Initial Validation Examples

User instruction:
Response: FALSE
User instruction: Create a google calendar event
Response: TRUE
User instruction: Create a google doc event
Response: FALSE
User instruction: aiorduktvnsler8uy47qotnrpet8ywervp
Response: FALSE
User instruction: Delete a google calendar event
Response: TRUE

Documentation for some tasks

User prompt: Create a **calendar** event named “Project Meeting” on Nov. 1 at 9am.

Documentation: https://developers.google.com/calendar/api/v3/reference/events/insert#python

- Service description: Provides methods to create Google **Calendar** events.
- Service Scope: https://www.googleapis.com/auth/calendar.events
- Service API: https://www.googleapis.com/calendar/v3/calendars/primary/events
- Service Method: POST
- Service Params: {}
- Service Body: {'summary': 'Project Meeting', 'description': '', 'start': {'dateTime': '2024-11-01T09:00:00', 'timeZone': 'America/Los_Angeles'}, 'end': {'dateTime': ‘2024-11-01T10:00:00’, 'timeZone': 'America/Los_Angeles'}}

User prompt: Get all **calendar** events on Nov. 1

Documentation: https://developers.google.com/calendar/api/v3/reference/events/list

- Service description: Provides methods to read Google **Calendar** events.
- Service Scope: https://www.googleapis.com/auth/calendar.events.readonly
- Service API: https://www.googleapis.com/calendar/v3/calendars/primary/events
- Service Method: GET
- Service Params: {'timeMin': '2023-11-01T00:00:00Z', 'timeMax': '2023-11-01T23:59:59Z', 'singleEvents': True, 'orderBy': 'startTime', 'timeZone': 'America/Los_Angeles'}
- Service Body: {}

User prompt: Create a Google **Doc** named “Test Doc” in root folder

Documentation: https://developers.google.com/docs/api/reference/rest/v1/documents/create

- Service description: Provides methods to create Google **Docs** without the ability to add initial contents.
- Service Scope: https://www.googleapis.com/auth/drive.file
- Service API: https://docs.googleapis.com/v1/documents
- Service Method: POST
- Service Params: {}
- Service Body: {‘title’:’Test Doc’}

User prompt: Create a Google **Sheet** named “My New Google Sheet” in root folder

Documentation: https://developers.google.com/sheets/api/reference/rest

- Service description: Provides methods to create, read, update, and delete Google **Sheets**.
- Service Scope: https://www.googleapis.com/auth/drive.file
- Service API: https://sheets.googleapis.com/v4/spreadsheets
- Service Method: POST
- Service Params: {}
- Service Body: {"properties": { "title": "My New Google Sheet" } }

Workflow steps example for some instructions

User prompt: Update the calendar event “Project Meeting” from Nov. 1 9am to 10am.

List out all calendar events with query of “Project Meeting” and within time frame of Nov. 1 9am.
Using the event ID retrieved from the last step, update the event to have start time at 10am.

Google Calendar API overview

bookmark_border

The Google Calendar API is a RESTful API that can be accessed through explicit HTTP calls or using the Google Client Libraries. The API exposes most of the features available in the Google Calendar Web interface.

Following is a list of common terms used in the Google Calendar API:

Event

An event on a calendar containing information such as the title, start and end times, and attendees. Events can be either single events or recurring events. An event is represented by an Event resource.

Calendar

A collection of events. Each calendar has associated metadata, such as calendar description or default calendar time zone. The metadata for a single calendar is represented by a Calendar resource.

Calendar List

A list of all calendars on a user's calendar list in the Calendar UI. The metadata for a single calendar that appears on the calendar list is represented by a CalendarListEntry resource. This

metadata includes user-specific properties of the calendar, such as its color or notifications for new events.

Setting

A user preference from the Calendar UI, such as the user's time zone. A single user preference is represented by a Setting Resource.

ACL

An access control rule granting a user (or a group of users) a specified level of access to a calendar. A single access control rule is represented by an ACL resource.

Related topics

To learn about developing with Google Workspace APIs, including handling authentication and authorization, refer to Get started as a Google Workspace developer.

To learn how to configure and run a simple Google Calendar API app, read the Quickstarts overview.

Calendar API Resource Types

bookmark_border

Calendar API background

Calendar concepts

Google Calendar is built on several basic concepts:

Event

A single event on a calendar containing information such as the title of event, start and end times, and attendees.

Calendar

A single calendar entry containing metadata for the calendar such as a description.

Calendar List

A list of all calendars on a user's calendar list in the Calendar UI.

Setting

A user preference from the Calendar UI, such as the user's time zone.

ACL

A single access control rule containing information such as the type and scope of the rule.

Calendar API data model

A resource is an individual data entity with a unique identifier. The Calendar API operates on five types of resources:

Event Resource

Represents a single event on a calendar.

Calendars Resource

Represents metadata for an individual calendar.

CalendarList Resource

Represents metadata for an individual calendar that appears on the user's calendar list in the UI.

Settings Resource

Represents a single user preference from the Calendar UI.

ACL Resource

Represents an ACL.

The Calendar API data model is based on groups of resources, called collections:

Events Collection

Consists of all the Event Resources within a specific Calendar Resource.

CalendarList Collection

Consists of all the CalendarList Resources for a specific user.

Settings Collection

Consists of all the Settings Resources for a specific user.

ACL Collection

Consists of all the ACL Resources applied to a specific calendar.

This guide describes calendars, events, and their relationship to each other.

Calendars

A calendar is a collection of related events, along with additional metadata such as summary, default time zone, location, etc. Each calendar is identified by an ID which is an email address. Calendars can have multiple owners.

Events

An event is an object associated with a specific date or time range. Events are identified by a unique ID. Besides a start and end date-time, events contain other data such as summary, description, location, status, reminders, attachments, etc.

Types of events

Google Calendar supports single and recurring events:

A single event represents a unique occurrence.

A recurring event defines multiple occurrences.

Events may also be timed or all-day:

A timed event occurs between two specific points in time. Timed events use the start.dateTime and end.dateTime fields to specify when they occur.

An all-day event spans an entire day or consecutive series of days. All-day events use the start.date and end.date fields to specify when they occur. Note that the timezone field has no significance for all-day events.

The start and end of the event must both be timed or both be all-day. For example, it is not valid to specify start.date and end.dateTime.

Organizers

Events have a single organizer which is the calendar containing the main copy of the event. Events can also have multiple attendees. An attendee is usually the primary calendar of an invited user.

The following diagram shows the conceptual relationship between calendars, events, and other related elements:

Primary calendars & other calendars

A primary calendar is a special type of calendar associated with a single user account. This calendar is created automatically for each new user account and its ID usually matches the user's primary email address. As long as the account exists, its primary calendar can never be deleted or "un-owned" by the user. However, it can still be shared with other users.

In addition to the primary calendar, you can explicitly create any number of other calendars; these calendars can be modified, deleted, and shared among multiple users.

Calendar & calendar list

The Calendars collection represents all existing calendars. It can be used to create and delete calendars. You can also retrieve or set global properties shared across all users with access to a calendar. For example, a calendar's title and default time zone are global properties.

The CalendarList is a collection of all calendar entries that a user has added to their list (shown in the left panel of the web UI). You can use it to add and remove existing calendars to/from the users' list. You also use it to retrieve and set the values of user-specific calendar properties, such as default reminders. Another example is foreground color, since different users can have different colors set for the same calendar.

The following table compares the meaning of operations for the two collections:

Operation Calendars CalendarList

insert Creates a new secondary calendar. By default, this calendar is also added to the creator's calendar list. Inserts an existing calendar into the user's list.

delete Deletes a secondary calendar. Removes a calendar from the user's list.

get Retrieves calendar metadata e.g. title, time zone. Retrieves metadata plus user-specific customization such as color or override reminders.

patch/update Modifies calendar metadata. Modifies user-specific calendar properties.

Recurring events

Some events occur multiple times on a regular schedule, such as weekly meetings, birthdays, and holidays. Other than having different start and end times, these repeated events are often identical.

Events are called recurring if they repeat according to a defined schedule. Single events are non-recurring and happen only once.

Recurrence rule

The schedule for a recurring event is defined in two parts:

Its start and end fields (which define the first occurrence, as if this were just a stand-alone single event), and

Its recurrence field (which defines how the event should be repeated over time).

The recurrence field contains an array of strings representing one or several RRULE, RDATE or EXDATE properties as defined in RFC 5545.

The RRULE property is the most important as it defines a regular rule for repeating the event. It is composed of several components. Some of them are:

FREQ — The frequency with which the event should be repeated (such as DAILY or WEEKLY). Required.

INTERVAL — Works together with FREQ to specify how often the event should be repeated. For example, FREQ=DAILY;INTERVAL=2 means once every two days.

COUNT — Number of times this event should be repeated.

You can use either COUNT or UNTIL to specify the end of the event recurrence. Don't use both in the same rule.

UNTIL — The date or date-time until which the event should be repeated (inclusive).

BYDAY — Days of the week on which the event should be repeated (SU, MO, TU, etc.). Other similar components include BYMONTH, BYYEARDAY, and BYHOUR.

The RDATE property specifies additional dates or date-times when the event occurrences should happen. For example, RDATE;VALUE=DATE:19970101,19970120. Use this to add extra occurrences not covered by the RRULE.

The EXDATE property is similar to RDATE, but specifies dates or date-times when the event should not happen. That is, those occurrences should be excluded. This must point to a valid instance generated by the recurrence rule.

EXDATE and RDATE can have a time zone, and must be dates (not date-times) for all-day events.

Each of the properties may occur within the recurrence field multiple times. The recurrence is defined as the union of all RRULE and RDATE rules, minus the ones excluded by all EXDATE rules.

Here are some examples of recurrent events:

An event that happens from 6am until 7am every Tuesday and Friday starting from September 15th, 2015 and stopping after the fifth occurrence on September 29th:

...

"start": {

"dateTime": "2015-09-15T06:00:00+02:00",

"timeZone": "Europe/Zurich"

},

"end": {

"dateTime": "2015-09-15T07:00:00+02:00",

"timeZone": "Europe/Zurich"

},

"recurrence": [

"RRULE:FREQ=WEEKLY;COUNT=5;BYDAY=TU,FR"

],

…

An all-day event starting on June 1st, 2015 and repeating every 3 days throughout the month, excluding June 10th but including June 9th and 11th:

...

"start": {

"date": "2015-06-01"

},

"end": {

"date": "2015-06-02"

},

"recurrence": [

"EXDATE;VALUE=DATE:20150610",

"RDATE;VALUE=DATE:20150609,20150611",

"RRULE:FREQ=DAILY;UNTIL=20150628;INTERVAL=3"

],

…

Instances & exceptions

A recurring event consists of several instances: its particular occurrences at different times. These instances act as events themselves.

Recurring event modifications can either affect the whole recurring event (and all of its instances), or only individual instances. Instances that differ from their parent recurring event are called exceptions.

For example, an exception may have a different summary, a different start time, or additional attendees invited only to that instance. You can also cancel an instance altogether without removing the recurring event (instance cancellations are reflected in the event status).

Examples of how to work with recurring events and instances via the Google Calendar API can be found here.

Time zones

A time zone specifies a region that observes a uniform standard time. In the Google Calendar API, you specify time zones using IANA time zone identifiers.

You can set the time zone for both calendars and events. The following sections describe the effects of these settings.

Calendar time zone

The time zone of the calendar is also known as the default time zone because of its implications for query results. The calendar time zone affects the way time values are interpreted or presented by the events.get(), events.list(), and events.instances() methods.

Query result time-zone conversion

Results of the get(), list(), and instances() methods are returned in the time zone that you specify in the timeZone parameter. If you omit this parameter, then these methods all use the calendar time zone as the default.

Matching all-day events to time-bracketed queries

The list(), and instances() methods let you specify start- and end-time filters, with the method returning instances that fall in the specified range. The calendar time zone is used to calculate start and end times of all-day events to determine whether they fall within the filter specification.

Event time zone

Event instances have a start and end time; the specification for these times may include the time zone. You can specify the time zone in several ways; the following all specify the same time:

Include a time zone offset in the dateTime field, for example 2017-01-25T09:00:00-0500.

Specify the time with no offset, for example 2017-01-25T09:00:00, leaving the timeZone field empty (this implicitly uses the default time zone).

Specify the time with no offset, for example 2017-01-25T09:00:00, but use the timeZone field to specify the time zone.

You can also specify event times in UTC if you prefer:

Specify the time in UTC: 2017-01-25T14:00:00Z or use a zero offset 2017-01-25T14:00:00+0000.

The internal representation of the event time is the same in all these cases, but setting the timeZone field attaches a time zone to the event, just as when you set an event time zone using the Calendar UI:

Screenshot fragment showing time zone on an event

For single events, you can specify different time zones for an event's start and end times. (This can help with events—such as travel—that actually start and end in different time zones.) For recurring events, see below.

Recurring event time zone

For recurring events a single timezone must always be specified. It is needed in order to expand the recurrences of the event.

Create events

bookmark_border

Imagine an app that helps users find the best hiking routes. By adding the hiking plan as a calendar event, the users get a lot of help in staying organized automatically. Google Calendar helps them to share the plan and reminds them about it so they can get prepared with no stress. Also, thanks to seamless integration of Google products, Google Now pings them about the time to leave and Google Maps direct them to the meeting spot on time.

This article explains how to create calendar events and add them to your users' calendars.

Create events

bookmark_border

Imagine an app that helps users find the best hiking routes. By adding the hiking plan as a calendar event, the users get a lot of help in staying organized automatically. Google Calendar helps them to share the plan and reminds them about it so they can get prepared with no stress. Also, thanks to seamless integration of Google products, Google Now pings them about the time to leave and Google Maps direct them to the meeting spot on time.

This article explains how to create calendar events and add them to your users' calendars.

Add an event

To create an event, call the events.insert() method providing at least these parameters:

- calendarId is the calendar identifier and can either be the email address of the calendar on which to create the event or a special keyword 'primary' which will use the primary calendar of the logged in user. If you don't know the email address of the calendar you would like to use, you can check it either in the calendar's settings of the Google Calendar web UI (in the section "Calendar Address") or you can look for it in the result of the calendarList.list() call.
- event is the event to create with all the necessary details such as start and end. The only two required fields are the start and end times. See the event reference for the full set of event fields.Specify timed events using the **start.dateTime** and **end.dateTime** fields. For all-day events, use **start.date** and **end.date** instead.

In order to successfully create events, you need to:

- Set your OAuth scope to https://www.googleapis.com/auth/calendar so that you have edit access to the user's calendar.
- Ensure the authenticated user has write access to the calendar with the calendarId you provided (for example by calling calendarList.get() for the calendarId and checking the accessRole).

Add event metadata

You can optionally add event metadata when you create a calendar event. If you choose not to add metadata during creation, you can update many fields using the events.update(); however, some fields, such as the event ID, can only be set during an events.insert() operation.

Location

Adding an address into the location field enables features such as

"time to leave" or displaying a map with the directions.

Event ID

When creating an event, you can choose to generate your own event ID

that conforms to our format requirements. This enables you to keep entities in your local database in sync with events in Google Calendar. It also prevents duplicate event creation if the operation fails at some point after it is successfully executed in the Calendar backend. If no event ID is provided, the server generates one for you. See the event ID reference for more information.

Attendees

The event you create appears on all the primary Google Calendars of

the attendees you included with the same event ID. If you set sendNotifications to true on your insert request, the attendees will also receive an email notification for your event. See the events with multiple attendees guide for more information.

The following examples demonstrate creating an event and setting its metadata:

GoJavaJavaScriptNode.jsPHPPythonRuby

```
# Refer to the Python quickstart on how to setup the environment:

# https://developers.google.com/calendar/quickstart/python

# Change the scope to 'https://www.googleapis.com/auth/calendar' and delete any
```

```
# stored credentials.


event = {
  'summary': 'Google I/O 2015',
  'location': '800 Howard St., San Francisco, CA 94103',
  'description': 'A chance to hear more about Google\'s developer products.',
  'start': {
          'dateTime': '2015-05-28T09:00:00-07:00',
          'timeZone': 'America/Los_Angeles',
  },
  'end': {
          'dateTime': '2015-05-28T17:00:00-07:00',
          'timeZone': 'America/Los_Angeles',
  },
  'recurrence': [
    'RRULE:FREQ=DAILY;COUNT=2'
  ],
  'attendees': [
          {'email': 'lpage@example.com'},
          {'email': 'sbrin@example.com'},
  ],
  'reminders': {
          'useDefault': False,
          'overrides': [
```

```
        {'method': 'email', 'minutes': 24 * 60},
        {'method': 'popup', 'minutes': 10},
        ],
  },
}

event = service.events().insert(calendarId='primary', body=event).execute()
print 'Event created: %s' % (event.get('htmlLink'))
```

Add Drive attachments to events

You can attach Google Drive files such as meeting notes in Docs, budgets in Sheets, presentations in Slides, or any other relevant Google Drive files to your calendar events. You can add the attachment when you create an event with events.insert() or later as part of an update such as with events.patch()

The two parts of attaching a Google Drive file to an event are:

1. Get the file alternateLink URL, title, and mimeType from the Drive API Files resource, typically with the files.get() method.
2. Create or update an event with the attachments fields set in the request body and the supportsAttachments parameter set to true.

The following code example demonstrates how to update an existing event to add an attachment:

JavaPHPPython

```
def add_attachment(calendarService, driveService, calendarId, eventId, fileId):
        file = driveService.files().get(fileId=fileId).execute()
        event = calendarService.events().get(calendarId=calendarId,
                                             eventId=eventId).execute()

        attachments = event.get('attachments', [])
```

```
attachments.append({
        'fileUrl': file['alternateLink'],
        'mimeType': file['mimeType'],
        'title': file['title']
        })

        changes = {
        'attachments': attachments
        }
calendarService.events().patch(calendarId=calendarId, eventId=eventId,
                        body=changes,
                        supportsAttachments=True).execute()
```

Important: You must perform a full sync of all events before enabling the **supportsAttachments** parameter for event modifications when adding attachments support into your existing app that stores events locally. If you do not perform a sync first, you may inadvertently remove existing attachments from user's events.

Add video and phone conferences to events

You can associate events with Hangouts and Google Meet conferences to allow your users to meet remotely via a phone call or a video call.

The conferenceData field can be used to read, copy, and clear existing conference details; it can also be used to request generation of new conferences. To allow creation and modification of conference details, set the conferenceDataVersion request parameter to 1.

There are three types of conferenceData currently supported, as denoted by the conferenceData.conferenceSolution.key.type:

1. Hangouts for consumers (eventHangout)
2. Classic Hangouts for Google Workspace users (deprecated; eventNamedHangout)
3. Google Meet (hangoutsMeet)

You can learn which conference type is supported for any given calendar of a user by looking at the conferenceProperties.allowedConferenceSolutionTypes in the calendars and calendarList collections. You can also learn whether the user prefers to have Hangouts created for all their newly created events by checking the autoAddHangouts setting in the settings collection.

Besides the type, the conferenceSolution also provides the name and the iconUri fields that you can use to represent the conference solution as shown below:

JavaScript

```
const solution = event.conferenceData.conferenceSolution;



const content = document.getElementById("content");

const text = document.createTextNode("Join " + solution.name);

const icon = document.createElement("img");

icon.src = solution.iconUri;



content.appendChild(icon);

content.appendChild(text);
```

You can create a new conference for an event by providing a createRequest with a newly generated requestId which can be a random string. Conferences are created asynchronously, but you can always check the status of your request to let your users know what's happening.

For example, to request conference generation for an existing event:

JavaScript

```
const eventPatch = {

  conferenceData: {

        createRequest: {requestId: "7qxalsvy0e"}

  }

};
```

```
gapi.client.calendar.events.patch({
  calendarId: "primary",
  eventId: "7cbh8rpc10lrc0ckih9tafss99",
  resource: eventPatch,
  sendNotifications: true,
  conferenceDataVersion: 1
}).execute(function(event) {
  console.log("Conference created for event: %s", event.htmlLink);
});
```

The immediate response to this call might not yet contain the fully-populated conferenceData; this is indicated by a status code of pending in the status field. The status code changes to success after the conference information is populated. The entryPoints field contains information about which video and phone URIs are available for your users to dial in.

If you wish to schedule multiple Calendar events with the same conference details, you can copy the entire conferenceData from one event to another.

Copying is useful in certain situations. For example, suppose you are developing a recruiting application that sets up separate events for the candidate and the interviewer—you want to protect the interviewer's identity, but you also want to make sure all participants join the same conference call.