Introduction

This document is intended for developers who want to write applications that interact with YouTube. It explains basic concepts of YouTube and of the API itself. It also provides an overview of the different functions that the API supports.

Before you start

1. You need a Google Account to access the Google API Console, request an API key, and register your application.
2. Create a project in the Google Developers Console and obtain authorization credentials so your application can submit API requests.
3. After creating your project, make sure the YouTube Data API is one of the services that your application is registered to use:
    a. Go to the API Console and select the project that you just registered.
    b. Visit the Enabled APIs page. In the list of APIs, make sure the status is **ON** for the **YouTube Data API v3**.
4. If your application will use any API methods that require user authorization, read the authentication guide to learn how to implement OAuth 2.0 authorization.
5. Select a client library to simplify your API implementation.
6. Familiarize yourself with the core concepts of the JSON (JavaScript Object Notation) data format. JSON is a common, language-independent data format that provides a simple text representation of arbitrary data structures. For more information, see json.org.

Resources and resource types

A resource is an individual data entity with a unique identifier. The table below describes the different types of resources that you can interact with using the API.

Resources

activity	Contains information about an action that a particular user has taken on the YouTube site. User actions that are reported in activity feeds include rating a video, sharing a video, marking a video as a favorite, and posting a channel bulletin, among others.
channel	Contains information about a single YouTube channel.
channelBanner	Identifies the URL to use to set a newly uploaded image as the banner image for a channel.
channelSection	Contains information about a set of videos that a channel has chosen to feature. For example, a section could feature a channel's latest uploads, most popular uploads, or videos from one or more playlists.
guideCategory	Identifies a category that YouTube associates with channels based on their content or other indicators, such as popularity. Guide categories seek to organize channels in a way that makes it easier for YouTube users to find the content they're looking for. While channels could be associated with one or more guide categories, they are not guaranteed to be in any guide categories.
i18nLanguage	Identifies an application language that the YouTube website supports. The application language can

	also be referred to as a UI language.
i18nRegion	Identifies a geographic area that a YouTube user can select as the preferred content region. The content region can also be referred to as a content locale.
playlist	Represents a single YouTube playlist. A playlist is a collection of videos that can be viewed sequentially and shared with other users.
playlistItem	Identifies a resource, such as a video, that is part of a playlist. The playlistItem resource also contains details that explain how the included resource is used in the playlist.
search result	Contains information about a YouTube video, channel, or playlist that matches the search parameters specified in an API request. While a search result points to a uniquely identifiable resource, like a video, it does not have its own persistent data.
subscription	Contains information about a YouTube user subscription. A subscription notifies a user when new videos are added to a channel or when another user takes one of several actions on YouTube, such as uploading a video, rating a video, or commenting on a video.
thumbnail	Identifies thumbnail images associated with a resource.
video	Represents a single YouTube video.
videoCategory	Identifies a category that has been or could be associated with uploaded videos.
watermark	Identifies an image that displays during playbacks of a specified channel's videos. The channel owner can also specify a target channel to which the image links as well as timing details that determine when the watermark appears during video playbacks and then length of time it is visible.

Note that, in many cases, a resource contains references to other resources. For example, a playlistItem resource's snippet.resourceId.videoId property identifies a video resource that, in turn, contains complete information about the video. As another example, a search result contains either a videoId, playlistId, or channelId property that identifies a particular video, playlist, or channel resource.

Supported operations

The following table shows the most common methods that the API supports. Some resources also support other methods that perform functions more specific to those resources. For example, the videos.rate method associates a user rating with a video, and the thumbnails.set method uploads a video thumbnail image to YouTube and associates it with a video.

Operations

list	Retrieves (GET) a list of zero or more resources.
insert	Creates (POST) a new resource.
update	Modifies (PUT) an existing resource to reflect data in your request.
delete	Removes (DELETE) a specific resource.

The API currently supports methods to list each of the supported resource types, and it supports write operations for many resources as well.

The table below identifies the operations that are supported for different types of resources. Operations that insert, update, or delete resources always require user authorization. In some cases, list methods support both authorized and unauthorized requests, where unauthorized requests only retrieve public data while authorized requests can also retrieve information about or private to the currently authenticated user.

Quota usage

The YouTube Data API uses a quota to ensure that developers use the service as intended and do not create applications that unfairly reduce service quality or limit access for others. All API requests, including invalid requests, incur at least a one-point quota cost. You can find the quota available to your application in the API Console.

Projects that enable the YouTube Data API have a default quota allocation of 10,000 units per day, an amount sufficient for the overwhelming majority of our API users. Default quota, which is subject to change, helps us optimize quota allocations and scale our infrastructure in a way that is more meaningful to our API users. You can see your quota usage on the Quotas page in the API Console.

Note: If you reach the quota limit, you can request additional quota by completing the Quota extension request form for YouTube API Services.

Calculating quota usage

Google calculates your quota usage by assigning a cost to each request. Different types of operations have different quota costs. For example:

- A read operation that retrieves a list of resources -- channels, videos, playlists -- usually costs 1 unit.
- A write operation that creates, updates, or deletes a resource usually has costs 50 units.
- A search request costs 100 units.
- A video upload costs 1600 units.

The Quota costs for API requests table shows the quota cost of each API method. With these rules in mind, you can estimate the number of requests that your application could send per day without exceeding your quota.

Partial resources

The API allows, and actually requires, the retrieval of partial resources so that applications avoid transferring, parsing, and storing unneeded data. This approach also ensures that the API uses network, CPU, and memory resources more efficiently.

The API supports two request parameters, which are explained in the following sections, that enable you to identify the resource properties that should be included in API responses.

- The part parameter identifies groups of properties that should be returned for a resource.
- The fields parameter filters the API response to only return specific properties within the requested resource parts.

How to use the part parameter

The part parameter is a required parameter for any API request that retrieves or returns a resource. The parameter identifies one or more top-level (non-nested) resource properties that should be included in an API response. For example, a video resource has the following parts:

- snippet
- contentDetails
- fileDetails
- player
- processingDetails
- recordingDetails
- statistics
- status
- suggestions
- topicDetails

All of these parts are objects that contain nested properties, and you can think of these objects as groups of metadata fields that the API server might (or might not) retrieve. As such, the part parameter requires you to select the resource components that your application actually uses. This requirement serves two key purposes:

- It reduces latency by preventing the API server from spending time retrieving metadata fields that your application doesn't use.
- It reduces bandwidth usage by reducing (or eliminating) the amount of unnecessary data that your application might retrieve.

Over time, as resources add more parts, these benefits will only increase since your application will not be requesting newly introduced properties that it doesn't support.

How to use the fields parameter

The fields parameter filters the API response, which only contains the resource parts identified in the part parameter value, so that the response only includes a specific set of fields. The fields parameter lets you remove nested properties from an API response and thereby further reduce your bandwidth usage. (The part parameter cannot be used to filter nested properties from a response.)

The following rules explain the supported syntax for the fields parameter value, which is loosely based on XPath syntax:

- Use a comma-separated list (fields=a,b) to select multiple fields.
- Use an asterisk (fields=*) as a wildcard to identify all fields.
- Use parentheses (fields=a(b,c)) to specify a group of nested properties that will be included in the API response.
- Use a forward slash (fields=a/b) to identify a nested property.

In practice, these rules often allow several different fields parameter values to retrieve the same API response. For example, if you want to retrieve the playlist item ID, title, and position for every item in a playlist, you could use any of the following values:

- fields=items/id,playlistItems/snippet/title,playlistItems/snippet/position
- fields=items(id,snippet/title,snippet/position)
- fields=items(id,snippet(title,position))

Note: As with all query parameter values, the **fields** parameter value must be URL encoded. For better readability, the examples in this document omit the encoding.

Sample partial requests

The examples below demonstrate how you can use the part and fields parameters to ensure that API responses only include the data that your application uses:

1. Example 1 returns a video resource that includes four parts as well as kind and etag properties.
2. Example 2 returns a video resource that includes two parts as well as kind and etag properties.
3. Example 3 returns a video resource that includes two parts but excludes kind and etag properties.
4. Example 4 returns a video resource that includes two parts but excludes kind and etag as well as some nested properties in the resource's snippet object.

Example 1Example 2Example 3Example 4

URL:

https://www.googleapis.com/youtube/v3/videos?id=7lCDEYXw3mM&key=YOUR_API_KEY
&part=snippet,contentDetails,statistics,status

Description: This example retrieves a video resource and identifies several resource parts that should be included in the API response.

API response


```
{
 "kind": "youtube#videoListResponse",
 "etag": "\"UCBpFjp2h75_b92t44sqraUcyu0/sDAlsG9NGKfr6v5AlPZKSEZdtqA\"",
 "videos": [
  {
   "id": "7lCDEYXw3mM",
   "kind": "youtube#video",
   "etag": "\"UCBpFjp2h75_b92t44sqraUcyu0/iYynQR8AtacsFUwWmrVaw4Smb_Q\"",
   "snippet": {
    "publishedAt": "2012-06-20T22:45:24.000Z",
    "channelId": "UC_x5XG1OV2P6uZZ5FSM9Ttw",
    "title": "Google I/O 101: Q&A On Using Google APIs",
    "description": "Antonio Fuentes speaks to us and takes questions on working with Google APIs and OAuth 2.0.",
    "thumbnails": {
     "default": {
      "url": "https://i.ytimg.com/vi/7lCDEYXw3mM/default.jpg"
     },
     "medium": {
      "url": "https://i.ytimg.com/vi/7lCDEYXw3mM/mqdefault.jpg"
     },
     "high": {
      "url": "https://i.ytimg.com/vi/7lCDEYXw3mM/hqdefault.jpg"
     }
    },
    "categoryId": "28"
   },
   "contentDetails": {
    "duration": "PT15M51S",
    "aspectRatio": "RATIO_16_9"
   },
   "statistics": {
    "viewCount": "3057",
    "likeCount": "25",
    "dislikeCount": "0",
    "favoriteCount": "17",
    "commentCount": "12"
   },
   "status": {
    "uploadStatus": "STATUS_PROCESSED",
    "privacyStatus": "PRIVACY_PUBLIC"
   }
```

```
 }
]
}
```

Optimizing performance

Using ETags

ETags, a standard part of the HTTP protocol, allow applications to refer to a specific version of a particular API resource. The resource could be an entire feed or an item in that feed. This functionality supports the following use cases:

- **Caching and conditional retrieval** – Your application can cache API resources and their ETags. Then, when your application requests a stored resource again, it specifies the ETag associated with that resource. If the resource has changed, the API returns the modified resource and the ETag associated with that version of the resource. If the resource has not changed, the API returns an HTTP 304 response (Not Modified), which indicates that the resource has not changed. Your application can reduce latency and bandwidth usage by serving cached resources in this manner.

  The client libraries for Google APIs differ in their support of ETags. For example, the JavaScript client library supports ETags via a whitelist for allowed request headers that includes If-Match and If-None-Match. The whitelist allows normal browser caching to occur so that if a resource's ETag has not changed, the resource can be served from the browser cache. The Obj-C client, on the other hand, does not support ETags.
- **Protecting against inadvertent overwrites of changes** – ETags help to ensure that multiple API clients don't inadvertently overwrite each other's changes. When updating or deleting a resource, your application can specify the resource's ETag. If the ETag doesn't match the most recent version of that resource, then the API request fails.

Using ETags in your application provides several benefits:

- The API responds more quickly to requests for cached but unchanged resources, yielding lower latency and lower bandwidth usage.
- Your application will not inadvertently overwrite changes to a resource that were made from another API client.

The Google APIs Client Library for JavaScript supports If-Match and If-None-Match HTTP request headers, thereby enabling ETags to work within the context of normal browser caching.

Using gzip

You can also reduce the bandwidth needed for each API response by enabling gzip compression. While your application will need additional CPU time to uncompress API responses, the benefit of consuming fewer network resources usually outweighs that cost.

To receive a gzip-encoded response you must do two things:

- Set the Accept-Encoding HTTP request header to gzip.
- Modify your user agent to contain the string gzip.

The sample HTTP headers below demonstrate these requirements for enabling gzip compression:

Accept-Encoding: gzip
User-Agent: my program (gzip)