JAO PUBLICATION TOOL

API Usage and Best Practice Guide

How to retrieve published data reliably and quickly

For: Transmission System Operators and Market Participants

Applies to: the publicly available data API of the JAO Publication Tool

Version 1.0 - September 2026

Contents

1. About this guide

This guide explains how to retrieve data from the JAO Publication Tool API in the way that works best - both for you and for everyone else using the service. It is written for the people who configure and operate data collection at a TSO or market party. It does not assume software engineering knowledge.

The recommendations here are not arbitrary. They come from measuring how the API actually behaves under real use. Where a recommendation is supported by a measurement, the figure is given so you can judge it for yourself.

Why it matters

The API allows the same data to be requested in several different ways. Those ways are not equivalent. In our measurements, retrieving one business day of data took between roughly one and a half minutes and more than forty minutes, depending only on how it was asked for - the data returned was identical.

The difference matters twice over. It determines how long you wait, and it determines how much load the service carries. A small number of inefficient collection routines can slow the service for every other user, particularly in the minutes after new data is published, when demand is highest.

What this guide covers

- The structure of the API and the two ways to retrieve data
- Six practices that make collection faster and more reliable
- The limits that apply, and how to stay within them
- How to read a response and how to handle errors
- A complete worked example of collecting one business day

Retrieval principles are the same across all JAO regional instances.

2. How the API is organised

2.1 Addresses

Every request follows the same shape: a base address for the regional instance, followed by the name of the dataset you want.

```
https://publicationtool.jao.eu/core/api/data/{dataset}
```

The final part is the dataset name - for example `finalComputation`, `netPos` or `maxExchanges`. Dataset names are not case-sensitive.

2.2 Two ways to retrieve data

The API offers two routes, and choosing the right one for the job is the single most effective thing you can do.

ROUTE	WHAT IT RETURNS	BEST FOR
The data endpoint `/api/data/{dataset}`	Rows as JSON, in the response itself	Regular automated collection; recent data; when you need to filter or process rows directly
The download endpoint `/api/data/{dataset}/download`	A file (CSV or XML), delivered in two steps	Getting a copy of a period for storage or offline analysis; when you want a file rather than rows

Both respect the same limits, including the maximum period per request described in Section 5.

2.3 Checking whether data has arrived

Before requesting data, you can ask whether it is there. The monitoring endpoint reports, for each dataset and business day, whether the files behind it have been received.

```
https://publicationtool.jao.eu/core/api/system/monitoring
    ?FromUtc=2026-09-17T00:00:00.000Z
    &ToUtc=2026-09-18T00:00:00.000Z
```

The response contains one entry per dataset per business day. The two fields that matter are page - the dataset name - and status:

STATUS	MEANING
Received	Everything expected for that dataset and business day has arrived. The data is ready to collect.
Expected	Not yet delivered, but still within the expected delivery time. Check again shortly.
Pending	Not yet delivered and past the expected delivery time. The data is late; collecting will not help until it arrives.
(empty)	Nothing is expected for that dataset on that business day.

> **Received means complete.** Where a dataset is built from several files, the status reflects the least advanced of them. A dataset only reports Received once every file expected for that business day has arrived - so you are not at risk of collecting a partially delivered day.

This is the right way to find out whether data is ready. It is a far better signal than requesting the data itself and seeing whether anything comes back, and it does not place the same load on the service.

Two practical points. Ask for a narrow period - the business day you care about, not a wide range. And check at a sensible interval, a few minutes apart rather than continuously; this endpoint answers from live data and is not free to produce.

3. Six practices for reliable collection

These are listed in order of impact. The first two account for most of the difference between a collection routine that finishes in under two minutes and one that takes the better part of an hour.

3.1 Select data by time, do not page through it

Every dataset is selected using a time range: `FromUtc` and `ToUtc`. These are the primary way to ask for data, and they should do nearly all the work.

Some datasets additionally accept `Skip` and `Take`, which move through a result set a page at a time. These exist for browsing on screen. They are not an efficient way to collect data, because the service must count past every row you skipped before it can return the page you asked for. The further into the data you go, the more work each request causes.

Our measurements show this clearly. Requesting the earliest pages of a day failed around 3% of the time; requesting the deepest pages of the same day failed around 28% of the time - the identical request shape, differing only in how far in it reached.

> **In short.** Use `FromUtc` and `ToUtc` to narrow what you ask for. Reach for `Skip` and `Take` only when you genuinely need a small sample, never to walk through a full day.

3.2 Retrieve a business day hour by hour

A published business day contains a large amount of data - for final computation, in the order of 475,000 records. Requesting all of it in one go produces a single very large response that is slow to prepare, slow to transfer, and likely to fail partway through.

Requesting the same day as twenty-four separate hourly requests is consistently faster and far more reliable. In a controlled comparison over forty-five rounds spread across a full day, hour-by-hour collection completed roughly two to three times faster than the paged equivalent, and was the faster method in forty-four of those forty-five rounds.

It also fails better. If one hour fails you retry one hour, rather than starting the whole day again.

> **In short.** Collect a business day as twenty-four hourly requests, one after another. See Section 8 for a worked example.

3.3 Ask only for the rows you need

Most datasets accept a filter, which is applied by the service before anything is sent to you. Using it reduces the work the service does, the amount transferred, and the time you spend waiting.

The alternative - downloading everything and discarding most of it locally - is common and is the most wasteful pattern we see. If you only ever use a few TSOs or a few critical network elements, say so in the request.

Filters available on the computation datasets include the critical network element name, TSO, hub from and hub to, contingency, and whether the element was presolved. Filters are combined, and text filters match on partial names.

3.4 Retrieve data once and keep it

Published data does not change after publication, except in the uncommon case of a correction being republished. Once you have retrieved a period successfully, there is no need to retrieve it again.

In our analysis of production traffic, a substantial share of all requests were exact repeats of a request the same client had already made successfully. Storing what you retrieve - even briefly - removes that load entirely and makes your own collection faster.

Where a dataset returns a `lastModifiedOn` value, you can use it to tell whether data has been republished since you last collected it, and re-fetch only when it has.

3.5 Send requests one at a time

Requests from the same client should be sent sequentially: start the next one when the previous one has finished. Opening several connections at once to collect faster is counterproductive - it does not reduce total time appreciably, and it raises the chance of requests being rejected or timing out, for you and for others.

This matters most during the publication window, when many organisations are collecting simultaneously and the service is at its busiest.

> **In short.** One request at a time per organisation. If collection is too slow, make each request smaller rather than making more of them at once.

3.6 Accept compressed responses

The service compresses responses when the client indicates it can accept them. This reduces the amount transferred by roughly a factor of eight, which is a substantial saving on large datasets and costs nothing to enable.

Most modern tools do this automatically. If yours does not, add the standard header:

```
Accept-Encoding: gzip
```

4. When to collect

Timing has a large effect on how long collection takes, because demand on the service is far from evenly spread.

4.1 The publication window

When final domain data for the next business day becomes available in the morning, a very large number of organisations begin collecting at almost the same moment. This is the busiest period of the day by a wide margin, and response times during it are materially longer than at any other time.

If your collection is time-critical, this is unavoidable, and the practices in Section 3 matter most here. If it is not time-critical - for example a routine daily archive, or a backfill of older periods - schedule it outside the publication window. Both you and everyone else will see better performance.

4.2 Waiting for data to appear

A common and costly pattern is to repeatedly request the data itself until something comes back. This concentrates load at exactly the wrong moment.

Instead, check the monitoring endpoint described in Section 2.3, a few minutes apart, and start collecting once it reports the dataset as Received. One modest check every few minutes replaces dozens of large data requests that were always going to return nothing.

If you must retry a data request, leave a growing gap between attempts - for example ten seconds, then twenty, then forty. Retrying immediately, or in a tight loop, makes a busy period worse and rarely succeeds sooner.

5. Limits that apply

The service applies a small number of limits. They exist to keep it responsive for everyone. Staying comfortably within them is straightforward if you follow Section 3.

LIMIT	VALUE	WHAT HAPPENS IF EXCEEDED
Period per request	2 days	The request is refused with a clear message. Split the period into shorter requests.
Requests per minute	100 per client	Further requests are refused until the next minute begins. Pace your collection.
Time per request	5 minutes	The request is abandoned and an error returned. Ask for a shorter period.

The two-day limit applies to the download route as well as the data route. To retrieve a longer period, split it into consecutive requests of ten days or fewer.

> **Collecting long historical periods.** If you need to load months or years of history - for example when onboarding or rebuilding a data store - please contact JAO before doing so. Large historical loads can be arranged in a way that does not affect day-to-day service, and doing this unannounced during business hours is the pattern most likely to cause disruption for others. JAO is actively working on a solution to make historical snapshots available for large range data downloads.

6. Understanding the response

Responses from the data endpoint are JSON. The rows you asked for are in data. Depending on the dataset, several additional fields describe what you received.

FIELD	MEANING
`data`	The rows themselves.
`rejected`	Whether the request was refused. See the note below - this can be true even on an apparently successful response.
`messages`	Any explanation of why a request was refused.
`totalRows`	How many rows exist in the period you asked for, before any filter.

FIELD	MEANING
`totalRowsWithFilter`	How many rows match after your filter is applied. Equal to totalRows when no filter was sent.
`appliedFilter`	The filter the service actually used. Worth checking if results are unexpected.
`lastModifiedOn`	When the returned data was last changed. Useful for detecting republished corrections.
`skip, take`	Echo of the paging values used, where applicable.

> **Always check the rejected field.** A refused request is not always reported as a failure at the transport level. A response can arrive apparently successfully while carrying rejected set to true and an explanation in messages. Collection routines should check this field, not only whether a response was received.

Not every dataset returns every field. The larger computation datasets return all of them; simpler datasets return only the rows, together with rejected and messages.

7. When something goes wrong

The table below covers what you are most likely to encounter and what to do about it.

RESPONSE	WHAT IT MEANS	WHAT TO DO
Bad request (400)	Something about the request is not valid - most often the period is longer than ten days, or one of the dates is missing.	Read the message, which states the problem directly. Correct the request; do not retry it unchanged.
Too many requests (429)	You have exceeded the rate limit, or have more than one request in flight at once.	Wait before retrying. If a Retry-After value is supplied, honour it. Reduce how quickly you are sending.
Not found (404)	The dataset name was not recognised.	The response body is empty in this case, so the name is the thing to check.
Server error (500)	Something failed on the service side. The message contains a reference code.	Retry once after a pause. If it persists, contact JAO and quote the reference code - it identifies your exact request in our logs.

7.1 Retrying well

How you retry matters as much as whether you retry.

- Leave a gap, and increase it with each attempt.
- Give up after a few attempts and report the failure rather than retrying indefinitely.
- Never retry a request while the original is still running.
- Never retry a rejected request unchanged - it will be rejected again.

8. Worked example: collecting one business day

This example collects final computation data for the business day of 17 September 2026 from the Core instance, following the practices in this guide.

Step 1 - Confirm the data has arrived

```
curl "https://publicationtool.jao.eu/core/api/system/monitoring\
  ?FromUtc=2026-09-17T00:00:00.000Z&ToUtc=2026-09-18T00:00:00.000Z"
```

Look for the entry whose page is `finalComputation` and proceed once its status is Received. If it is Expected, wait a few minutes and check again.

Step 2 - Work out the day in UTC

A business day runs from midnight to midnight in Central European time, which is not the same as midnight UTC. During summer time the day runs from 22:00 UTC on the previous day to 22:00 UTC on the day itself; during winter time, from 23:00 to 23:00.

For 17 September 2026 (summer time), the day is 16 September 22:00 UTC to 17 September 22:00 UTC.

Step 3 - Request each hour in turn

Make twenty-four requests, one per hour, each waiting for the previous to finish. The first is:

```
curl --compressed "https://publicationtool.jao.eu/core/api/data/finalComputation\
  ?FromUtc=2026-09-16T22%3A00%3A00.000Z\
  &ToUtc=2026-09-16T23%3A00%3A00.000Z"
```

The second covers 23:00 to 00:00, the third 00:00 to 01:00, and so on, until the twenty-fourth covers 21:00 to 22:00 on 17 September.

The `--compressed` option asks for a compressed response, as described in Section 3.6. The colons in the timestamps are written as `%3A` because they appear inside a web address.

Step 4 - Check each response

For each hour, confirm that rejected is not true and that rows were returned. If an hour fails, retry that hour after a short pause rather than restarting the day.

Step 5 - Store what you collected

Keep the result. If you need this day again - for a report, a recalculation, or a comparison - use your stored copy rather than requesting it a second time.

> **Narrowing it further.** If you only need certain TSOs or certain critical network elements, add a filter to each request as described in Section 3.3. This reduces both the time you wait and the load your collection places on the service.

9. Quick reference

Request parameters

PARAMETER	REQUIRED	NOTES
`FromUtc`	Yes	Start of the period, in UTC. Inclusive.
`ToUtc`	Yes	End of the period, in UTC. Must not be more than 10 days after FromUtc.
`Filter`	No	Restricts which rows are returned. Strongly recommended where applicable.
`Take`	No	Maximum rows to return. Available on the larger computation datasets only.
`Skip`	No	Rows to skip before returning. Avoid for bulk collection - see Section 3.1.

Do and do not

DO	DO NOT
Collect a business day as 24 hourly requests	Request a whole day, or several days, in one call
Select with FromUtc and ToUtc	Page through a day with Skip and Take
Filter on the service where you can	Download everything and filter locally
Send one request at a time	Open several connections at once to go faster
Check the monitoring endpoint before collecting	Repeatedly request data until it appears
Store what you have already retrieved	Re-request data you already hold
Accept compressed responses	Ignore compression on large datasets
Back off and increase the gap when retrying	Retry immediately, or in a tight loop
Contact JAO before large historical loads	Start a multi-year backfill during business hours

Questions about this guide, or about arranging a large historical data load, should be directed to JAO.