Speech generation (text-to-speech)

The Gemini API can transform text input into single speaker or multi-speaker audio using native text-to-speech (TTS) generation capabilities. Text-to-speech (TTS) generation is *controllable* (#controllable), meaning you can use natural language to structure interactions and guide the *style*, *accent*, *pace*, and *tone* of the audio.

The TTS capability differs from speech generation provided through the Live API (/gemini-api/docs/live), which is designed for interactive, unstructured audio, and multimodal inputs and outputs. While the Live API excels in dynamic conversational contexts, TTS through the Gemini API is tailored for scenarios that require exact text recitation with fine-grained control over style and sound, such as podcast or audiobook generation.

This guide shows you how to generate single-speaker and multi-speaker audio from text.

Preview: Native text-to-speech (TTS) is in Preview (/gemini-api/docs/models#preview).

Before you begin

Ensure you use a Gemini 2.5 model variant with native text-to-speech (TTS) capabilities, as listed in the Supported models (/gemini-api/docs/speech-generation#supported-models) section. For optimal results, consider which model best fits your specific use case.

You may find it useful to test the Gemini 2.5 TTS models in AI Studio (https://aistudio.google.com/generate-speech) before you start building.

Note: TTS models accept text-only inputs and produce audio-only outputs. For a complete list of restrictions specific to TTS models, review the Limitations (/gemini-api/docs/speech-generation#limitations) section.

Single-speaker text-to-speech

To convert text to single-speaker audio, set the response modality to "audio", and pass a `SpeechConfig` object with `VoiceConfig` set. You'll need to choose a voice name from the prebuilt output voices (#voices).

This example saves the output audio from the model in a wave file:

Python JavaScript (#javascript) REST (#rest)
(#python)

```
from google import genai
from google.genai import types
import wave

# Set up the wave file to save the output:
def wave_file(filename, pcm, channels=1, rate=24000, sample_width=2):
   with wave.open(filename, "wb") as wf:
      wf.setnchannels(channels)
      wf.setsampwidth(sample_width)
      wf.setframerate(rate)
      wf.writeframes(pcm)

client = genai.Client()

response = client.models.generate_content(
   model="gemini-2.5-flash-preview-tts",
   contents="Say cheerfully: Have a wonderful day!",
   config=types.GenerateContentConfig(
      response_modalities=["AUDIO"],
      speech_config=types.SpeechConfig(
         voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
               voice_name='Kore',
            )
         )
      ),
   )
)

data = response.candidates[0].content.parts[0].inline_data.data

file_name='out.wav'
```

```
wave_file(file_name, data) # Saves the file to current directory
```

For more code samples, refer to the "TTS - Get Started" file in the cookbooks repository:

View on GitHub (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_TTS.ipynb)

Multi-speaker text-to-speech

For multi-speaker audio, you'll need a `MultiSpeakerVoiceConfig` object with each speaker (up to 2) configured as a `SpeakerVoiceConfig`. You'll need to define each `speaker` with the same names used in the prompt (#controllable):

Python JavaScript (#javascript) REST (#rest)
(#python)

```
from google import genai
from google.genai import types
import wave

# Set up the wave file to save the output:
def wave_file(filename, pcm, channels=1, rate=24000, sample_width=2):
   with wave.open(filename, "wb") as wf:
      wf.setnchannels(channels)
      wf.setsampwidth(sample_width)
      wf.setframerate(rate)
      wf.writeframes(pcm)

client = genai.Client()

prompt = """TTS the following conversation between Joe and Jane:
         Joe: How's it going today Jane?
         Jane: Not too bad, how about you?"""

response = client.models.generate_content(
   model="gemini-2.5-flash-preview-tts",
   contents=prompt,
   config=types.GenerateContentConfig(
```

```
        response_modalities=["AUDIO"],
        speech_config=types.SpeechConfig(
           multi_speaker_voice_config=types.MultiSpeakerVoiceConfig(
              speaker_voice_configs=[
                 types.SpeakerVoiceConfig(
                    speaker='Joe',
                    voice_config=types.VoiceConfig(
                       prebuilt_voice_config=types.PrebuiltVoiceConfig(
                          voice_name='Kore',
                       )
                    )
                 ),
                 types.SpeakerVoiceConfig(
                    speaker='Jane',
                    voice_config=types.VoiceConfig(
                       prebuilt_voice_config=types.PrebuiltVoiceConfig(
                          voice_name='Puck',
                       )
                    )
                 ),
              ]
           )
        )
     )
)

data = response.candidates[0].content.parts[0].inline_data.data

file_name='out.wav'
wave_file(file_name, data) # Saves the file to current directory
```

Controlling speech style with prompts

You can control style, tone, accent, and pace using natural language prompts for both single- and multi-speaker TTS. For example, in a single-speaker prompt, you can say:

```
Say in an spooky whisper:
"By the pricking of my thumbs...
Something wicked this way comes"
```

In a multi-speaker prompt, provide the model with each speaker's name and corresponding transcript. You can also provide guidance for each speaker individually:

```
Make Speaker1 sound tired and bored, and Speaker2 sound excited and happy:

Speaker1: So... what's on the agenda today?
Speaker2: You're never going to guess!
```

Try using a voice option (#voices) that corresponds to the style or emotion you want to convey, to emphasize it even more. In the previous prompt, for example, *Enceladus*'s breathiness might emphasize "tired" and "bored", while *Puck*'s upbeat tone could complement "excited" and "happy".

Generating a prompt to convert to audio

The TTS models only output audio, but you can use other models (/gemini-api/docs/models) to generate a transcript first, then pass that transcript to the TTS model to read aloud.

Python JavaScript (#javascript)
(#python)

```
from google import genai
from google.genai import types

client = genai.Client()

transcript = client.models.generate_content(
   model="gemini-2.0-flash",
   contents="""Generate a short transcript around 100 words that reads
            like it was clipped from a podcast by excited herpetologists.
            The hosts names are Dr. Anya and Liam.""").text

response = client.models.generate_content(
   model="gemini-2.5-flash-preview-tts",
   contents=transcript,
   config=types.GenerateContentConfig(
      response_modalities=["AUDIO"],
      speech_config=types.SpeechConfig(
         multi_speaker_voice_config=types.MultiSpeakerVoiceConfig(
```

```
            speaker_voice_configs=[
               types.SpeakerVoiceConfig(
                  speaker='Dr. Anya',
                  voice_config=types.VoiceConfig(
                     prebuilt_voice_config=types.PrebuiltVoiceConfig(
                        voice_name='Kore',
                     )
                  )
               ),
               types.SpeakerVoiceConfig(
                  speaker='Liam',
                  voice_config=types.VoiceConfig(
                     prebuilt_voice_config=types.PrebuiltVoiceConfig(
                        voice_name='Puck',
                     )
                  )
               ),
            ]
         )
      )
   )
)

# ...Code to stream or save the output
```

Voice options

TTS models support the following 30 voice options in the `voice_name` field:

Zephyr -- *Bright*	**Puck** -- *Upbeat*	**Charon** -- *Informative*
Kore -- *Firm*	**Fenrir** -- *Excitable*	**Leda** -- *Youthful*
Orus -- *Firm*	**Aoede** -- *Breezy*	**Callirrhoe** -- *Easy-going*
Autonoe -- *Bright*	**Enceladus** -- *Breathy*	**Iapetus** -- *Clear*
Umbriel -- *Easy-going*	**Algieba** -- *Smooth*	**Despina** -- *Smooth*

Erinome -- *Clear*	**Algenib** -- *Gravelly*	**Rasalgethi** -- *Informative*
Laomedeia -- *Upbeat*	**Achernar** -- *Soft*	**Alnilam** -- *Firm*
Schedar -- *Even*	**Gacrux** -- *Mature*	**Pulcherrima** -- *Forward*
Achird -- *Friendly*	**Zubenelgenubi** -- *Casual*	**Vindemiatrix** -- *Gentle*
Sadachbia -- *Lively*	**Sadaltager** -- *Knowledgeable*	**Sulafat** -- *Warm*

You can hear all the voice options in AI Studio (https://aistudio.google.com/generate-speech).

Supported languages

The TTS models detect the input language automatically. They support the following 24 languages:

Language	BCP-47 Code	Language	BCP-47 Code
Arabic (Egyptian)	`ar-EG`	German (Germany)	`de-DE`
English (US)	`en-US`	Spanish (US)	`es-US`
French (France)	`fr-FR`	Hindi (India)	`hi-IN`
Indonesian (Indonesia)	`id-ID`	Italian (Italy)	`it-IT`
Japanese (Japan)	`ja-JP`	Korean (Korea)	`ko-KR`
Portuguese (Brazil)	`pt-BR`	Russian (Russia)	`ru-RU`
Dutch (Netherlands)	`nl-NL`	Polish (Poland)	`pl-PL`

Language	BCP-47 Code	Language	BCP-47 Code
Thai (Thailand)	`th-TH`	Turkish (Turkey)	`tr-TR`
Vietnamese (Vietnam)	`vi-VN`	Romanian (Romania)	`ro-RO`
Ukrainian (Ukraine)	`uk-UA`	Bengali (Bangladesh)	`bn-BD`
English (India)	`en-IN` & `hi-IN` bundle	Marathi (India)	`mr-IN`
Tamil (India)	`ta-IN`	Telugu (India)	`te-IN`

Supported models

Model	Single speaker	Multispeaker
Gemini 2.5 Flash Preview TTS (/gemini-api/docs/models#gemini-2.5-flash-preview-tts)	✔️	✔️
Gemini 2.5 Pro Preview TTS (/gemini-api/docs/models#gemini-2.5-pro-preview-tts)	✔️	✔️

Limitations

- TTS models can only receive text inputs and generate audio outputs.
- A TTS session has a context window (/gemini-api/docs/long-context) limit of 32k tokens.
- Review Languages (/gemini-api/docs/speech-generation#languages) section for language support.

What's next

- Try the audio generation cookbook (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_TTS.ipynb) .
- Gemini's Live API (/gemini-api/docs/live) offers interactive audio generation options you can interleave with other modalities.
- For working with audio *inputs*, visit the Audio understanding (/gemini-api/docs/audio) guide.

Last updated 2025-07-08 UTC.