# Using files

The Gemini API supports uploading media files separately from the prompt input, allowing your media to be reused across multiple requests and multiple prompts. For more details, check out the Prompting with media guide.

## Method: media.upload

Creates a File.

### Endpoint

#### Upload URI, for media upload requests:

post https://generativelanguage.googleapis.com/upload/v1beta/files

#### Metadata URI, for metadata-only requests:

post https://generativelanguage.googleapis.com/v1beta/files

### Request body

The request body contains data with the following structure:

Fields

file object (File)

Optional. Metadata for the file to create.

### Response body

Response for media.upload.

If successful, the response body contains data with the following structure:

Fields

#### fileobject (File)

Metadata for the created file.

## JSON representation:

```
{
  "file": {
    object (File)
  }
}
```

## Example request

### IMAGE TYPE Files:

#### Python code:

```
from google import genai


client = genai.Client()

myfile = client.files.upload(file=media / "Cajun_instruments.jpg")

print(f"{myfile=}")


result = client.models.generate_content(
    model="gemini-2.0-flash",
    contents=[
        myfile,
        "\n\n",
        "Can you tell me about the instruments in this photo?",
    ],
)
```

```
print(f"{result.text=}")
```

### Node.js code:

```
// Make sure to include the following import:
// import {GoogleGenAI} from '@google/genai';
const ai = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY });
const myfile = await ai.files.upload({
    file: path.join(media, "Cajun_instruments.jpg"),
    config: { mimeType: "image/jpeg" },
});
console.log("Uploaded file:", myfile);

const result = await ai.models.generateContent({
    model: "gemini-2.0-flash",
    contents: createUserContent([
        createPartFromUri(myfile.uri, myfile.mimeType),
        "\n\n",
        "Can you tell me about the instruments in this photo?",
    ]),
});
console.log("result.text=", result.text);
```

## AUDIO TYPE Files:

### Python code:

```
from google import genai
```

```
client = genai.Client()

myfile = client.files.upload(file=media / "sample.mp3")

print(f"{myfile=}")


result = client.models.generate_content(

        model="gemini-2.0-flash", contents=[myfile, "Describe this audio clip"]

)

print(f"{result.text=}")
```

Node.js code:

```
// Make sure to include the following import:

// import {GoogleGenAI} from '@google/genai';

const ai = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY });

const myfile = await ai.files.upload({

    file: path.join(media, "sample.mp3"),

    config: { mimeType: "audio/mpeg" },

});

console.log("Uploaded file:", myfile);


const result = await ai.models.generateContent({

    model: "gemini-2.0-flash",

    contents: createUserContent([

        createPartFromUri(myfile.uri, myfile.mimeType),
```

```
        "Describe this audio clip",
    ]),
});
console.log("result.text=", result.text);
```

## TEXT TYPE Files:

### Python code:

```
from google import genai


client = genai.Client()
myfile = client.files.upload(file=media / "poem.txt")
print(f"{myfile=}")


result = client.models.generate_content(
        model="gemini-2.0-flash",
        contents=[myfile, "\n\n", "Can you add a few more lines to this poem?"],
)
print(f"{result.text=}")
```

### Node.js code:

```
// Make sure to include the following import:
// import {GoogleGenAI} from '@google/genai';
const ai = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY });
const myfile = await ai.files.upload({
    file: path.join(media, "poem.txt"),
```

```
});

console.log("Uploaded file:", myfile);


const result = await ai.models.generateContent({
    model: "gemini-2.0-flash",
    contents: createUserContent([
        createPartFromUri(myfile.uri, myfile.mimeType),
        "\n\n",
        "Can you add a few more lines to this poem?",
    ]),
});

console.log("result.text=", result.text);
```

## VIDEO TYPE Files:

### Python code:

```
from google import genai

import time


client = genai.Client()

# Video clip (CC BY 3.0) from https://peach.blender.org/download/

myfile = client.files.upload(file=media / "Big_Buck_Bunny.mp4")

print(f"{myfile=}")


# Poll until the video file is completely processed (state becomes ACTIVE).
```

```
while not myfile.state or myfile.state.name != "ACTIVE":
        print("Processing video...")
        print("File state:", myfile.state)
        time.sleep(5)
        myfile = client.files.get(name=myfile.name)

result = client.models.generate_content(
        model="gemini-2.0-flash", contents=[myfile, "Describe this video clip"]
)
print(f"{result.text=}")
```

Node.js code:

```
// Make sure to include the following import:
// import {GoogleGenAI} from '@google/genai';
const ai = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY });
let myfile = await ai.files.upload({
    file: path.join(media, "Big_Buck_Bunny.mp4"),
    config: { mimeType: "video/mp4" },
});
console.log("Uploaded video file:", myfile);

// Poll until the video file is completely processed (state becomes ACTIVE).
while (!myfile.state || myfile.state.toString() !== "ACTIVE") {
    console.log("Processing video...");
```

```
    console.log("File state: ", myfile.state);
    await sleep(5000);
    myfile = await ai.files.get({ name: myfile.name });
}

const result = await ai.models.generateContent({
    model: "gemini-2.0-flash",
    contents: createUserContent([
        createPartFromUri(myfile.uri, myfile.mimeType),
        "Describe this video clip",
    ]),
});
console.log("result.text=", result.text);
```

## PDF Files:

### Python code:

```
from google import genai

client = genai.Client()
sample_pdf = client.files.upload(file=media / "test.pdf")
response = client.models.generate_content(
        model="gemini-2.0-flash",
        contents=["Give me a summary of this pdf file.", sample_pdf],
)
```

print(response.text)

# Method: files.get

Gets the metadata for the given File.

## Endpoint

get https://generativelanguage.googleapis.com/v1beta/{name=files/*}

## Path parameters

name string

Required. The name of the File to get. Example: files/abc-123 It takes the form files/{file}.

## Request body

The request body must be empty.

## Response body

If successful, the response body contains an instance of File.

## Example request

### Python code:

```
from google import genai


client = genai.Client()

myfile = client.files.upload(file=media / "poem.txt")

file_name = myfile.name

print(file_name)    # "files/*"
```

```
myfile = client.files.get(name=file_name)

print(myfile)
```

### Node.js code:

```
// Make sure to include the following import:

// import {GoogleGenAI} from '@google/genai';

const ai = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY });

const myfile = await ai.files.upload({

    file: path.join(media, "poem.txt"),

});

const fileName = myfile.name;

console.log(fileName);


const fetchedFile = await ai.files.get({ name: fileName });

console.log(fetchedFile);
```

# Method: files.list

Lists the metadata for Files owned by the requesting project.

## Endpoint

get https://generativelanguage.googleapis.com/v1beta/files

## Query parameters

### pageSize integer

Optional. Maximum number of Files to return per page. If unspecified, defaults to 10. Maximum pageSize is 100.

### pageToken string

Optional. A page token from a previous files.list call.

## Request body

The request body must be empty.

## Response body

Response for files.list.

If successful, the response body contains data with the following structure:

Fields

### files[] object (File)

The list of Files.

### nextPageToken string

A token that can be sent as a pageToken into a subsequent files.list call.

### JSON representation

```
{
  "files": [
    {
      object (File)
    }
  ],
```

### JSON representation

```
  "nextPageToken": string
}
```

## Example request

### Python code:

```
from google import genai


client = genai.Client()

print("My files:")

for f in client.files.list():

    print("    ", f.name)
```

### Node.js code:

```
// Make sure to include the following import:

// import {GoogleGenAI} from '@google/genai';

const ai = new GoogleGenAI({ apiKey: process.env.GEMINI_API_KEY });

console.log("My files:");

// Using the pager style to list files

const pager = await ai.files.list({ config: { pageSize: 10 } });

let page = pager.page;

const names = [];

while (true) {
```

```
    for (const f of page) {
      console.log("    ", f.name);
      names.push(f.name);
    }
    if (!pager.hasNextPage()) break;
    page = await pager.nextPage();
}
```

# Method: files.delete

Deletes the File.

## Endpoint

delete https://generativelanguage.googleapis.com/v1beta/{name=files/*}

## Path parameters

name string

Required. The name of the File to delete. Example: files/abc-123 It takes the form files/{file}.

## Request body

The request body must be empty.

## Response body

If successful, the response body is an empty JSON object.

## Example request

### Python code:

```
from google import genai


client = genai.Client()

myfile = client.files.upload(file=media / "poem.txt")


client.files.delete(name=myfile.name)


try:
    result = client.models.generate_content(
        model="gemini-2.0-flash", contents=[myfile, "Describe this file."]
    )
    print(result)
except genai.errors.ClientError:
    pass
```

### Node.js code:

```
// The Gen AI SDK for TypeScript and JavaScript is in preview.

// Some features have not been implemented.
```

# REST Resource: files

## Resource: File

A file uploaded to the API. Next ID: 15

### JSON representation

```
{
  "name": string,
  "displayName": string,
  "mimeType": string,
  "sizeBytes": string,
  "createTime": string,
  "updateTime": string,
  "expirationTime": string,
  "sha256Hash": string,
  "uri": string,
  "downloadUri": string,
  "state": enum (State),
  "source": enum (Source),
  "error": {
    object (Status)
  },

  // metadata
  "videoMetadata": {
```

object (VideoFileMetadata)

}

// Union type

}

## Fields

### Name string

Immutable. Identifier. The File resource name. The ID (name excluding the "files/" prefix) can contain up to 40 characters that are lowercase alphanumeric or dashes (-). The ID cannot start or end with a dash. If the name is empty on create, a unique name will be generated. Example: files/123-456

### displayName string

Optional. The human-readable display name for the File. The display name must be no more than 512 characters in length, including spaces. Example: "Welcome Image"

### mimeType string

Output only. MIME type of the file.

### sizeBytes string (int64 format)

Output only. Size of the file in bytes.

### createTime string (Timestamp format)

Output only. The timestamp of when the File was created.

Uses RFC 3339, where generated output will always be Z-normalized and uses 0, 3, 6 or 9 fractional digits. Offsets other than "Z" are also accepted. Examples: "2014-10-02T15:01:23Z", "2014-10-02T15:01:23.045123456Z" or "2014-10-02T15:01:23+05:30".

### updateTime string (Timestamp format)

Output only. The timestamp of when the File was last updated.

Uses RFC 3339, where generated output will always be Z-normalized and uses 0,

3, 6 or 9 fractional digits. Offsets other than "Z" are also accepted.
Examples: "2014-10-02T15:01:23Z", "2014-10-02T15:01:23.045123456Z" or "2014-10-02T15:01:23+05:30".

### expirationTime string (Timestamp format)

Output only. The timestamp of when the File will be deleted. Only set if the File is scheduled to expire.

Uses RFC 3339, where generated output will always be Z-normalized and uses 0, 3, 6 or 9 fractional digits. Offsets other than "Z" are also accepted.
Examples: "2014-10-02T15:01:23Z", "2014-10-02T15:01:23.045123456Z" or "2014-10-02T15:01:23+05:30".

### sha256 Hash string (bytes format)

Output only. SHA-256 hash of the uploaded bytes.

A base64-encoded string.

### Uri string

Output only. The uri of the File.

### downloadUri string

Output only. The download uri of the File.

### State enum (State)

Output only. Processing state of the File.

### Source enum (Source)

Source of the File.

### Error object (Status)

Output only. Error status if File processing failed.

Metadata for the File. metadata can be only one of the following:

Metadata Union type:

- videoMetadata object (VideoFileMetadata)
- Output only. Metadata for a video.

# VideoFileMetadata

Metadata for a video File.

## JSON representation

```
{
  "videoDuration": string
}
```

## Fields

### videoDuration string (Duration format)

Duration of the video.

A duration in seconds with up to nine fractional digits, ending with 's'.

Example: "3.5s".

# State

States for the lifecycle of a File.

| Enums | |
|---|---|
| STATE_UNSPECIFIED | The default value. This value is used if the state is omitted. |
| PROCESSING | File is being processed and cannot be used for inference yet. |

| Enums | |
|---|---|
| ACTIVE | File is processed and available for inference. |
| FAILED | File failed processing. |

## Source

| Enums | |
|---|---|
| SOURCE_UNSPECIFIED | Used if source is not specified. |
| UPLOADED | Indicates the file is uploaded by the user. |
| GENERATED | Indicates the file is generated by Google. |

## Status

The Status type defines a logical error model that is suitable for different programming environments, including REST APIs and RPC APIs. It is used by gRPC. Each Status message contains three pieces of data: error code, error message, and error details.

You can find out more about this error model and how to work with it in the API Design Guide.

### JSON representation


{

"code": integer,

"message": string,

"details": [

```
    {
      "@type": string,
      field1: ...,
      ...
    }
  ]
}
```

## Fields

### code integer

The status code, which should be an enum value of google.rpc.Code.

### message string

A developer-facing error message, which should be in English. Any user-facing error message should be localized and sent in the google.rpc.Status.details field, or localized by the client.

### details[] object

A list of messages that carry the error details. There is a common set of message types for APIs to use.

An object containing fields of an arbitrary type. An additional field "@type" contains a URI identifying the type. Example: { "id": 1234, "@type": "types.example.com/standard/id" }.