UI need be present, U7 Rerecognition, U8 Voice Activity Detection, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.2.4 R18. User perceived latency of synthesis must be minimized

For longer stretches of spoken output, it may be necessary to stream the synthesis without knowing the full rendering of the TTS/SSML nor the Content-Length of the rendered audio format. To enable high quality applications user agents should support streaming of synthesis results without needing the content length header to be present with a correct full synthesized file length. I.e., consider a TTS processor that can process one sentence at a time, and is requested to read an email consisting of three paragraphs.

Relevant Use Cases Include: U9 Temporal Structure of Synthesis to Provide Visual Feedback, U10 Hello World Use Case, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.2.5 R19. End user extensions should be available both on desktop and in cloud

End-user extensions should be accessible either from the desktop or from the cloud.

Relevant Use Cases Include:

A.3.2.6 R20. Web author selected TTS service should be available both on device and in the cloud

It should be possible to specify a target TTS engine not only via the "URI" attribute, but via a more generic "source" attribute, which can point to a local TTS engine as well. To achieve this, it'd be useful to think about extendability and flexibility of the framework, so that it is easy for third parties to provide high quality TTS engines.

Relevant Use Cases Include: U9 Temporal Structure of Synthesis to Provide Visual Feedback, U10 Hello World Use Case, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.2.7 R21. Any public interface for creating extensions should be speakable

Any public interfaces for creating extensions should be "speakable". A user should never need to touch the keyboard in order to expand a grammar, reference data, or add functionality.

Relevant Use Cases Include:

A.3.2.8 R22. Web application author wants to provide a consistent user experience across all modalities

A developer creating a (multimodal) interface combining speech input with graphical output needs to have the ability to provide a consistent user experience not just for graphical elements but also for voice. In addition, high quality speech applications often involve a lot of tuning of recognition parameters and grammars to work with different recognition technologies. A web author may wish for her application to only need to tune the speech recognition with one technology stack, and not have to tune and special case different grammars and parameters for different user agents. There exists enough browser detection in the web developer world to deal with accidental incompatibility and legacy implementations without causing speech to require it by design for quality speech recognition. This is one reason to allow author specified networked speech services.

Relevant Use Cases Include: U1 Voice Web Search, U2 Speech Command Interface, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U5 Domain Specific Grammars Filling Multiple Input Fields, U6 Speech UI present when no visible UI need be present, U7 Rerecognition, U8 Voice Activity Detection, U9 Temporal Structure of Synthesis to Provide Visual Feedback, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.2.9 R23. Speech as an input on any application should be able to be optional

If the user can't speak, can't speak the language well enough to be recognized, the speech recognizer just doesn't work well for them, or they are in an environment where speaking would be inappropriate they should be able to interact with the web application some other way.

Relevant Use Cases Include: U14 Multimodal Interaction.

A.3.2.10 R24. End user should be able to use speech in a hands-free mode

There should be a way to speech-enable every aspect of a web application that you would do with a mouse, a touchscreen, or by typing.

Relevant Use Cases Include: U1 Voice Web Search, U2 Speech Command Interface, U4 Continuous Recognition of Open Dialog, U8 Voice Activity Detection, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.2.11 R25. It should be easy to extend the standard without effecting existing speech applications

If recognizers support new capabilities like language detection or gender detection, it should be easy to add the results of those new capabilities to the speech recognition result, without requiring a new version of the standard.

Relevant Use Cases Include: U13 Dialog Systems, U14 Multimodal Interaction.

A.3.2.12 R26. There should exist a high quality default speech recognition visual user interface

Multimodal speech recognition apps are typically accompanied by a GUI experience to (i) provide a means to invoke SR; and (ii) indicate progress of recognition through various states (listening to the user speak; waiting for the recognition result; displaying errors; displaying alternates; etc). Many applications, at least in their initial development, and in some cases the finished product, will not implement their own GUI for controlling speech recognition. These applications will rely on the browser to implement a default control to begin speech recognition, such as a GUI button on the screen or a physical button on the device, keyboard or microphone. They will also rely on a default GUI to indicate the state of recognition (listening, waiting, error, etc).

Relevant Use Cases Include: U1 Voice Web Search, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U6 Speech UI present when no visible UI need be present, U13 Dialog Systems, U14 Multimodal Interaction.

A.3.2.13 R27. Grammars, TTS, media composition, and recognition results should all use standard formats

No developer likes to be locked into a particular vendor's implementation. In some cases this will be unavoidable due to differentiation in capabilities between vendors. But general concepts like grammars, TTS and media composition, and recognition results should use standard formats (e.g. SRGS, SSML, SMIL, EMMA).

Relevant Use Cases Include: U1 Voice Web Search, U2 Speech Command Interface, U3 Domain Specific Grammars Contingent on Earlier Inputs, U4 Continuous Recognition of Open Dialog, U5 Domain Specific Grammars Filling Multiple Input Fields, U7 Rerecognition, U9 Temporal Structure of Synthesis to

Provide Visual Feedback, U11 Speech Translation, U12 Speech Enabled Email Client, U13 Dialog Systems, U14 Multimodal Interaction, U15 Speech Driving Directions.

A.3.3 Security and Privacy Requirements

These requirements have to do with security, privacy, and user expectations. These often don't have specific use cases, and maybe mitigations should be explored in appropriate user agent permissions to allow some of these actions on certain trusted sites while forbidding them on others.

A.3.3.1 R28. Web application must not be allowed access to raw audio

Some users may be concerned if their audio may be recorded and then controlled by the web application author so user agents must prevent this.

Relevant Use Cases Include:

A.3.3.2 R29. Web application may only listen in response to user action

Some users may be concerned if their audio may be recognized without being aware of it so user agents must make sure that recognition only occurs in response to explicit end user actions.

Relevant Use Cases Include:

A.3.3.3 R30. End users should not be forced to store anything about their speech recognition environment in the cloud

For reasons of privacy, the user should not be forced to store anything about their speech recognition environment on the cloud.

Relevant Use Cases Include:

A.3.3.4 R31. End users, not web application authors, should be the ones to select speech recognition resources

Selection of the speech engine should be a user-setting in the browser, not a Web developer setting. The security bar is much higher for an audio recording solution that can be pointed at an arbitrary destination.

Relevant Use Cases Include:

A.3.3.5 R32. End users need a clear indication whenever microphone is listening to the user

Many users are sensitive about who or what is listening to them, and will not tolerate an application that listens to the user without the user's knowledge. A browser needs to provide clear indication to the user either whenever it will listen to the user or whenever it is using a microphone to listen to the user.

Relevant Use Cases Include:

A.3.3.6 R33. User agents need a way to enable end users to grant permission to an application to listen to them

Some users will want to explicitly grant permission for the user agent, or an application, to listen to them. Whether this is a setting that is global, applies to a subset of applications/domains, etc, depends somewhat on the security & privacy expectations of the user agent's customers.

Relevant Use Cases Include:

A.3.3.7 R34. A trust relation is needed between end user and whatever is doing recognition

The user also needs to be able to trust and verify that their utterance is processed by the application that's on the screen (or its backend servers), or at least by a service the user trusts.

Relevant Use Cases Include:

B. Acknowledgements

This report was developed by the HTML Speech XG.

Special thanks to the members of the XG: Andrei Popescu, Andy Mauro, Björn Bringert, Chaitanya Gharpure, Charles Chen, Dan Druta, Daniel Burnett, Dave Burke, David Bolter, Deborah Dahl, Fabio Paternò, Glen Shires, Ingmar Kliche, Jerry Carter, Jim Larson, Kazuyuki Ashimura, Marc Schröder, Markus Gylling, Masahiro Araki, Matt Womer, Michael Bodell, Michael Johnston, Milan Young, Olli Pettay, Paolo Baggia, Patrick Ehlen, Raj Tumuluri, Rania Elnaggar, Ravi Reddy, Robert Brown, Satish Kumar Sampath, Somnath Chandra, and T.V. Raman.

C. References

C.1 Normative references

[EMMA]

Michael Johnston. *EMMA: Extensible MultiModal Annotation markup language.* 10 February 2009. W3C Recommendation. URL:

http://www.w3.org/TR/2009/REC-emma-20090210

[HTML5]
Ian Hickson; David Hyatt. *HTML5.* 25 May 2011. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/html5

[MRCPv2]
Burnett, D. Shanmugham, S. *Media Resource Control Protocol Version 2* 15 November 2011. URL: http://tools.ietf.org/html/draft-ietf-speechsc-mrcpv2-27

[NTP]
D. Mills. *Network Time Protocol (Version 3)*. March 1992. IETF RFC 1305. URL: http://www.ietf.org/rfc/rfc1305.txt

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* March 1997. Internet RFC 2119. URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC3339]
G. Klyne, C. Newman. *Date and Time on the Internet: Timestamps*. July 2002. Internet RFC 3339. URL: http://www.ietf.org/rfc/rfc3339.txt

[RFC3555]
S. Casner; P. Hoschka. *MIME Type Registration of RTP Payload Formats*. July 2003. Internet RFC 3555. URL: http://www.rfc-editor.org/rfc/rfc3555.txt

[RFC5646]
A. Phillips, M. Davis. *Tags for Identifying Languages.* September 2009. Internet RFC 5646. URL: http://www.rfc-editor.org/rfc/rfc5646.txt

[SPEECH-GRAMMAR]
Andrew Hunt; Scott McGlashan. *Speech Recognition Grammar Specification Version 1.0.* 16 March 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-speech-grammar-20040316

[SPEECH-SYNTHESIS]
Daniel C. Burnett; Mark R. Walker; Andrew Hunt. *Speech Synthesis Markup Language (SSML) Version 1.0.* 7 September 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-speech-synthesis-20040907

[WEBSOCKETS-PROTOCOL]
C. Holmberg, S. Hakansson, G. Eriksson. *The WebSocket protocol.* URL: http://tools.ietf.org/id/draft-ietf-hybi-thewebsocketprotocol-09.txt

C.2 Informative references

[HTTP11]

R. Fielding; et al. *Hypertext Transfer Protocol - HTTP/1.1.* June 1999. Internet RFC 2616. URL: http://www.ietf.org/rfc/rfc2616.txt

[TLS]

T. Dierks, E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.2.* August 2008. Internet RFC 5246. URL: http://tools.ietf.org/html/rfc5246

[WEBSOCKETS-API]

I. Hickson. *The WebSocket API.* 29 September 2011. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2011/WD-websockets-20110929/