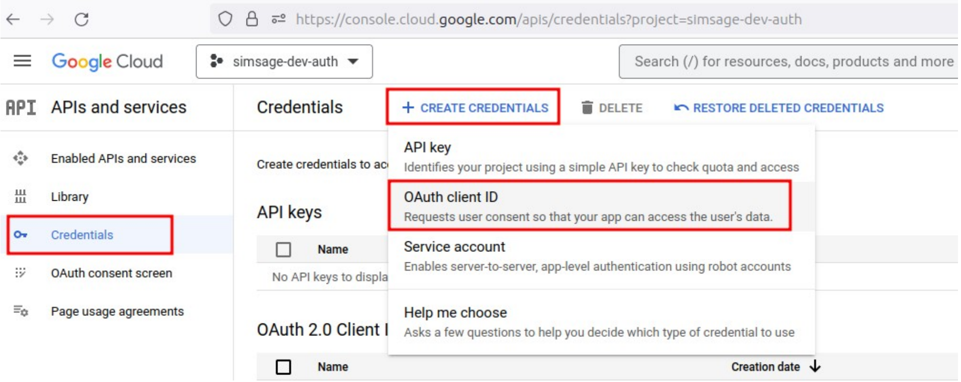
Open Authentication Set Up Guide

The SimSage web crawler can be set up to use OIDC (Open ID Consortium), and to use Google Drive. At the time of writing Google Drive needs its own token, as Google doesn't accept JWT tokens for Google Drive. This document is about setting up OIDC for SimSage to talk to your remote Google systems that DO accept OIDC authentication.

Google Set Up

1. Navigate to https://console.cloud.google.com/apis/credentials



https://console.cloud.google.com/apis/credentials?project=simsage-dev-auth

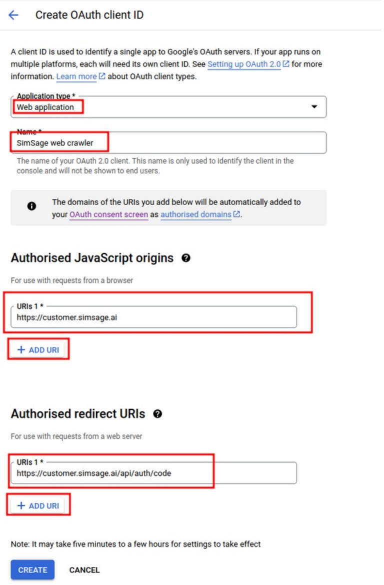
API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so that your app can access the user's data.

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

2. Select "Credentials" in the left-hand side menu
3. Click "CREATE CREDENTIALS"
4. select "OAuth client ID".



Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See Setting up OAuth 2.0 for more information. Learn more about OAuth client types.

Application type *: Web application

Name *: SimSage web crawler

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorised domains.

Authorised JavaScript origins

For use with requests from a browser

URIs 1 *: https://customer.simsage.ai

Authorised redirect URIs

For use with requests from a web server

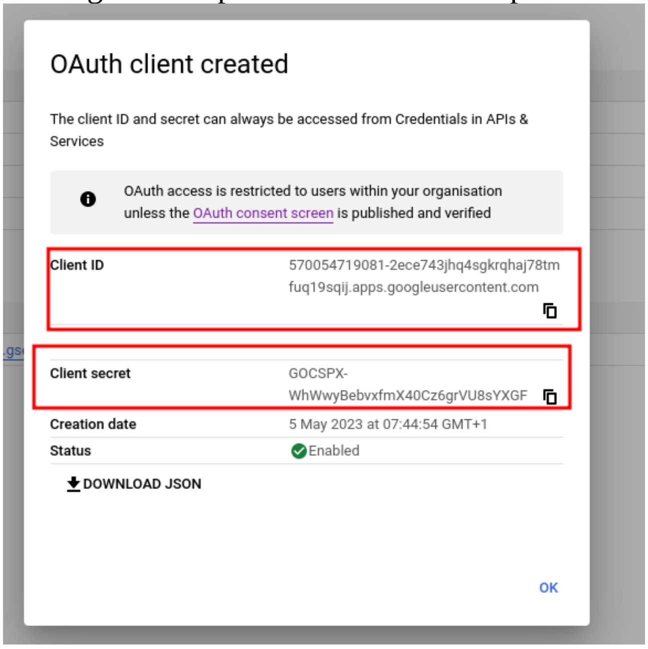
URIs 1 *: https://customer.simsage.ai/api/auth/code

Note: It may take five minutes to a few hours for settings to take effect

5. Populate the form with the following values.
   Application type: Web application
   Name: We suggest you use whatever helps you remember what this token is for. Our example shows "SimSage web crawler".

Authorised JavaScript origins: This should be the address of your SimSage instance e.g. https://customer.simsage.ai. (This address is the server's FQDN and will be different for your instance)
Authorised redirect URIs: This must be the same server FQDN you used for the Authorised JavaScript origin with /api/auth/code appended. e.g. https://customer.simsage.ai/api/auth/code

6. Click create to complete the process.
7. Copy both the Client ID and Client Secret from the pop up. These values are used by SimSage to set up the OIDC relationship with Google.



OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services

OAuth access is restricted to users within your organisation unless the OAuth consent screen is published and verified

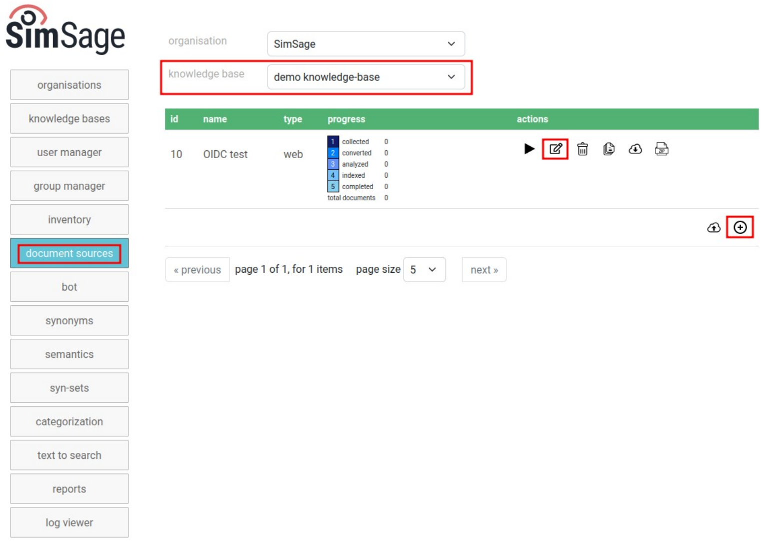
Client ID	570054719081-2ece743jhq4sgkrqhaj78tmfuq19sqij.apps.googleusercontent.com
Client secret	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
Creation date	5 May 2023 at 07:44:54 GMT+1
Status	✓ Enabled

8. Once copied click ok to close the window.

SimSage Set Up

There is one restriction around the OIDC relationship imposed by Google. The relationship can only be established once. You can keep using the Client ID and Client secret for other sources / crawlers inside SimSage, however, you only need to set up this relationship once.
(NB: The person that establishes the relationship becomes the user that is associated with this token. Make sure that you as a user setting this up are the user that has access to the intended systems.)

1. Navigate to the "document sources" section in SimSage.



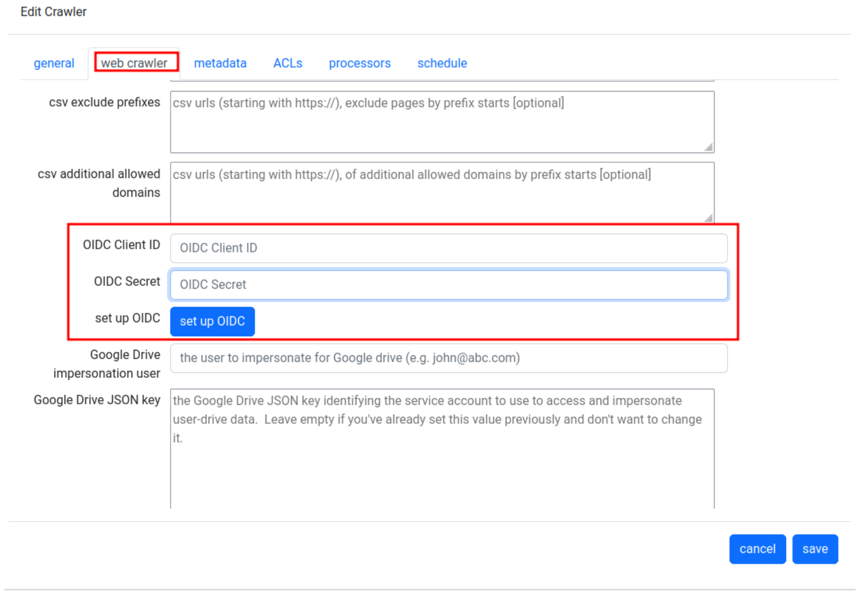
organisation: SimSage

knowledge base: demo knowledge-base

id	name	type	progress	actions
10	OIDC test	web	1 collected 0 2 converted 0 3 analyzed 0 4 indexed 0 5 completed 0 total documents 0	

« previous   page 1 of 1, for 1 items   page size 5   next »

2. Select the right knowledge-base.
3. Either edit an existing source, or create new source with web source type.
4. Navigate to the web crawler tab.
5. Copy both the Client ID and Client secret into the corresponding fields.
6. Click “Set up OIDC” if this is the first time you have used these keys.



Edit Crawler

csv exclude prefixes: csv urls (starting with https://), exclude pages by prefix starts [optional]

csv additional allowed domains: csv urls (starting with https://), of additional allowed domains by prefix starts [optional]

OIDC Client ID: OIDC Client ID

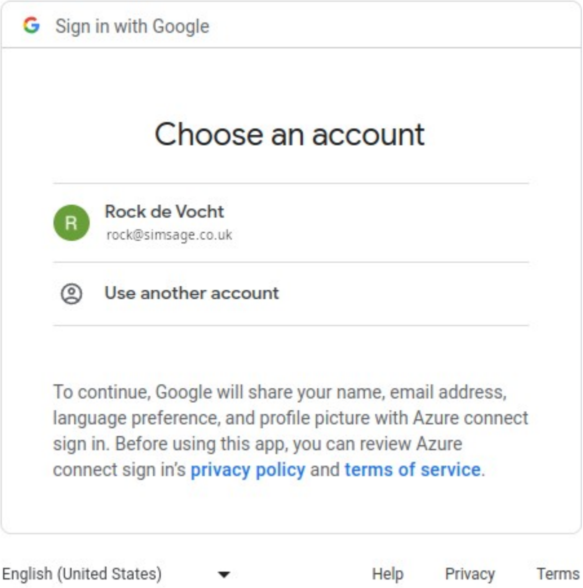
OIDC Secret: OIDC Secret

set up OIDC: set up OIDC

Google Drive impersonation user: the user to impersonate for Google drive (e.g. john@abc.com)

Google Drive JSON key: the Google Drive JSON key identifying the service account to use to access and impersonate user-drive data. Leave empty if you've already set this value previously and don't want to change it.

7. This will open up a new tab in your browser (NB: ensure you have enabled popups from this site)



Sign in with Google

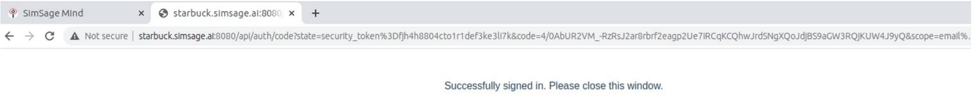
Choose an account

Rock de Vocht
rock@simsage.co.uk

Use another account

To continue, Google will share your name, email address, language preference, and profile picture with Azure connect sign in. Before using this app, you can review Azure connect sign in's privacy policy and terms of service.

8. Log in with the account that you want to be associated with all future requests made by the web crawler.
9. Once logged in, SimSage should reply with a Success message.



starbuck.simsage.ai:8080/api/auth/code?state=security_token%3Dfjh4h8804cto1r1def3ke3li7k&code=4/0AbUR2VM_-RzRsJ2ar8rbrf2eagp2Ue7IRCqKCQhwJrdSNgXQoJdjBS9aGW3RQJKUW4J9yQ&scope=email%.

Successfully signed in. Please close this window.

What is a JWT

This section is purely informative and isn't required for the set up process.

A JWT is an entity that consists of three parts. The first part is the HEADER. The header declares that it is a JWT (typ), has the ID of the key used to encrypt the token (kid), and expresses the encryption and hash algorithm used (alg) to one-way hash and encrypt.

Here is an example JWT header

```
{
  "alg": "RS256",
  "kid": "c9afda3682ebf19eb3055c1d4bd39b751fbf8115",
  "typ": "JWT"
}
```

The next part of the JWT is the PAYLOAD. This expresses who we claim to be (email) as well as the issuer (iss), the issue-at time (iat) and the expiry time (exp) of the token. Here is an example payload.

```
{
  "iss": "https://accounts.google.com",
  "azp": "570054719081-de7kbeom1b5o60aokd0rcr6531nn4nlc.apps.googleusercontent.com",
  "aud": "570054719081-de7kbeom1b5o60aokd0rcr6531nn4nlc.apps.googleusercontent.com",
  "sub": "100971117556722785565",
  "hd": "simsage.co.uk",
  "email": "rock@simsage.co.uk",
  "email_verified": true,
  "at_hash": "nCKy7JdKhXB6E558dR3UMg",
  "iat": 1583267769,
  "exp": 1583271369
}
```

The last part of the JWT is the encrypted hash (the signature). Our example shows the algorithm as RS256, which m,eans that in our example the last part can be expressed as:

RSASHA256(base64UrlEncode(HEADER) + “.” + base64UrlEncode(PAYLOAD))

Upon receiving a JWT, it is the implementer that reads the JWT and uses the key id (kid) to read the public key from the issuer (Google in our example) and re-applies the encryption using the formula above to check if the encrypted header and payload match the signature. If it does, we have just proved that the token came from the issuer, as only the issuer can decrypt the payload (the workings of RSA asymmetric encryption are beyond the scope of this document).