Google code labs

Google Tasks API (Labs)

Developer's Guide (v1): Using the API

Important: This version of the Google Tasks API is in Labs, and its features might change unexpectedly until it graduates.

This document describes how to use RESTful calling style and client libraries for various programming languages (currently **Java**, **Python**, and **PHP**) to access and edit Google Tasks data.

Contents

Introduction

This document is intended for developers who want to write applications that can interact with the Google Tasks API.

If you're unfamiliar with Google Tasks API concepts, you should read Getting Started before starting to code.

Setup

How to set up a client library varies by programming language. Select the tab for the language you're using for development.

Using the Google APIs Client Library for Java requires that you download the core Java client library and the the Tasks API Java library .jar files. Download the files and include them in your build path:

Core client library .jar files	Lists the latest .jar files for the Google APIs Client Library for Java.
Tasks API library .jar file	Lists the latest .jar file for the Tasks API library for Java.

You can now import the classes you will need using the following statements:

```
import com.google.api.client.http.HttpTransport;
import com.google.api.client.http.javanet.NetHttpTransport;
import com.google.api.client.json.jackson.JacksonFactory;
import com.google.api.services.tasks.Tasks;
import com.google.api.services.tasks.model.*;
```

Using the Google APIs Client Library for Python requires that you download the Python source. In the future, packages will be provided. Refer to the project page for more details.

Run the following commands to download and install the source:

```
$ hg clone https://google-api-python-client.googlecode.com/hg/ google-
api-python-client
$ cd google-api-python-client
$ sudo python setup.py install
```

You can now import the classes you will need using the following statements:

```
from apiclient.discovery import build
from apiclient.oauth import OAuthCredentials

import httplib2
import oauth2 as oauth
```

Using the Google APIs Client Library for PHP requires that you download the PHP source. In the future, packages will be provided. Refer to the project page for more details.

Run the following commands to download and install the source:

```
$ svn checkout http://google-api-php-client.googlecode.com/svn/trunk/
google-api-php-client-read-only
$ cd google-api-php-read-only
$ # Copy the src directory to your working directory.
```

You can now import the classes you will need using the following statements:

```
require_once "../src/apiClient.php";
require_once "../src/contrib/apiTasksService.php";
```

Modelling data representation classes

One of the main tasks of the Google APIs client libraries is translating JSON to native objects and back. In the Java library, this is done by a set of classes provided by the tasksapi.jar file. In Python, JSON is converted directly into a dictionary or hash, respectively.

Authorizing requests

Every request your application sends to the Google Tasks API must include an authorization token. The token also identifies your application to Google.

About authorization

Before your application can get access to data from a user's Google Account, the application must request authorization from the user.

OAuth 2.0 is the recommended authorization protocol for Google APIs.

If your application has certain unusual authorization requirements, such as logging in at the same time as requesting data access (hybrid) or domain-wide delegation of authority (2LO), then you cannot currently use OAuth 2.0 tokens. In such cases, you must use an API key **in addition to** a valid OAuth 1.0 or AuthSub token. You can find your application's API key in the Google APIs Console, in the Simple API Access section of the API Access pane.

Authorizing requests with OAuth 2.0

All requests to the Google Tasks API must be authorized by an authenticated user.

The details of the authorization process, or "flow," for OAuth 2.0 vary somewhat depending on what kind of application you're writing. The following general process applies to all application types:

1. When you create your application, you register it with Google. Google then provides information you'll need later, such as a client ID and a client secret.
2. Activate the Google Tasks API in the Services pane of the Google APIs Console. (If it isn't listed in the Console, then skip this step.)
3. When your application needs access to user data, it asks Google for a particular **scope** of access.
4. Google displays an **OAuth dialog** to the user, asking them to authorize your application to request some of their data.
5. If the user approves, then Google gives your application a short-lived **access token**.
6. Your application requests user data, attaching the access token to the request.
7. If Google determines that your request and the token are valid, it returns the requested data.

Some flows include additional steps, such as using **refresh tokens** to acquire new access tokens. For detailed information about flows for various types of applications, see Google's OAuth 2.0 documentation.

Here's the OAuth 2.0 scope information for the Google Tasks API:

Scope	Meaning
`https://www.googleapis.com/auth/tasks`	read/write access to Tasks
`https://www.googleapis.com/auth/tasks.readonly`	read-only access to Tasks

To request access using OAuth 2.0, your application needs the scope information, as well as information that Google supplies during application registration (such as the client ID and/or the client secret).

Tip: The Google APIs client libraries can handle some of the authorization process for you. They are available for a variety of programming languages; check the Libraries and Samples page for more details.

Instantiating the client

You must instantiate a client to make requests to the API. All requests to the Google Tasks API require authentication.

The following code demonstrates how to configure your client and authenticate using OAuth 2.0 for native applications.

```
import com.google.api.client.auth.oauth2.draft10.AccessTokenResponse;
import
com.google.api.client.googleapis.auth.oauth2.draft10.GoogleAccessProtect
edResource;
import
com.google.api.client.googleapis.auth.oauth2.draft10.GoogleAccessTokenRe
quest.GoogleAuthorizationCodeGrant;
import
com.google.api.client.googleapis.auth.oauth2.draft10.GoogleAuthorization
RequestUrl;

import com.google.api.client.http.HttpTransport;
import com.google.api.client.http.javanet.NetHttpTransport;
import com.google.api.client.json.jackson.JacksonFactory;

import com.google.api.services.tasks.v1.Tasks;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

...

  public void setUp() throws IOException {
    HttpTransport httpTransport = new NetHttpTransport();
    JacksonFactory jsonFactory = new JacksonFactory();

    // The clientId and clientSecret are copied from the API Access tab
on
    // the Google APIs Console
    String clientId = "YOUR_CLIENT_ID";
    String clientSecret = "YOUR_CLIENT_SECRET";

    // Or your redirect URL for web based applications.
    String redirectUrl = "urn:ietf:wg:oauth:2.0:oob";
    String scope = "https://www.googleapis.com/auth/tasks";

    // Step 1: Authorize -->
    String authorizationUrl = new
GoogleAuthorizationRequestUrl(clientId, redirectUrl, scope)
        .build();

    // Point or redirect your user to the authorizationUrl.
    System.out.println("Go to the following link in your browser:");
    System.out.println(authorizationUrl);

    // Read the authorization code from the standard input stream.
    BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
    System.out.println("What is the authorization code?");
    String code = in.readLine();
    // End of Step 1 <--

    // Step 2: Exchange -->
    AccessTokenResponse response = new
GoogleAuthorizationCodeGrant(httpTransport, jsonFactory,
        clientId, clientSecret, code, redirectUrl).execute();
    // End of Step 2 <--

    GoogleAccessProtectedResource accessProtectedResource = new
GoogleAccessProtectedResource(
        response.accessToken, httpTransport, jsonFactory, clientId,
clientSecret,
        response.refreshToken);

    Tasks service = new Tasks(httpTransport, accessProtectedResource,
jsonFactory);
    service.setApplicationName("YOUR_APPLICATION_NAME");
    ...
  }
...
```

```
import gflags
import httplib2

from apiclient.discovery import build
from oauth2client.file import Storage
from oauth2client.client import OAuth2WebServerFlow
from oauth2client.tools import run

FLAGS = gflags.FLAGS

# Set up a Flow object to be used if we need to authenticate. This
# sample uses OAuth 2.0, and we set up the OAuth2WebServerFlow with
# the information it needs to authenticate. Note that it is called
# the Web Server Flow, but it can also handle the flow for native
# applications
# The client_id and client_secret are copied from the API Access tab on
# the Google APIs Console
FLOW = OAuth2WebServerFlow(
    client_id='YOUR_CLIENT_ID',
    client_secret='YOUR_CLIENT_SECRET',
    scope='https://www.googleapis.com/auth/tasks',
    user_agent='YOUR_APPLICATION_NAME/YOUR_APPLICATION_VERSION')

# To disable the local server feature, uncomment the following line:
# FLAGS.auth_local_webserver = False

# If the Credentials don't exist or are invalid, run through the native
client
# flow. The Storage object will ensure that if successful the good
# Credentials will get written back to a file.
storage = Storage('tasks.dat')
credentials = storage.get()
if credentials is None or credentials.invalid == True:
  credentials = run(FLOW, storage)

# Create an httplib2.Http object to handle our HTTP requests and
authorize it
# with our good Credentials.
http = httplib2.Http()
http = credentials.authorize(http)

# Build a service object for interacting with the API. Visit
# the Google APIs Console
# to get a developerKey for your own application.
service = build(serviceName='tasks', version='v1', http=http,
       developerKey='YOUR_DEVELOPER_KEY')
```

Edit the `src/config.php` file to put in your developer API information.

```
global $apiConfig;

$apiConfig = array(
    // Site name to show in Google's OAuth authentication screen
    'site_name' => 'www.example.org',

    // OAuth2 Setting, you can get these keys on the API Access tab on
    // the Google APIs Console
    'oauth2_client_id' => 'YOUR_CLIENT_ID',
    'oauth2_client_secret' => 'YOUR_CLIENT_SECRET',
    'oauth2_redirect_uri' => 'YOUR_REDIRECT_URL',

    // The developer key; you get this from the Google APIs Console
    'developer_key' => 'YOUR_DEVELOPER_KEY',
    ...

    // Which Authentication, Storage and HTTP IO classes to use.
    'authClass' => 'apiOAuth2',
    ....

    // Definition of service specific values like scopes, OAuth token
URLs, etc
    'services' => array(
        'tasks' => array('scope' =>
'https://www.googleapis.com/auth/tasks'),
    )
);
```

Invoke the Tasks API service and authenticate:

```
<?php
session_start();

require_once "../src/apiClient.php";
require_once "../src/contrib/apiTasksService.php";

$apiClient = new apiClient();
$service = new apiTasksService($apiClient);

if (isset($_SESSION['oauth_access_token'])) {
  $apiClient->setAccessToken($_SESSION['oauth_access_token']);
} else {
  $token = $apiClient->authenticate();
  $_SESSION['oauth_access_token'] = $token;
}
...
```

Working with task lists

The Tasks API allows users to have different task lists to organize their tasks regarding their preferences.

For example, a user can have a task list for work, another one for hobbies, and so on.

This section gives an overview and some examples of the capabilities that the API provides for working with task lists.

Retrieving a user's task lists

To retrieve all of a user's task lists, send an authenticated `GET` request to the following URL:

```
https://www.googleapis.com/tasks/v1/users/username/lists
```

With the appropriate value in place of *`username`*.

Note: The Tasks API currently only supports the special *`username`* value `@me`, indicating the currently authenticated user.

Upon success, the server responds with an HTTP `200 OK` status code and the user's task lists.

Request:

```
GET /tasks/v1/users/@me/lists
```

Response:

```
HTTP/1.1 200 OK

{
  kind: "tasks#taskLists",
  nextPageToken: "nextPageToken",
  items: [
    {
      id: "taskListID",
      kind: "tasks#taskList",
      selfLink:
"https://www.googleapis.com/tasks/v1/users/@me/lists/taskListID",
      title: "Foo's Tasks",
    },
    ...
  ]
}
```

```
TaskLists taskLists = service.tasklists.list().execute();

for (TaskList taskList : taskLists.items) {
  System.out.println(taskList.title);
}
```

```
tasklists = service.tasklists().list().execute()
```

```
for tasklist in tasklists['items']:
  print tasklist['title']
```

```
$taskLists = $service->listTasklists();

foreach ($taskLists['items'] as $taskList) {
  echo $taskList['title'];
}
```

Retrieving a single task list

To retrieve a single task list, send an authenticated `GET` request to the task list's `selfLink` URL.

Note: This request retrieves the task list's title and other information, but doesn't retrieve the tasks in the list. To request a collection of the tasks in the list, see the "Retrieving tasks" section of this document.

The URL is of the form:

```
https://www.googleapis.com/tasks/v1/users/username/lists/taskListID
```

With the appropriate values in place of *`username`* and *`taskListID`*.

Note: The Tasks API currently only supports the special *`username`* value `@me`, indicating the currently authenticated user.

Upon success, the server responds with an HTTP `200 OK` status code and the requested task list.

Request:

```
GET /tasks/v1/users/@me/lists/taskListID
```

Response:

```
HTTP/1.1 200 OK

{
  id: "taskListID",
  kind: "tasks#taskList",
  selfLink:
"https://www.googleapis.com/tasks/v1/users/@me/lists/taskListID",
  title: "My Tasks",
}
```

```
TaskList taskList = service.tasklists.get("taskListID").execute();

System.out.println(taskList.title);
```

```
tasklist = service.tasklists().get(tasklist='tasklistID').execute()

print tasklist['title']
```

```
$tasklist = $service->getTasklists('taskListID');

echo $tasklist['title'];
```

Creating a task list

To create a task list, send an authenticated `POST` request to the user's task lists URL with the task list information in the body.

```
https://www.googleapis.com/tasks/v1/users/username/lists
```

With the appropriate value in place of *`username`*.

Note: The Tasks API currently only supports the special *`username`* value `@me`, indicating the currently authenticated user.

Upon success, the server responds with an HTTP `201 Created` status code and the newly created task list data with some additional elements and properties (shown in bold) that are set by the server, such as `id`, `kind`, and various link elements and properties.

Request:

```
POST /tasks/v1/users/@me/lists

{
  title: "New Task List",
}
```

Response:

```
HTTP/1.1 201 CREATED
Location: https://www.googleapis.com/tasks/v1/users/@me/lists/taskListID

{
  id: "taskListID",
  kind: "tasks#taskList",
  selfLink:
"https://www.googleapis.com/tasks/v1/users/@me/lists/taskListID",
  title: "New Task List",
}
```

```
TaskList taskList = new tasklists();
taskList.title = "New Task List";

TaskList result = service.tasklists.insert(taskList).execute();
System.out.println(result.id);
```

```
tasklist = {
  'title': 'New Task List',
  }

result = service.tasklists().insert(body=tasklist).execute()
print result['id']
```

```
$tasklist = array('title' => 'New Task List');

$result = $service->insertTasklists($tasklist);
echo $result['title'];
```

Updating a task list

To update a task list, first retrieve the task list, then modify the data, then send an authenticated `PUT` request to the task list's `selfLink` URL with the modified task list in the body.

The URL is of the form:

```
https://www.googleapis.com/tasks/v1/users/username/lists/taskListID
```

With the appropriate values in place of *`username`* and *`taskListID`*.

Note: The Tasks API currently only supports the special *`username`* value `@me`, indicating the currently authenticated user.

Upon success, the server responds with an HTTP `200 OK` status code and the updated task list.

Request:

```
PUT /tasks/v1/users/@me/lists/taskListID

{
  id: "taskListID",
  kind: "tasks#taskList",
  selfLink: "https://www.googleapis.com/tasks/v1/users/@me/taskListID",
  title: "New Task List Name",
}
```

Response:

```
HTTP/1.1 200 OK
```

```
Location: https://www.googleapis.com/tasks/v1/users/@me/lists/taskListID

{
  id: "taskListID",
  kind: "tasks#taskList",
  selfLink:
"https://www.googleapis.com/tasks/v1/users/@me/lists/taskListID",
  title: "New Task List Name",
}
```

```
// First retrieve the tasklist to update.
TaskList tasklist = service.tasklists.get("taskListID").execute();
taskList.title = "New Task List Name";

TaskList result = service.tasklists.update(taskList.id,
taskList).execute();
System.out.println(result.title);
```

```
# First retrieve the tasklist to update.
tasklist = service.tasklists().get(tasklist='taskListID').execute()
tasklist['title'] = 'New Task List Name'

result = service.tasklists().update(tasklist=tasklist['id'],
body=tasklist).execute()
print result['title']
```

```
/*
 * First retrieve the tasklist to update.
 */
$tasklist = service->getTasklists('taskListID');
$tasklist['title'] = 'New Task List Name';

$result = $service->updateTasklists($tasklist['id'], $tasklist);
echo $result['title']
```

Deleting a task list

To delete a task list, send an authenticated `DELETE` request to the task list's `selfLink` URL.

The URL is of the form:

```
https://www.googleapis.com/tasks/v1/users/username/lists/taskListID
```

With the appropriate values in place of *`username`* and *`taskListID`*.

Note: The Tasks API currently only supports the special *`username`* value `@me`, indicating the currently authenticated user.

Upon success, the server responds with an HTTP `200 OK` status code.

Request:

```
DELETE /tasks/v1/users/@me/lists/taskListID
```

Response:

```
HTTP/1.1 200 OK
```

```
service.tasklists.delete("taskListID").execute();
```

```
service.tasklists().delete(tasklist='taskListID').execute()
```

```
$service->deleteTasklists('taskListID');
```

Working with tasks

This section gives an overview and some examples of the capabilities that the API provides for working with tasks.

Retrieving tasks

To retrieve tasks that are in a particular task list, send an authenticated `GET` request to the following URL:

```
https://www.googleapis.com/tasks/v1/lists/taskListID/tasks
```

With the appropriate value in place of *`taskListID`*.

Note: The special *`taskListID`* value `@default` can be used to refer to the authenticated user's default task list.

Upon success, the server responds with an HTTP `200 OK` status code and the requested task list's tasks.

Request:

```
GET /tasks/v1/lists/@default/tasks
```

Response:

```
HTTP/1.1 200 OK
```

```
{
  kind: "tasks#tasks",
  nextPageToken: "nextPageToken",
  items: [
    ...,
    {
      id: "taskID",
      kind: "tasks#task",
      selfLink:
"https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/taskID",
      etag: "ETag",
      title: "Something to do",
      notes: "To do",
      parent: "parentTaskID",
      position: "position",
      updated: "2010-10-15T11:03:47.681",
      due: "2010-10-15T11:03:47.681",
      hidden: "false",
      status: "needsAction",
      deleted: "false",
    },
    ...
  ]
}
```

```
Tasks tasks = service.tasks.list("@default").execute();

for (Task task : tasks.items) {
  System.out.println(task.title);
}
```

```
tasks = service.tasks().list(tasklist='@default').execute()

for task in tasks['items']:
  print task['title']
```

```
$tasks = $service->listTasks('@default');

foreach($tasks['items'] as $task) {
  echo $task['title'];
}
```

Retrieving a single task

To retrieve a single task, send an authenticated `GET` request to the task's `selfLink` URL.

The URL is of the form:

```
https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/taskID
```

With the appropriate values in place of *`taskListID`* and *`taskID`*.

Note: The special *`taskListID`* value `@default` can be used to refer to the authenticated user's default task list.

Upon success, the server responds with an HTTP `200 OK` status code and the requested task data.

Request:

```
GET /tasks/v1/lists/@default/tasks/taskID
```

Response:

```
HTTP/1.1 200 OK

{
  id: "taskID",
  kind: "tasks#task",
  selfLink:
"https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/taskID",
  etag: "ETag",
  title: "Something to do",
  notes: "To do",
  parent: "parentTaskID",
  position: "position",
  updated: "2010-10-15T11:03:47.681",
  due: "2010-10-15T11:03:47.681",
  hidden: "false",
  status: "needsAction",
  deleted: "false",
}
```

```
Task task = service.tasks.get("@default", "taskID").execute();

System.out.println(task.title);
```

```
task = service.tasks().get(tasklist='@default', task='taskID').execute()

print task['title']
```

```
$task = $service->getTasks('taskID', '@default');

echo $task['title'];
```

Creating a task

To create a new task, send an authenticated `POST` request to the following URL with the new task data in the body:

```
https://www.googleapis.com/tasks/v1/lists/taskListID/tasks
```

With the appropriate value in place of *`taskListID`*.

Note: The special *`taskListID`* value `@default` can be used to refer to the authenticated user's default task list.

Upon success, the server responds with an HTTP `201 Created` status code and the newly created task data with some additional elements and properties (shown in bold) that are set by the server, such as `id`, `updated`, and various link elements and properties.

Request:

```
POST /tasks/v1/lists/@default/tasks

{
  title: "New Task",
  notes: "Please complete me",
  due: "2010-10-15T12:00:00.000Z",
}
```

Response:

```
HTTP/1.1 201 CREATED
Location:
https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/newTaskID

{
  id: "newTaskID",
  kind: "tasks#task",
  selfLink:
"https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/newTaskID",
  etag: "ETag",
  title: "New Task",
  notes: "Please complete me",
  updated: "2010-10-15T11:03:47.681",
  position: "position",
  due: "2010-10-15T12:00:00.000Z",
  hidden: "false",
  status: "needsAction",
  deleted: "false",
}
```

```
Task task = new Task();
task.title = "New Task";
task.notes = "Please complete me";
task.due = "2010-10-15T12:00:00.000Z";

Task result = service.tasks.insert("@default", task).execute();
System.out.println(result.title);
```

```
task = {
  'title': 'New Task',
  'notes': 'Please complete me',
  'due': '2010-10-15T12:00:00.000Z',
  }

result = service.tasks().insert(tasklist='@default',
body=task).execute()
print result['id']
```

```
$task = array(
  'title' => 'New Task',
  'notes' => 'Please complete me',
  'due' => '2010-10-15T12:00:00.000Z',
);

$result = $service->insertTasks('@default', $task);
echo $result['id'];
```

By default, the new task is prepended to the top level of the task list. To insert the new task under a specific position, use the following query parameters:

The *`parent`* parameter

Specifies the ID of the parent task under which the new task should be inserted; omitting this parameter moves the task to the top level of the list.

The *`previous`* parameter

Specifies the ID of the task after which the new task should be inserted; omitting this parameter moves the task to the first position of the sublist.

Note: For more information about these parameters, see Ordering a task.

Updating a task

To update a task, send an authenticated `PUT` request to the task's `selfLink` URL with the updated task data in the body.

The URL is of the form:

```
https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/taskID
```

With the appropriate values in place of *`taskListID`* and *`taskID`*.

Note: The special *`taskListID`* value `@default` can be used to refer to the authenticated user's default task list.

Upon success, the server responds with an HTTP `200 OK` status code and the updated task.

Request:

```
PUT /tasks/v1/lists/@default/tasks/taskID
```

```
{
  id: "taskID",
  kind: "tasks#task",
  selfLink:
"https://www.googleapis.com/tasks/v1/lists/@default/tasks/taskID",
  etag: "ETag",
  title: "New Task",
  notes: "Please complete me",
  updated: "2010-10-15T11:03:47.681Z",
  due: "2010-10-15T12:00:00.000Z",
  position: "position",
  hidden: "false",
  status: "completed",
  deleted: "false",
}
```

Response:

```
HTTP/1.1 200 OK

{
  id: "taskID",
  kind: "tasks#task",
  selfLink:
"https://www.googleapis.com/tasks/v1/lists/@default/tasks/taskID",
  etag: "newETag",
  title: "New Task",
  notes: "Please complete me",
  updated: "2010-10-15T11:30:00.000Z",
  due: "2010-10-15T12:00:00.000Z",
  position: "position",
  completed: "2010-10-15T11:30:00.000Z",
  hidden: "false",
  status: "completed",
  deleted: "false",
}
```

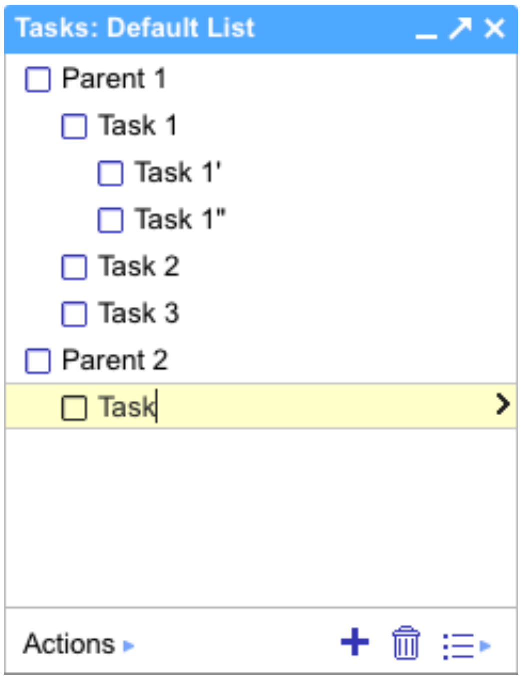
```
// First retrieve the task to update.
Task task = service.tasks.get("@default", "taskID").execute();
task.status = "completed";

Task result = service.tasks.update("@default", task.id, task).execute();
// Print the completed date.
System.out.println(result.completed);
```

```
# First retrieve the task to update.
task = service.tasks().get(tasklist='@default', task='taskID').execute()
task['status'] = 'completed'

result = service.tasks().update(tasklist='@default', task=task['id'],
body=task).execute()
# Print the completed date.
print result['completed']
```

```
/*
 * First retrieve the task to update.
 */
$task = $service->getTasks('@default', 'taskID');
$task['status'] = 'completed';

$result = $service->updateTasks($task['id'], '@default', $task);
/*
 * Print the completed date.
 */
echo $result['completed'];
```

Ordering a task

You can organize tasks to suit the user's needs and preferences. A task can be moved under another task (`parent` task) and/or move to be after another task (`previous`).



To move a task, send an authenticated `POST` request to the following URL with these special parameters:

The *`parent`* parameter

Specifies the ID of the parent task under which the new task should be inserted; omitting this parameter places the task in the top level of the list.

The *`previous`* parameter

Specifies the ID of the task after which the new task should be inserted; omitting this parameter places the task in the first position of the sublist.

The URL is of the form:

```
https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/taskID/move?
parent=parentTaskID&previous=previousTaskID
```

With the appropriate values in place of *`taskListID`*, *`taskID`*, *`parentTaskID`* and *`previousTaskID`*.

Note: The special *`taskListID`* value `@default` can be used to refer to the authenticated user's default task list.

Upon success, the server responds with an HTTP `200 OK` status code and the new task data.

Request:

```
POST
/tasks/v1/lists/@default/tasks/taskID/move?parent=parentTaskID&previous=
previousTaskID
```

Response:

```
HTTP/1.1 200 OK

{
  id: "taskID",
  kind: "tasks#task",
  selfLink:
"https://www.googleapis.com/tasks/v1/lists/@default/tasks/taskID",
  etag: "newETag",
  title: "New Task",
  notes: "Please complete me",
  updated: "2010-10-15T11:30:00.000Z",
  ...,
  parent: "parentTaskID",
  position: "newPosition",
  ...
}
```

```
import com.google.api.services.tasks.v1.Tasks.TasksOperations.Move;
...

Move move = service.tasks.move("@default", "taskID");
move.parent = "parentTaskID";
move.previous = "previousTaskID";
Task result = move.execute();

// Print the new values.
System.out.println(result.parent);
System.out.println(result.position);
```

```
result = service.tasks().move(tasklist='@default', task='taskID',
parent='parentTaskID', previous='previousTaskID').execute()

# Print the new values.
print result['parent']
print result['position']
```

```
$result = $service->moveTasks('taskID', '@default', null,
'parentTaskID', 'previousTaskID');

/*
 * Print the new values.
 */
echo $result['parent'];
echo $result['position'];
```

Note: The `parent` and `previous` parameters can also be used while creating a new task.

Deleting a task

To delete a task, send an authenticated `DELETE` request to the task's `selfLink` URL.

The URL is of the form:

```
https://www.googleapis.com/tasks/v1/lists/taskListID/tasks/taskID
```

With the appropriate values in place of *`taskListID`* and *`taskID`*.

Note: The special *`taskListID`* value `@default` can be used to refer to the authenticated user's default task list.

Upon success, the server responds with an HTTP `200 OK` status code.

Request:

```
DELETE /tasks/v1/lists/@default/tasks/taskID
```

Response:

```
HTTP/1.1 200 OK
```

```
service.tasks.delete("@default", "taskID").execute();
```

```
service.tasks().delete(tasklist='@default', task='taskID').execute()
```

```
$service->deleteTasks('@default', 'taskID');
```

Clearing completed tasks from a task list

To hide all completed tasks in a task list, send an authenticated `POST` request to the following URL:

```
https://www.googleapis.com/tasks/v1/lists/taskListID/clear
```

With the appropriate value in place of *taskListID*.

Upon success, the server responds with an HTTP `200 OK` status code.

Request:

```
POST /tasks/v1/lists/taskListID/clear
```

Response:

```
HTTP/1.1 200 OK
```

```
service.tasks.clear("taskListID").execute();
```

```
service.tasks().clear(tasklist='taskListID').execute()
```

```
$service->clearTasks('taskListID', null);
```

Query parameter reference

The query parameters you can use with the Tasks API are summarized in the following tables. All parameter values need to be URL-encoded.

Standard query parameters

Notes (on API keys and auth tokens):

1. The `key` parameter is required with every request, **unless you provide an OAuth 2.0 token with the request.**
2. You **must** send an authorization token with every request that is marked (AUTHENTICATED). OAuth 2.0 is the preferred authorization protocol.
3. You can provide an OAuth 2.0 token with any request in one of two ways:
   - Using the `oauth_token` query parameter like this: `?oauth_token={oauth2-token}`
   - Using the HTTP `Authorization` header like this: `Authorization: OAuth {oauth2-token}`

All parameters are optional except where noted.

Parameter	Meaning	Notes	Applicability
`alt`	Data format for the response.	• Valid values: `json`, `atom`. • Default value: varies per API.	• Applies to all operations

		• Not all values are available for every API.	operations for all resources.
`callback`	Callback function.	• Name of the JavaScript callback function that handles the response. • Used in JavaScript [JSON-P](#) requests.	
`key`	API key. (REQUIRED*)	• *Required unless you provide an OAuth 2.0 token. • Your [API key](#) identifies your project and provides you with API access, quota, and reports. • Obtain your project's API key from the [APIs console](#).	
`oauth_token`	OAuth 2.0 token for the current user.	• One possible way to provide an [OAuth 2.0 token](#). • See above notes on API keys and auth tokens.	
`prettyprint`	Returns response with identations and line breaks.	• Returns the response in a human-readable format if `true`. • Default value: `true`. • When this is `false`, it can reduce the response payload size, which might lead to better performance in some environments.	
`userIp`	IP address of the site where the request originates.	• Use this if you want to enforce [per-user limits](#).	

Tasks-specific query parameters

Parameter	Meaning	Notes	Applicability
`completedMax`	Upper bound for a task's completion date (as an RFC 3339 timestamp) to filter by.	• Default: `completedMax=2031-01-01` • Use the RFC 3339 timestamp format. For example: `2010-08-09T10:57:00.000-08:00Z`	• [Retrieving tasks](#)
`completedMin`	Lower bound for a task's completion date (as an RFC 3339 timestamp) to filter by.	• Default: `completedMin=1970-01-01` • Use the RFC 3339 timestamp format. For example: `2010-08-09T10:57:00.000-08:00Z`	• [Retrieving tasks](#)

`dueMax`	Upper bound for a task's due date (as an RFC 3339 timestamp) to filter by.	• Default: `dueMax=2031-01-01` • Use the RFC 3339 timestamp format. For example: `2010-08-09T10:57:00.000-08:00Z`	• Retrieving tasks
`dueMin`	Lower bound for a task's due date (as an RFC 3339 timestamp) to filter by.	• Default: `dueMin=1970-01-01` • Use the RFC 3339 timestamp format. For example: `2010-08-09T10:57:00.000-08:00Z`	• Retrieving tasks
`maxResults`	The maximum number of elements to return with this request.	• Default: `maxResults=20` • Maximum allowable value: `maxResults=100.`	• Retrieving a user's task lists • Retrieving tasks
`pageToken`	Token specifying the result page to return.	• The default is to return the first page.	• Retrieving a user's task lists • Retrieving tasks
`parent`	Specify the task's parent task ID.	• No parameter indicates an insertion or a move to the top level of the task list.	• Creating a task • Ordering a task
`previous`	Specify the task's previous task ID.	• No parameter indicates an insertion or a move to the first position in the sublist.	• Creating a task • Ordering a task
`showCompleted`	Specify whether or not to show completed tasks.	• Default: `showCompleted=true`	• Retrieving tasks
`showDeleted`	Specify whether or not to show deleted tasks.	• Default: `showDeleted=false`	• Retrieving tasks
`showHidden`	Specify whether or not to show hidden tasks.	• Default: `showHidden=true`	• Retrieving tasks
`updatedMin`	Lower bound for a task's last modification time (as an RFC 3339 timestamp) to filter by.	• Use the RFC 3339 timestamp format. For example: `2010-08-09T10:57:00.000-08:00Z.` • The default is not to filter by	• Retrieving tasks

		last modification time.	