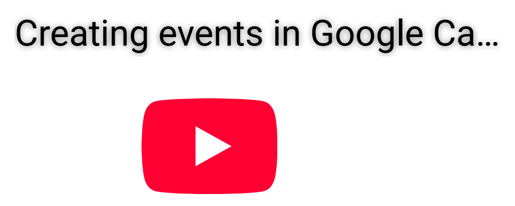
Create events

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()` (/workspace/calendar/v3/reference/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()` (/workspace/calendar/v3/reference/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 (/workspace/calendar/v3/reference/events) 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()`](/workspace/calendar/v3/reference/calendarList/get) (/workspace/calendar/v3/reference/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()` (/workspace/calendar/v3/reference/events/update); however, some fields, such as the event ID, can only be set during an `events.insert()` (/workspace/calendar/v3/reference/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 (/workspace/calendar/v3/reference/events#id) 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 `sendUpdates` to `"all"` or `"externalOnly"` in your insert request, the corresponding attendees receive an email notification for your event. To learn more, see events with multiple attendees (/workspace/calendar/concepts#events_with_attendees).

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

(#go)
Go Java (#java) JavaScript (#javascript)Node.js (#node.js)PHP (#php)Python (#python)Ruby (#ruby)

```
// Refer to the Go quickstart on how to setup the environment:
// https://developers.google.com/workspace/calendar/quickstart/go
// Change the scope to calendar.CalendarScope and delete any stored creden

event := &calendar.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: &calendar.EventDateTime{
    DateTime: "2015-05-28T09:00:00-07:00",
    TimeZone: "America/Los_Angeles",
  },
  End: &calendar.EventDateTime{
    DateTime: "2015-05-28T17:00:00-07:00",
    TimeZone: "America/Los_Angeles",
  },
  Recurrence: []string{"RRULE:FREQ=DAILY;COUNT=2"},
  Attendees: []*calendar.EventAttendee{
    &calendar.EventAttendee{Email:"lpage@example.com"},
    &calendar.EventAttendee{Email:"sbrin@example.com"},
  },
}

calendarId := "primary"
event, err = srv.Events.Insert(calendarId, event).Do()
if err != nil {
  log.Fatalf("Unable to create event. %v\n", err)
}
fmt.Printf("Event created: %s\n", event.HtmlLink)
```

Add Drive attachments to events

You can attach Google Drive (//drive.google.com) 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()`** (/workspace/calendar/v3/reference/events/insert) or later as part of an update such as with **`events.patch()`** (/workspace/calendar/v3/reference/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 (/workspace/drive/v3/reference/files), typically with the **`files.get()`** (/workspace/drive/v3/reference/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:

Java (#java) PHP (#php) Python (#python)

```
public static void addAttachment(Calendar calendarService, Drive driveServ
    String eventId, String fileId) throws IOException {
  File file = driveService.files().get(fileId).execute();
  Event event = calendarService.events().get(calendarId, eventId).execute(

  List<EventAttachment> attachments = event.getAttachments();
  if (attachments == null) {
    attachments = new ArrayList<EventAttachment>();
  }
  attachments.add(new EventAttachment()
      .setFileUrl(file.getAlternateLink())
      .setMimeType(file.getMimeType())
      .setTitle(file.getTitle()));

  Event changes = new Event()
      .setAttachments(attachments);
  calendarService.events().patch(calendarId, eventId, changes)
      .setSupportsAttachments(true)
      .execute();
}
```

Important: You must perform a full sync (/workspace/calendar/v3/sync#initial_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 (//hangouts.google.com) and Google Meet (//meet.google.com) conferences to allow your users to meet remotely via a phone call or a video call.

The `conferenceData` (/workspace/calendar/v3/reference/events#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` (/workspace/calendar/v3/reference/calendars) and `calendarList` (/workspace/calendar/v3/reference/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` (/workspace/calendar/v3/reference/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 (#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
(#javascript)

```
const eventPatch = {
  conferenceData: {
    createRequest: {requestId: "7qxalsvy0e"}
  }
};

gapi.client.calendar.events.patch({
  calendarId: "primary",
  eventId: "7cbh8rpc10lrc0ckih9tafss99",
  resource: eventPatch,
  sendUpdates: "all",
  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 (/workspace/calendar/v3/reference/events#conferenceData.createRequest.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.

Important: You must perform a full sync (/workspace/calendar/v3/sync#initial_full_sync) of all events before enabling the conference data support (by setting of the **`conferenceDataVersion`** request parameter to **`1`** for event modifications) when adding conference support into your existing app that stores events locally. If you do not perform a sync first, you may inadvertently remove existing conferences from users' events.