GOOGLE API ASSIGNMENT

*By Christopher Sigouin & Nimna Nekanayaka*

Table of Contents

[Timeline of Research 2](#_Toc435903100)

[Demonstration Information 7](#_Toc435903101)

[Next steps 7](#_Toc435903102)

[Terminology 8](#_Toc435903103)

[URLs of Interest 9](#_Toc435903104)

# Timeline of Research

START: Monday, November 9, 2015 @ 11:38 AM

**GOAL: Research and analysis**

**OUTCOME:**

Google API Library can be quickly installed with use of ' Composer ' dependency manager. If it is not available you can also download it manually and

Command line to install the library. This will download all dependencies to a vendor folder in the root of the directory that you execute the command.

*php composer.phar require google/apiclient:1.\**

\*\*\* VERY IMPORTANT TO KNOW

<https://developers.google.com/api-client-library/php/auth/web-app>

When you use a Google API Client Library to handle your application's OAuth 2.0 flow, the client library keeps track of when a stored access token can be used and when the application must re-acquire consent, generates correct redirect URLs, and helps to implement redirect handlers that exchange authorization codes for access tokens. An application that carries out the OAuth 2.0 flow without using a client library must correctly complete the same steps.

JSON file types hold configuration information for the example. These can be replaced with literal strings

END: Monday, November 9, 2015 @ 12:38 AM

START: Friday, November 13, 2015 @ 1:00pm

**GOALS: Complete example on Google API Calendar page**

**OUTCOME:**

Completed the Google API PHP quickstart. This will allow access to the API via command line.

To test via command line we completed the following:

Access your cpanel via the following URL:

http://<your-username>.hccis.info/cpanel

Enter your credentials for access

Setup a SSH Key for SSH shell access. (This will allow you to use php via command line)

Create the key

Activate authorization on the key

Download the private key to your client computer that will be used to access the web space hosting the Google API example files. Depending on what SSH client you use you may need to convert the key ( OpenSSH for Linux )

Use the key with your SSH client and ssh into the web space ( Command line should be <your-username>@<your-hostname> )

It will display a URL. Visit the URL, login to your google account and copy the passcode

Enter the passcode into the command line prompt that is requesting one.

*This will auto generate a .json file within a hidden folder “.credentials”. This is so subsequent requests for authorization will be handled*

Although this is not web based, it does allow us to edit the .php file and test the functionality of the API.

In order to setup access from a Web Application (from a website), there is a different process.

END: Friday, November 13, 2015 @ 3:00pm

START: Friday, November 13, 2015 @ 7:00pm

**GOALS: Setup OAuth2.0 authorization to access Google Calendar API**

**OUTCOME:**

We began with the example code from the prior tutorial for the console based output.

A project with type “Web Application " is required to be setup in the Google Developer Console. This will allow you to download a JSON file that contains the following structure:

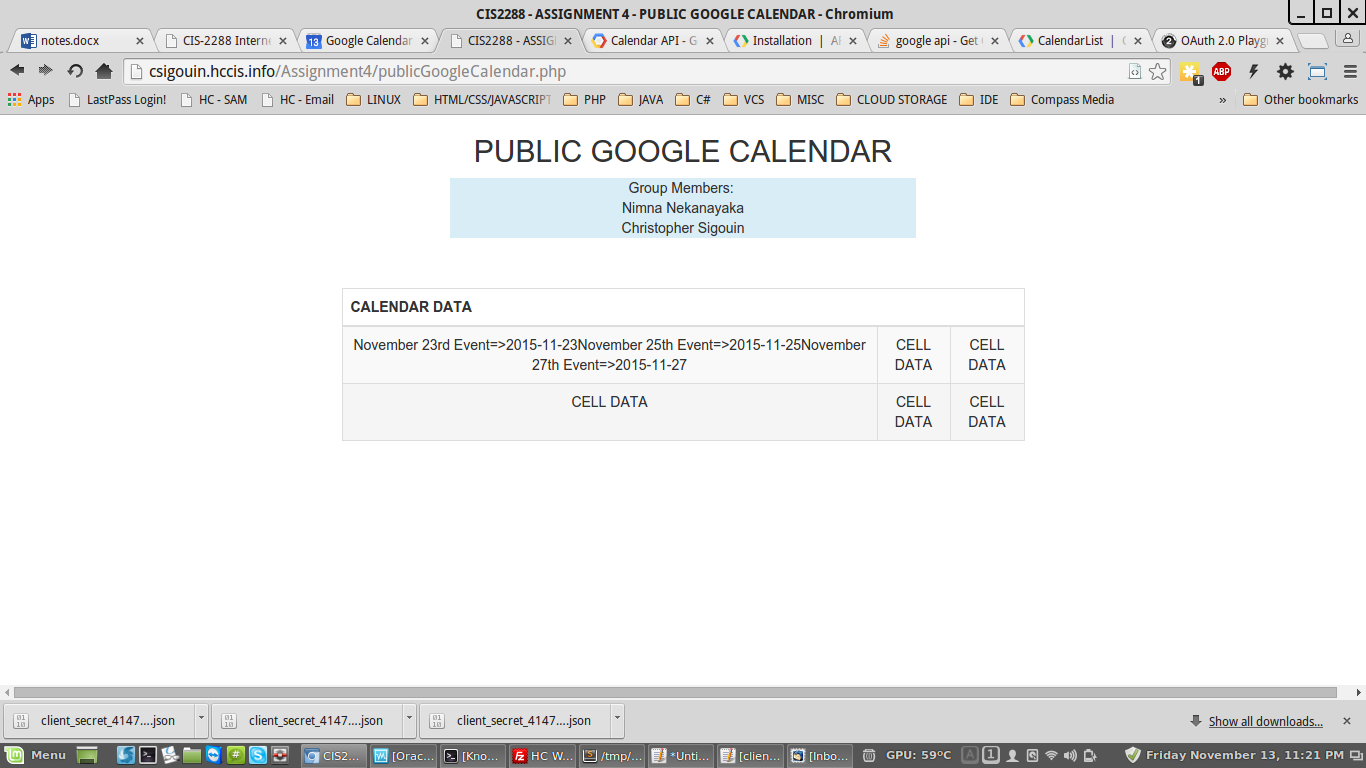
|  |  |
| --- | --- |
| client\_id | This is the ID that matches the project / application created within the Google Developer Console. |
| auth\_uri | <https://accounts.google.com/o/oauth2/auth>  This endpoint is the target of the initial request. It handles active session lookup, authenticating the user, and user consent. This is sent back to the application for use with acquiring a token |
| token\_uri | <https://accounts.google.com/o/oauth2/token>  Each access token is only valid for a short time. Once the current access token expires, the server will need to use the refresh token to get a new one. An application can send a post request to do this. |
| auth\_provider | <https://www.googleapis.com/oauth2/v1/certs>  Google certification ( Version 1…there are 3 versions ) that is used to generate a public key with verification of tokens. It is in the format of a JSON web key.  REF: <http://javadoc.google-api-java-client.googlecode.com/hg-history/1122186c131afd7247ae666fddcd70d5f1308900/1.7.0-beta/com/google/api/client/googleapis/auth/oauth2/GoogleIdTokenVerifier.html>  REF: <https://tools.ietf.org/html/rfc7517> |
| client\_secret | Much like a password. It is the client secret for the project / application created in the Google developer console. |
| redirect\_uris | Authorized redirect URIs are the path to the php script that will receive the response back from the OAuth2 server when a request is made to the API. |

Once created, using OAuth2 Playground, the Client ID and Client Secret are pasted into the settings menu at the bottom.

Clicking through each of the steps produces a small JSON snippet with the request and access token IDs. As we are using the example code that was initially used, we copied that snippet and pasted into the credentials file that script verifies for access to Google API.

***NOTE:***  *The documentation on Google suggests that the credentials be placed outside of your public directory as this would expose the token information and allow someone unauthorized access if they were able to get that information. Both JSON files used ( client\_secret\_web.json & calendar-php-quickstart.json ( our token JSON file )) are stored in this manner.*

Success! Although not formatted very well...



END: Friday, November 13, 2015 @ 10:00pm

START: Friday, November 16, 2015 @ 10:30pm

**GOALS: Manipulate and customize the data on the public facing page**

**OUTCOME:**

Our research utilized a ' read-only ' functionality ( into the Google Calendar API. It appears that it you can also enable editing, adding new events and manipulating the calendar remotely.

Found a reference for the event variables today. This link lists an associative array that holds property types for each event. Utilizing these properties will display the event data onto the website. For example

$event->summary (Provides the title of the event )

$event->description (Provides the description of the event)

$event->location (Provides the location data of the event)

REF: <https://developers.google.com/google-apps/calendar/create-events>

REF: <https://developers.google.com/google-apps/calendar/v3/reference/events>

Event date and time ( As quoted in the above references ):

*The time, as a combined date-time value (formatted according to RFC3339). A time zone offset is required unless a time zone is explicitly specified in timeZone.*

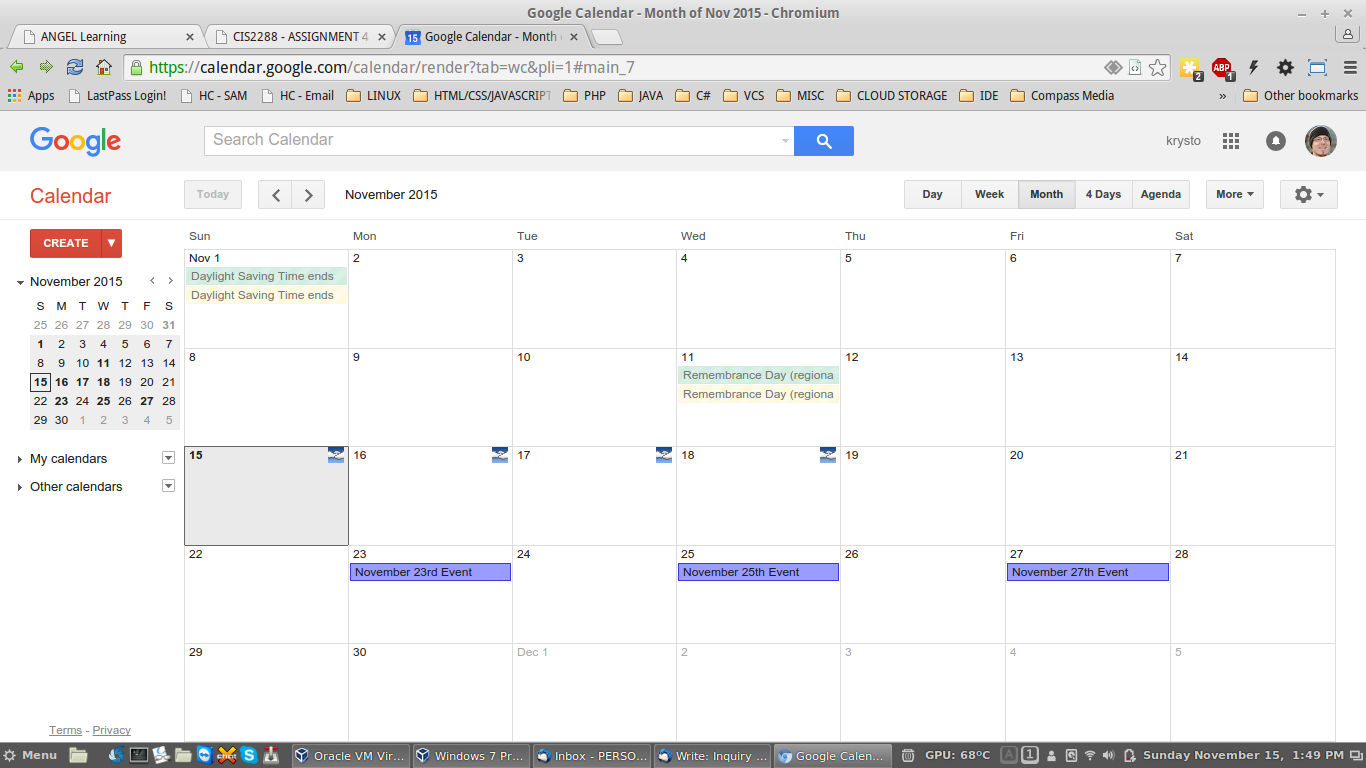
There are two possible attributes within the API for event date and time information ( date and dateTime ). Using 'dateTime', we can format the date and/or time as we wish. It would be required to use the following to produce a more human readable format.

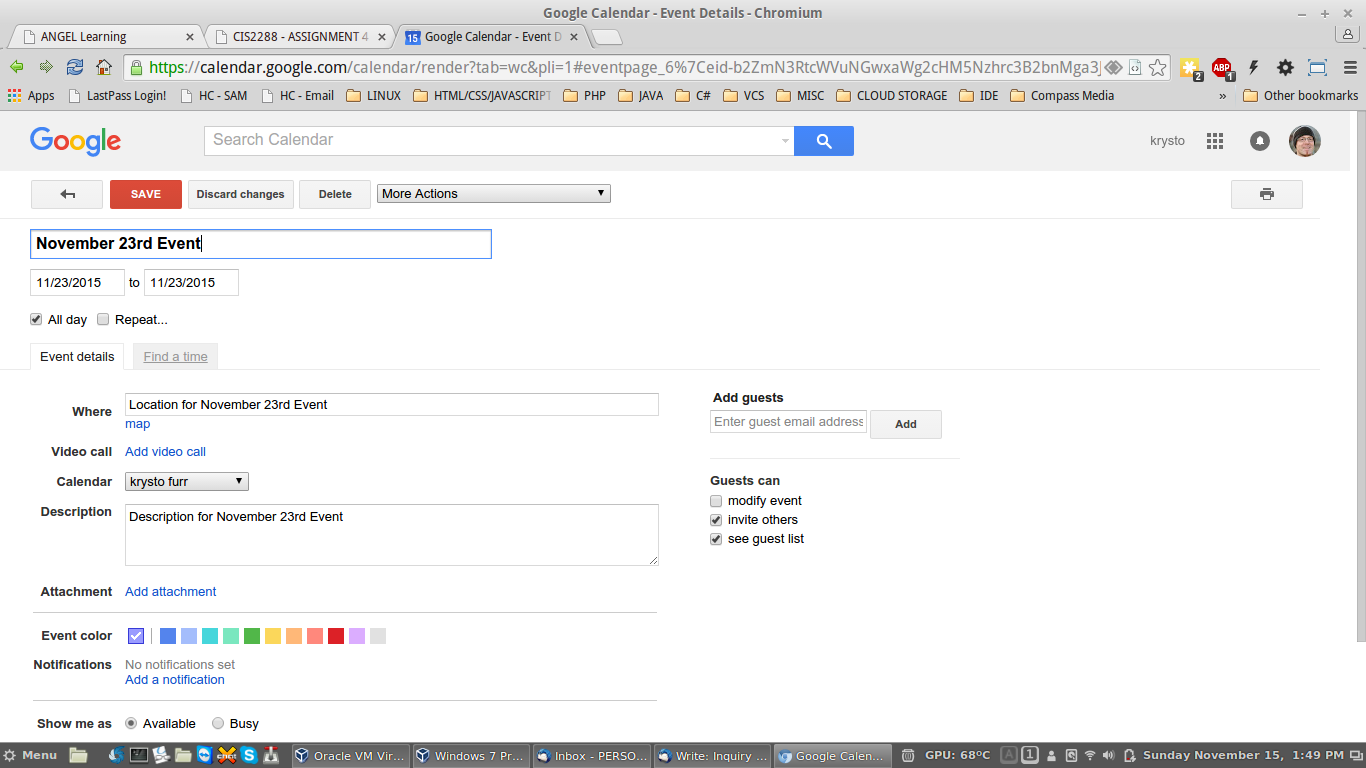
*date("F j, Y, g:i a",strtotime($event->start->dateTime)) - Date and Time*

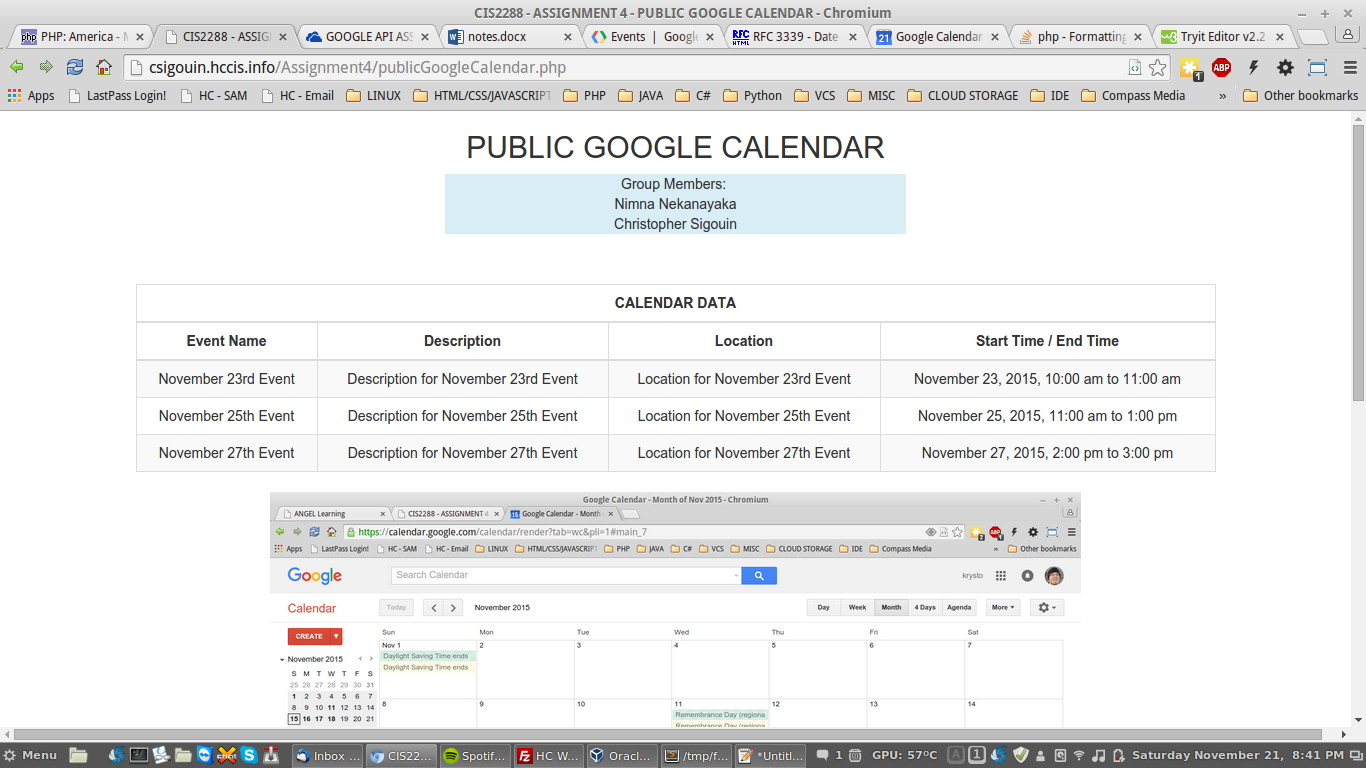
*date("g:i a",strtotime($event->end->dateTime)) - Time Only*

*date("F j, Y",strtotime($event->start->dateTime)) - Date Only*

Included are screenshots showing the event data and a screenshot of the data showing on a public facing website.







END: Friday, November 16, 2015 @ 12:00pm

# Demonstration Information

<http://csigouin.hccis.info/Assignment4/publicGoogleCalendar.php>

# Next steps

|  |
| --- |
| Research the ability to utilize the Google Calendar API for different users instead of a single static user |
| Research the ability to manipulate the calendar further via CRUD |
| Learn to manipulate the API further by limiting the number of events displayed |

# Terminology

**REFERENCE:** <https://en.wikipedia.org/wiki/OAuth#OAuth_2.0>

**OAuth**

OAuth is an [open standard](https://en.wikipedia.org/wiki/Open_standard) for [authorization](https://en.wikipedia.org/wiki/Authorization). OAuth provides client applications a 'secure delegated access' to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with [Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (HTTP), OAuth essentially allows [access tokens](https://en.wikipedia.org/wiki/Access_token) to be issued to third-party clients by an authorization server, with the approval of the resource owner. The client then uses the access token to access the protected resources hosted by the resource server.[[1]](https://en.wikipedia.org/wiki/OAuth#cite_note-1) OAuth is commonly used as a way for Internet users to log into third party websites using their Microsoft, Google, Facebook or Twitter accounts without exposing their password.

**OAuth 2.0**

...is the next evolution of the OAuth protocol and is not backwards compatible with OAuth 1.0. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. The specification and associated RFCs are developed by the IETF OAuth WG;[[5]](https://en.wikipedia.org/wiki/OAuth" \l "cite_note-5) the main framework was published in October 2012. (It was expected to be finalized by the end of 2010, according to [Eran Hammer](https://en.wikipedia.org/w/index.php?title=Eran_Hammer&action=edit&redlink=1).[[6]](https://en.wikipedia.org/wiki/OAuth#cite_note-6) However, due to discordant views about the evolution of OAuth, Hammer left the working group.[[7]](https://en.wikipedia.org/wiki/OAuth#cite_note-oauthtwoandtheroadtohell-7))

[Facebook](https://en.wikipedia.org/wiki/Facebook)'s [Graph API](https://en.wikipedia.org/wiki/Facebook_Platform#Graph_API) only supports OAuth 2.0.[[8]](https://en.wikipedia.org/wiki/OAuth#cite_note-8) [Google](https://en.wikipedia.org/wiki/Google) supports OAuth 2.0 as the recommended authentication mechanism for all of its APIs.[[9]](https://en.wikipedia.org/wiki/OAuth#cite_note-9) As of 2011 [Microsoft](https://en.wikipedia.org/wiki/Microsoft)[[10]](https://en.wikipedia.org/wiki/OAuth#cite_note-10) has added OAuth 2.0 experimental support to their APIs.

The OAuth 2.0 Framework[[11]](https://en.wikipedia.org/wiki/OAuth" \l "cite_note-Dick_Hardt-11) and Bearer Token Usage[[12]](https://en.wikipedia.org/wiki/OAuth#cite_note-12) were published in October 2012. Other documents are still being worked on within the OAuth working group.

**REFERENCE:** <https://developers.google.com/google-apps/calendar/concepts>

**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](https://developers.google.com/google-apps/calendar/concepts#recurring_events). An event is represented by an [Event resource](https://developers.google.com/google-apps/calendar/concepts#events_resource). The Events collection for a given calendar contains all event resources for that calendar.

**Calendar**

A calendar is 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. The Calendars collection contains Calendar resources for all existing calendars.

**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](https://developers.google.com/google-apps/calendar/concepts#calendarLists_resource). This metadata includes user-specific properties of the calendar, such as its color or notifications for new events. The CalendarList collection contains all CalendarListEntry resources for a given user. See also a further explanation of [the difference betweeen the Calendars and CalendarList collections.](https://developers.google.com/google-apps/calendar/concepts#calendar_vs_calendarList)

**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. The Settings collection contains all Setting resources for a given user.

**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. The ACL collection for a given calendar contains all ACL resources that grant access to that calendar.

**Color**

A color presented in the Calendar UI. The Colors resource represents the set of all colors available in the Calendar UI, in two groups: colors available for events and colors available for calendars.

**Free/busy**

A time when a calendar has events scheduled is considered "busy", a time when a calendar has no events is considered "free". The Freebusy resource allows querying for the set of busy times for a given calendar or set of calendars.

# URLs of Interest

Google Calendar API

<https://developers.google.com/google-apps/calendar/>

[API Client Library for PHP (Beta)](https://developers.google.com/api-client-library/php/)

<https://developers.google.com/api-client-library/php/start/get_started>

Google Developer Console

<https://console.developers.google.com>

OAuth 2.0 Playground

<https://developers.google.com/oauthplayground/>

Google API variables ( As shown in the associative array )

<https://developers.google.com/google-apps/calendar/create-events>

<https://developers.google.com/google-apps/calendar/v3/reference/events>