Emissions Product Documentation and API Guide

Summary

This document describes how to access data available in the Emissions data product available through the CSIRO Data Shop. For details about the methodology used to generate the data please refer to *Aryai, V., Goldsworthy, M.* "Controlling electricity storage to balance electricity costs and greenhouse gas emissions in buildings", **Energy Inform 5, 11 (2022).** https://doi.org/10.1186/s42162-022-00216-5.

Prerequisites

- Authorization and license agreement. You can access a subscription through our CSIRO Data Shop. More details on Authorisation processes/mechanisms are in the section "Authentication for CSIRO Data Shop Products APIs" below.
- Some programming experience to consume REST API data services. An example is provided using the python programming language, but python is not required at all.

Data Structure and Variables

The Emissions data is organized as a collection of time series estimates of the carbon emissions intensity of grid-consumed power for the five regions of the National Electricity Market (NEM) updated at five-minute intervals with the most up-to-date information available. Estimates are calculated using an energy balance model combined with publicly available generator SCADA data, interstate power flows/losses and state-level regional demand data sourced from the market operator. Generator carbon emissions intensity factors include Scope 1 and Scope 3 emissions.

Multiple Stream Requests

The following structure shows a single data point for the `nsw` and `qld` streams for illustration purposes, but it is worth noting that all data is JSON encoded with the following structure:


```
{
  "_links":{
    "self":{
      "href":"https://senaps.io/api/sensor/v2/observations?limit=99999999&start=2023-05-01T00:00:00.000Z&end=2023-05-10T00:00:00.000Z&streamid=csiro.energy.dch.agshop.regional_global_emissions.nsw,csiro.energy.dch.agshop.regional_global_emissions.qld"
    }
  },
  "_embedded":{
    "stream":[
      {
```

```
        "_links":{
          "self":{
            "href":"https://senaps.io/api/sensor/v2/streams/csiro.energy.dch.
agshop.regional_global_emissions.nsw",
            "id":"csiro.energy.dch.agshop.regional_global_emissions.nsw"
          }
        }
      },
      {
        "_links":{
          "self":{

"href":"https://senaps.io/api/sensor/v2/streams/csiro.energy.dch.agshop.regio
nal_global_emissions.qld",
            "id":"csiro.energy.dch.agshop.regional_global_emissions.qld"
          }
        }
      }
    ]
  },
  "streamCount":2,
  "results":[
    {
      "2023-05-01T00:00:00.000Z":{
        "csiro.energy.dch.agshop.regional_global_emissions.nsw":{
          "v":728.2350489926174
        },
        "csiro.energy.dch.agshop.regional_global_emissions.qld":{
          "v":545.8727223676001
        }
      }
    },
    ...
  ],
  "count":2593
}
```


- The actual data for consumption appears in the `"results"` collection.
- The `"streamCount" and "count" metadata fields refer to the number of columns and rows (resp.) in the response data.`
- All timestamp data is provided as RFC339, UTC formatted string data with the value of each timestamp defining the key for the data object.
- The value of the data object is another collection of key-value pairs, with the key denoting the stream id under consideration and the value consisting of another key-value pair.
- The innermost key-value pair has the special key `"v"` which denotes the actual (floating point) value of the data point.
- Valid `stream_id` values for this dataset are:
    - `csiro.energy.dch.agshop.regional_global_emissions.nsw`
    - `csiro.energy.dch.agshop.regional_global_emissions.qld`
    - `csiro.energy.dch.agshop.regional_global_emissions.sa`

    - `csiro.energy.dch.agshop.regional_global_emissions.vic`
    - `csiro.energy.dch.agshop.regional_global_emissions.tas`
- Units of emissions data in this data is **gCO_2/kWh**.

Single Stream Requests

The following structure shows a single data point for the `nsw` stream for illustration purposes, but it is worth noting that all data is JSON encoded with the following structure:

```
{
  "_links":{
    "self":{
      "href":"https://senaps.io/api/sensor/v2/observations"
    }
  },
  "_embedded":{
    "stream":{
      "_links":{
        "self":{
          "href":"https://senaps.io/api/sensor/v2/streams/csiro.energy.dch.ag
shop.regional_global_emissions.nsw",
          "id":"csiro.energy.dch.agshop.regional_global_emissions.nsw"
        }
      }
    }
  },
  "results":[
    {
      "t": "2023-05-01T00:00:00.000Z",
      "v":{
          "v":728.2350489926174
      },
    ...
  ],
  "count":2593,
  "streamCount": 1
}
```

- The actual data for consumption appears in the `"results"` collection.
- The `"streamCount" and "count" metadata fields refer to the number of columns and rows (resp.) in the response data.`
- All timestamp data is provided as RFC339, UTC formatted string data with the value of each timestamp corresponding to the `"t"` key.
- The value of the data object is given by the nested `"v"` field, and the `stream_id` value no longer appears in the `"results"` field.
- Valid `stream_id` values for this dataset are:
    - `csiro.energy.dch.agshop.regional_global_emissions.nsw`
    - `csiro.energy.dch.agshop.regional_global_emissions.qld`
    - `csiro.energy.dch.agshop.regional_global_emissions.sa`
    - `csiro.energy.dch.agshop.regional_global_emissions.vic`
    - `csiro.energy.dch.agshop.regional_global_emissions.tas`
- Units of emissions data in this data is **gCO_2/kWh**.

Sample Use (python)

Upon purchasing access to the data, you will be provided with access credentials.

Accessing Your Credentials

After obtaining access to data through the Data Shop, you will receive your access credentials, which are crucial for accessing the data. To find these credentials, follow the steps below:

1. Sign into your **Data Shop account**.

2. Navigate to the **My Account** tab.

3. Click on the **Orders** tab.

4. Under **Recurring payments**, select **View order** (make sure to note your Order number - XXXX).

5. On the Order information page, check for the **Note(s)** tab. Here, you will find your **client_id** and **client_secret**.

```
client_id: <UUID>
client_secret: <string>
```

Your newly acquired CSIRO Data Shop credentials will permit you access to the data itself which is in the Senaps cloud platform. The following sample code shows how to use the credentials to make a GET request to the data, as well as parse the data and write the (parsed) response to disk in parquet format for use downstream.
Further details on how to authenticate with Senaps using your credentials can be found in the "Authentication for CSIRO Data Shop Products APIs" section below.

Example Code

The following example can be used to make a request using the above credentials with some time boundaries, with the response data written straight to disk.

Whilst this example has been constructed in python, any language can be employed by following a similar pattern. It is also worth noting that the polars library used is actually a rust library, so the above workflow can be reconstructed in a straightforward manner in rust or any of the wrappers that are provided, including python, NodeJS, and R.

Note the section "Authentication for CSIRO Data Shop Products APIs" below also provides alternate examples of authentication for accessing CSIRO Data Shop products.

Prerequisites

- Ensure you have Python 3.10 or higher installed on your system to avoid errors related to language features such as the **match** functionality.
- Install necessary Python packages (polars and requests_oauth2client) if they are not already installed:

```
pip install polars requests_oauth2client
```

Steps to Use the Code

1. **Set Up Authentication:**
   Replace **<YOUR CLIENT ID>** and **<YOUR CLIENT SECRET>** in the following code with your credentials (see Accessing Your Credentials).

2. **Configure Parameters:**
   Adjust the **regions**, **start**, **end**, and **write_path** parameters in the code to match your data retrieval needs:
   - **regions**: List of region codes for which you want emissions data (e.g., **["nsw", "qld" , "sa","tas","vic"]**).
   - **start**: Start date and time for the data retrieval in ISO 8601 format (e.g., **"2023-05-01T00:00:00.000Z"**).
   - **end**: Enddate and time for the data retrieval in ISO 8601 format (e.g., **"2023-05-10T00:00:00.000Z"**).
   - **write_path**: Path to save the output data in Parquet format (e.g., **Path("C:\demo_response.parquet")**).

3. **Run the Code**.

```
import json
import polars as pl
import requests
import tempfile
from pathlib import Path
from requests_oauth2client import OAuth2Client, OAuth2ClientCredentialsAuth
from typing import List
CLIENT_ID = r"<YOUR CLIENT SECRET>"
CLIENT_SECRET = r"<YOUR CLIENT SECRET>"
class MyEmissionsData(requests.Session):
    _auth_url = "https://login.microsoftonline.com/a815c246-a01f-4d10-bc3e-
eeb6a48ef48a/oauth2/v2.0/token"
    _senaps_url = "https://senaps.eratos.com/api/sensor/v2/observations"
    def __init__(
        self,
        client_id: str = CLIENT_ID,
        client_secret: str = CLIENT_SECRET,
    ) -> None:
        super().__init__()
        oauth2client = OAuth2Client(
            self._auth_url,
            (client_id, client_secret),
```

```
    )
    self.auth = OAuth2ClientCredentialsAuth(
        oauth2client,
        scope=f"{client_id}/.default",
    )
    self.headers = {
        "accept": "*/*",
        "content-type": "application/json",
    }
def download_and_parse_data(
    self,
    *,
    write_path: Path,
    regions: List[str],
    start: str,
    end: str,
    limit: int = 99_999_999,
) -> None:
    match len(regions):
        case 0:
            raise ValueError("`regions` list cannot be empty")
        case 1:
            parser = self._parse_single_stream
        case _:
            parser = self._parse_multiple_streams
    streamid = ",".join(
        (
            f"csiro.energy.dch.agshop.regional_global_emissions.{region}"
            for region in regions
        )
    )
    # we stream the response directly to disk to go easy on memory
    with tempfile.TemporaryDirectory() as tmpdir:
        fname = Path(tmpdir) / "response.json"
        with self.get(
            url=self._senaps_url,
            params=dict(
                streamid=streamid,
                start=start,
                end=end,
                limit=limit,
            ),
        ) as response:
            response.raise_for_status()
            with open(fname, "wb") as fp:
                for chunk in response.iter_content(chunk_size=1024):
                    fp.write(chunk)
        # parse the JSON to parquet data
        write_path.parent.mkdir(parents=True, exist_ok=True)
        with open(fname, "r") as fp:
            data = json.load(fp)
            parser(data, write_path)
@staticmethod
def _parse_single_stream(data, write_path) -> None:
    col_name = (
        data.get("_embedded")
        .get("stream")
```

```
        .get("_links")
        .get("self")
        .get("id")
    )
    (
        pl.LazyFrame(
            [
                {
                    "timestamp": elem.get("t"),
                    col_name: elem.get("v").get("v"),
                }
                for elem in data.get("results")
            ]
        )
        .with_columns(
            pl.col("timestamp")
            .str.strptime(
                dtype=pl.Datetime,
                format="%Y-%m-%dT%H:%M:%S%.fZ",
                strict=True,
                exact=True,
            )
            .cast(
                pl.Datetime(
                    time_unit="ms",
                    time_zone="UTC",
                )
            )
        )
        .sort(by="timestamp")
        .sink_parquet(write_path)
    )
@staticmethod
def _parse_multiple_streams(data, write_path) -> None:
    (
        pl.LazyFrame(
            [
                {
                    "timestamp": key,
                    "struct": {
                        obs_key: obs_val.get("v")
                        for obs_key, obs_val in values.items()
                    },
                }
                for elem in data.get("results")
                for key, values in elem.items()
            ]
        )
        .unnest("struct")
        .with_columns(
            pl.col("timestamp")
            .str.strptime(
                dtype=pl.Datetime,
                format="%Y-%m-%dT%H:%M:%S%.fZ",
                strict=True,
                exact=True,
            )
```

```
                .cast(
                    pl.Datetime(
                        time_unit="ms",
                        time_zone="UTC",
                    )
                )
            )
            .sort(by="timestamp")
            .sink_parquet(write_path)
        )
if __name__ == "__main__":
    e = MyEmissionsData()
    e.download_and_parse_data(
        regions=["nsw", "qld"],
        start="2023-05-01T00:00:00.000Z",
        end="2023-05-10T00:00:00.000Z",
        write_path=Path("./demo_response.parquet"),
    )
```

Authentication for CSIRO Data Shop Products APIs

Introduction

This documentation is provided as a reference where the use of an open source OAuth2 client library for authentication is not available.

DataShop products accessible via API require a Json Web Token (JWT) as a bearer access token to authenticate every API request. You need to get a bearer token and then use it in the Authorization header of each API request to use the API to access the product data successfully. It is used by the API endpoint to confirm you have access to the product before providing product data. Most programming languages will have libraries with support for OAuth2 which can automate this process, but it is explained in detail on this page for reference and to help debug and test product API calls using interactive API-docs.

Retrieving an access token

To obtain an access token, a request needs to be made to the CSIRO identity provider's token endpoint following the Client Credentials flow, which is part of the commonly used OAuth 2.0 specification. The client credentials flow accepts your *client_id* and *client_secret* and provides you an Access Token. Your *client_id* and *client_secret* can be found in the order details page after purchasing a product. You can find your order history page via your account on the shop website or the 'order details' link provided in the order confirmation email.

Access Token Request

```
POST https://login.microsoftonline.com/a815c246-a01f-4d10-bc3e-
eeb6a48ef48a/oauth2/v2.0/token
```

Parameters:

Name	In	Type	Required	Description
Content-Type	Header	String	Yes	Set to "application/x-www-form-urlencoded"
grant_type	Body	String	Yes	Set to "client_credentials"
client_id	Body	String	Yes	Set to the client_id that you have been supplied. (Can also be retrieved from the CSIRO Data Shop order history)
client_secret	Body	String	Yes	Set to the client_secret that you have been supplied. (Can also be retrieved from the CSIRO Data Shop order history)
scope	Body	String	Yes	Must be set to the client_id + "/.default" For example: 12345678-1234-1234-1234-1234567890AB/.default

Note: the client_id value needs to be inserted in two different places!

Sample request:

```
POST https://login.microsoftonline.com/a815c246-a01f-4d10-bc3e-
eeb6a48ef48a/oauth2/v2.0/token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_id=12345678-1234-1234-1234-1234567890AB
&client_secret=C1ient$ecr3t#
&scope=12345678-1234-1234-1234-1234567890AB/.default
```

Access Token Responses

Access Token Response Status Codes

Status	Meaning	Description

200	OK	The request was valid and an access token has been returned:
		
		`{`
		`"token_type": "Bearer",`
		`"expires_in": 3599,`
		`"ext_expires_in": 3599,`
		`"access_token": "eyJ0eXAiOiJKV1QiLCJub2…"`
		`}`

400	Bad Request	The request was invalid, such as a missing parameter.
401	Unauthorized	Invalid client credentials were supplied in the request.

Access Token Response Properties

A successful response (200 OK) will return the access_token as well as additional details that describe the token usage.

Property	Description
token_type	Outlines that the token is a bearer token (i.e. give access to the bearer of this token) and should be passed to the API through the *Authorization* header using the *Bearer* scheme.
expires_in	The amount of seconds until the access_token expires. **Note:** Once the access_token has expired it can no longer be used to call the API. When this occurs a new access_token must be requested from the identity provider.
ext_expires_in	This indicates the extended lifetime of the token. How long the access token is valid (in seconds) if the server isn't responding.
access_token	The value of the access_token. This is the value to be passed to the API in the *Authorization* header using the *Bearer* scheme.

Sample response:

```
200 OK

{
 "token_type": "Bearer",
 "expires_in": 3599,
 "ext_expires_in": 3599,
 "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJ...NOiYYE910ABO_A"
}
```

Using the Access Token

When a 200 OK status is returned from the token endpoint, the access_token value from the response body can be extracted. This value is then set as the bearer token when calling the relevant CSIRO Data Shop Product API:

```
GET /product_api_call/
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJ...NOiYYE910ABO_A
```

Examples

Postman Collection

This Postman Collection provides a pre-configured 'get token' request where only the the client_id, client_secret and scope parameters are required to test the request.

```
{
  "info": {
    "_postman_id": "564e9091-75a9-4f97-a3ec-44c72486e405",
    "name": "AgData Shop Authentication",
    "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
  "item": [{
      "name": "Get Token",
      "event": [{
          "listen": "prerequest",
          "script": {
            "exec": [""],
            "type": "text/javascript"
          }
        }],
      "request": {
        "method": "POST",
        "header": [],
        "body": {
          "mode": "formdata",
          "formdata": [{
              "key": "client_id",
```

```
              "value": "<ENTER CLIENT ID>",
              "description": "Set to the client_id that you have been
supplied. (Can also be retrieved from the AgData shop order history)",
              "type": "text"
            }, {
              "key": "client_secret",
              "value": "<ENTER CLIENT SECRET>",
              "description": "Set to the client_secret that you have been
supplied. (Can also be retrieved from the AgData shop order history)",
              "type": "text"
            }, {
              "key": "scope",
              "value": "<ENTER CLIENT ID>/.default",
              "description": "Must be set to the client_id + \"./default\"
\n\nFor example: \n\n12345678-1234-1234-1234-1234567890AB/.default ",
              "type": "text"
            }, {
              "key": "grant_type",
              "value": "client_credentials",
              "description": "Must be 'client_credentials'",
              "type": "text"
            }]
        },
        "url": {
          "raw": "https://login.microsoftonline.com/a815c246-a01f-4d10-bc3e-
eeb6a48ef48a/oauth2/v2.0/token ",
          "protocol": "https",
          "host": ["login", "microsoftonline", "com"],
          "path": ["a815c246-a01f-4d10-bc3e-eeb6a48ef48a", "oauth2", "v2.0",
"token "]
        }
      },
      "response": []
    }]
}
```

cURL example

```
curl --request POST 'https://login.microsoftonline.com/a815c246-a01f-4d10-
bc3e-eeb6a48ef48a/oauth2/v2.0/token'  --form 'client_id="<ENTER CLIENT ID
HERE>"' --form 'client_secret="<ENTER CLIENT SECRET HERE>"' --form
'scope="<ENTER CLIENT ID HERE>/.default"' --form
'grant_type="client_credentials"'
```

Python example (includes call to Senaps using the acquired token)

Many of the CSIRO Data Shop products provide Python examples using the popular requests library. The follow code snippet demonstrates how to use the requests-oauth2client library to perform CSIRO Data Shop API requests.

The requests-oauth2client library can be installed using pip as follows:

```
pip install requests-oauth2client
```

Get Token example:

```
from requests_oauth2client import *
import requests
CLIENT_ID = '<ENTER CLIENT ID HERE>'
CLIENT_SECRET = '<ENTER CLIENT SECRET HERE>'
oauth2client = OAuth2Client('https://login.microsoftonline.com/a815c246-a01f-
4d10-bc3e-eeb6a48ef48a/oauth2/v2.0/token', (CLIENT_ID, CLIENT_SECRET))
session = requests.Session()
session.auth = OAuth2ClientCredentialsAuth(oauth2client,
scope=f'{CLIENT_ID}/.default')
# The session object can be used to make AgData Shop product API calls.
#response = session.get('<product api request>')
#e.g. if the product is Eratos Senaps based:
response = session.get('https://senaps.eratos.com/api/sensor/v2/')
print(f"Successful connection to Eratos Senaps for user
{response.json()['_embedded']['user'][0]['id']}")
#This won't be needed in your code - it displays the token for API Docs use.
token =
session.auth.client.client_credentials(**session.auth.token_kwargs).access_to
ken
print(f"Token:\n{token}")
```