Web Speech API

Draft Community Group Report, 21 January 2020

This version:
https://wicg.github.io/speech-api/

Issue Tracking:
GitHub
Inline In Spec

Editors:
André Natal (Mozilla)
Glen Shires (Google)
Marcos Cáceres (Mozilla)
Philip Jägenstedt (Google)

Former Editor:
Hans Wennborg (Google)

Tests:
web-platform-tests speech-api/ (ongoing work)

Abstract

This specification defines a JavaScript API to enable web developers to incorporate speech recognition and synthesis into their web pages. It enables developers to use scripting to generate text-to-speech output and to use speech recognition as an input for forms, continuous dictation and control. The JavaScript API allows web pages to control activation and timing and to handle results and alternatives.

Status of this document

This specification was published by the Web Platform Incubator Community Group. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C

Community Contributor License Agreement (CLA) there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.

Table of Contents

1 Introduction

2 Use Cases

3 Security and privacy considerations

3.1 Implementation considerations

4 API Description

4.1 The SpeechRecognition Interface

4.1.1 SpeechRecognition Attributes

4.1.2 SpeechRecognition Methods

4.1.3 SpeechRecognition Events

4.1.4 SpeechRecognitionErrorEvent

4.1.5 SpeechRecognitionAlternative

4.1.6 SpeechRecognitionResult

4.1.7 SpeechRecognitionResultList

4.1.8 SpeechRecognitionEvent

4.1.9 SpeechGrammar

4.1.10 SpeechGrammarList

4.2 The SpeechSynthesis Interface

4.2.1 SpeechSynthesis Attributes

4.2.2 SpeechSynthesis Methods

4.2.3 SpeechSynthesis Events

4.2.4 SpeechSynthesisUtterance Attributes

4.2.5 SpeechSynthesisUtterance Events

4.2.6 SpeechSynthesisEvent Attributes

4.2.7 SpeechSynthesisErrorEvent Attributes

4.2.8 SpeechSynthesisVoice Attributes

5 Examples

5.1 Speech Recognition Examples
5.2 Speech Synthesis Examples

Acknowledgments

Conformance

Index
Terms defined by this specification
Terms defined by reference

References
Normative References
Informative References

IDL Index

Issues Index

§ 1. Introduction

This section is non-normative.

The Web Speech API aims to enable web developers to provide, in a web browser, speech-input and text-to-speech output features that are typically not available when using standard speech-recognition or screen-reader software. The API itself is agnostic of the underlying speech recognition and synthesis implementation and can support both server-based and client-based/embedded recognition and synthesis. The API is designed to enable both brief (one-shot) speech input and continuous speech input. Speech recognition results are provided to the web page as a list of hypotheses, along with other relevant information for each hypothesis.

This specification is a subset of the API defined in the HTML Speech Incubator Group Final Report. That report is entirely informative since it is not a standards track document. All portions of that report may be considered informative with regards to this document, and provide an informative background to this document. This specification is a fully-functional subset of that report. Specifically, this subset excludes the underlying transport protocol, the proposed additions to HTML markup, and it defines a simplified subset of the JavaScript API. This subset supports the majority of use-cases and

sample code in the Incubator Group Final Report. This subset does not preclude future standardization of additions to the markup, API or underlying transport protocols, and indeed the Incubator Report defines a potential roadmap for such future work.

§ 2. Use Cases

This section is non-normative.

This specification supports the following use cases, as defined in Section 4 of the Incubator Report.

- Voice Web Search
- Speech Command Interface
- Domain Specific Grammars Contingent on Earlier Inputs
- Continuous Recognition of Open Dialog
- Domain Specific Grammars Filling Multiple Input Fields
- Speech UI present when no visible UI need be present
- Voice Activity Detection
- Temporal Structure of Synthesis to Provide Visual Feedback
- Hello World
- Speech Translation
- Speech Enabled Email Client
- Dialog Systems
- Multimodal Interaction
- Speech Driving Directions
- Multimodal Video Game
- Multimodal Search

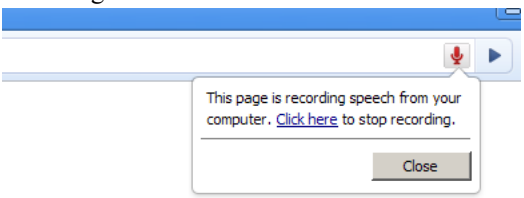
To keep the API to a minimum, this specification does not directly support the following use case. This does not preclude adding support for this as a future API enhancement, and indeed the Incubator report provides a roadmap for doing so.

- Rerecognition

Note that for many usages and implementations, it is possible to avoid the need for Rerecognition by using a larger grammar, or by combining multiple grammars — both of these techniques are supported in this specification.

§ 3. Security and privacy considerations

1. User agents must only start speech input sessions with explicit, informed user consent. User consent can include, for example:
    - User click on a visible speech input element which has an obvious graphical representation showing that it will start speech input.
    - Accepting a permission prompt shown as the result of a call to `SpeechRecognition.start`.
    - Consent previously granted to always allow speech input for this web page.
2. User agents must give the user an obvious indication when audio is being recorded.
    - In a graphical user agent, this could be a mandatory notification displayed by the user agent as part of its chrome and not accessible by the web page. This could for example be a pulsating/blinking record icon as part of the browser chrome/address bar, an indication in the status bar, an audible notification, or anything else relevant and accessible to the user. This UI element must also allow the user to stop recording.



    - In a speech-only user agent, the indication may for example take the form of the system speaking the label of the speech input element, followed by a short beep.
3. The user agent may also give the user a longer explanation the first time speech input is used, to let the user know what it is and how they can tune their privacy settings to disable speech recording if required.

§ 3.1. Implementation considerations

This section is non-normative.

1. Spoken password inputs can be problematic from a security perspective, but it is up to the user to decide if they want to speak their password.

2. Speech input could potentially be used to eavesdrop on users. Malicious webpages could use tricks such as hiding the input element or otherwise making the user believe that it has stopped recording speech while continuing to do so. They could also potentially style the input element to appear as something else and trick the user into clicking them. An example of styling the file input element can be seen at https://www.quirksmode.org/dom/inputfile.html. The above recommendations are intended to reduce this risk of such attacks.

§ 4. API Description

This section is normative.

§ 4.1. The SpeechRecognition Interface

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

The term "final result" indicates a SpeechRecognitionResult in which the final attribute is true. The term "interim result" indicates a SpeechRecognitionResult in which the final attribute is false.

```
[Exposed=Window]
interface SpeechRecognition : EventTarget {
    constructor();

    // recognition parameters
    attribute SpeechGrammarList grammars;
    attribute DOMString lang;
    attribute boolean continuous;
    attribute boolean interimResults;
    attribute unsigned long maxAlternatives;

    // methods to drive the speech interaction
    void start();
    void stop();
    void abort();

    // event methods
    attribute EventHandler onaudiostart;
    attribute EventHandler onsoundstart;
    attribute EventHandler onspeechstart;
    attribute EventHandler onspeechend;
    attribute EventHandler onsoundend;
    attribute EventHandler onaudioend;
    attribute EventHandler onresult;
    attribute EventHandler onnomatch;
    attribute EventHandler onerror;
    attribute EventHandler onstart;
    attribute EventHandler onend;
};

enum SpeechRecognitionErrorCode {
    "no-speech",
    "aborted",
    "audio-capture",
    "network",
    "not-allowed",
    "service-not-allowed",
    "bad-grammar",
    "language-not-supported"
};

[Exposed=Window]
interface SpeechRecognitionErrorEvent : Event {
```

```
    constructor(DOMString type, SpeechRecognitionErrorEventInit eventInitD
    readonly attribute SpeechRecognitionErrorCode error;
    readonly attribute DOMString message;
};

dictionary SpeechRecognitionErrorEventInit : EventInit {
    required SpeechRecognitionErrorCode error;
    DOMString message = "";
};

// Item in N-best list
[Exposed=Window]
interface SpeechRecognitionAlternative {
    readonly attribute DOMString transcript;
    readonly attribute float confidence;
};

// A complete one-shot simple response
[Exposed=Window]
interface SpeechRecognitionResult {
    readonly attribute unsigned long length;
    getter SpeechRecognitionAlternative item(unsigned long index);
    readonly attribute boolean isFinal;
};

// A collection of responses (used in continuous mode)
[Exposed=Window]
interface SpeechRecognitionResultList {
    readonly attribute unsigned long length;
    getter SpeechRecognitionResult item(unsigned long index);
};

// A full response, which could be interim or final, part of a continuous
[Exposed=Window]
interface SpeechRecognitionEvent : Event {
    constructor(DOMString type, SpeechRecognitionEventInit eventInitDict);
    readonly attribute unsigned long resultIndex;
    readonly attribute SpeechRecognitionResultList results;
};

dictionary SpeechRecognitionEventInit : EventInit {
    unsigned long resultIndex = 0;
    required SpeechRecognitionResultList results;
};
```

```
// The object representing a speech grammar
[Exposed=Window]
interface SpeechGrammar {
    attribute DOMString src;
    attribute float weight;
};

// The object representing a speech grammar collection
[Exposed=Window]
interface SpeechGrammarList {
    constructor();
    readonly attribute unsigned long length;
    getter SpeechGrammar item(unsigned long index);
    void addFromURI(DOMString src,
                    optional float weight = 1.0);
    void addFromString(DOMString string,
                    optional float weight = 1.0);
};
```

4.1.1. SpeechRecognition Attributes

grammars **attribute, of type SpeechGrammarList**

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

lang **attribute, of type DOMString**

This attribute will set the language of the recognition for the request, using a valid BCP 47 language tag. [BCP47] If unset it remains unset for getting in script, but will default to use the language of the html document root element and associated hierarchy. This default value is computed and used when the input request opens a connection to the recognition service.

continuous **attribute, of type boolean**

When the continuous attribute is set to false, the user agent must return no more than one final result in response to starting recognition, for example a single turn pattern of interaction. When the continuous attribute is set to true, the user agent must return zero or more final results representing multiple consecutive recognitions in response to starting recognition, for example a dictation. The default value must be false. Note, this attribute setting does not affect interim results.

interimResults **attribute, of type boolean**

Controls whether interim results are returned. When set to true, interim results should be returned. When set to false, interim results must not be returned. The default value must be false. Note, this attribute setting does not affect final results.

***`maxAlternatives`* attribute, of type unsigned long**

This attribute will set the maximum number of `SpeechRecognitionAlternative`s per result. The default value is 1.

> ISSUE 1 The group has discussed whether WebRTC might be used to specify selection of audio sources and remote recognizers. See Interacting with WebRTC, the Web Audio API and other external sources thread on public-speech-api@w3.org.

§ 4.1.2. SpeechRecognition Methods

***`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. Once the system is successfully listening to the recognition the user agent must raise a start event. If the start method is called on an already started object (that is, start has previously been called, and no error or end event has fired on the object), the user agent must throw an "`InvalidStateError`" `DOMException` and ignore the call.

***`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 already received for this recognition. 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 already collected for this recognition. If the stop method is called on an object which is already stopped or being stopped (that is, start was never called on it, the end or error event has fired on it, or stop was previously called on it), the user agent must ignore the call.

***`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 abort method is called, the speech service must stop recognizing. The user agent must raise an end event once the speech service is no longer connected. If the abort method is called on an object which is already stopped or aborting (that is, start was never called on it, the end or error event has fired on it, or abort was previously called on it), the user agent must ignore the call.

§ 4.1.3. SpeechRecognition 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 SpeechRecognition interface also contains convenience attributes for registering a single event handler for each event type. The events do not bubble and are not cancelable.

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**
: Fired when some sound, possibly speech, has been detected. This must be fired with low latency, e.g. by using a client-side energy detector. The audiostart event must always have been fired before the soundstart event.

***speechstart* event**
: Fired when the speech that will be used for speech recognition has started. The audiostart event must always have been fired before the speechstart event.

***speechend* event**
: Fired when the speech that will be used for speech recognition has ended. The speechstart event must always have been fired before speechend.

***soundend* event**

Fired when some sound is no longer detected. This must be fired with low latency, e.g. by using a client-side energy detector. The soundstart event must always have been fired before soundend.

***audioend* event**

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

***result* event**

Fired when the speech recognizer returns a result. The event must use the `SpeechRecognitionEvent` interface. The audiostart event must always have been fired before the result event.

***nomatch* event**

Fired when the speech recognizer returns a final result with no recognition hypothesis that meet or exceed the confidence threshold. The event must use the `SpeechRecognitionEvent` interface. The `results` attribute in the event may contain speech recognition results that are below the confidence threshold or may be null. The `audiostart` event must always have been fired before the nomatch event.

***error* event**

Fired when a speech recognition error occurs. The event must use the `SpeechRecognitionErrorEvent` interface.

***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.

§ 4.1.4. SpeechRecognitionErrorEvent

The `SpeechRecognitionErrorEvent` interface is used for the error event.

***error* attribute, of type SpeechRecognitionErrorCode, readonly**

The errorCode is an enumeration indicating what has gone wrong. The values are:

"no-speech"

No speech was detected.

"aborted"

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

"audio-capture"
: Audio capture failed.

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

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

"service-not-allowed"
: 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"
: There was an error in the speech recognition grammar or semantic tags, or the grammar format or semantic tag format is unsupported.

"language-not-supported"
: The language was not supported.

message **attribute, of type DOMString, readonly**
: 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.

§ 4.1.5. SpeechRecognitionAlternative

The SpeechRecognitionAlternative represents a simple view of the response that gets used in a n-best list.

transcript **attribute, of type DOMString, readonly**
: The transcript string represents the raw words that the user spoke. For continuous recognition, leading or trailing whitespace MUST be included where necessary such that concatenation of consecutive SpeechRecognitionResults produces a proper transcript of the session.

confidence **attribute, of type float, readonly**
: 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.

ISSUE 2 The group has discussed whether confidence can be specified in a speech-recognition-engine-independent manner and whether confidence threshold and nomatch should be included, because this is not a dialog API. See Confidence property thread on public-speech-api@w3.org.

§ 4.1.6. SpeechRecognitionResult

The SpeechRecognitionResult 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.

***length* attribute, of type unsigned long, readonly**

The long attribute represents how many n-best alternatives are represented in the item array.

***item(index)* getter**

The item getter returns a SpeechRecognitionAlternative from the index into an array of n-best values. If index is greater than or equal to length, this returns null. 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).

***isFinal* attribute, of type boolean, readonly**

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.

§ 4.1.7. SpeechRecognitionResultList

The SpeechRecognitionResultList 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* attribute, of type unsigned long, readonly**

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

***item(index)* getter**

The item getter returns a SpeechRecognitionResult from the index into an array of result values. If index is greater than or equal to length, this returns null. The user agent must ensure that the length attribute is set to the number of elements in the array.

§ 4.1.8. SpeechRecognitionEvent

The SpeechRecognitionEvent is the event that is raised each time there are any changes to interim or final results.

resultIndex **attribute, of type unsigned long, readonly**

The resultIndex must be set to the lowest index in the "results" array that has changed.

results **attribute, of type SpeechRecognitionResultList, readonly**

The array of all current recognition results for this session. Specifically all final results that have been returned, followed by the current best hypothesis for all interim results. It must consist of zero or more final results followed by zero or more interim results. On subsequent SpeechRecognitionResultEvent events, interim results may be overwritten by a newer interim result or by a final result or may be removed (when at the end of the "results" array and the array length decreases). Final results must not be overwritten or removed. All entries for indexes less than resultIndex must be identical to the array that was present when the last SpeechRecognitionResultEvent was raised. All array entries (if any) for indexes equal or greater than resultIndex that were present in the array when the last SpeechRecognitionResultEvent was raised are removed and overwritten with new results. The length of the "results" array may increase or decrease, but must not be less than resultIndex. Note that when resultIndex equals results.length, no new results are returned, this may occur when the array length decreases to remove one or more interim results.

§ 4.1.9. SpeechGrammar

The SpeechGrammar object represents a container for a grammar.

> ISSUE 3 The group has discussed options for which grammar formats should be supported, how builtin grammar types are specified, and default grammars when not specified. See Default value of SpeechRecognition.grammars thread on public-speech-api@w3.org.

This structure has the following attributes:

src **attribute, of type DOMString**

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, of type float**

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.

§ 4.1.10. SpeechGrammarList

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

***`length`* attribute, of type unsigned long, readonly**

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

***`item(`index`)`* getter**

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.

***`addFromURI(`src, weight`)`* 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. The *weight* parameter represents this grammar's weight relative to the other grammar.

***`addFromString(`string, weight`)`* 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. This content should be encoded into a data: URI when the SpeechGrammar object is created. The *weight* parameter represents this grammar's weight relative to the other grammar.

§ 4.2. The SpeechSynthesis Interface

The SpeechSynthesis interface is the scripted web API for controlling a text-to-speech output.

```
[Exposed=Window]
interface SpeechSynthesis : EventTarget {
    readonly attribute boolean pending;
    readonly attribute boolean speaking;
    readonly attribute boolean paused;

    attribute EventHandler onvoiceschanged;

    void speak(SpeechSynthesisUtterance utterance);
    void cancel();
    void pause();
    void resume();
    sequence<SpeechSynthesisVoice> getVoices();
};

partial interface Window {
    [SameObject] readonly attribute SpeechSynthesis speechSynthesis;
};

[Exposed=Window]
interface SpeechSynthesisUtterance : EventTarget {
    constructor(optional DOMString text);

    attribute DOMString text;
    attribute DOMString lang;
    attribute SpeechSynthesisVoice? voice;
    attribute float volume;
    attribute float rate;
    attribute float pitch;

    attribute EventHandler onstart;
    attribute EventHandler onend;
    attribute EventHandler onerror;
    attribute EventHandler onpause;
    attribute EventHandler onresume;
    attribute EventHandler onmark;
    attribute EventHandler onboundary;
};

[Exposed=Window]
interface SpeechSynthesisEvent : Event {
    constructor(DOMString type, SpeechSynthesisEventInit eventInitDict);
    readonly attribute SpeechSynthesisUtterance utterance;
```

```
    readonly attribute unsigned long charIndex;
    readonly attribute unsigned long charLength;
    readonly attribute float elapsedTime;
    readonly attribute DOMString name;
};

dictionary SpeechSynthesisEventInit : EventInit {
    required SpeechSynthesisUtterance utterance;
    unsigned long charIndex = 0;
    unsigned long charLength = 0;
    float elapsedTime = 0;
    DOMString name = "";
};

enum SpeechSynthesisErrorCode {
    "canceled",
    "interrupted",
    "audio-busy",
    "audio-hardware",
    "network",
    "synthesis-unavailable",
    "synthesis-failed",
    "language-unavailable",
    "voice-unavailable",
    "text-too-long",
    "invalid-argument",
    "not-allowed",
};

[Exposed=Window]
interface SpeechSynthesisErrorEvent : SpeechSynthesisEvent {
    constructor(DOMString type, SpeechSynthesisErrorEventInit eventInitDic
    readonly attribute SpeechSynthesisErrorCode error;
};

dictionary SpeechSynthesisErrorEventInit : SpeechSynthesisEventInit {
    required SpeechSynthesisErrorCode error;
};

[Exposed=Window]
interface SpeechSynthesisVoice {
    readonly attribute DOMString voiceURI;
    readonly attribute DOMString name;
    readonly attribute DOMString lang;
```

```
    readonly attribute boolean localService;
    readonly attribute boolean default;
};
```

§ 4.2.1. SpeechSynthesis Attributes

pending **attribute, of type boolean, readonly**

This attribute is true if the queue for the global SpeechSynthesis instance contains any utterances which have not started speaking.

speaking **attribute, of type boolean, readonly**

This attribute is true if an utterance is being spoken. Specifically if an utterance has begun being spoken and has not completed being spoken. This is independent of whether the global SpeechSynthesis instance is in the paused state.

paused **attribute, of type boolean, readonly**

This attribute is true when the global SpeechSynthesis instance is in the paused state. This state is independent of whether anything is in the queue. The default state of a the global SpeechSynthesis instance for a new window is the non-paused state.

§ 4.2.2. SpeechSynthesis Methods

speak(utterance) **method**

This method appends the SpeechSynthesisUtterance object *utterance* to the end of the queue for the global SpeechSynthesis instance. It does not change the paused state of the SpeechSynthesis instance. If the SpeechSynthesis instance is paused, it remains paused. If it is not paused and no other utterances are in the queue, then this utterance is spoken immediately, else this utterance is queued to begin speaking after the other utterances in the queue have been spoken. If changes are made to the SpeechSynthesisUtterance object after calling this method and prior to the corresponding end or error event, it is not defined whether those changes will affect what is spoken, and those changes may cause an error to be returned. The SpeechSynthesis object takes exclusive ownership of the SpeechSynthesisUtterance object. Passing it as a speak() argument to another SpeechSynthesis object should throw an exception. (For example, two frames may have the same origin and each will contain a SpeechSynthesis object.)

cancel() **method**

This method removes all utterances from the queue. If an utterance is being spoken, speaking ceases immediately. This method does not change the paused state of the global SpeechSynthesis instance.

***pause()* method**

This method puts the global SpeechSynthesis instance into the paused state. If an utterance was being spoken, it pauses mid-utterance. (If called when the SpeechSynthesis instance was already in the paused state, it does nothing.)

***resume()* method**

This method puts the global SpeechSynthesis instance into the non-paused state. If an utterance was speaking, it continues speaking the utterance at the point at which it was paused, else it begins speaking the next utterance in the queue (if any). (If called when the SpeechSynthesis instance was already in the non-paused state, it does nothing.)

***getVoices()* method**

This method returns the available voices. It is user agent dependent which voices are available. If there are no voices available, or if the the list of available voices is not yet known (for example: server-side synthesis where the list is determined asynchronously), then this method must return a SpeechSynthesisVoiceList of length zero.

§ 4.2.3. SpeechSynthesis Events

***voiceschanged* event**

Fired when the contents of the SpeechSynthesisVoiceList, that the getVoices method will return, have changed. Examples include: server-side synthesis where the list is determined asynchronously, or when client-side voices are installed/uninstalled.

§ 4.2.4. SpeechSynthesisUtterance Attributes

***text* attribute, of type DOMString**

This attribute specifies the text to be synthesized and spoken for this utterance. This may be either plain text or a complete, well-formed SSML document. [SSML] For speech synthesis engines that do not support SSML, or only support certain tags, the user agent or speech engine must strip away the tags they do not support and speak the text. There may be a maximum length of the text, it may be limited to 32,767 characters.

***lang* attribute, of type DOMString**

This attribute specifies the language of the speech synthesis for the utterance, using a valid BCP 47 language tag. [BCP47] If unset it remains unset for getting in script, but will default to use the language of the html document root element and associated hierarchy. This default value is computed and used when the input request opens a connection to the recognition service.

***voice* attribute, of type SpeechSynthesisVoice, nullable**

This attribute specifies the speech synthesis voice that the web application wishes to use. When a `SpeechSynthesisUtterance` object is created this attribute must be initialized to null. If, at the time of the `speak()` method call, this attribute has been set to one of the `SpeechSynthesisVoice` objects returned by `getVoices()`, then the user agent must use that voice. If this attribute is unset or null at the time of the `speak()` method call, then the user agent must use a user agent default voice. The user agent default voice should support the current language (see `lang`) and can be a local or remote speech service and can incorporate end user choices via interfaces provided by the user agent such as browser configuration parameters.

***volume* attribute, of type float**

This attribute specifies the speaking volume for the utterance. It ranges between 0 and 1 inclusive, with 0 being the lowest volume and 1 the highest volume, with a default of 1. If SSML is used, this value will be overridden by prosody tags in the markup.

***rate* attribute, of type float**

This attribute specifies the speaking rate for the utterance. It is relative to the default rate for this voice. 1 is the default rate supported by the speech synthesis engine or specific voice (which should correspond to a normal speaking rate). 2 is twice as fast, and 0.5 is half as fast. Values below 0.1 or above 10 are strictly disallowed, but speech synthesis engines or specific voices may constrain the minimum and maximum rates further, for example, a particular voice may not actually speak faster than 3 times normal even if you specify a value larger than 3. If SSML is used, this value will be overridden by prosody tags in the markup.

***pitch* attribute, of type float**

This attribute specifies the speaking pitch for the utterance. It ranges between 0 and 2 inclusive, with 0 being the lowest pitch and 2 the highest pitch. 1 corresponds to the default pitch of the speech synthesis engine or specific voice. Speech synthesis engines or voices may constrain the minimum and maximum rates further. If SSML is used, this value will be overridden by prosody tags in the markup.

§ 4.2.5. SpeechSynthesisUtterance Events

Each of these events must use the `SpeechSynthesisEvent` interface, except the error event which must use the `SpeechSynthesisErrorEvent` interface. These events bubble up to SpeechSynthesis.

***start* event**

Fired when this utterance has begun to be spoken.

***end* event**

Fired when this utterance has completed being spoken. If this event fires, the error event must not be fired for this utterance.

***error* event**

Fired if there was an error that prevented successful speaking of this utterance. If this event fires, the end event must not be fired for this utterance.

***pause* event**

Fired when and if this utterance is paused mid-utterance.

***resume* event**

Fired when and if this utterance is resumed after being paused mid-utterance. Adding the utterance to the queue while the global SpeechSynthesis instance is in the paused state, and then calling the resume method does not cause the resume event to be fired, in this case the utterance's start event will be called when the utterance starts.

***mark* event**

Fired when the spoken utterance reaches a named "mark" tag in SSML. [SSML] The user agent must fire this event if the speech synthesis engine provides the event.

***boundary* event**

Fired when the spoken utterance reaches a word or sentence boundary. The user agent must fire this event if the speech synthesis engine provides the event.

§ 4.2.6. SpeechSynthesisEvent Attributes

***utterance* attribute, of type SpeechSynthesisUtterance, readonly**

This attribute contains the SpeechSynthesisUtterance that triggered this event.

***charIndex* attribute, of type unsigned long, readonly**

This attribute indicates the zero-based character index into the original utterance string that most closely approximates the current speaking position of the speech engine. No guarantee is given as to where charIndex will be with respect to word boundaries (such as at the end of the previous word or the beginning of the next word), only that all text before charIndex has already been spoken, and all text after charIndex has not yet been spoken. The user agent must return this value if the speech synthesis engine supports it, otherwise the user agent must return 0.

***charLength* attribute, of type unsigned long, readonly**

This attribute indicates the length of the text (word or sentence) that will be spoken corresponding to this event. This attribute is the length, in characters, starting from this event's `charIndex`. The user agent must return this value if the speech synthesis engine supports it or the user agent can otherwise determine it, otherwise the user agent must return 0.

***elapsedTime* attribute, of type float, readonly**

This attribute indicates the time, in seconds, that this event triggered, relative to when this utterance has begun to be spoken. The user agent must return this value if the speech synthesis engine supports it or the user agent can otherwise determine it, otherwise the user agent must return 0.

***name* attribute, of type DOMString, readonly**

For mark events, this attribute indicates the name of the marker, as defined in SSML as the name attribute of a mark element. [SSML] For boundary events, this attribute indicates the type of boundary that caused the event: "word" or "sentence". For all other events, this value should return "".

§ 4.2.7. SpeechSynthesisErrorEvent Attributes

The SpeechSynthesisErrorEvent is the interface used for the SpeechSynthesisUtterance error event.

***error* attribute, of type SpeechSynthesisErrorCode, readonly**

The errorCode is an enumeration indicating what has gone wrong. The values are:

"canceled"

A cancel method call caused the SpeechSynthesisUtterance to be removed from the queue before it had begun being spoken.

"interrupted"

A cancel method call caused the SpeechSynthesisUtterance to be interrupted after it has begun being spoken and before it completed.

"audio-busy"

The operation cannot be completed at this time because the user-agent cannot access the audio output device. (For example, the user may need to correct this by closing another application.)

"audio-hardware"

The operation cannot be completed at this time because the user-agent cannot identify an audio output device. (For example, the user may need to connect a speaker or configure system settings.)

"network"

The operation cannot be completed at this time because some required network communication failed.

"synthesis-unavailable"

The operation cannot be completed at this time because no synthesis engine is available. (For example, the user may need to install or configure a synthesis

engine.)

"synthesis-failed"
: The operation failed because synthesis engine had an error.

"language-unavailable"
: No appropriate voice is available for the language designated in SpeechSynthesisUtterance lang.

"voice-unavailable"
: The voice designated in SpeechSynthesisUtterance voice attribute is not available.

"text-too-long"
: The contents of the SpeechSynthesisUtterance text attribute is too long to synthesize.

"invalid-argument"
: The contents of the SpeechSynthesisUtterance rate, pitch or volume attribute is not supported by synthesizer.

"not-allowed"
: Synthesis was not allowed to start by the user agent or system in the current context.

§ 4.2.8. SpeechSynthesisVoice Attributes

***voiceURI* attribute, of type DOMString, readonly**
: The voiceURI attribute specifies the speech synthesis voice and the location of the speech synthesis service for this voice. Note that the voiceURI is a generic URI and can thus point to local or remote 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.

***name* attribute, of type DOMString, readonly**
: This attribute is a human-readable name that represents the voice. There is no guarantee that all names returned are unique.

***lang* attribute, of type DOMString, readonly**
: This attribute is a BCP 47 language tag indicating the language of the voice. [BCP47]

***localService* attribute, of type boolean, readonly**
: This attribute is true for voices supplied by a local speech synthesizer, and is false for voices supplied by a remote speech synthesizer service. (This may be useful because remote services may imply additional latency, bandwidth or cost, whereas local voices may imply lower quality, however there is no guarantee that any of these implications are true.)

***default* attribute, of type boolean, readonly**

This attribute is true for at most one voice per language. There may be a different default for each language. It is user agent dependent how default voices are determined.

§ 5. Examples

This section is non-normative.

§ 5.1. Speech Recognition Examples

EXAMPLE 1

Using speech recognition to fill an input-field and perform a web search.

```
<script type="text/javascript">
  var recognition = new SpeechRecognition();
  recognition.onresult = function(event) {
    if (event.results.length > 0) {
      q.value = event.results[0][0].transcript;
      q.form.submit();
    }
  }
</script>

<form action="https://www.example.com/search">
  <input type="search" id="q" name="q" size=60>
  <input type="button" value="Click to Speak" onclick="recognition.start
</form>
```

EXAMPLE 2

Using speech recognition to fill an options list with alternative speech results.

```
<script type="text/javascript">
  var recognition = new SpeechRecognition();
  recognition.maxAlternatives = 10;
  recognition.onresult = function(event) {
    if (event.results.length > 0) {
      var result = event.results[0];
      for (var i = 0; i < result.length; ++i) {
        var text = result[i].transcript;
        select.options[i] = new Option(text, text);
      }
    }
  }

  function start() {
    select.options.length = 0;
    recognition.start();
  }
</script>

<select id="select"></select>
<button onclick="start()">Click to Speak</button>
```

EXAMPLE 3

Using continuous speech recognition to fill a textarea.

```
<textarea id="textarea" rows=10 cols=80></textarea>
<button id="button" onclick="toggleStartStop()"></button>

<script type="text/javascript">
  var recognizing;
  var recognition = new SpeechRecognition();
  recognition.continuous = true;
  reset();
  recognition.onend = reset;

  recognition.onresult = function (event) {
    for (var i = event.resultIndex; i < event.results.length; ++i) {
      if (event.results[i].isFinal) {
        textarea.value += event.results[i][0].transcript;
      }
    }
  }

  function reset() {
    recognizing = false;
    button.innerHTML = "Click to Speak";
  }

  function toggleStartStop() {
    if (recognizing) {
      recognition.stop();
      reset();
    } else {
      recognition.start();
      recognizing = true;
      button.innerHTML = "Click to Stop";
    }
  }
</script>
```

EXAMPLE 4

Using continuous speech recognition, showing final results in black and interim results in grey.

```
<button id="button" onclick="toggleStartStop()"></button>
<div style="border:dotted;padding:10px">
  <span id="final_span"></span>
  <span id="interim_span" style="color:grey"></span>
</div>

<script type="text/javascript">
  var recognizing;
  var recognition = new SpeechRecognition();
  recognition.continuous = true;
  recognition.interimResults = true;
  reset();
  recognition.onend = reset;

  recognition.onresult = function (event) {
    var final = "";
    var interim = "";
    for (var i = 0; i < event.results.length; ++i) {
      if (event.results[i].isFinal) {
        final += event.results[i][0].transcript;
      } else {
        interim += event.results[i][0].transcript;
      }
    }
    final_span.innerHTML = final;
    interim_span.innerHTML = interim;
  }

  function reset() {
    recognizing = false;
    button.innerHTML = "Click to Speak";
  }

  function toggleStartStop() {
    if (recognizing) {
      recognition.stop();
      reset();
    } else {
      recognition.start();
      recognizing = true;
      button.innerHTML = "Click to Stop";
      final_span.innerHTML = "";
      interim_span.innerHTML = "";
```

```
    }
  }
</script>
```

§ 5.2. Speech Synthesis Examples

EXAMPLE 5

Spoken text.

```
<script type="text/javascript">
  speechSynthesis.speak(new SpeechSynthesisUtterance('Hello World'));
</script>
```

EXAMPLE 6

Spoken text with attributes and events.

```
<script type="text/javascript">
  var u = new SpeechSynthesisUtterance();
  u.text = 'Hello World';
  u.lang = 'en-US';
  u.rate = 1.2;
  u.onend = function(event) { alert('Finished in ' + event.elapsedTime
  speechSynthesis.speak(u);
</script>
```

§ Acknowledgments

Adam Sobieski (Phoster)
Björn Bringert (Google)
Charles Pritchard
Dominic Mazzoni (Google)
Gerardo Capiel (Benetech)

Jerry Carter
Kagami Sascha Rosylight
Marcos Cáceres (Mozilla)
Nagesh Kharidi (Openstream)
Olli Pettay (Mozilla)
Peter Beverloo (Google)
Raj Tumuluri (Openstream)
Satish Sampath (Google)

Also, the members of the HTML Speech Incubator Group, and the corresponding Final Report, which created the basis for this specification.

§ Conformance

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

> EXAMPLE 7
> This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

> Note, this is an informative note.

§ Index

§ Terms defined by this specification

abort(), in §4.1.2

"aborted", in §4.1.4

addFromString(string), in §4.1.10

addFromString(string, weight), in §4.1.10

addFromURI(src), in §4.1.10

addFromURI(src, weight), in §4.1.10

"audio-busy", in §4.2.7

"audio-capture", in §4.1.4

audioend, in §4.1.3

"audio-hardware", in §4.2.7

audiostart, in §4.1.3

"bad-grammar", in §4.1.4

boundary, in §4.2.5

cancel(), in §4.2.2

"canceled", in §4.2.7

charIndex

- attribute for SpeechSynthesisEvent, in §4.2.6
- dict-member for SpeechSynthesisEventInit, in §4.2

charLength

- attribute for SpeechSynthesisEvent, in §4.2.6
- dict-member for SpeechSynthesisEventInit, in §4.2

confidence, in §4.1.5

constructor()

- constructor for SpeechGrammarList, in §4.1
- constructor for SpeechRecognition, in §4.1
- constructor for SpeechSynthesisUtterance, in §4.2

constructor(text), in §4.2

constructor(type, eventInitDict)

- constructor for SpeechRecognitionErrorEvent, in §4.1
- constructor for SpeechRecognitionEvent, in §4.1
- constructor for SpeechSynthesisErrorEvent, in §4.2
- constructor for SpeechSynthesisEvent, in §4.2

continuous, in §4.1.1

default, in §4.2.8

elapsedTime

- attribute for SpeechSynthesisEvent, in §4.2.6
- dict-member for SpeechSynthesisEventInit, in §4.2

end

- event for SpeechRecognition, in §4.1.3
- event for SpeechSynthesisUtterance, in §4.2.5

error

- attribute for SpeechRecognitionErrorEvent, in §4.1.4
- attribute for SpeechSynthesisErrorEvent, in §4.2.7
- dict-member for SpeechRecognitionErrorEventInit, in §4.1
- dict-member for SpeechSynthesisErrorEventInit, in §4.2
- event for SpeechRecognition, in §4.1.3
- event for SpeechSynthesisUtterance, in §4.2.5

getVoices(), in §4.2.2

grammars, in §4.1.1

interimResults, in §4.1.1

"interrupted", in §4.2.7

"invalid-argument", in §4.2.7

isFinal, in §4.1.6

item(index)

- method for SpeechGrammarList, in §4.1.10
- method for SpeechRecognitionResult, in §4.1.6
- method for SpeechRecognitionResultList, in §4.1.7

lang

- attribute for SpeechRecognition, in §4.1.1
- attribute for SpeechSynthesisUtterance, in §4.2.4
- attribute for SpeechSynthesisVoice, in §4.2.8

"language-not-supported", in §4.1.4

"language-unavailable", in §4.2.7

length

- attribute for SpeechGrammarList, in §4.1.10
- attribute for SpeechRecognitionResult, in §4.1.6
- attribute for SpeechRecognitionResultList, in §4.1.7

localService, in §4.2.8

mark, in §4.2.5

maxAlternatives, in §4.1.1

message

- attribute for SpeechRecognitionErrorEvent, in §4.1.4
- dict-member for SpeechRecognitionErrorEventInit, in §4.1

name

- attribute for SpeechSynthesisEvent, in §4.2.6
- attribute for SpeechSynthesisVoice, in §4.2.8
- dict-member for SpeechSynthesisEventInit, in §4.2

"network"

- enum-value for SpeechRecognitionErrorCode, in §4.1.4
- enum-value for SpeechSynthesisErrorCode, in §4.2.7

nomatch, in §4.1.3

"no-speech", in §4.1.4

"not-allowed"

- enum-value for SpeechRecognitionErrorCode, in §4.1.4
- enum-value for SpeechSynthesisErrorCode, in §4.2.7

onaudioend, in §4.1

onaudiostart, in §4.1

onboundary, in §4.2

onend

- attribute for SpeechRecognition, in §4.1
- attribute for SpeechSynthesisUtterance, in §4.2

onerror

- attribute for SpeechRecognition, in §4.1
- attribute for SpeechSynthesisUtterance, in §4.2

onmark, in §4.2

onnomatch, in §4.1

onpause, in §4.2

onresult, in §4.1

onresume, in §4.2

onsoundend, in §4.1

onsoundstart, in §4.1

onspeechend, in §4.1

onspeechstart, in §4.1

onstart
attribute for SpeechRecognition, in §4.1
attribute for SpeechSynthesisUtterance, in §4.2
onvoiceschanged, in §4.2
pause(), in §4.2.2
pause, in §4.2.5
paused, in §4.2.1
pending, in §4.2.1
pitch, in §4.2.4
rate, in §4.2.4
result, in §4.1.3
resultIndex
attribute for SpeechRecognitionEvent, in §4.1.8
dict-member for SpeechRecognitionEventInit, in §4.1
results
attribute for SpeechRecognitionEvent, in §4.1.8
dict-member for SpeechRecognitionEventInit, in §4.1
resume, in §4.2.5
resume(), in §4.2.2
"service-not-allowed", in §4.1.4
soundend, in §4.1.3
soundstart, in §4.1.3
speaking, in §4.2.1
speak(utterance), in §4.2.2
speechend, in §4.1.3
SpeechGrammar, in §4.1
SpeechGrammarList, in §4.1
SpeechGrammarList(), in §4.1
SpeechRecognition(), in §4.1
SpeechRecognition, in §4.1
SpeechRecognitionAlternative, in §4.1
SpeechRecognitionErrorCode, in §4.1
SpeechRecognitionErrorEvent, in §4.1
SpeechRecognitionErrorEventInit, in §4.1
SpeechRecognitionErrorEvent(type, eventInitDict), in §4.1
SpeechRecognitionEvent, in §4.1
SpeechRecognitionEventInit, in §4.1
SpeechRecognitionEvent(type, eventInitDict), in §4.1
SpeechRecognitionResult, in §4.1
SpeechRecognitionResultList, in §4.1
speechstart, in §4.1.3
speechSynthesis, in §4.2
SpeechSynthesis, in §4.2
SpeechSynthesisErrorCode, in §4.2
SpeechSynthesisErrorEvent, in §4.2
SpeechSynthesisErrorEventInit, in §4.2
SpeechSynthesisErrorEvent(type, eventInitDict), in §4.2
SpeechSynthesisEvent, in §4.2
SpeechSynthesisEventInit, in §4.2
SpeechSynthesisEvent(type, eventInitDict), in §4.2
SpeechSynthesisUtterance, in §4.2
SpeechSynthesisUtterance(), in §4.2
SpeechSynthesisUtterance(text), in §4.2
SpeechSynthesisVoice, in §4.2
src, in §4.1.9

start

- event for SpeechRecognition, in §4.1.3
- event for SpeechSynthesisUtterance, in §4.2.5

start(), in §4.1.2

stop(), in §4.1.2

"synthesis-failed", in §4.2.7

"synthesis-unavailable", in §4.2.7

text, in §4.2.4

"text-too-long", in §4.2.7

transcript, in §4.1.5

utterance

- attribute for SpeechSynthesisEvent, in §4.2.6
- dict-member for SpeechSynthesisEventInit, in §4.2

voice, in §4.2.4

voiceschanged, in §4.2.3

"voice-unavailable", in §4.2.7

voiceURI, in §4.2.8

volume, in §4.2.4

weight, in §4.1.9

§ Terms defined by reference

[DOM] defines the following terms:

- Event
- EventInit
- EventTarget

[HTML] defines the following terms:

- EventHandler
- Window
- language

[WebIDL] defines the following terms:

- DOMException
- DOMString
- Exposed
- InvalidStateError
- SameObject
- boolean
- float
- unsigned long

§ References

§ Normative References

[BCP47]
A. Phillips; M. Davis. Tags for Identifying Languages. September 2009. IETF Best Current Practice. URL: https://tools.ietf.org/html/bcp47

[DOM]
Anne van Kesteren. DOM Standard. Living Standard. URL: https://dom.spec.whatwg.org/

[HTML]

Anne van Kesteren; et al. HTML Standard. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[RFC2119]

S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[SSML]

Daniel Burnett; Zhi Wei Shuang. Speech Synthesis Markup Language (SSML) Version 1.1. 7 September 2010. REC. URL: https://www.w3.org/TR/speech-synthesis11/

[WebIDL]

Boris Zbarsky. Web IDL. 15 December 2016. ED. URL: https://heycam.github.io/webidl/

§ Informative References

[HTMLSPEECH]

Michael Bodell; et al. HTML Speech Incubator Group Final Report. URL: https://www.w3.org/2005/Incubator/htmlspeech/XGR-htmlspeech-20111206/

§ IDL Index

```
[Exposed=Window]
interface SpeechRecognition : EventTarget {
    constructor();

    // recognition parameters
    attribute SpeechGrammarList grammars;
    attribute DOMString lang;
    attribute boolean continuous;
    attribute boolean interimResults;
    attribute unsigned long maxAlternatives;

    // methods to drive the speech interaction
    void start();
    void stop();
    void abort();

    // event methods
    attribute EventHandler onaudiostart;
    attribute EventHandler onsoundstart;
    attribute EventHandler onspeechstart;
    attribute EventHandler onspeechend;
    attribute EventHandler onsoundend;
    attribute EventHandler onaudioend;
    attribute EventHandler onresult;
    attribute EventHandler onnomatch;
    attribute EventHandler onerror;
    attribute EventHandler onstart;
    attribute EventHandler onend;
};

enum SpeechRecognitionErrorCode {
    "no-speech",
    "aborted",
    "audio-capture",
    "network",
    "not-allowed",
    "service-not-allowed",
    "bad-grammar",
    "language-not-supported"
};

[Exposed=Window]
interface SpeechRecognitionErrorEvent : Event {
```

```
    constructor(DOMString type, SpeechRecognitionErrorEventInit eventInitD
    readonly attribute SpeechRecognitionErrorCode error;
    readonly attribute DOMString message;
};

dictionary SpeechRecognitionErrorEventInit : EventInit {
    required SpeechRecognitionErrorCode error;
    DOMString message = "";
};

// Item in N-best list
[Exposed=Window]
interface SpeechRecognitionAlternative {
    readonly attribute DOMString transcript;
    readonly attribute float confidence;
};

// A complete one-shot simple response
[Exposed=Window]
interface SpeechRecognitionResult {
    readonly attribute unsigned long length;
    getter SpeechRecognitionAlternative item(unsigned long index);
    readonly attribute boolean isFinal;
};

// A collection of responses (used in continuous mode)
[Exposed=Window]
interface SpeechRecognitionResultList {
    readonly attribute unsigned long length;
    getter SpeechRecognitionResult item(unsigned long index);
};

// A full response, which could be interim or final, part of a continuous
[Exposed=Window]
interface SpeechRecognitionEvent : Event {
    constructor(DOMString type, SpeechRecognitionEventInit eventInitDict);
    readonly attribute unsigned long resultIndex;
    readonly attribute SpeechRecognitionResultList results;
};

dictionary SpeechRecognitionEventInit : EventInit {
    unsigned long resultIndex = 0;
    required SpeechRecognitionResultList results;
};
```

```
// The object representing a speech grammar
[Exposed=Window]
interface SpeechGrammar {
    attribute DOMString src;
    attribute float weight;
};

// The object representing a speech grammar collection
[Exposed=Window]
interface SpeechGrammarList {
    constructor();
    readonly attribute unsigned long length;
    getter SpeechGrammar item(unsigned long index);
    void addFromURI(DOMString src,
                    optional float weight = 1.0);
    void addFromString(DOMString string,
                    optional float weight = 1.0);
};

[Exposed=Window]
interface SpeechSynthesis : EventTarget {
    readonly attribute boolean pending;
    readonly attribute boolean speaking;
    readonly attribute boolean paused;

    attribute EventHandler onvoiceschanged;

    void speak(SpeechSynthesisUtterance utterance);
    void cancel();
    void pause();
    void resume();
    sequence<SpeechSynthesisVoice> getVoices();
};

partial interface Window {
    [SameObject] readonly attribute SpeechSynthesis speechSynthesis;
};

[Exposed=Window]
interface SpeechSynthesisUtterance : EventTarget {
    constructor(optional DOMString text);

    attribute DOMString text;
```

```
    attribute DOMString lang;
    attribute SpeechSynthesisVoice? voice;
    attribute float volume;
    attribute float rate;
    attribute float pitch;

    attribute EventHandler onstart;
    attribute EventHandler onend;
    attribute EventHandler onerror;
    attribute EventHandler onpause;
    attribute EventHandler onresume;
    attribute EventHandler onmark;
    attribute EventHandler onboundary;
};

[Exposed=Window]
interface SpeechSynthesisEvent : Event {
    constructor(DOMString type, SpeechSynthesisEventInit eventInitDict);
    readonly attribute SpeechSynthesisUtterance utterance;
    readonly attribute unsigned long charIndex;
    readonly attribute unsigned long charLength;
    readonly attribute float elapsedTime;
    readonly attribute DOMString name;
};

dictionary SpeechSynthesisEventInit : EventInit {
    required SpeechSynthesisUtterance utterance;
    unsigned long charIndex = 0;
    unsigned long charLength = 0;
    float elapsedTime = 0;
    DOMString name = "";
};

enum SpeechSynthesisErrorCode {
    "canceled",
    "interrupted",
    "audio-busy",
    "audio-hardware",
    "network",
    "synthesis-unavailable",
    "synthesis-failed",
    "language-unavailable",
    "voice-unavailable",
    "text-too-long",
```

```
    "invalid-argument",
    "not-allowed",
};

[Exposed=Window]
interface SpeechSynthesisErrorEvent : SpeechSynthesisEvent {
    constructor(DOMString type, SpeechSynthesisErrorEventInit eventInitDic
    readonly attribute SpeechSynthesisErrorCode error;
};

dictionary SpeechSynthesisErrorEventInit : SpeechSynthesisEventInit {
    required SpeechSynthesisErrorCode error;
};

[Exposed=Window]
interface SpeechSynthesisVoice {
    readonly attribute DOMString voiceURI;
    readonly attribute DOMString name;
    readonly attribute DOMString lang;
    readonly attribute boolean localService;
    readonly attribute boolean default;
};
```

§ Issues Index

ISSUE 1 The group has discussed whether WebRTC might be used to specify selection of audio sources and remote recognizers. See Interacting with WebRTC, the Web Audio API and other external sources thread on public-speech-api@w3.org. ↲

ISSUE 2 The group has discussed whether confidence can be specified in a speech-recognition-engine-independent manner and whether confidence threshold and nomatch should be included, because this is not a dialog API. See Confidence property thread on public-speech-api@w3.org. ↲

ISSUE 3 The group has discussed options for which grammar formats should be supported, how builtin grammar types are specified, and default grammars when not specified. See Default value of SpeechRecognition.grammars thread on public-speech-api@w3.org. ↵