EXHIBIT 29

SpeechRecognizer Interface

version 2.0

Table of Contents

Every user utterance leverages SpeechRecognizer. It is the core interface of the Alexa Voice Service (AVS). It exposes directives and events for capturing user speech and prompting a client when Alexa needs additional speech input.

Additionally, this interface allows your client to inform AVS of how an interaction with Alexa was initiated (press and hold, tap and release, voice-initiated/wake word enabled (/docs/alexa-voice-service/audio-hardware-configurations.html#applications)), and choose the appropriate Automatic Speech Recognition (ASR) profile (/docs/alexa-voice-service/audio-hardware-configurations.html#asr) for your product, which allows Alexa to understand user speech and respond with precision.

⚠ **Important:** Cloud-based wake word verification is required for voice-initiated products. It improves wake word accuracy by reducing false wakes that are caused by utterances that sound similar to the wake word. See Enable Cloud-based Wake Word Verification (/docs/alexa-voice-service/enable-cloud-based-wake-word-verification.html) for implementation details.

Version Changes

- Opus (http://opus-codec.org/) is now a supported format for captured audio. For more details, see the specification under the `Recognize` event.

State Diagram

The following diagram illustrates state changes driven by SpeechRecognizer components. Boxes represent SpeechRecognizer states and the connectors indicate state transitions.

SpeechRecognizer has the following states: (htt

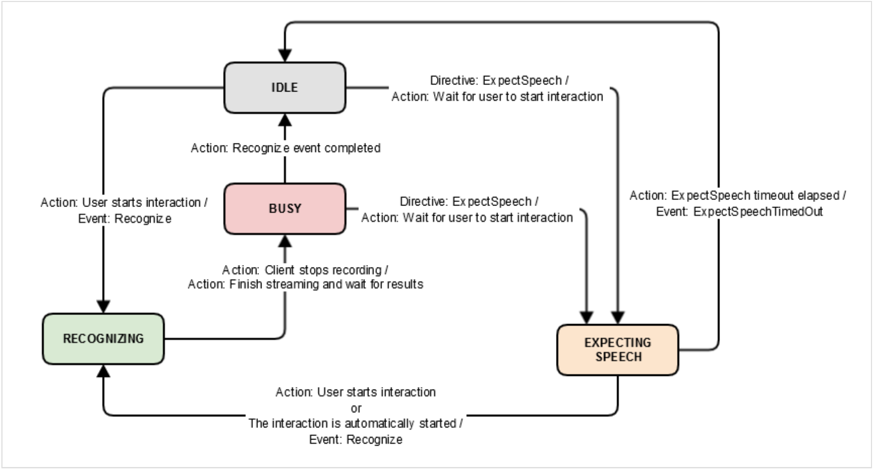
IDLE: Prior to capturing user speech, SpeechRecognizer should be in an *idle* state. SpeechRecognizer should also return to an *idle* state after a speech interaction with AVS has concluded. This can occur when a speech request has been successfully processed or when an `ExpectSpeechTimedOut` event has elapsed.

Additionally, SpeechRecognizer may return to an *idle* state during a multiturn interaction, at which point, if additional speech is required from the user, it should transition from the *idle* state to the *expecting speech* state without a user starting a new interaction.

RECOGNIZING: When a user begins interacting with your client, specifically when captured audio is streamed to AVS, SpeechRecognizer should transition from the *idle* state to the *recognizing state*. It should remain in the *recognizing state* until the client stops recording speech (or streaming is complete), at which point your SpeechRecognizer component should transition from the *recognizing* state to the *busy* state.

BUSY: While processing the speech request, SpeechRecognizer should be in the *busy* state. You cannot start another speech request until the component transitions out of the *busy* state. From the *busy* state, SpeechRecognizer will transition to the *idle* state if the request is successfully processed (completed) or to the *expecting speech* state if Alexa requires additional speech input from the user.

EXPECTING SPEECH: SpeechRecognizer should be in the *expecting speech* state when additional audio input is required from a user. From *expecting speech*, SpeechRecognizer should transition to the *recognizing state* when a user interaction occurs or the interaction is automatically started on the user's behalf. It should transition to the *idle* state if no user interaction is detected within the specified timeout window.


IDLE
Directive: ExpectSpeech / Action: Wait for user to start interaction
Action: Recognize event completed
Action: User starts interaction / Event: Recognize
BUSY
Directive: ExpectSpeech / Action: Wait for user to start interaction
Action: ExpectSpeech timeout elapsed / Event: ExpectSpeechTimedOut
Action: Client stops recording / Action: Finish streaming and wait for results
RECOGNIZING
EXPECTING SPEECH
Action: User starts interaction or The interaction is automatically started / Event: Recognize

(https://images-na.ssl-images-amazon.com/images/G/01/mobile-apps/dex/alexa/alexa-voice-service/docs/speechrecognizer-state.png)
Click to enlarge

Capabilities API

To use version 2.0 of the SpeechRecognizer interface, it must be declared in your call to the Capabilities API. For additional details, see Capabilities API (../alexa-voice-service/capabilities-api.html).

Sample Object

```
{
    "type": "AlexaInterface",
    "interface": "SpeechRecognizer",
    "version": "2.0"
}
```

(htt

SpeechRecognizer Context

Alexa expects all clients to report the currently set wake word, if wake word enabled.

To learn more about reporting Context, see Context Overview (../alexa-voice-service/context.html).

Sample Message

```
{
    "header": {
        "namespace": "SpeechRecognizer",
        "name": "RecognizerState"
    },
    "payload": {
        "wakeword": "ALEXA"
    }
}
```

Payload Parameters

Parameter	Description	Type
wakeword	Identifies the current wake word. **Accepted Value**: "ALEXA"	string

Recognize Event

The `Recognize` event is used to send user speech to AVS and translate that speech into one or more directives. This event must be sent as a multipart message, consisting of two parts:

- A JSON-formatted object
- The binary audio captured by the product's microphone.

Captured audio that is streamed to AVS should be chunked to reduce latency. The stream should contain 10ms of captured audio per chunk (320 bytes).

After an interaction with Alexa is initiated, the microphone must remain open until:

- A `StopCapture` directive is received.
- The stream is closed by the Alexa service.
- The user manually closes the microphone. For example, a press and hold implementation (/docs/alexa-voice-service/audio-hardware-configurations.html#applications).

The `profile` parameter and `initiator` object tell Alexa which ASR profile should be used to best understand the captured audio, and how the interaction was initiated.

All captured audio must be sent to AVS in either PCM or Opus, and adhere to the following specifications:

PCM	Opus
16bit Linear PCM	16bit Opus
16kHz sample rate	16kHz sample rate
Single channel	32k bit rate
Little endian byte order	Little endian byte order

(htt

⚠ **Important:** If your product is voice-initiated it must adhere to the Requirements for Cloud-Based Wake Word Verification (/docs/alexa-voice-service/streaming-requirements-for-cloud-based-wake-word-verification.html).

For a protocol specific example, see Structuring an HTTP/2 Request (/docs/alexa-voice-service/structure-http2-request.html#examples).

Sample Message

```
{
    "context": [
        // This is an array of context objects that are used to communicate the
        // state of all client components to Alexa. See Context for details.
    ],
    "event": {
        "header": {
            "namespace": "SpeechRecognizer",
            "name": "Recognize",
            "messageId": "{{STRING}}",
            "dialogRequestId": "{{STRING}}"
        },
        "payload": {
            "profile": "{{STRING}}",
            "format": "{{STRING}}",
            "initiator": {
                "type": "{{STRING}}",
                "payload": {
                    "wakeWordIndices": {
                        "startIndexInSamples": {{LONG}},
                        "endIndexInSamples": {{LONG}}
                    },
                    "token": "{{STRING}}"
                }
            }
        }
    }
}
```

Binary Audio Attachment

Each `Recognize` event requires a corresponding binary audio attachment as one part of the multipart message. The following headers are required for each binary audio attachment:

```
Content-Disposition: form-data; name="audio"
Content-Type: application/octet-stream

{{BINARY AUDIO ATTACHMENT}}
```

Context

This event requires your product to report the status of all client component states to Alexa in the context object. For additional information see Context (/docs/alexa-voice-service/context.html).

Header Parameters

Parameter	Description	Type
messageId	A unique ID used to represent a specific message.	string
dialogRequestId	A unique identifier that your client must create for each `Recognize` event sent to Alexa. This parameter is used to correlate directives sent in response to a specific `Recognize` event.	string

Payload Parameters

Parameter	Description	Type

(htt

Parameter	Description	Type
profile	Identifies the Automatic Speech Recognition (ASR) profile associated with your product. AVS supports three distinct ASR profiles optimized for user speech from varying distances. **Accepted values:** `CLOSE_TALK`, `NEAR_FIELD`, `FAR_FIELD`.	string
format	Identifies the format of captured audio. **Accepted value:** `AUDIO_L16_RATE_16000_CHANNELS_1` (PCM), `OPUS`.	string
initiator	Lets Alexa know how an interaction was initiated. This object is required when an interaction is originated by the end user (wake word, tap, push and hold). If `initiator` is present in an `ExpectSpeech` directive then it must be returned in the following `Recognize` event. If `initiator` is absent from the `ExpectSpeech` directive, then it should **not** be included in the following `Recognize` event.	object
initiator.type	Represents the action taken by a user to initiate an interaction with Alexa. **Accepted values:** `PRESS_AND_HOLD`, `TAP`, and `WAKEWORD`. If an `initiator.type` is provided in an `ExpectSpeech` directive, that string must be returned as `initiator.type` in the following `Recognize` event.	string
initiator.payload	Includes information about the initiator.	object
initiator.payload.wakeWordIndices	This object is required when `initiator.type` is set to `WAKEWORD`. `wakeWordIndices` includes the `startIndexInSamples` and `endIndexInSamples`. For additional details, see Requirements for Cloud-Based Wake Word Verification (/docs/alexa-voice-service/streaming-requirements-for-cloud-based-wake-word-verification.html).	object
initiator.payload.wakeWordIndices.startIndexInSamples	Represents the index in the audio stream where the wake word starts (in samples). The start index should be accurate to within 50ms of wake word detection.	long
initiator.payload.wakeWordIndices.endIndexInSamples	Represents the index in the audio stream where the wake word ends (in samples). The end index should be accurate to within 150ms of the end of the detected wake word.	long
initiator.payload.token	An opaque string. This value is only required if present in the payload of a preceding `ExpectSpeech` (/docs/alexa-voice-service/speechrecognizer.html#expectspeech) directive.	string

Profiles

ASR profiles are tuned for different products, form factors, acoustic environments and use cases. Use the table below to learn more about accepted values for the `profile` parameter.

(htt

Value	Optimal Listening Distance
CLOSE_TALK	0 to 2.5 ft.
NEAR_FIELD	0 to 5 ft.
FAR_FIELD	0 to 20+ ft.

Note: See Audio Hardware Configurations (/docs/alexa-voice-service/audio-hardware-configurations.html) to determine the appropriate ASR Profile for your Alexa-enabled product.

Initiator

The `initiator` parameter tells AVS how an interaction with Alexa was triggered; and determines two things:

1. If `StopCapture` will be sent to your client when the end of speech is detected in the cloud.
2. If cloud-based wake word verification will be performed on the stream.

`initiator` must be included in the payload of each `SpeechRecognizer.Recognize` event. The following values are accepted:

Value	Description	Supported Profile(s)	StopCapture Enabled	Wake Word Verification Enabled	Wake Word Indices Required
PRESS_AND_HOLD	Audio stream initiated by pressing a button (physical or GUI) and terminated by releasing it.	`CLOSE_TALK`	N	N	N
TAP	Audio stream initiated by the tap and release of a button (physical or GUI) and terminated when a `StopCapture` directive is received.	`NEAR_FIELD`, `FAR_FIELD`	Y	N	N
WAKEWORD	Audio stream initiated by the use of a wake word and terminated when a `StopCapture` directive is received.	`NEAR_FIELD`, `FAR_FIELD`	Y	Y	Y

StopCapture Directive

(htt

This directive instructs your client to stop capturing a user's speech after AVS has identified the user's intent or when end of speech is detected. When this directive is received, your client must immediately close the microphone and stop listening for the user's speech.

Note: `StopCapture` is sent to your client on the downchannel stream and may be received while speech is still being streamed to AVS. To receive the `StopCapture` directive, you must use a `profile` in your `Recognize` event that supports cloud-endpointing, such as `NEAR_FIELD` or `FAR_FIELD`.

Sample Message

```
{
    "directive": {
        "header": {
            "namespace": "SpeechRecognizer",
            "name": "StopCapture",
            "messageId": "{{STRING}}",
            "dialogRequestId": "{{STRING}}"
        },
        "payload": {
        }
    }
}
```

Header Parameters

Parameter	Description	Type
messageId	A unique ID used to represent a specific message.	string
dialogRequestId	A unique ID used to correlate directives sent in response to a specific `Recognize` event.	string

ExpectSpeech Directive

`ExpectSpeech` is sent when Alexa requires additional information to fulfill a user's request. It instructs your client to open the microphone and begin streaming user speech. If the microphone is not opened within the specified timeout window, an `ExpectSpeechTimedOut` event must be sent from your client to AVS.

During a multi-turn interaction with Alexa, your device will receive at least one `ExpectSpeech` directive instructing your client to start listening for user speech. If present, the `initiator` object included in the payload of the `ExpectSpeech` directive must be passed back to Alexa as the `initiator` object in the following `Recognize` event. If `initiator` is absent from the payload, the following `Recognize` event should **not** include `initiator`.

For information on the rules that govern audio prioritization, please review the Interaction Model (/docs/alexa-voice-service/interaction-model.html).

Sample Message

(htt

```
{
    "directive": {
        "header": {
            "namespace": "SpeechRecognizer",
            "name": "ExpectSpeech",
            "messageId": "{{STRING}}",
            "dialogRequestId": "{{STRING}}"
        },
        "payload": {
            "timeoutInMilliseconds": {{LONG}},
            "initiator": {
                "type": "{{STRING}}",
                "payload": {
                    "token": "{{STRING}}"
                }
            }
        }
    }
}
```

Header Parameters

Parameter	Description	Type
messageId	A unique ID used to represent a specific message.	string
dialogRequestId	A unique ID used to correlate directives sent in response to a specific `Recognize` event.	string

Payload Parameters

Parameter	Description	Type
timeoutInMilliseconds	Specifies, in milliseconds, how long your client should wait for the microphone to open and begin streaming user speech to AVS. If the microphone is not opened within the specified timeout window, then the ExpectSpeechTimedOut event must be sent. The primary use case for this behavior is a PRESS_AND_HOLD implementation.	long
initiator	Contains information about the interaction. If present it must be sent back to Alexa in the following `Recognize` event.	object
initiator.type	An opaque string. If present it must be sent back to Alexa in the following `Recognize` event.	string
initiator.payload	Includes information about the initiator.	object
initiator.payload.token	An opaque string. If present it must be sent back to Alexa in the following `Recognize` event.	string

ExpectSpeechTimedOut Event

This event must be sent to AVS if an `ExpectSpeech` directive was received, but was not satisfied within the specified timeout window.

Sample Message

(htt

```
{
    "event": {
        "header": {
            "namespace": "SpeechRecognizer",
            "name": "ExpectSpeechTimedOut",
            "messageId": "{{STRING}}",
        },
        "payload": {
        }
    }
}
```

Header Parameters

Parameter	Description	Type
messageId	A unique ID used to represent a specific message.	string

Payload Parameters

An empty payload should be sent.

(htt