Last updated February 27, 2017. | **Unknown author**

Examples

In this section, we provide examples of each access type and application type.

Simple API example

For this example, we use simple API access for a command-line application. It calls the Google Books API to list all books about Android.

Setup for example

1. **Activate the Books API**: Read about API activation and activate the Books API.
2. **Get your Simple API key**: See the API keys documentation.

Code for example

Download or expand the code below. This script is well commented to explain each step.

▸
Click to expand

```
#!/usr/bin/python
#
# Copyright 2012 Google Inc. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
```

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import pprint
import sys
from apiclient.discovery import build

# For this example, the API key is provided as a command-line argument.
api_key = sys.argv[1]

# The apiclient.discovery.build() function returns an instance of an API
service
# object that can be used to make API calls. The object is constructed with
# methods specific to the books API. The arguments provided are:
#   name of the API ('books')
#   version of the API you are using ('v1')
#   API key
service = build('books', 'v1', developerKey=api_key)

# The books API has a volumes().list() method that is used to list books
# given search criteria. Arguments provided are:
#   volumes source ('public')
#   search query ('android')
# The method returns an apiclient.http.HttpRequest object that
encapsulates
# all information needed to make the request, but it does not call the API.
request = service.volumes().list(source='public', q='android')

# The execute() function on the HttpRequest object actually calls the API.
# It returns a Python object built from the JSON response. You can print
this
# object or refer to the Books API documentation to determine its structure.
response = request.execute()
pprint.pprint(response)

# Accessing the response like a dict object with an 'items' key returns a list
# of item objects (books). The item object is a dict object with a
```

```
'volumeInfo'
# key. The volumeInfo object is a dict with keys 'title' and 'authors'.
print 'Found %d books:' % len(response['items'])
for book in response.get('items', []):
  print 'Title: %s, Authors: %s' % (
    book['volumeInfo']['title'],
    book['volumeInfo']['authors'])
```

Note that the build() function is used to create an API-specific service object. You will always use this function for this purpose. In this case, the API key is passed to the build() function; however, API keys are only relevant to simple API calls like this. The authorization examples below require more code.

Run the example

Copy the script to some directory on your computer, open a terminal, go to the directory, and execute the following command:

$ python simple_api_cmd_line_books.py *your_api_key*

The command outputs a list of Android books.

Authorized API for installed application example

For this example, we use authorized API access for a command-line application. It calls the Google Calendar API to list a user's calendar events.

Setup for example

1. **Activate the Calendar API**: Read about API activation and activate the Calendar API.
2. **Get your client ID and client secret**: Get a client ID and secret for *installed applications*.
3. **Create calendar events**: In this example, you will read a user's calendar events. You can use any Google user account that you own, including the account associated with the application's Google APIs Console project. For the target user, create a few calendar events if none exist already.

Code for example

Download or expand the code below. This script is well commented to explain each step.

▸
Click to expand

```
#!/usr/bin/python
#
# Copyright 2012 Google Inc. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import httplib2
import sys

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

# For this example, the client id and client secret are command-line arguments.
client_id = sys.argv[1]
client_secret = sys.argv[2]

# The scope URL for read/write access to a user's calendar data
scope = 'https://www.googleapis.com/auth/calendar'
```

```
# Create a flow object. This object holds the client_id, client_secret, and
# scope. It assists with OAuth 2.0 steps to get user authorization and
# credentials.
flow = OAuth2WebServerFlow(client_id, client_secret, scope)

def main():

  # Create a Storage object. This object holds the credentials that your
  # application needs to authorize access to the user's data. The name of the
  # credentials file is provided. If the file does not exist, it is
  # created. This object can only hold credentials for a single user, so
  # as-written, this script can only handle a single user.
  storage = Storage('credentials.dat')

  # The get() function returns the credentials for the Storage object. If no
  # credentials were found, None is returned.
  credentials = storage.get()

  # If no credentials are found or the credentials are invalid due to
  # expiration, new credentials need to be obtained from the authorization
  # server. The oauth2client.tools.run_flow() function attempts to open an
  # authorization server page in your default web browser. The server
  # asks the user to grant your application access to the user's data.
  # If the user grants access, the run_flow() function returns new
credentials.
  # The new credentials are also stored in the supplied Storage object,
  # which updates the credentials.dat file.
  if credentials is None or credentials.invalid:
    credentials = tools.run_flow(flow, storage, tools.argparser.parse_args())

  # Create an httplib2.Http object to handle our HTTP requests, and
authorize it
  # using the credentials.authorize() function.
  http = httplib2.Http()
  http = credentials.authorize(http)

  # The apiclient.discovery.build() function returns an instance of an API
service
  # object can be used to make API calls. The object is constructed with
  # methods specific to the calendar API. The arguments provided are:
  #   name of the API ('calendar')
```

```
  #   version of the API you are using ('v3')
  #   authorized httplib2.Http() object that can be used for API calls
  service = build('calendar', 'v3', http=http)

  try:

    # The Calendar API's events().list method returns paginated results, so
we
    # have to execute the request in a paging loop. First, build the
    # request object. The arguments provided are:
    #   primary calendar for user
    request = service.events().list(calendarId='primary')
    # Loop until all pages have been processed.
    while request != None:
      # Get the next page.
      response = request.execute()
      # Accessing the response like a dict object with an 'items' key
      # returns a list of item objects (events).
      for event in response.get('items', []):
        # The event object is a dict object with a 'summary' key.
        print repr(event.get('summary', 'NO SUMMARY')) + '\n'
      # Get the next request object by passing the previous request object to
      # the list_next method.
      request = service.events().list_next(request, response)

  except AccessTokenRefreshError:
    # The AccessTokenRefreshError exception is raised if the credentials
    # have been revoked by the user or they have expired.
    print ('The credentials have been revoked or expired, please re-run'
           'the application to re-authorize')

if __name__ == '__main__':
  main()
```

Note how the Storage object is used to to retrieve and store credentials. When no credentials are found, the run_flow() function is used to get user acceptance and credentials. Once credentials are obtained, they are used to authorize an Http object. The authorized Http object is then passed to the build() function to create the service. The calendar API request and response

handling shows how to loop through paginated results from a collection. For more information about pagination, see the pagination page in the Developer's Guide.

Run the example

1. Copy the script to an **empty** directory on your computer.
2. Open a terminal and go to the directory.
3. Execute the following command:

   $ python authorized_api_cmd_line_calendar.py *your_client_id your_client_secret*

4. The script will open an authorization server page in your default web browser.
5. Follow instructions to authorize the application's access to your calendar data.
6. Calendar events are sent to stdout by the command.

Subsequent runs of this script will not require the browser because credentials are saved in a file. As mentioned above, these credential files should be kept private.

Authorized API for web application example

For this example, we use authorized API access for a simple web server. It calls the Google Calendar API to list a user's calendar events. Python's built-in BaseHTTPServer is used to create the server. Actual production code would normally use a more sophisticated web server framework, but the simplicity of BaseHTTPServer allows the example to focus on using this library.

Setup for example

1. **Activate the Calendar API**: Read about API activation and activate the Calendar API.
2. **Get your client ID and client secret**: Get a client ID and secret for *web applications*. Use http://localhost as your domain. After creating the client ID, edit the *Redirect URIs* field to contain only http://localhost:8080/.
3. **Create calendar events**: In this example, user calendar events will be read. You can use any Google account you own, including the account

associated with the application's Google APIs Console project. For the target user, create a few calendar events if none exist already.

Code for example

Download or expand the code below. This script is well commented to explain each step.

▸
Click to expand

```
#!/usr/bin/python
#
# Copyright 2012 Google Inc. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import BaseHTTPServer
import Cookie
import httplib2
import StringIO
import urlparse
import sys

from apiclient.discovery import build
from oauth2client.client import AccessTokenRefreshError
from oauth2client.client import OAuth2WebServerFlow
from oauth2client.file import Storage
```

```
class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
  """Child class of BaseHTTPRequestHandler that only handles GET
request."""

  # Create a flow object. This object holds the client_id, client_secret, and
  # scope. It assists with OAuth 2.0 steps to get user authorization and
  # credentials. For this example, the client ID and secret are command-line
  # arguments.
  flow = OAuth2WebServerFlow(sys.argv[1],
                             sys.argv[2],
                             'https://www.googleapis.com/auth/calendar',
                             redirect_uri='http://localhost:8080/')

  def do_GET(self):
    """Handler for GET request."""
    print '\nNEW REQUEST, Path: %s' % (self.path)
    print 'Headers: %s' % self.headers

    # To use this server, you first visit
    # http://localhost:8080/?fake_user=<some_user_name>. You can use
any name you
    # like for the fake_user. It's only used as a key to store credentials,
    # and has no relationship with real user names. In a real system, you
would
    # only use logged-in users for your system.
    if self.path.startswith('/?fake_user='):
      # Initial page entered by user
      self.handle_initial_url()

    # When you redirect to the authorization server below, it redirects back
    # to to http://localhost:8080/?code=<some_code> after the user grants
access
    # permission for your application.
    elif self.path.startswith('/?code='):
      # Page redirected back from auth server
      self.handle_redirected_url()
    # Only the two URL patterns above are accepted by this server.
    else:
      # Either an error from auth server or bad user entered URL.
      self.respond_ignore()
```

```
  def handle_initial_url(self):
    """Handles the initial path."""
    # The fake user name should be in the URL query parameters.
    fake_user = self.get_fake_user_from_url_param()

    # Call a helper function defined below to get the credentials for this user.
    credentials = self.get_credentials(fake_user)

    # If there are no credentials for this fake user or they are invalid,
    # we need to get new credentials.
    if credentials is None or credentials.invalid:
      # Call a helper function defined below to respond to this GET request
      # with a response that redirects the browser to the authorization server.
      self.respond_redirect_to_auth_server(fake_user)
    else:
      try:
        # Call a helper function defined below to get calendar data for this
        # user.
        calendar_output = self.get_calendar_data(credentials)

        # Call a helper function defined below which responds to this
        # GET request with data from the calendar.
        self.respond_calendar_data(calendar_output)
      except AccessTokenRefreshError:
        # This may happen when access tokens expire. Redirect the browser to
        # the authorization server
        self.respond_redirect_to_auth_server(fake_user)

  def handle_redirected_url(self):
    """Handles the redirection back from the authorization server."""
    # The server should have responded with a "code" URL query parameter.
This
    # is needed to acquire credentials.
    code = self.get_code_from_url_param()

    # Before we redirected to the authorization server, we set a cookie to
save
    # the fake user for retrieval when handling the redirection back to this
    # server. This is only needed because we are using this fake user
```

```
    # name as a key to access credentials.
    fake_user = self.get_fake_user_from_cookie()

    #
    # This is an important step.
    #
    # We take the code provided by the authorization server and pass it to
the
    # flow.step2_exchange() function. This function contacts the
authorization
    # server and exchanges the "code" for credentials.
    credentials = RequestHandler.flow.step2_exchange(code)

    # Call a helper function defined below to save these credentials.
    self.save_credentials(fake_user, credentials)

    # Call a helper function defined below to get calendar data for this user.
    calendar_output = self.get_calendar_data(credentials)

    # Call a helper function defined below which responds to this GET
request
    # with data from the calendar.
    self.respond_calendar_data(calendar_output)

  def respond_redirect_to_auth_server(self, fake_user):
    """Respond to the current request by redirecting to the auth server."""
    #
    # This is an important step.
    #
    # We use the flow object to get an authorization server URL that we
should
    # redirect the browser to. We also supply the function with a
redirect_uri.
    # When the auth server has finished requesting access, it redirects
    # back to this address. Here is pseudocode describing what the auth
server
    # does:
    #   if (user has already granted access):
    #     Do not ask the user again.
    #     Redirect back to redirect_uri with an authorization code.
```

```
    #   else:
    #     Ask the user to grant your app access to the scope and service.
    #     if (the user accepts):
    #       Redirect back to redirect_uri with an authorization code.
    #     else:
    #       Redirect back to redirect_uri with an error code.
    uri = RequestHandler.flow.step1_get_authorize_url()

    # Set the necessary headers to respond with the redirect. Also set a
cookie
    # to store our fake_user name. We will need this when the auth server
    # redirects back to this server.
    print 'Redirecting %s to %s' % (fake_user, uri)
    self.send_response(301)
    self.send_header('Cache-Control', 'no-cache')
    self.send_header('Location', uri)
    self.send_header('Set-Cookie', 'fake_user=%s' % fake_user)
    self.end_headers()

  def respond_ignore(self):
    """Responds to the current request that has an unknown path."""
    self.send_response(200)
    self.send_header('Content-type', 'text/plain')
    self.send_header('Cache-Control', 'no-cache')
    self.end_headers()
    self.wfile.write(
      'This path is invalid or user denied access:\n%s\n\n' % self.path)
    self.wfile.write(
      'User entered URL should look like: http://localhost:8080/?
fake_user=johndoe')

  def respond_calendar_data(self, calendar_output):
    """Responds to the current request by writing calendar data to
stream."""
    self.send_response(200)
    self.send_header('Content-type', 'text/plain')
    self.send_header('Cache-Control', 'no-cache')
    self.end_headers()
    self.wfile.write(calendar_output)
```

```
  def get_calendar_data(self, credentials):
    """Given the credentials, returns calendar data."""
    output = StringIO.StringIO()

    # Now that we have credentials, calling the API is very similar to
    # other authorized access examples.

    # Create an httplib2.Http object to handle our HTTP requests, and
authorize
    # it using the credentials.authorize() function.
    http = httplib2.Http()
    http = credentials.authorize(http)

    # The apiclient.discovery.build() function returns an instance of an API
    # service object that can be used to make API calls.
    # The object is constructed with methods specific to the calendar API.
    # The arguments provided are:
    #   name of the API ('calendar')
    #   version of the API you are using ('v3')
    #   authorized httplib2.Http() object that can be used for API calls
    service = build('calendar', 'v3', http=http)

    # The Calendar API's events().list method returns paginated results, so
we
    # have to execute the request in a paging loop. First, build the request
    # object. The arguments provided are:
    #   primary calendar for user
    request = service.events().list(calendarId='primary')
    # Loop until all pages have been processed.
    while request != None:
      # Get the next page.
      response = request.execute()
      # Accessing the response like a dict object with an 'items' key
      # returns a list of item objects (events).
      for event in response.get('items', []):
        # The event object is a dict object with a 'summary' key.
        output.write(repr(event.get('summary', 'NO SUMMARY')) + '\n')
      # Get the next request object by passing the previous request object to
      # the list_next method.
      request = service.events().list_next(request, response)
```

```
    # Return the string of calendar data.
    return output.getvalue()

  def get_credentials(self, fake_user):
    """Using the fake user name as a key, retrieve the credentials."""
    storage = Storage('credentials-%s.dat' % (fake_user))
    return storage.get()

  def save_credentials(self, fake_user, credentials):
    """Using the fake user name as a key, save the credentials."""
    storage = Storage('credentials-%s.dat' % (fake_user))
    storage.put(credentials)

  def get_fake_user_from_url_param(self):
    """Get the fake_user query parameter from the current request."""
    parsed = urlparse.urlparse(self.path)
    fake_user = urlparse.parse_qs(parsed.query)['fake_user'][0]
    print 'Fake user from URL: %s' % fake_user
    return fake_user

  def get_fake_user_from_cookie(self):
    """Get the fake_user from cookies."""
    cookies = Cookie.SimpleCookie()
    cookies.load(self.headers.get('Cookie'))
    fake_user = cookies['fake_user'].value
    print 'Fake user from cookie: %s' % fake_user
    return fake_user

  def get_code_from_url_param(self):
    """Get the code query parameter from the current request."""
    parsed = urlparse.urlparse(self.path)
    code = urlparse.parse_qs(parsed.query)['code'][0]
    print 'Code from URL: %s' % code
    return code

def main():
  try:
    server = BaseHTTPServer.HTTPServer(('', 8080), RequestHandler)
    print 'Starting server. Use Control+C to stop.'
    server.serve_forever()
  except KeyboardInterrupt:
```

```
    print 'Shutting down server.'
    server.socket.close()

if __name__ == '__main__':
  main()
```

Note how the Storage object is used to to retrieve and store credentials. If no credentials are found, the flow.step1_get_authorize_url() function is used to redirect the user to the authorization server. Once the user has granted access, the authorization server redirects back to the local server with a code query parameter. This code is passed to the flow.step2_exchange() function, which returns credentials. From that point forward, this script is very similar to the command-line access example above.

Run the example

1. Copy the script to an **empty** directory on your computer.
2. Open a terminal and go to the directory.
3. Execute the following command to run the local server:

   $ python authorized_api_web_server_calendar.py *your_client_id your_client_secret*

4. Open a web browser and log in to your Google account as the target user.
5. Go to this URL: http://localhost:8080/?fake_user=*target_user_name* replacing *target_user_name* with the user name for the target user.
6. If this is the first time the target user has accessed this local server, the target user is redirected to the authorization server. The authorization server asks the target user to grant the application access to calendar data. Click the button that allows access.
7. The authorization server redirects the browser back to the local server.
8. Calendar events for the target user are listed on the page.
