Streaming Avatar SDK API Reference

Streaming Avatar SDK API Reference

The `@heygen/streaming-avatar` package provides a TypeScript SDK for interacting with HeyGen's streaming avatar service. For detailed information about the available methods, interfaces, enums, and event handling, refer to the API reference provided below.

Classes

`StreamingAvatar`

This class is the core of the SDK, responsible for managing avatar streaming sessions, including session creation, controlling avatars, and handling real-time communication.

TypeScript TypeScript

```
const avatar = new StreamingAvatar({ token: "access-token" });
```

Interfaces

`StreamingAvatarApiConfig`

Configuration object for initializing `StreamingAvatar`.

Property	Type	Description
`token`	`string`	Authentication token for the session. **Please note this is not your HeyGen API key.** You can retrieve this 'Session Token' by calling the create_token endpoint: https://docs.heygen.com/reference/create-session-token
`basePath`	`string`	Base API URL (optional, defaults to `https://api.heygen.com`).

TypeScript

```typescript
const config: StreamingAvatarApiConfig = {
  token: "access-token",
};
```

StartAvatarRequest ↗

Request payload to initiate a new avatar streaming session.

Property	Type	Description
`avatarName`	`string`	Interactive Avatar ID for the session. (default, `default`)
`quality`	`AvatarQuality`	(Optional) The desired quality level of the avatar stream.
`voice`	`VoiceSetting`	(Optional) Voice settings for the avatar.
`knowledgeId`	`string`	(Optional) Knowledge base ID for the avatar's knowledge / prompt. Retrieve from labs.heygen.com.
`knowledgeBase`	`string`	(Optional) This is used as a custom 'system prompt' for the LLM that powers the Avatar's responses when using the Talk task type in the Speak request method.
`disableIdleTimeout`	`boolean`	(Optional) Controls the avatar's session timeout behavior. When true, prevents automatic session termination during periods of inactivity. ⚠️ *Do not use this feature without proper session management, as open sessions can consume your API credits!*

TypeScript

```typescript
const startRequest: StartAvatarRequest = {
  quality: AvatarQuality.High,
  avatarName: avatarId,
  knowledgeId: knowledgeId,
  // knowledgeBase: knowledgeBase,
  voice: {
    voiceId: voiceId,
    rate: 1.5, // 0.5 ~ 1.5
    emotion: VoiceEmotion.EXCITED,
  },
  language: language,
  disableIdleTimeout: true
};
```

StartAvatarResponse

The response received when an avatar session is successfully started.

Property	Type	Description
`session_id`	`string`	The unique ID of the streaming session.
`access_token`	`string`	Token to authenticate further interactions.
`url`	`string`	WebSocket URL for establishing the streaming session.
`is_paid`	`boolean`	Indicates whether the session is under a paid plan.
`session_duration_limit`	`number`	Maximum allowed duration for the session in seconds.

Response

```
{
  "session_id": "eba59f0d-71f5-11ef-b8af-d2e5560124bc",
  "sdp": null,
  "access_token": "eyJhbGc...",
  "url": "wss://heygen-feapbkvq.livekit.cloud",
  "ice_servers": null,
  "ice_servers2": null,
  "is_paid": true,
  "session_duration_limit": 600
}
```

SpeakRequest ↗

Request payload for sending a speaking command to the avatar.

Property	Type	Description
`text`	`string`	The textual content the avatar will vocalize and synchronize with its movements.

Property	Type	Description
`taskType`	`string`	Defines the speaking behavior mode. Options include TaskType.TALK and TaskType.REPEAT.
`taskMode`	`string`	Specifies synchronization strategy. `SYNC` blocks further actions until speaking completes, `ASYNC` allows concurrent processing.

TypeScript

```typescript
const speakRequest: SpeakRequest = {
  text: "Hello, there!",
  task_type: TaskType.REPEAT
};
```

Methods

`createStartAvatar`

Starts a new avatar session using the provided configuration and returns session information.

TypeScript   TypeScript

```typescript
createStartAvatar(requestData: StartAvatarRequest): Promise<any>
```

`startVoiceChat`

Starts a voice chat within the active avatar session. You can optionally enable or disable silence prompts during the chat by setting the `useSilencePrompt` flag

TypeScript   TypeScript

```typescript
startVoiceChat(requestData: { useSilencePrompt?: boolean } = {}): Promise<any>
```

Property	Type	Description
`useSilencePrompt`	`boolean`	(Optional) Controls automatic conversational prompts during periods of user inactivity. Enables fallback conversational strategies.
`isInputAudioMuted`	`boolean`	(Optional) Determines whether the user's microphone input is muted during the voice chat session. When set

Property	Type	Description
		to `true` , the avatar will not receive audio input from the user.

`closeVoiceChat`

Ends the active voice chat session within the avatar interaction.

TypeScript TypeScript

```
closeVoiceChat(): Promise<any>
```

`newSession`

Creates and starts a new session using the provided `StartAvatarRequest` data, returning detailed session information such as the session ID and other metadata.

TypeScript TypeScript

```
newSession(requestData: StartAvatarRequest): Promise<StartAvatarResponse>
```

`startSession`

Starts an existing avatar session by using the previously stored session ID or configuration from a `StartAvatarRequest` .

TypeScript TypeScript

```
startSession(): Promise<any>
```

`speak`

Sends a command to the avatar to speak the provided text. Additional parameters like `task_type` allow for more advanced control, like repeating or talking.

TypeScript TypeScript

```
speak(requestData: SpeakRequest): Promise<any>
```

`startListening`

Activates the avatar’s listening mode, allowing it to process incoming audio or messages from the user.

TypeScript TypeScript

```
startListening(): Promise<any>
```

`stopListening`

Stops the avatar from listening to incoming audio or messages.

TypeScript TypeScript

```
stopListening(): Promise<any>
```

`interrupt`

Interrupts the current speaking task.

TypeScript TypeScript

```
interrupt(): Promise<any>
```

`stopAvatar`

Stops the avatar session.

TypeScript TypeScript

```
stopAvatar(): Promise<any>
```

`on`

Registers an event listener for specific streaming events.

TypeScript TypeScript

```
on(eventType: string, listener: EventHandler): this
```

off

Unregisters an event listener.

TypeScript TypeScript

```
off(eventType: string, listener: EventHandler): this
```

Types and Enums

`AvatarQuality`

Defines the quality settings for the avatar.

- **High**: `'high'` - 2000kbps and 720p.
- **Medium**: `'medium'` - 1000kbps and 480p.
- **Low**: `'low'` - 500kbps and 360p.

`VoiceEmotion`

- **`EXCITED`** : Excited voice emotion.
- **`SERIOUS`** : Serious voice emotion.
- **`FRIENDLY`** : Friendly voice emotion.
- **`SOOTHING`** : Soothing voice emotion.
- **`BROADCASTER`** : Broadcaster voice emotion.

`TaskType`

- **`TALK`** : Avatar will talk in response to the `Text` sent in tasks of this type; the response will be provided by HeyGen's connection to GPT-4o mini, and influenced by the `KnowledgeID` or `KnowledgeBase` that were provided when calling the `StartAvatarRequest` method.
- **`REPEAT`** : Avatar will simply repeat the `Text` sent in tasks of this type; this task type is commonly used by developers who process a user's input independently, via an LLM of their choosing, and send the LLM's response as a **Repeat** task for the Avatar to say.

`StreamingEvents`

Enumerates the event types for streaming. See Event Handling for details.

- **`AVATAR_START_TALKING`** : Emitted when the avatar starts speaking.
- **`AVATAR_STOP_TALKING`** : Emitted when the avatar stops speaking.
- **`AVATAR_TALKING_MESSAGE`** : Triggered when the avatar sends a speaking message.
- **`AVATAR_END_MESSAGE`** : Triggered when the avatar finishes sending messages.

- `USER_TALKING_MESSAGE` : Emitted when the user sends a speaking message.
- `USER_END_MESSAGE` : Triggered when the user finishes sending messages.
- `USER_START` : Indicates when the user starts interacting.
- `USER_STOP` : Indicates when the user stops interacting.
- `USER_SILENCE` : Indicates when the user is silent.
- `STREAM_READY` : Indicates that the stream is ready for display.
- `STREAM_DISCONNECTED` : Triggered when the stream disconnects.

Event Handling

The SDK emits a variety of events during a streaming session, which can be captured to update the UI or trigger additional logic. Use the `on` and `off` methods to manage event listeners.

`AVATAR_START_TALKING`

This event is emitted when the avatar begins speaking.

TypeScript

```
avatar.on(StreamingEvents.AVATAR_START_TALKING, (event) => {
  console.log('Avatar has started talking:', event);
  // You can update the UI to reflect that the avatar is talking
});
```

`AVATAR_STOP_TALKING`

TypeScript

```
avatar.on(StreamingEvents.AVATAR_STOP_TALKING, (event) => {
  console.log('Avatar has stopped talking:', event);
  // You can reset the UI to indicate the avatar has stopped speaking
});
```

`AVATAR_TALKING_MESSAGE`

Fired when the avatar sends a message while talking. This event can be useful for real-time updates on what the avatar is currently saying.

TypeScript

```
avatar.on(StreamingEvents.AVATAR_TALKING_MESSAGE, (message) => {
  console.log('Avatar talking message:', message);
  // You can display the message in the UI
});
```

AVATAR_END_MESSAGE

Fired when the avatar sends the final message before ending its speech.

TypeScript

```
avatar.on(StreamingEvents.AVATAR_END_MESSAGE, (message) => {
  console.log('Avatar end message:', message);
  // Handle the end of the avatar's message, e.g., indicate the end of the conversation
});
```

USER_TALKING_MESSAGE

Fired when the user sends a message to the avatar.

TypeScript

```
avatar.on(StreamingEvents.USER_TALKING_MESSAGE, (message) => {
  console.log('User talking message:', message);
  // Handle the user's message input to the avatar
});
```

USER_END_MESSAGE

Fired when the user has finished sending their message to the avatar.

TypeScript

```
avatar.on(StreamingEvents.USER_END_MESSAGE, (message) => {
  console.log('User end message:', message);
  // Handle the end of the user's message, e.g., process the user's response
});
```

USER_START

Fired when the user has finished sending their message to the avatar.

TypeScript

```
avatar.on(StreamingEvents.USER_START, (event) => {
  console.log('User has started interaction:', event);
  // Handle the start of the user's interaction, such as activating a listening indicat
});
```

USER_STOP

Triggered when the user stops interacting or speaking with the avatar.

TypeScript

```
avatar.on(StreamingEvents.USER_STOP, (event) => {
  console.log('User has stopped interaction:', event);
  // Handle the end of the user's interaction, such as deactivating a listening indicat
});
```

USER_SILENCE

Triggered when the user is silent for a certain period.

TypeScript

```
avatar.on(StreamingEvents.USER_SILENCE, () => {
  console.log('User is silent');
});
```

STREAM_READY

Fired when the avatar's streaming session is ready.

TypeScript

```
avatar.on(StreamingEvents.STREAM_READY, (event) => {
  console.log('Stream is ready:', event.detail);
  // Use event.detail to attach the media stream to a video element, for example
});
```

STREAM_DISCONNECTED

Triggered when the streaming connection is lost or intentionally disconnected.

TypeScript

```
avatar.on(StreamingEvents.STREAM_DISCONNECTED, () => {
  console.log('Stream has been disconnected');
  // Handle the disconnection, e.g., clean up the UI or try to reconnect the session
});
```

Error Handling

Always handle errors gracefully when dealing with asynchronous requests to avoid disruptions in the user experience.

TypeScript

```
try {
  await avatar.speak({ text: 'Hello!' });
} catch (error) {
  console.error('Error sending speak command:', error);
}
```

Conclusion

This reference offers a comprehensive overview of HeyGen's streaming avatar SDK, complete with examples and descriptions of methods, events, and configuration options.

Updated 24 days ago

← Streaming Avatar SDK

Demo: Create a Vite Project with Streaming SDK →