

# Ogre SDK - Demand Forecast Flow

End-to-end guide for running a demand forecast with the **ogre-sdk** Python client.

---

## Overview

The Ogre SDK wraps the Ogre Forecasting API into a concise Python interface. A single **Ogre** client exposes three namespaces - **meteo**, **demand**, and **generation** - each grouping related resources. All long-running operations return a **PendingJob** whose `.wait()` method blocks until completion and returns the result.

Shell

```
pip install ogre-sdk
```

---

## Quick-start (full pipeline)

Python

```
from datetime import datetime
from pathlib import Path
from ogre_sdk import Ogre

ogre_ai = Ogre(api_key=API_KEY, project_id=PROJECT_ID)

consumer = ogre_ai.demand.consumers.create(
    name="Demo Consumer",
    latitude=53.350, longitude=-6.260,
    dso_id=DSO_ID, profile_id=PROFILE_ID,
    external_id="external-consumer-id",
)

consumer.import_historical(
    CSV_PATH, start_date=TRAIN_START, end_date=TRAIN_END).wait()
consumer.ingest_meteo(TRAIN_START, TRAIN_END).wait()
consumer.train(TRAIN_START, TRAIN_END).wait()

if FORECAST_END > TRAIN_END:
    consumer.ingest_meteo(FORECAST_START, FORECAST_END).wait()

result = consumer.run_forecast(FORECAST_START, FORECAST_END).wait()
```

---

## Configuration

All credentials and project identifiers are provided by the OGRE team. Set them once at the top of your script or notebook.

```
Python
from datetime import datetime
from pathlib import Path

API_KEY      = ""          # your OGRE API key
PROJECT_ID   = "737f49b8-..."
DSO_ID       = "7cefd274-..."
PROFILE_ID   = "2c3c831f-..."

CSV_PATH = Path("historical.csv")

TRAIN_START  = datetime(2016, 1, 1)
TRAIN_END    = datetime(2017, 12, 31)
FORECAST_START = datetime(2018, 1, 1)
FORECAST_END  = datetime(2018, 2, 1)
```

---

## Create the client

Instantiate `Ogre` once and reuse it across all calls. Passing `project_id` here makes it the default for every demand request.

```
Python
import logging
from ogre_sdk import Ogre

logging.basicConfig(
    level=logging.INFO, format="%(asctime)s | %(levelname)s | %(message)s")

ogre_ai = Ogre(api_key=API_KEY, project_id=PROJECT_ID)
```

---

## Sanity check - list available meteo sources

A read-only call that confirms your API key and network connectivity are working before running the full pipeline.

Python

```
sources = ogre_ai.meteo.sources.list()
print(f"{len(sources)} meteo source(s) available")
```

---

## Step 1 - Create a consumer

A *consumer* represents a single metered site defined by GPS coordinates, a DSO, and a load profile. The SDK automatically provisions the meteo location, ingestion settings, and forecast settings in the same call, and returns the enriched `Consumer` object with `.meteo_location` and `.forecast_settings` already attached. If a meteo location with the same coordinates already exists it is reused rather than duplicated. **Run once per site.**

Python

```
consumer = ogre_ai.demand.consumers.create(
    name="Demo Consumer",
    latitude=53.350,
    longitude=-6.260,
    dso_id=DSO_ID,
    profile_id=PROFILE_ID,
    external_id="external-consumer-id",
)

print(f"Consumer id:           {consumer.id}")
print(f"Meteo location id:    {consumer.meteo_location.id}")
print(f"Forecast settings id: {consumer.forecast_settings.id}")
```

`external_id` is your internal identifier for this site and is used for data integration (e.g. real-time feeds, forecast delivery).

---

## Step 2 - Import historical consumption data

Upload the historical consumption CSV for the training window. `.wait()` blocks until the import job is complete (typically ~2 minutes). Run before each retrain.

Python

```
import_job = consumer.import_historical(
    CSV_PATH,
    start_date=TRAIN_START,
    end_date=TRAIN_END,
)
import_job.wait()
```

The CSV must contain a `Timestamp` column and one consumption column per consumer (DEFAULT\_HISTORICAL\_OGRE\_1 format). The consumption column name must match the `external_id` set on the consumer in Step 1.

---

### Step 3 - Ingest historical meteo data

Fetches historical weather data for the training window from the configured meteo source(s). `.wait()` blocks for the duration of the ingest (typically ~10 minutes).

Python

```
meteo_job = consumer.ingest_meteo(TRAIN_START, TRAIN_END)
meteo_job.wait()
```

---

### Step 4 - Train the forecast model

Trains a forecast model on the historical consumption and meteo data. `.wait()` blocks until training is done (~10 minutes) and returns the trained model metadata.

Python

```
training_job = consumer.train(TRAIN_START, TRAIN_END)
trained_model = training_job.wait()
print(f"Trained model id: {trained_model['id']}")
```

---

### Step 4b - Ingest meteo data for the forecast period

Only required when the forecast window extends beyond the training window. Skipped automatically if the training ingest already covers the full forecast period.

Python

```
if FORECAST_END > TRAIN_END:
    meteo_job = consumer.ingest_meteo(FORECAST_START, FORECAST_END)
    meteo_job.wait()
else:
    print("Forecast period within training window - skipping.")
```

---

## Steps 5-6 - Run forecast & download results

Triggers an on-demand forecast run and waits until the results are ready (~5 minutes). `.wait()` returns the forecast records directly as a list.

Python

```
forecast_job = consumer.run_forecast(FORECAST_START, FORECAST_END)
result = forecast_job.wait()
print(f"Forecast records: {len(result)}")
```

```
# To re-download a previously computed forecast without triggering a new run:
# result = consumer.get_forecast(FORECAST_START, FORECAST_END)
```

---

## Pipeline flow summary

Step	Method	Typical wait	Run when
1	<code>consumers.create(...)</code>	immediate	Once per site
2	<code>consumer.import_historical(...)</code>	~2 min	Before each retrain
3	<code>consumer.ingest_meteo(...)</code>	~10 min	After new meteo location
4	<code>consumer.train(...)</code>	~10 min	Before first forecast
4b	<code>consumer.ingest_meteo(...)</code>	~10 min	If forecast > train window
5	<code>consumer.run_forecast(...)</code>	~5 min	Each forecast run
6	<code>consumer.get_forecast(...)</code>	immediate	Re-download only