Session management with Live API

In the Live API, a session refers to a persistent connection where input and output are streamed continuously over the same connection (read more about how it works (/gemini-api/docs/live)). This unique session design enables low latency and supports unique features, but can also introduce challenges, like session time limits, and early termination. This guide covers strategies for overcoming the session management challenges that can arise when using the Live API.

Session lifetime

Without compression, audio-only sessions are limited to 15 minutes, and audio-video sessions are limited to 2 minutes. Exceeding these limits will terminate the session (and therefore, the connection), but you can use context window compression (#context-window-compression) to extend sessions to an unlimited amount of time.

The lifetime of a connection is limited as well, to around 10 minutes. When the connection terminates, the session terminates as well. In this case, you can configure a single session to stay active over multiple connections using session resumption (#session-resumption). You'll also receive a GoAway message (#goaway-message) before the connection ends, allowing you to take further actions.

Context window compression

To enable longer sessions, and avoid abrupt connection termination, you can enable context window compression by setting the contextWindowCompression (/api/live#BidiGenerateContentSetup.FIELDS.ContextWindowCompressionConfig.BidiGenerateContentSetup.context_window_compression) field as part of the session configuration.

In the ContextWindowCompressionConfig (/api/live#contextwindowcompressionconfig), you can configure a sliding-window mechanism (/api/live#ContextWindowCompressionConfig.FIELDS.ContextWindowCompressionConfig.SlidingWindow.ContextWindowCompressionConfig.sliding_window) and the number of tokens (/api/live#ContextWindowCompressionConfig.FIELDS.int64.ContextWindowCompressionConfig.trigger_tokens) that triggers compression.

Python (#python)JavaScript (#javascript)

```
const config = {
  responseModalities: [Modality.AUDIO],
  contextWindowCompression: { slidingWindow: {} }
};
```

Session resumption

To prevent session termination when the server periodically resets the WebSocket connection, configure the sessionResumption (/api/live#BidiGenerateContentSetup.FIELDS.SessionResumptionConfig.BidiGenerateContentSetup.session_resumption) field within the setup configuration (/api/live#BidiGenerateContentSetup).

Passing this configuration causes the server to send SessionResumptionUpdate (/api/live#SessionResumptionUpdate) messages, which can be used to resume the session by passing the last resumption token as the `SessionResumptionConfig.handle` (/api/live#SessionResumptionConfig.FIELDS.string.SessionResumptionConfig.handle) of the subsequent connection.

Resumption tokens are valid for 2 hr after the last sessions termination.

Python (#python)JavaScript (#javascript)

```
import { GoogleGenAI, Modality } from '@google/genai';

const ai = new GoogleGenAI({});
const model = 'gemini-2.5-flash-native-audio-preview-12-2025';

async function live() {
  const responseQueue = [];

  async function waitMessage() {
    let done = false;
    let message = undefined;
    while (!done) {
      message = responseQueue.shift();
      if (message) {
        done = true;
      } else {
        await new Promise((resolve) => setTimeout(resolve, 100));
      }
    }
    return message;
  }

  async function handleTurn() {
    const turns = [];
    let done = false;
    while (!done) {
      const message = await waitMessage();
      turns.push(message);
      if (message.serverContent && message.serverContent.turnComplete) {
        done = true;
      }
    }
    return turns;
  }

console.debug('Connecting to the service with handle %s...', previousSes
const session = await ai.live.connect({
  model: model,
  callbacks: {
    onopen: function () {
      console.debug('Opened');
    },
    onmessage: function (message) {
      responseQueue.push(message);
    },
    onerror: function (e) {
      console.debug('Error:', e.message);
    },
    onclose: function (e) {
      console.debug('Close:', e.reason);
    },
  },
  config: {
    responseModalities: [Modality.AUDIO],
    sessionResumption: { handle: previousSessionHandle }
    // The handle of the session to resume is passed here, or else null
  }
});

const inputTurns = 'Hello how are you?';
session.sendClientContent({ turns: inputTurns });

const turns = await handleTurn();
```

```
for (const turn of turns) {
  if (turn.sessionResumptionUpdate) {
    if (turn.sessionResumptionUpdate.resumable && turn.sessionResumptior
      let newHandle = turn.sessionResumptionUpdate.newHandle
      // ...Store newHandle and start new session with this handle here
    }
  }
}

  session.close();
}

async function main() {
  await live().catch((e) => console.error('got error', e));
}

main();
```

Receiving a message before the session disconnects

The server sends a GoAway (/api/live#GoAway) message that signals that the current connection will soon be terminated. This message includes the timeLeft (/api/live#GoAway.FIELDS.google.protobuf.Duration.GoAway.time_left), indicating the remaining time and lets you take further action before the connection will be terminated as ABORTED.

Python (#python)JavaScript (#javascript)

```
const turns = await handleTurn();

for (const turn of turns) {
  if (turn.goAway) {
    console.debug('Time left: %s\n', turn.goAway.timeLeft);
  }
}
```

Receiving a message when the generation is complete

The server sends a generationComplete (/api/live#BidiGenerateContentServerContent.FIELDS.bool.BidiGenerateContentServerContent.generation_complete) message that signals that the model finished generating the response.

Python (#python)JavaScript (#javascript)

```
const turns = await handleTurn();

for (const turn of turns) {
  if (turn.serverContent && turn.serverContent.generationComplete) {
    // The generation is complete
  }
}
```

What's next

Explore more ways to work with the Live API in the full Capabilities (/gemini-api/docs/live) guide, the Tool use (/gemini-api/docs/live-tools) page, or the Live API cookbook (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI.ipynb) .