Google Calendar Simple API Documentation

Yevhen Kuzmovych

Sep 06, 2022

Contents

Google Calendar Simple API or *gcsa* is a library that simplifies event management in a Google Calendars. It is a Pythonic object oriented adapter for the official API.

CHAPTER 1

Example usage

1.1 List events

```
from gcsa.google_calendar import GoogleCalendar

calendar = GoogleCalendar('your_email@gmail.com')
for event in calendar:
    print(event)
```

1.2 Create event

```
from gcsa.event import Event

event = Event(
    'The Glass Menagerie',
    start=datetime(2020, 7, 10, 19, 0),
    location='Záhřebská 468/21'
    minutes_before_popup_reminder=15
)
calendar.add_event(event)
```

1.3 Create recurring event

```
from gcsa.recurrence import Recurrence, DAILY

event = Event(
    'Breakfast',
    start=date(2020, 7, 16),
```

(continues on next page)

(continued from previous page)

```
    recurrence=Recurrence.rule(freq=DAILY)
)
calendar.add_event(event)
```

CHAPTER 2

Contents

2.1 Getting started

2.1.1 Installation

To install `gcsa` run the following command:

```
pip install gcsa
```

from sources:

```
git clone git@github.com:kuzmoyev/google-calendar-simple-api.git
cd google-calendar-simple-api
python setup.py install
```

2.1.2 Credentials

Now you need to get your API credentials:

1. Create a new Google Cloud Platform (GCP) project

   Note: You will need to enable the "Google Calendar API" for your project.

2. Configure the OAuth consent screen
3. Create a OAuth client ID credential and download the `credentials.json` file
4. Put downloaded `credentials.json` file into `~/.credentials/` directory

See more options in *Authentication*.

Note: You can put `credentials.json` file anywhere you want and specify the path to it in your code afterwords. But remember not to share it (e.g. add it to `.gitignore`) as it is your private credentials.

Note:

On the first run, your application will prompt you to the default browser to get permissions from you to use your calendar. This will create `token.pickle` file in the same folder (unless specified otherwise) as your `credentials.json`. So don't forget to also add it to `.gitignore` if it is in a GIT repository.

If you don't want to save it in `.pickle` file, you can use `save_token=False` when initializing the `GoogleCalendar`.

2.1.3 Quick example

The following code will create a recurrent event in your calendar starting on January 1 and repeating everyday at 9:00am except weekends and two holidays (April 19, April 22).

Then it will list all events for one year starting today.

For `date`/`datetime` objects you can use Pythons datetime module or as in the example beautiful_date library (*because it's beautiful... just like you*).

```
from gcsa.event import Event
from gcsa.google_calendar import GoogleCalendar
from gcsa.recurrence import Recurrence, DAILY, SU, SA

from beautiful_date import Jan, Apr


calendar = GoogleCalendar('your_email@gmail.com')
event = Event(
    'Breakfast',
    start=(1 / Jan / 2019)[9:00],
    recurrence=[
        Recurrence.rule(freq=DAILY),
        Recurrence.exclude_rule(by_week_day=[SU, SA]),
        Recurrence.exclude_times([
            (19 / Apr / 2019)[9:00],
            (22 / Apr / 2019)[9:00]
        ])
    ],
    minutes_before_email_reminder=50
)

calendar.add_event(event)

for event in calendar:
    print(event)
```

2.2 Authentication

There are several ways to authenticate in `GoogleCalendar`.

2.2.1 Credentials file

If you have a `credentials.json` file (see *Getting started*), `GoogleCalendar` will read all the needed data to generate the token and refresh-token from it.

To read `credentials.json` from the default path (`~/.credentials/credentials.json`) use:

```
gc = GoogleCalendar()
```

In this case, if `~/.credentials/token.pickle` file exists, it will read it and refresh only if needed. If `token.pickle` does not exist, it will be created during authentication flow and saved alongside with `credentials.json` in `~/.credentials/token.pickle`.

To **avoid saving** the token use:

```
gc = GoogleCalendar(save_token=False)
```

After token is generated during authentication flow, it can be accessed in `gc.credentials` field.

To specify `credentials.json` file path use `credentials_path` parameter:

```
gc = GoogleCalendar(credentials_path='path/to/credentials.json')
```

Similarly, if `token.pickle` file exists in the same folder (`path/to/`), it will be used and refreshed only if needed. If it doesn't exist, it will be generated and stored alongside the `credentials.json` (in `path/to/token.pickle`).

To specify different path for the pickled token file use `token_path` parameter:

```
gc = GoogleCalendar(credentials_path='path/to/credentials.json',
                    token_path='another/path/user1_token.pickle')
```

That could be useful if you want to save the file elsewhere, or if you have multiple google accounts.

2.2.2 Token object

If you store/receive/generate the token in a different way, you can pass loaded token directly:

```
from google.oauth2.credentials import Credentials

token = Credentials(
    token='<access_token>',
    refresh_token='<refresh_token>',
    client_id='<client_id>',
    client_secret='<client_secret>',
    scopes=['https://www.googleapis.com/auth/calendar'],
    token_uri='https://oauth2.googleapis.com/token'
)
gc = GoogleCalendar(credentials=token)
```

It will be refreshed using `refresh_token` during initialization of `GoogleCalendar` if needed.

2.2.3 Multiple calendars

To authenticate multiple Google Calendars you should specify different *token_path* for each of them. Otherwise, *gcsa* would overwrite default token file location:

```
gc_primary = GoogleCalendar(token_path='path/to/tokens/token_primary.pickle')
gc_secondary = GoogleCalendar(calendar='f7c1gf7av3g6f2dave17gan4b8@group.calendar.
↪google.com',
                              token_path='path/to/tokens/token_secondary.pickle')
```

2.2.4 Browser authentication timeout

If you'd like to avoid your script hanging in case user closes the browser without finishing authentication flow, you can use the following solution with the help of Pebble.

First, install *Pebble* with `pip install pebble`.

```
from gcsa.google_calendar import GoogleCalendar
from concurrent.futures import TimeoutError
from pebble import concurrent


@concurrent.process(timeout=60)
def create_process():
    return GoogleCalendar()


if __name__ == '__main__':
    try:
        process = create_process()
        gc = process.result()
    except TimeoutError:
        print("User hasn't authenticated in 60 seconds")
```

Thanks to Teraskull for the idea and the example.

2.3 Events

Event in *gcsa* is represented by the class *Event*. It stores all the needed information about the event including its summary, starting and ending dates/times, attachments, reminders, recurrence rules, etc.

Current version of *gcsa* allows you to create a new events, retrieve, update, move and delete existing events.

To do so, create a `GoogleCalendar` instance (see *Getting started* to get your credentials):

```
from gcsa.google_calendar import GoogleCalendar

calendar = GoogleCalendar()
```

2.3.1 List events

This code will print out events for one year starting today:

```
for event in calendar:
    print(event)
```

Specify range of listed events in two ways:

```
calendar.get_events(start_date, end_date, order_by='updated')
```

or

```
calendar[start_date:end_date:'updated']
```

`start_date` and `end_date` can be `date` or `datetime` objects. `order_by` can be *'startTime'* or *'updated'*. If not specified, unspecified stable order is used.

Use `query` parameter for free text search through all event fields (except for extended properties):

```
calendar.get_events(query='Meeting')
calendar.get_events(query='John') # Name of attendee
```

Use `single_events` parameter to expand recurring events into instances and only return single one-off events and instances of recurring events, but not the underlying recurring events themselves.

```
calendar.get_events(single_events=True)
```

2.3.2 Get event by id

```
calendar.get_event('<event_id>')
```

2.3.3 List recurring event instances

```
calendar.get_instances('<recurring_event_id>')
```

or

```
calendar.get_instances(recurring_event)
```

where `recurring_event` is *Event* object with set `event_id`. You'd probably get it from the `get_events` method.

2.3.4 Create event

```
from beautiful_date import Apr, hours
from gcsa.event import Event

start = (22/Apr/2019)[12:00]
end = start + 2 * hours
event = Event('Meeting',
              start=start,
              end=end)
```

or to create an all-day event, use a *date* object:

```
from beautiful_date import Aug, days

start = 1/Aug/2021
end = start + 7 * days
event = Event('Vacation',
              start=start,
              end=end)
```

For `date`/`datetime` objects you can use Pythons datetime module or as in the example beautiful_date library (*because it's beautiful... just like you*).

Now **add** your event to the calendar:

```
calendar.add_event(event)
```

See dedicated pages on how to add *Attendees*, *Attachments*, *Conference*, *Reminders*, and *Recurrence* to an event.

2.3.5 Update event

```
event.location = 'Prague'
calendar.update_event(event)
```

2.3.6 Import event

```
calendar.import_event(event)
```

This operation is used to add a private copy of an existing event to a calendar.

2.3.7 Move event to another calendar

```
calendar.move_event(event, destination_calendar_id='primary')
```

2.3.8 Delete event

```
calendar.delete_event(event)
```

Event has to have `event_id` to be updated, moved or deleted. Events that you get from `get_events()` method already have their ids.

2.3.9 Clear calendar

Remove all events from the calendar:

```
calendar.clear()
```

2.4 Attendees

If you want to add attendee(s) to your event, just create *Attendee* (s) and pass as an `attendees` parameter (you can also pass just an email of the attendee and the *Attendee* will be created for you):

```
from gcsa.attendee import Attendee

attendee = Attendee(
    'attendee@gmail.com',
    display_name='Friend',
    additional_guests=3
)

event = Event('Meeting',
              start=(17/Jul/2020)[12:00],
              attendees=attendee)
```

or

```
event = Event('Meeting',
              start=(17/Jul/2020)[12:00],
              attendees='attendee@gmail.com')
```

You can pass multiple attendees at once in a list.

```
event = Event('Meeting',
              start=(17/Jul/2020)[12:00],
              attendees=[
                  'attendee@gmail.com',
                  Attendee('attendee2@gmail.com', display_name='Friend')
              ])
```

To **notify** attendees about created/updated/deleted event use *send_updates* parameter in *add_event*, *update_event*, and *delete_event* methods. See `SendUpdatesMode` for possible values.

To add attendees to an existing event use its *add_attendee()* method:

```
event.add_attendee(
        Attendee('attendee@gmail.com',
            display_name='Friend',
            additional_guests=3
        )
)
```

or

```
event.add_attendee('attendee@gmail.com')
```

Update event using `update_event()` method to save the changes.

2.5 Attachments

If you want to add attachment(s) to your event, just create *Attachment* (s) and pass as a `attachments` parameter:

```
from gcsa.attachment import Attachment

attachment = Attachment(file_url='https://bit.ly/3lZo0Cc',
                        title='My file',
                        mime_type='application/vnd.google-apps.document')

event = Event('Meeting',
              start=(22/Apr/2019)[12:00],
              attachments=attachment)
```

You can pass multiple attachments at once in a list.

```
event = Event('Meeting',
              start=(22/Apr/2019)[12:00],
              attachments=[attachment1, attachment2])
```

To add attachment to an existing event use its *add_attachment()* method:

```
event.add_attachment('My file',
                     file_url='https://bit.ly/3lZo0Cc',
                     mime_type='application/vnd.google-apps.document')
```

Update event using `update_event()` method to save the changes.

2.6 Conference

To add conference (such as Hangouts or Google Meet) to an event you can use *ConferenceSolution* (for existing conferences) or *ConferenceSolutionCreateRequest* (to create new conference) and pass it as a `conference_solution` parameter:

2.6.1 Existing conference

To add existing conference you need to specify its `solution_type` (see *SolutionType* for available values) and at least one *EntryPoint* in `entry_points` parameter. You can pass single *EntryPoint*:

```
from gcsa.conference import ConferenceSolution, EntryPoint, SolutionType

event = Event(
    'Meeting',
    start=(22 / Nov / 2020)[15:00],
    conference_solution=ConferenceSolution(
        entry_points=EntryPoint(
            EntryPoint.VIDEO,
            uri='https://meet.google.com/aaa-bbbb-ccc'
        ),
        solution_type=SolutionType.HANGOUTS_MEET,
    )
)
```

or multiple entry points in a list:

```
event = Event(
    'Event with conference',
```

(continues on next page)

(continued from previous page)

```
    start=(22 / Nov / 2020)[15:00],
    conference_solution=ConferenceSolution(
        entry_points=[
            EntryPoint(
                EntryPoint.VIDEO,
                uri='https://meet.google.com/aaa-bbbb-ccc'
            ),
            EntryPoint(
                EntryPoint.PHONE,
                uri='tel:+12345678900'
            )
        ],
        solution_type=SolutionType.HANGOUTS_MEET,
    )
)
```

See more parameters for *ConferenceSolution* and *EntryPoint*.

2.6.2 New conference

To generate new conference you need to specify its `solution_type` (see *SolutionType* for available values).

```
from gcsa.conference import ConferenceSolutionCreateRequest, SolutionType

event = Event(
    'Meeting',
    start=(22 / Nov / 2020)[15:00],
    conference_solution=ConferenceSolutionCreateRequest(
        solution_type=SolutionType.HANGOUTS_MEET,
    )
)
```

See more parameters for *ConferenceSolutionCreateRequest*.

Note: Create requests are asynchronous. Check `status` field of event's `conference_solution` to find it's status. If the status is `"success"`, `conference_solution` will contain a *ConferenceSolution* object and you'll be able to access its fields (like `entry_points`). Otherwise (if `status` is `"pending"` or `"failure"`), `conference_solution` will contain a *ConferenceSolutionCreateRequest* object.

```
event = calendar.add_event(
    Event(
        'Meeting',
        start=(22 / Nov / 2020)[15:00],
        conference_solution=ConferenceSolutionCreateRequest(
            solution_type=SolutionType.HANGOUTS_MEET,
        )
    )
)

if event.conference_solution.status == 'success':
    print(event.conference_solution.solution_id)
    print(event.conference_solution.entry_points)
elif event.conference_solution.status == 'pending':
```

(continues on next page)

(continued from previous page)

```
    print('Conference request has not been processed yet.')
elif event.conference_solution.status == 'failure':
    print('Conference request has failed.')
```

2.7 Reminders

To add reminder(s) to an event you can create *EmailReminder* or *PopupReminder* and pass them as a `reminders` parameter (single reminder or list of reminders):

```
from gcsa.reminders import EmailReminder, PopupReminder

event = Event('Meeting',
              start=(22/Apr/2019)[12:00],
              reminders=EmailReminder(minutes_before_start=30))
```

or

```
event = Event('Meeting',
              start=(22/Apr/2019)[12:00],
              reminders=[
                    EmailReminder(minutes_before_start=30),
                    EmailReminder(minutes_before_start=60),
                    PopupReminder(minutes_before_start=15)
              ])
```

You can also simply add reminders by specifying `minutes_before_popup_reminder` and/or `minutes_before_email_reminder` parameter of the *Event* object:

```
event = Event('Meeting',
              start=(22/Apr/2019)[12:00],
              minutes_before_popup_reminder=15,
              minutes_before_email_reminder=30)
```

If you want to add a reminder to an existing event use *add_email_reminder()* and/or *add_popup_reminder()* methods:

```
event.add_popup_reminder(minutes_before_start=30)
event.add_email_reminder(minutes_before_start=50)
```

Update event using `update_event()` method to save the changes.

To use default reminders of the calendar, set `default_reminders` parameter of the *Event* to `True`.

Note: You can add up to 5 reminders to one event.

2.8 Recurrence

With `gcsa` you can create recurrent events. Use *recurrence* module.

There are 8 methods that you can use to define recurrence rules:

- *rule()* - rule that defines recurrence
- *exclude_rule()* - rule that defines excluded dates/datetimes
- *dates()* - date or list of dates to include
- *exclude_dates()* - date or list of dates to exclude
- *times()* - datetime or list of datetimes to include
- *exclude_times()* - datetime or list of datetimes to exclude
- *periods()* - period or list of periods to include
- *exclude_periods()* - period or list of periods to exclude

Note: Methods `{method}` have the same format and parameters as their `exclude_{method}` counterparts. So all examples for `{method}` also apply to `exclude_{method}`.

These methods return strings in `RRULE` format that you can pass as a `recurrence` parameter to the *Event* objects. You can pass one string or list of strings. For example:

```
Event('Breakfast',
      (1/Jan/2020)[9:00],
      (1/Jan/2020)[10:00],
      recurrence=Recurrence.rule(freq=DAILY))
```

or

```
Event('Breakfast',
      (1/Jan/2019)[9:00],
      (1/Jan/2020)[9:00],
      recurrence=[
         Recurrence.rule(freq=DAILY),
         Recurrence.exclude_rule(by_week_day=[SU, SA])
      ])
```

2.8.1 Examples

You will need to import *Recurrence* class and optionally other auxiliary classes and objects:

```
from gcsa.recurrence import Recurrence

# days of the week
from gcsa.recurrence import SU, MO, TU, WE, TH, FR, SA

# possible repetition frequencies
from gcsa.recurrence import SECONDLY, MINUTELY, HOURLY, \
                            DAILY, WEEKLY, MONTHLY, YEARLY
```

Examples were taken from the Internet Calendaring and Scheduling Core Object Specification (iCalendar) and adapted to `gcsa`.

Daily for 10 occurrences:

```
Recurrence.rule(freq=DAILY, count=10)
```

or as `DAILY` is a default frequency:

```
Recurrence.rule(count=10)
```

Every other day:

```
Recurrence.rule(freq=DAILY, interval=2)
```

Every 10 days, 5 occurrences:

```
Recurrence.rule(count=5, interval=10)
```

Every day in January:

```
Recurrence.rule(freq=YEARLY,
                by_month=1,
                by_week_day=[SU,MO,TU,WE,TH,FR,SA])
```

or

```
Recurrence.rule(freq=DAILY, by_month=1)
```

Weekly for 10 occurrences:

```
Recurrence.rule(freq=WEEKLY, count=10)
```

Weekly on Tuesday and Thursday:

```
Recurrence.rule(freq=WEEKLY,
                by_week_day=[TU, TH])
```

Every other week on Monday, Wednesday, and Friday:

```
Recurrence.rule(freq=WEEKLY,
                interval=2,
                by_week_day=[MO, WE, FR])
```

Every other week on Tuesday and Thursday, for 8 occurrences:

```
Recurrence.rule(freq=WEEKLY,
                interval=2,
                count=8,
                by_week_day=[TU, TH])
```

Monthly on the first Friday for 10 occurrences:

```
Recurrence.rule(freq=MONTHLY,
                count=10,
                by_week_day=FR(1))
```

Every other month on the first and last Sunday of the month for 10 occurrences:

```
Recurrence.rule(freq=MONTHLY,
                interval=2,
                count=10,
                by_week_day=[SU(1), SU(-1)])
```

Monthly on the second-to-last Monday of the month for 6 months:

```
Recurrence.rule(freq=MONTHLY,
                count=6,
                by_week_day=MO(-2))
```

Monthly on the third-to-the-last day of the month:

```
Recurrence.rule(freq=MONTHLY,
                by_month_day=-3)
```

Monthly on the 2nd and 15th of the month for 10 occurrences:

```
Recurrence.rule(freq=MONTHLY,
                count=10,
                by_month_day=[2, 15])
```

Monthly on the first and last day of the month for 10 occurrences:

```
Recurrence.rule(freq=MONTHLY,
                count=10,
                by_month_day=[1, -1])
```

Every 18 months on the 10th thru 15th of the month for 10 occurrences:

```
Recurrence.rule(freq=MONTHLY,
                interval=18,
                count=10,
                by_month_day=list(range(10, 16)))
```

Every Tuesday, every other month:

```
Recurrence.rule(freq=MONTHLY,
                interval=2,
                by_week_day=TU)
```

Yearly in June and July for 10 occurrences:

```
Recurrence.rule(freq=YEARLY,
                count=10,
                by_month=[6, 7])
```

Every third year on the 1st, 100th, and 200th day for 10 occurrences:

```
Recurrence.rule(freq=YEARLY,
                interval=3,
                count=10,
                by_year_day=[1, 100, 200])
```

Every 20th Monday of the year:

```
Recurrence.rule(freq=YEARLY,
                by_week_day=MO(20))
```

Monday of week number 20 (where the default start of the week is Monday):

```
Recurrence.rule(freq=YEARLY,
                by_week=20,
                week_start=MO)
```

Every Thursday in March:

```
Recurrence.rule(freq=YEARLY,
                by_month=3,
                by_week_day=TH)
```

The third instance into the month of one of Tuesday, Wednesday, or Thursday, for the next 3 months:

```
Recurrence.rule(freq=MONTHLY,
                count=3,
                by_week_day=[TU, WE, TH],
                by_set_pos=3)
```

The second-to-last weekday of the month:

```
Recurrence.rule(freq=MONTHLY,
                by_week_day=[MO, TU, WE, TH, FR],
                by_set_pos=-2)
```

Every 20 minutes from 9:00 AM to 4:40 PM every day:

```
Recurrence.rule(freq=DAILY,
                by_hour=list(range(9, 17)),
                by_minute=[0, 20, 40])
```

2.9 Serializers

The library implements the JSON serializers for all available Google Calendar objects. JSON format is as specified in the official API documentation. In general, you won't need to use them, `gcsa` serializes everything as needed under the hood. It is documented just so you know they exist and can be used if necessary.

Note: Note that serializer's `to_json` methods ignore read-only fields of the objects. Read only fields of the objects are ones that are passed to the parameters of their `__init__` with underscores, e.g. `Event(_updated=25/Nov/2020)`.

2.9.1 Events serializer

To json

```
from gcsa.event import Event
from gcsa.serializers.event_serializer import EventSerializer


event = Event(
    'Meeting',
    start=(22 / Nov / 2020)[18:00]
)

EventSerializer.to_json(event)
```

```
{
    'summary': 'Meeting',
    'start': {
        'dateTime': '2020-11-22T18:00:00+01:00',
        'timeZone': 'Europe/Prague'
    },
    'end': {
        'dateTime': '2020-11-22T19:00:00+01:00',
        'timeZone': 'Europe/Prague'
    },
    'attachments': [],
    'attendees': [],
    'recurrence': [],
    'reminders': {'useDefault': False},
    'visibility': 'default'
}
```

To object

```
event_json = {
    'start': {
        'dateTime': '2020-11-22T18:00:00+01:00',
        'timeZone': 'Europe/Prague'
    },
    'end': {
        'dateTime': '2020-11-22T19:00:00+01:00',
        'timeZone': 'Europe/Prague'
    },
    'attachments': [],
    'attendees': [],
    'recurrence': [],
    'reminders': {'useDefault': False},
    'summary': 'Meeting',
    'visibility': 'default'
}

EventSerializer.to_object(event_json)
```

```
<Event 2020-11-22 18:00:00+01:00 - Meeting>
```

2.9.2 Attachments serializer

To json

```
from gcsa.attachment import Attachment
from gcsa.serializers.attachment_serializer import AttachmentSerializer

attachment = Attachment(
    file_url='https://bit.ly/3lZo0Cc',
    title='My file',
    mime_type='application/vnd.google-apps.document'
)
```

(continues on next page)

(continued from previous page)

```
AttachmentSerializer.to_json(attachment)
```

```
{
    'title': 'My file',
    'fileUrl': 'https://bit.ly/3lZo0Cc',
    'mimeType': 'application/vnd.google-apps.document'
}
```

To object

```
attachment_json = {
    'fileUrl': 'https://bit.ly/3lZo0Cc',
    'mimeType': 'application/vnd.google-apps.document',
    'title': 'My file'
}

AttachmentSerializer.to_object(attachment_json)
```

```
<Attachment 'My file' - 'https://bit.ly/3lZo0Cc'>
```

2.9.3 Person serializer

To json

```
from gcsa.person import Person
from gcsa.serializers.person_serializer import PersonSerializer

person = Person(
    'john@gmail.com',
    display_name='BFF',
)

PersonSerializer.to_json(person)
```

```
{
    'email': 'john@gmail.com'
    'displayName': 'BFF',
}
```

To object

```
person_json = {
    'email': 'john@gmail.com',
    'displayName': 'BFF',
    'id': '123123',
    'self': True
}

PersonSerializer.to_object(person_json)
```

```
<Person 'john@gmail.com' - 'BFF'>
```

2.9.4 Attendees serializer

To json

```
from gcsa.attendee import Attendee
from gcsa.serializers.attendee_serializer import AttendeeSerializer

attendee = Attendee(
    'john@gmail.com',
    display_name='BFF',
    additional_guests=2
)

AttendeeSerializer.to_json(attendee)
```

```
{
    'email': 'john@gmail.com'
    'displayName': 'BFF',
    'additionalGuests': 2,
}
```

To object

```
attendee_json = {
    'email': 'john@gmail.com',
    'displayName': 'BFF',
    'additionalGuests': 2,
    'responseStatus': 'needsAction'
}

AttendeeSerializer.to_object(attendee_json)
```

```
<Attendee 'john@gmail.com' - response: 'needsAction'>
```

2.9.5 Conference serializer

EntryPoint

To json

```
from gcsa.conference import EntryPoint
from gcsa.serializers.conference_serializer import EntryPointSerializer

entry_point = EntryPoint(
    EntryPoint.VIDEO,
    uri='https://meet.google.com/aaa-bbbb-ccc'
)
```

(continues on next page)

(continued from previous page)

```
EntryPointSerializer.to_json(entry_point)
```

```
{
    'entryPointType': 'video',
    'uri': 'https://meet.google.com/aaa-bbbb-ccc'
}
```

To object

```
entry_point_json = {
    'entryPointType': 'video',
    'uri': 'https://meet.google.com/aaa-bbbb-ccc'
}

EntryPointSerializer.to_object(entry_point_json)
```

```
<EntryPoint video - 'https://meet.google.com/aaa-bbbb-ccc'>
```

ConferenceSolution

To json

```
from gcsa.conference import ConferenceSolution, EntryPoint, SolutionType
from gcsa.serializers.conference_serializer import ConferenceSolutionSerializer

conference_solution = ConferenceSolution(
    entry_points=EntryPoint(
        EntryPoint.VIDEO,
        uri='https://meet.google.com/aaa-bbbb-ccc'
    ),
    solution_type=SolutionType.HANGOUTS_MEET,
)

ConferenceSolutionSerializer.to_json(conference_solution)
```

```
{
    'conferenceSolution': {
        'key': {
            'type': 'hangoutsMeet'
        }
    },
    'entryPoints': [
        {
            'entryPointType': 'video',
            'uri': 'https://meet.google.com/aaa-bbbb-ccc'
        }
    ]
}
```

To object

```
conference_solution_json = {
    'conferenceSolution': {
        'key': {
            'type': 'hangoutsMeet'
        }
    },
    'entryPoints': [
        {
            'entryPointType': 'video',
            'uri': 'https://meet.google.com/aaa-bbbb-ccc'
        }
    ]
}

ConferenceSolutionSerializer.to_object(conference_solution_json)
```

```
<ConferenceSolution hangoutsMeet - [<EntryPoint video - 'https://meet.google.com/aaa-
↪bbbb-ccc'>]>
```

ConferenceSolutionCreateRequest

To json

```
from gcsa.conference import ConferenceSolutionCreateRequest, SolutionType
from gcsa.serializers.conference_serializer import␣
↪ConferenceSolutionCreateRequestSerializer

conference_solution_create_request = ConferenceSolutionCreateRequest(
    solution_type=SolutionType.HANGOUTS_MEET,
)

ConferenceSolutionCreateRequestSerializer.to_json(conference_solution_create_request)
```

```
{
    'createRequest': {
        'conferenceSolutionKey': {
            'type': 'hangoutsMeet'
        },
        'requestId': '30b8e7c4d595445aa73c3feccf4b4f06'
    }
}
```

To object

```
conference_solution_create_request_json = {
    'createRequest': {
        'conferenceSolutionKey': {
            'type': 'hangoutsMeet'
        },
        'requestId': '30b8e7c4d595445aa73c3feccf4b4f06',
```

(continues on next page)

(continued from previous page)

```
        'status': {
            'statusCode': 'pending'
        }
    }
}

ConferenceSolutionCreateRequestSerializer.to_object(conference_solution_create_
↪request_json)
```

```
<ConferenceSolutionCreateRequest hangoutsMeet - status:'pending'>
```

2.9.6 Reminders serializer

To json

```
from gcsa.reminders import EmailReminder, PopupReminder
from gcsa.serializers.reminder_serializer import ReminderSerializer

reminder = EmailReminder(minutes_before_start=30)

ReminderSerializer.to_json(reminder)
```

```
{
    'method': 'email',
    'minutes': 30
}
```

```
reminder = PopupReminder(minutes_before_start=30)

ReminderSerializer.to_json(reminder)
```

```
{
    'method': 'popup',
    'minutes': 30
}
```

To object

```
reminder_json = {
    'method': 'email',
    'minutes': 30
}

ReminderSerializer.to_object(reminder_json)
```

```
<EmailReminder - minutes_before_start:30>
```

```
reminder_json = {
    'method': 'popup',
    'minutes': 30
```

(continues on next page)

(continued from previous page)

```
}

ReminderSerializer.to_object(reminder_json)
```

```
<PopupReminder - minutes_before_start:30>
```

2.10 Code documentation

2.10.1 GoogleCalendar

2.10.2 Event

class gcsa.event.**Event**(*summary*, *start*, *end=None*, ***, *timezone='Etc/UTC'*, *event_id=None*, *description=None*, *location=None*, *recurrence=None*, *color_id=None*, *visibility='default'*, *attendees=None*, *attachments=None*, *conference_solution=None*, *reminders=None*, *default_reminders=False*, *minutes_before_popup_reminder=None*, *minutes_before_email_reminder=None*, *guests_can_invite_others=True*, *guests_can_modify=False*, *guests_can_see_other_guests=True*, *transparency=None*, *_creator=None*, *_organizer=None*, *_created=None*, *_updated=None*, *_recurring_event_id=None*, ***other*)

Parameters

- **summary** – Title of the event.
- **start** – Starting date/datetime.
- **end** – Ending date/datetime. If 'end' is not specified, event is considered as a 1-day or 1-hour event if 'start' is date or datetime respectively.
- **timezone** – Timezone formatted as an IANA Time Zone Database name, e.g. "Europe/Zurich". By default, the computers local timezone is used if it is configured. UTC is used otherwise.
- **event_id** – Opaque identifier of the event. By default is generated by the server. You can specify id as a 5-1024 long string of characters used in base32hex ([a-vA-V0-9]). The ID must be unique per calendar.
- **description** – Description of the event. Can contain HTML.
- **location** – Geographic location of the event as free-form text.
- **recurrence** – RRULE/RDATE/EXRULE/EXDATE string or list of such strings. See *recurrence*
- **color_id** – Color id referring to an entry from colors endpoint (list_event_colors)
- **visibility** – Visibility of the event. Default is default visibility for events on the calendar. See *Visibility*
- **attendees** – Attendee or list of attendees. See *Attendee*. Each attendee may be given as email string or *Attendee* object.
- **attachments** – Attachment or list of attachments. See *Attachment*

- **conference_solution** – *ConferenceSolutionCreateRequest* object to create a new conference or *ConferenceSolution* object for existing conference.
- **reminders** – Reminder or list of reminder objects. See *reminders*
- **default_reminders** – Whether the default reminders of the calendar apply to the event.
- **minutes_before_popup_reminder** – Minutes before popup reminder or None if reminder is not needed.
- **minutes_before_email_reminder** – Minutes before email reminder or None if reminder is not needed.
- **guests_can_invite_others** – Whether attendees other than the organizer can invite others to the event.
- **guests_can_modify** – Whether attendees other than the organizer can modify the event.
- **guests_can_see_other_guests** – Whether attendees other than the organizer can see who the event's attendees are.
- **transparency** – Whether the event blocks time on the calendar. See *Transparency*
- **_creator** – The creator of the event. See *Person*
- **_organizer** – The organizer of the event. See *Person*. If the organizer is also an attendee, this is indicated with a separate entry in attendees with the organizer field set to True. To change the organizer, use the move operation see `move_event()`
- **_created** – Creation time of the event. Read-only.
- **_updated** – Last modification time of the event. Read-only.
- **_recurring_event_id** – For an instance of a recurring event, this is the id of the recurring event to which this instance belongs. Read-only.
- **other** – Other fields that should be included in request json. Will be included as they are. See more in https://developers.google.com/calendar/v3/reference/events

id

add_attendee(*attendee*)
Adds attendee to an event. See *Attendee*. Attendee may be given as email string or *Attendee* object.

add_attachment(*file_url*, *title=None*, *mime_type=None*)
Adds attachment to an event. See *Attachment*

add_email_reminder(*minutes_before_start=60*)
Adds email reminder to an event. See *EmailReminder*

add_popup_reminder(*minutes_before_start=30*)
Adds popup reminder to an event. See *PopupReminder*

add_reminder(*reminder*)
Adds reminder to an event. See *reminders*

is_recurring_instance

class gcsa.event.**Visibility**
Possible values of the event visibility.

- DEFAULT - Uses the default visibility for events on the calendar. This is the default value.
- PUBLIC - The event is public and event details are visible to all readers of the calendar.

- PRIVATE - The event is private and only event attendees may view event details.

DEFAULT = 'default'

PUBLIC = 'public'

PRIVATE = 'private'

class gcsa.event.**Transparency**

Possible values of the event transparency.

- **OPAQUE - Default value. The event does block time on the calendar.** This is equivalent to setting 'Show me as' to 'Busy' in the Calendar UI.
- **TRANSPARENT - The event does not block time on the calendar.** This is equivalent to setting 'Show me as' to 'Available' in the Calendar UI.

OPAQUE = 'opaque'

TRANSPARENT = 'transparent'

2.10.3 Person

class gcsa.person.**Person**(*email=None*, *display_name=None*, *_id=None*, *_is_self=None*)

Represents organizer's, creator's, or primary attendee's fields. For attendees see more in *Attendee*.

Parameters

- **email** – The person's email address, if available
- **display_name** – The person's name, if available
- **_id** – The person's Profile ID, if available. It corresponds to the id field in the People collection of the Google+ API
- **_is_self** – Whether the person corresponds to the calendar on which the copy of the event appears. The default is False (set by Google's API).

2.10.4 Attendees

class gcsa.attendee.**Attendee**(*email*, *display_name=None*, *comment=None*, *optional=None*, *is_resource=None*, *additional_guests=None*, *_id=None*, *_is_self=None*, *_response_status=None*)

Represents attendee of the event.

Parameters

- **email** – The attendee's email address, if available.
- **display_name** – The attendee's name, if available
- **comment** – The attendee's response comment
- **optional** – Whether this is an optional attendee. The default is False.
- **is_resource** – Whether the attendee is a resource. Can only be set when the attendee is added to the event for the first time. Subsequent modifications are ignored. The default is False.
- **additional_guests** – Number of additional guests. The default is 0.
- **_id** – The attendee's Profile ID, if available. It corresponds to the id field in the People collection of the Google+ API

- **_is_self** – Whether this entry represents the calendar on which this copy of the event appears. The default is False (set by Google's API).
- **_response_status** – The attendee's response status. See *ResponseStatus*

class gcsa.attendee.**ResponseStatus**

Possible values for attendee's response status

- NEEDS_ACTION - The attendee has not responded to the invitation.
- DECLINED - The attendee has declined the invitation.
- TENTATIVE - The attendee has tentatively accepted the invitation.
- ACCEPTED - The attendee has accepted the invitation.

NEEDS_ACTION = 'needsAction'

DECLINED = 'declined'

TENTATIVE = 'tentative'

ACCEPTED = 'accepted'

2.10.5 Attachments

class gcsa.attachment.**Attachment**(*file_url*, *title=None*, *mime_type=None*, *_icon_link=None*, *_file_id=None*)

File attachment for the event.

Currently only Google Drive attachments are supported.

Parameters

- **file_url** – A link for opening the file in a relevant Google editor or viewer.
- **title** – Attachment title
- **mime_type** – Internet media type (MIME type) of the attachment. See available MIME types
- **_icon_link** – URL link to the attachment's icon (read only)
- **_file_id** – Id of the attached file (read only)

2.10.6 Conference

class gcsa.conference.**ConferenceSolution**(*entry_points*, *solution_type=None*, *name=None*, *icon_uri=None*, *conference_id=None*, *signature=None*, *notes=None*)

Information about the conference solution, such as Hangouts or Google Meet.

Parameters

- **entry_points** – *EntryPoint* or list of *EntryPoint* s. Information about individual conference entry points, such as URLs or phone numbers. All of them must belong to the same conference.
- **solution_type** – Solution type. See *SolutionType*

  The possible values are:

  – HANGOUT - for Hangouts for consumers (hangouts.google.com)

  - NAMED_HANGOUT - for classic Hangouts for Google Workspace users (hangouts.google.com)
  - HANGOUTS_MEET - for Google Meet (meet.google.com)
  - ADD_ON - for 3P conference providers
- **name** – The user-visible name of this solution. Not localized.
- **icon_uri** – The user-visible icon for this solution.
- **conference_id** – The ID of the conference. Optional. Can be used by developers to keep track of conferences, should not be displayed to users.

  Values for solution types (see *SolutionType*):
  - HANGOUT: unset
  - NAMED_HANGOUT: the name of the Hangout
  - HANGOUTS_MEET: the 10-letter meeting code, for example "aaa-bbbb-ccc"
  - ADD_ON: defined by 3P conference provider
- **signature** – The signature of the conference data. Generated on server side. Must be preserved while copying the conference data between events, otherwise the conference data will not be copied. None for a conference with a failed create request. Optional for a conference with a pending create request.
- **notes** – String of additional notes (such as instructions from the domain administrator, legal notices) to display to the user. Can contain HTML. The maximum length is 2048 characters

class gcsa.conference.**EntryPoint**(*entry_point_type*, *uri=None*, *label=None*, *pin=None*, *access_code=None*, *meeting_code=None*, *passcode=None*, *password=None*)

Information about individual conference entry points, such as URLs or phone numbers.

When creating new conference data, populate only the subset of *meeting_code*, *access_code*, *passcode*, *password*, and *pin* fields that match the terminology that the conference provider uses.

Only the populated fields should be displayed.

Parameters

- **entry_point_type** – The type of the conference entry point.

  Possible values are:
  - VIDEO - joining a conference over HTTP.
    A conference can have zero or one *VIDEO* entry point.
  - PHONE - joining a conference by dialing a phone number.
    A conference can have zero or more *PHONE* entry points.
  - SIP - joining a conference over SIP.
    A conference can have zero or one *SIP* entry point.
  - MORE - further conference joining instructions, for example additional phone numbers.
    A conference can have zero or one *MORE* entry point.
    A conference with only a *MORE* entry point is not a valid conference.
- **uri** – The URI of the entry point. The maximum length is 1300 characters. Format:
  - for *VIDEO*, http: or https: schema is required.

  - for *PHONE*, tel: schema is required.
    The URI should include the entire dial sequence (e.g., tel:+12345678900,,,123456789;1234).
  - for *SIP*, sip: schema is required, e.g., sip:12345678@myprovider.com.
  - for *MORE*, http: or https: schema is required.
- **label** – The label for the URI. Visible to end users. Not localized. The maximum length is 512 characters.

  Examples:
  - for *VIDEO*: meet.google.com/aaa-bbbb-ccc
  - for *PHONE*: +1 123 268 2601
  - for *SIP*: 12345678@altostrat.com
  - for *MORE*: should not be filled
- **pin** – The PIN to access the conference. The maximum length is 128 characters.
- **access_code** – The access code to access the conference. The maximum length is 128 characters. Optional.
- **meeting_code** – The meeting code to access the conference. The maximum length is 128 characters.
- **passcode** – The passcode to access the conference. The maximum length is 128 characters.
- **password** – The password to access the conference. The maximum length is 128 characters.

```
VIDEO = 'video'
PHONE = 'phone'
SIP = 'sip'
MORE = 'more'
ENTRY_POINT_TYPES = ('video', 'phone', 'sip', 'more')
```

class gcsa.conference.**ConferenceSolutionCreateRequest**(*solution_type=None*, *request_id=None*, *_status=None*, *conference_id=None*, *signature=None*, *notes=None*)

A request to generate a new conference and attach it to the event. The data is generated asynchronously. To see whether the data is present check the status field.

Parameters

- **solution_type** – Solution type. See *SolutionType*

  The possible values are:
  - HANGOUT - for Hangouts for consumers (hangouts.google.com)
  - NAMED_HANGOUT - for classic Hangouts for Google Workspace users (hangouts.google.com)
  - HANGOUTS_MEET - for Google Meet (meet.google.com)
  - ADD_ON - for 3P conference providers

- **request_id** – The client-generated unique ID for this request. By default it is generated as UUID. If you specify request_id manually, they should be unique for every new CreateRequest, otherwise request will be ignored.
- **_status** – The current status of the conference create request. Should not be set by developer.

  The possible values are:

  – "pending": the conference create request is still being processed.

  – "failure": the conference create request failed, there are no entry points.

  – "success": the conference create request succeeded, the entry points are populated.
  In this case *ConferenceSolution* with created entry points is stored in the event's *conference_data*. And *ConferenceSolutionCreateRequest* is omitted.
- **conference_id** – The ID of the conference. Optional. Can be used by developers to keep track of conferences, should not be displayed to users.

  Values for solution types (see *SolutionType*):

  – HANGOUT: unset

  – NAMED_HANGOUT: the name of the Hangout

  – HANGOUTS_MEET: the 10-letter meeting code, for example "aaa-bbbb-ccc"

  – ADD_ON: defined by 3P conference provider
- **signature** – The signature of the conference data. Generated on server side. Must be preserved while copying the conference data between events, otherwise the conference data will not be copied. None for a conference with a failed create request. Optional for a conference with a pending create request.
- **notes** – String of additional notes (such as instructions from the domain administrator, legal notices) to display to the user. Can contain HTML. The maximum length is 2048 characters

class gcsa.conference.**SolutionType**

- HANGOUT - for Hangouts for consumers (hangouts.google.com)
- NAMED_HANGOUT - for classic Hangouts for Google Workspace users (hangouts.google.com)
- HANGOUTS_MEET - for Google Meet (meet.google.com)
- ADD_ON - for 3P conference providers

```
HANGOUT = 'eventHangout'

NAMED_HANGOUT = 'eventNamedHangout'

HANGOUTS_MEET = 'hangoutsMeet'

ADD_ON = 'addOn'
```

2.10.7 Reminders

class gcsa.reminders.**Reminder**(*method*, *minutes_before_start*)

Represents base reminder object

Parameters

- **method** – Method of the reminder. Possible values: email or popup

- **minutes_before_start** – Minutes before reminder

class gcsa.reminders.**EmailReminder**(*minutes_before_start=60*)
Represents email reminder object

Parameters **minutes_before_start** – Minutes before reminder

class gcsa.reminders.**PopupReminder**(*minutes_before_start=30*)
Represents popup reminder object

Parameters **minutes_before_start** – Minutes before reminder

2.10.8 Recurrence

class gcsa.recurrence.**Duration**(*w=None*, *d=None*, *h=None*, *m=None*, *s=None*)
Represents properties that contain a duration of time.

Parameters

- **w** – weeks
- **d** – days
- **h** – hours
- **m** – minutes
- **s** – seconds

class gcsa.recurrence.**Recurrence**

static rule(*freq='DAILY'*, *interval=None*, *count=None*, *until=None*, *by_second=None*, *by_minute=None*, *by_hour=None*, *by_week_day=None*, *by_month_day=None*, *by_year_day=None*, *by_week=None*, *by_month=None*, *by_set_pos=None*, *week_start=<gcsa.recurrence._DayOfTheWeek object>*)
This property defines a rule or repeating pattern for recurring events.

Parameters

- **freq** – Identifies the type of recurrence rule. Possible values are SECONDLY, HOURLY, MINUTELY, DAILY, WEEKLY, MONTHLY, YEARLY. Default: DAILY
- **interval** – Positive integer representing how often the recurrence rule repeats
- **count** – Number of occurrences at which to range-bound the recurrence
- **until** – End date of recurrence
- **by_second** – Second or list of seconds within a minute. Valid values are 0 to 60
- **by_minute** – Minute or list of minutes within a hour. Valid values are 0 to 59
- **by_hour** – Hour or list of hours of the day. Valid values are 0 to 23
- **by_week_day** – Day or list of days of the week. Possible values: :py:class:~SUNDAY, :py:class:~MONDAY, :py:class:~TUESDAY, :py:class:~WEDNESDAY, :py:class:~THURSDAY ,:py:class:~FRIDAY, :py:class:~SATURDAY
- **by_month_day** – Day or list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month.
- **by_year_day** – Day or list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year.

- **by_week** – Ordinal or list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1.
- **by_month** – Month or list of months of the year. Valid values are 1 to 12.
- **by_set_pos** – Value or list of values which corresponds to the nth occurrence within the set of events specified by the rule. Valid values are 1 to 366 or -366 to -1. It can only be used in conjunction with another by_xxx parameter.
- **week_start** – The day on which the workweek starts. Possible values: :py:class:~SUNDAY, :py:class:~MONDAY, :py:class:~TUESDAY, :py:class:~WEDNESDAY, :py:class:~THURSDAY ,:py:class:~FRIDAY, :py:class:~SATURDAY

Returns String representing specified recurrence rule in RRULE format.

Note: If none of the by_day, by_month_day, or by_year_day are specified, the day is gotten from start date.

---

static exclude_rule(*freq='DAILY'*, *interval=None*, *count=None*, *until=None*, *by_second=None*, *by_minute=None*, *by_hour=None*, *by_week_day=None*, *by_month_day=None*, *by_year_day=None*, *by_week=None*, *by_month=None*, *by_set_pos=None*, *week_start=<gcsa.recurrence._DayOfTheWeek object>*)

This property defines an exclusion rule or repeating pattern for recurring events.

Parameters

- **freq** – Identifies the type of recurrence rule. Possible values are SECONDLY, HOURLY, MINUTELY, DAILY, WEEKLY, MONTHLY, YEARLY. Default: DAILY
- **interval** – Positive integer representing how often the recurrence rule repeats
- **count** – Number of occurrences at which to range-bound the recurrence
- **until** – End date of recurrence
- **by_second** – Second or list of seconds within a minute. Valid values are 0 to 60
- **by_minute** – Minute or list of minutes within a hour. Valid values are 0 to 59
- **by_hour** – Hour or list of hours of the day. Valid values are 0 to 23
- **by_week_day** – Day or list of days of the week. Possible values: :py:class:~SUNDAY, :py:class:~MONDAY, :py:class:~TUESDAY, :py:class:~WEDNESDAY, :py:class:~THURSDAY ,:py:class:~FRIDAY, :py:class:~SATURDAY
- **by_month_day** – Day or list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month.
- **by_year_day** – Day or list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year.
- **by_week** – Ordinal or list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1.
- **by_month** – Month or list of months of the year. Valid values are 1 to 12.
- **by_set_pos** – Value or list of values which corresponds to the nth occurrence within the set of events specified by the rule. Valid values are 1 to 366 or -366 to -1. It can only be used in conjunction with another by_xxx parameter.

- **week_start** – The day on which the workweek starts. Possible values: :py:class:~SUNDAY, :py:class:~MONDAY, :py:class:~TUESDAY, :py:class:~WEDNESDAY, :py:class:~THURSDAY ,:py:class:~FRIDAY, :py:class:~SATURDAY

Returns String representing specified recurrence rule in RRULE format.

Note: If none of the by_day, by_month_day, or by_year_day are specified, the day is gotten from start date.

---

static dates(*ds*)
Converts date(s) set to RDATE format.

Parameters **ds** – date/datetime object or list of date/datetime objects

Returns RDATE string of dates.

static times(*dts*, *timezone='Etc/UTC'*)
Converts datetime(s) set to RDATE format.

Parameters

- **dts** – datetime object or list of datetime objects
- **timezone** – Timezone formatted as an IANA Time Zone Database name, e.g. "Europe/Zurich". By default, the computers local timezone is used if it is configured. UTC is used otherwise.

Returns RDATE string of datetimes with specified timezone.

static periods(*ps*, *timezone='Etc/UTC'*)
Converts date period(s) to RDATE format.

Period is defined as tuple of starting date/datetime and ending date/datetime or duration as Duration object:
(date/datetime, date/datetime/Duration)

Parameters

- **ps** – Period or list of periods.
- **timezone** – Timezone formatted as an IANA Time Zone Database name, e.g. "Europe/Zurich". By default, the computers local timezone is used if it is configured. UTC is used otherwise.

Returns RDATE string of periods.

static exclude_dates(*ds*)
Converts date(s) set to EXDATE format.

Parameters **ds** – date/datetime object or list of date/datetime objects

Returns EXDATE string of dates.

static exclude_times(*dts*, *timezone='Etc/UTC'*)
Converts datetime(s) set to EXDATE format.

Parameters

- **dts** – datetime object or list of datetime objects

- **timezone** – Timezone formatted as an IANA Time Zone Database name, e.g. "Europe/Zurich". By default, the computers local timezone is used if it is configured. UTC is used otherwise.

Returns EXDATE string of datetimes with specified timezone.

static exclude_periods(*ps*, *timezone='Etc/UTC'*)

Converts date period(s) to EXDATE format.

Period is defined as tuple of starting date/datetime and ending date/datetime or duration as Duration object:
(date/datetime, date/datetime/Duration)

Parameters

- **ps** – Period or list of periods.
- **timezone** – Timezone formatted as an IANA Time Zone Database name, e.g. "Europe/Zurich". By default, the computers local timezone is used if it is configured. UTC is used otherwise.

Returns EXDATE string of periods.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

CHAPTER 4

References

Template for *setup.py* was taken from kennethreitz/setup.py.

Python Module Index

g

Index

S

T

V