Google Ads

Getting Started: Credentials & Account Access

Google Ads API Migration Workshops - 2021

In this session, we will be learning how to authorize and generate credentials for the Google Ads API, and debug the credentials when something goes wrong. We'll also introduce a tool for debugging credentials issues called the Google Ads Doctor tool and show you how to use it.

The main steps involved in this session are listed below.

- Part 0 - Prerequisites
- Part 1 - OAuth
- Part 2 - Client Libraries
- Part 3 - Google Ads Doctor
- Part 4 - Fix Missing Scope Error
- Part 5 - Success
- Part 6 - Update Refresh Token

This is meant to be an interactive session in which you can follow along with the demonstration by performing each of the steps below. Please post any questions you have to the Q&A forum, and our team will be standing by to help you out.

Part 0 - Prerequisites

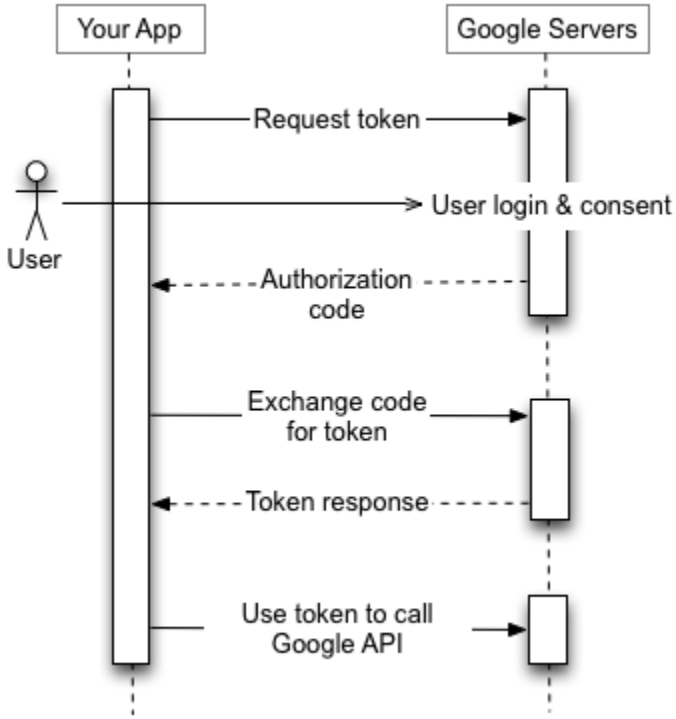
Before getting started you will need the following.

- A Google Ads Test Account
- A Developer Token
- A Google Cloud Platform project with the Google Ads API enabled
- An environment with
    - Your client library of choice installed
    - git

Note that a GCP project can only be linked to a single Developer Token.

Part 1 - OAuth

OAuth is an open standard for authorizing access to APIs and services. Many Google APIs, like the Google Ads API, use OAuth 2 to delegate access to Ads Accounts to third-party services, client libraries, and more without the authorizing party sharing their Google password.



All Google Ads API requests must include authorized OAuth credentials. While there are several types of credentials and flows for granting credentials, this codelab focuses on user Access Tokens and the Desktop flow.

Service accounts use an OAuth flow that avoids the token negotiation steps. They can be used with the Google Ads API, however they will not be covered in this session. The desktop flow we'll demonstrate here is the recommended approach.

Part 2 - Client Libraries

We recommend using the client libraries to interact with the Google Ads API. They provide friendlier interfaces that help you get started faster and have a large set of code examples for performing most common API tasks.

We also expose a REST API which you can use if we do not support a client library for your preferred programming language. This codelab will not be covering the REST API.

Part 2.0: Setup

We will be using Ruby in this codelab. If you choose to use another client library, the setup will be slightly different. From an empty directory we'll create the dependencies file.

2.0.0: Create the dependencies file

```
# Gemfile

source 'https://rubygems.org'

gem 'google-ads-googleads', '~> 14.0'
```

Once we have set the dependencies, they can be installed with a dependency manager, in this case bundler.

2.0.1: Install your dependencies

```
$ bundle install
```

Part 2.1: Script

From the same directory we'll create the `main` script. More functionality will be added later.

2.1.0: Create the main script

```
# main.rb

require 'google/ads/google_ads'

client = Google::Ads::GoogleAds::GoogleAdsClient.new('./google_ads_config.rb')
```

Running the script will demonstrate the error result of not having a configuration file.

2.1.1: Config file missing error

```
$ bundle exec ruby main.rb

No configuration file found at location "./google_ads_config.rb" (ArgumentError)
```

Part 2.2: Configure

Let's add the missing file where we'll store our configuration information by creating `google_ads_config`. This file name and setup will be different depending on what client library you are using. The links below contain client-library specific configuration information:

- Java
- .NET
- Perl
- PHP
- Python
- Ruby

2.2.0: Create a configuration file

```
$ touch google_ads_config.rb
```

Next, let's execute the script, which will produce an error message. The exact error message will vary depending on your client library.

2.2.1: Run script and see error

```
$ bundle exec ruby main.rb

Configuration file did not produce expected type Google::Ads::GoogleAds::Config,
got "NilClass" instead (ArgumentError)
```

Congratulations, you've successfully unsuccessfully configured a client library!

Part 3 - Google Ads Doctor

When using a client library and run into configuration or authentication issues, the Google Ads Doctor (or OAuthDoctor) tool should be your first stop. The OAuthDoctor will help you debug and troubleshoot issues related to configuration and account access.

Part 3.0: Install the OAuthDoctor

To install the OAuthDoctor, clone the git repository and execute the binary built for your environment. Do this within the same directory for easy access.

3.0.0: Clone the OAuthDoctor

```
$ git clone https://github.com/googleads/google-ads-doctor.git
```

Part 3.1: Test without Configuration

Execute the binary with the `--help` flag to make sure OAuthDoctor is installed correctly and show the possible options for the tool.

3.1.0: Run the OAuthDoctor with the –help option

```
$ ./google-ads-doctor/oauthdoctor/bin/darwin/amd64/oauthdoctor --help
```

With OAuthDoctor installed and running, execute the OAuthDoctor to test the environment to identify issues. Note, your command line parameters may be different depending on your client library.

3.1.1: Run the OAuthDoctor with language-specific parameters

```
$ ./google-ads-doctor/oauthdoctor/bin/darwin/amd64/oauthdoctor -language ruby
-oauthtype installed_app -configpath ./google_ads_config.rb
```

OAuthDoctor prints out all the missing configuration values and a prompt to enter an Account ID. Abort the script with `control + c` or the equivalent for your environment.

Part 3.2: Fix your Configuration File

Update your configuration file to include the client ID, client secret, and developer token with values from your GCP Project and Google Ads Manager account. The refresh token will be added in a later step. Each client library has a sample file showing all the configurable options.

- Java
- .NET
- Perl
- PHP
- Python
- Ruby

3.2.0: Update your configuration file

```
# google_ads_config.rb

Google::Ads::GoogleAds::Config.new do |c|
  # Treat deprecation warnings as errors will cause all deprecation warnings
  # to raise instead of calling `Warning#warn`. This lets you run your tests
  # against google-ads-googleads to make sure that you are not calling any
  # deprecated code
  c.treat_deprecation_warnings_as_errors = false

  # Warn on all deprecations. Setting this to `true` will cause the library to
  # warn every time a piece of deprecated code is called. The `false` (default)
  # behaviour is to only issue a warning once for each call site in your code.
  c.warn_on_all_deprecations = false

  # The developer token is required to authenticate that you are allowed to
  # make API calls.
  c.developer_token = 'INSERT_DEVELOPER_TOKEN_HERE'

  # Authentication tells the API that you are allowed to make changes to the
  # specific account you're trying to access.
  # The default method of authentication is to use a refresh token, client id,
  # and client secret to generate an access token.
  c.client_id = 'INSERT_CLIENT_ID_HERE'
  c.client_secret = 'INSERT_CLIENT_SECRET_HERE'
end
```

Running the OAuthDoctor command again will show several values are now configured. Continue to the next step by entering your Google Ads Manager Account ID in the format of 123-456-7890. This will log Google Identity authorization dialog URL.

3.2.1: Sample URL output

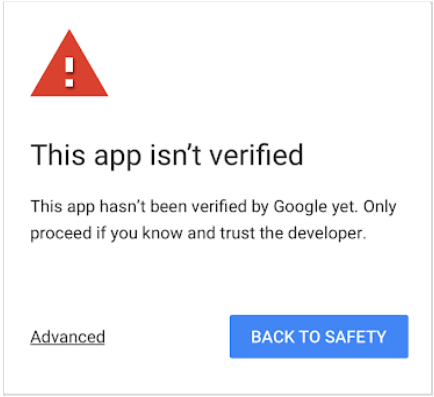
```
https://accounts.google.com/o/oauth2/auth?access_type=offline&client_id=123456789-b
5tsd9bh5.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoo
b&response_type=code&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fadwords&state=
state
```

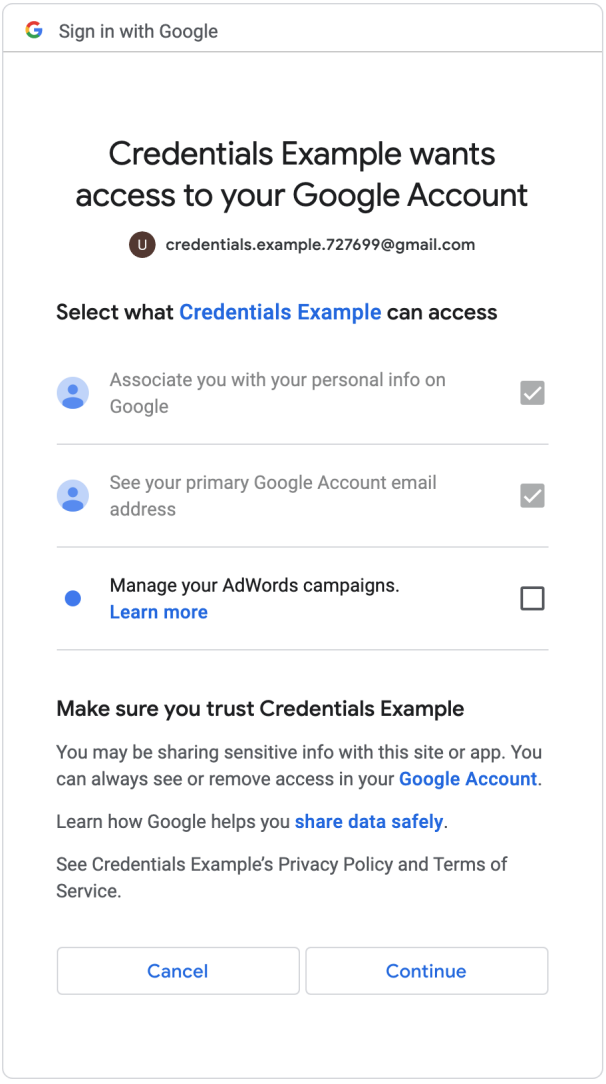
Part 3.3: Introducing a Scope Error

Before authorizing access, make one small change to add the `email` scope (`email` is shorthand for the `https://www.googleapis.com/auth/userinfo.email` Google Identity scope) followed by an encoded space. Update the `scope` value of the URL so that instead of `scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fadwords` it's `scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fadwords`.

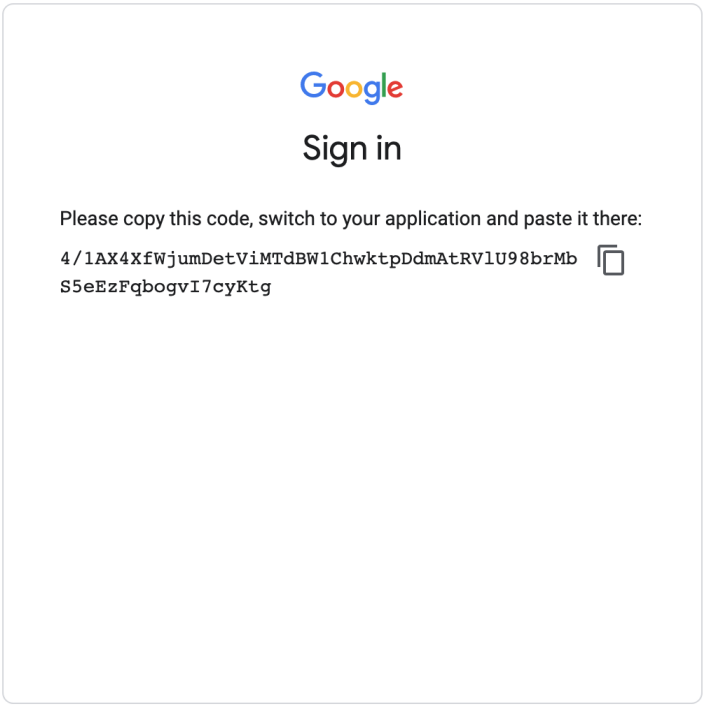
Open that URL in a browser where you will select the Google account you wish to authorize. Grant access to the Google Account **leaving "manage your AdWords campaigns" unchecked** for now.

If the GCP Project OAuth app is unverified you may have to click through a warning screen.





Account access has now been authorized for the email scope. Copy the desktop flow code and paste it into the OAuthDoctor prompt.

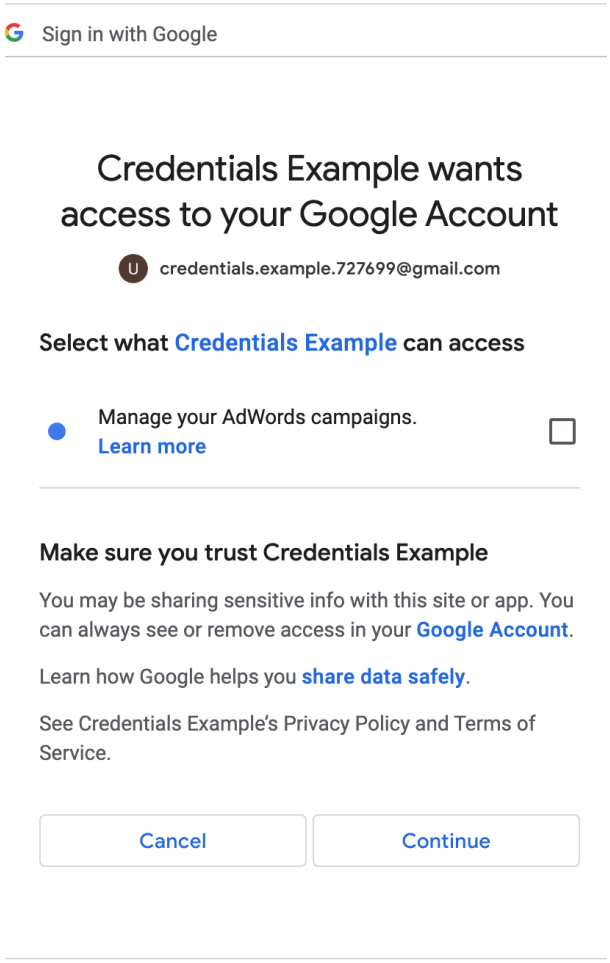


After continuing with OAuthDoctor you'll get an `ERROR: OAuth test failed` message. This is caused by missing authorization of the `https://www.googleapis.com/auth/adwords` scope.

Part 4 - Fix Missing Scope Error

Part 4.0: Enable missing scope

Go through the Desktop OAuth flow again, this time enabling “Manage your AdWords campaigns.”.



After enabling the Google Ads (Displayed as AdWords) and authorizing access, OAuthDoctor will display a success message.

4.0.0: Sample prompt from OAuthDoctor

```
2021/08/19 06:27:56 SUCCESS: OAuth test passed with given config file settings.
2021/08/19 06:27:56 Would you like to replace your refresh token in the client
library config file with the new one generated?
```

Confirm Y at the prompt to write the refresh token to the configuration file. A refresh token is a long lived OAuth credential that allows generating short lived Access Tokens that are used to authenticate each request.

Part 5 - Success

Configuration is set up, access has been granted, and a refresh token with the correct scope has been issued. Let's see a successful API call.

Part 5.0: Account Information

Update `main` script to include the primary function code from the get campaigns examples linked below. Update the example to select customers instead and include the Google Ads Customer ID without any dashes (e.g. `1234567890`).

- Java
- .NET
- Perl
- PHP
- Python
- Ruby

5.0.0: Get campaigns code

```
# main.rb

require 'google/ads/google_ads'

client = Google::Ads::GoogleAds::GoogleAdsClient.new('./google_ads_config.rb')

customer_id = 'INSERT_CLIENT_ID'

responses = client.service.google_ads.search_stream(
  customer_id: customer_id,
  query: 'SELECT campaign.id, campaign.name FROM campaign ORDER BY campaign.id',
)

responses.each do |response|
  response.results.each do |row|
    puts "Campaign with ID #{row.campaign.id} and name '#{row.campaign.name}' was
found."
  end
end
```

5.0.1: Get customer info code

```
# main.rb

require 'google/ads/google_ads'

client = Google::Ads::GoogleAds::GoogleAdsClient.new('./google_ads_config.rb')

customer_id = 'INSERT_CLIENT_ID'

responses = client.service.google_ads.search_stream(
  customer_id: customer_id,
  query: 'SELECT customer.id, customer.descriptive_name FROM customer',
)

responses.each do |response|
  response.results.each do |row|
    puts "Customer with ID #{row.customer.id} and name
'#{row.customer.descriptive_name}' was found."
  end
end
```


Run the example to see successful output.

5.0.2: See a successful run

```
$ bundle exec ruby main.rb
```


Part 6 - Update Refresh Token

One of the common pitfalls when working with OAuth credentials is missing a character when copy/pasting a client ID, a secret, etc. Let's take a look at what happens.

Part 6.0: Introduce Credentials Error

Back in the `google_ads_config.rb` config file, delete the last character of the refresh token.

6.0.0: See Authorization Error

```
$ bundle exec ruby main.rb
```

There will be a large error message that includes "Authorization failed".

A similar error can also be caused by [revoking the application's access] to the Google account.

Part 6.1: Update Refresh Token

Run the OAuthDoctor flow again and perform the following steps

- Enter an Account ID
- Select Google account in browser
- Enable managing AdWords campaigns
- Copy OAuth code and paste it into OAuthDoctor
- Confirm overwriting client library config file

6.1.0: Run the OAuthDoctor Again

```
$ ./google-ads-doctor/oauthdoctor/bin/darwin/amd64/oauthdoctor -language ruby
-oauthtype installed_app -configpath ./google_ads_config.rb
```

Great job! Run the example to see successful output.

6.1.1: See a successful run

```
$ bundle exec ruby main.rb
```

Resources

- Client library docs
- Google Ads API Authentication Docs
- Google Ads Doctor
- Using OAuth 2.0 to Access Google APIs