Processing SEC submissions data

Ian D. Gow

26 October 2026

This note describes how to use the `sec-submissions` package to obtain and process the SEC's submissions data. The package converts the bulk JSON archive into Parquet, preserves the original timestamp text, and uses comparisons with earlier data and filing headers to produce time-zone-aware acceptance timestamps. I explain the evidence behind those corrections and report a sample-based assessment of the resulting data.

> **i** Methods and data versions
>
> This note describes the package workflow and reports the original September 30, 2026 audit alongside subsequent findings. New candidates treat live-file inference as provisional and can use filing-index references where SGML timestamps are absent. The package methodology and audit documentation give the maintained details. Each dated assessment concerns its own frozen candidate; it does not assess a later file.

In an earlier note, I described the structure of the SEC's submissions data and showed how to process those data using R. That note introduces JSON, list-columns, and the tools used to turn hierarchical data into tables. Here I focus on doing the work using my `sec-submissions` Python package, which brings together the extraction, timestamp-correction, and audit code used to prepare the data discussed below. I switched the bulk processing to Python for efficiency; the resulting Parquet files can still be analysed using R and DuckDB.

Since I wrote the earlier note, I have also found a problem with the acceptance timestamps supplied in newer submissions data. Some represent UTC and others New York local time, even when the text ends in `Z`. The package therefore produces two filings tables: `filings_raw.parquet`, with the original timestamp text, and `filings.parquet`, with time-zone-aware timestamps and information about how they were established. Keeping the two separate allows the interpretation to improve without losing the source data or extracting the JSON archive again. New candidates also retain the original text in `acceptanceDateTime_raw`, allowing later updates to compare source clocks even when an earlier interpretation was unresolved. My note on dates and times discusses the additional issues that arise when analysing these timestamps.

The workflow has four stages: extract a dated snapshot, resolve timestamps using earlier evidence and new comparisons, collect filing headers for selected remaining cases, and audit a frozen candidate before release. Readers interested only in analysing the data can instead download the current corrected Parquet. The examples below show how to prepare an update using the package; the reported results concern the 30 September 2026 release. Running the download examples later obtains a different SEC snapshot, which needs its own checks and audit.

Getting started

The package requires Python 3.12 or later. It is not yet on PyPI; install it from GitHub, including the optional dependencies used to calculate audit confidence bounds:

```
python -m pip install 'sec-submissions[audit] @ git+https://github.com/iangow/sec_submissions.git'
```

The package uses `RAW_DATA_DIR` for original ZIP snapshots and `DATA_DIR` for Parquet files, supporting evidence, and audits. It reads these settings from a `.env` file in the working directory, preserving values already set in the process environment. An optional configuration is:

```
RAW_DATA_DIR=~/data/raw_data
DATA_DIR=~/data/pq_data
SEC_USER_AGENT="Your Name your.email@example.org"
```

Without these directory settings, the roots are `~/sec-submissions-data/raw_data` and `~/sec-submissions-data/pq_data`. Each contains a `submissions/` subdirectory, and explicit function arguments override the defaults. SEC requests require an identifying user agent; if one has not been configured, the first interactive request prompts for it and offers to save it. The package moderates concurrent requests in accordance with the SEC's developer guidance. The data and directory guide explains the layout, and the API reference documents the functions used below.

For a first update, obtain a corrected reference and its supporting evidence:

```
import sec_submissions as sec

reference = sec.fetch_reference(version="2026-09-30")
reference.filings
```

This downloads the dated corrected file, reusable JSON and SGML observations, source metadata, and audit artifacts. Every file is checked against its recorded size and SHA-256 checksum; interrupted downloads can resume, and verified local files are reused. Omit `version` to obtain the current release, or add `companions=True` for the other Parquet tables. These public GitHub downloads require neither a GitHub account nor an SEC user agent.

From the bulk archive to Parquet

The SEC provides a bulk submissions archive, which it recompiles nightly. Downloading that archive is more efficient than obtaining a separate JSON file for every CIK. Use the package to retain a dated snapshot and extract it:

```
snapshot = sec.download_submissions()
sec.extract_submissions(snapshot, max_seconds=0)
```

`download_submissions()` saves the ZIP under the raw-data root and records its source URL, retrieval time, HTTP modification headers, size, and checksum in a small Parquet manifest. It does not overwrite an earlier snapshot. `extract_submissions()` writes the following tables into a separate directory for that snapshot under the Parquet root.

File	Contents
`filings_raw.parquet`	Filing records, including the unaltered acceptance timestamp text
`companies.parquet`	Company metadata from each primary CIK file
`addresses.parquet`	Business and mailing addresses
`tickers.parquet`	Tickers and exchanges
`former_names.parquet`	Former names and their date ranges
`files.parquet`	References to historical filing files, including whether they are present in the archive

The extractor reads every primary CIK file and the historical files that it references, retaining `cik` and `source_file` so that each filing row can be traced to its source. It does not start from a selected list of companies. Historical files referenced more than once by a company are read only once for that company; absent files are recorded in `files.parquet` but their filing rows cannot be recovered from the ZIP. Company metadata describes the downloaded snapshot, not necessarily the company as it existed on each filing date.

Dates such as `filingDate` and `reportDate` are written as dates, while identifiers that may contain leading zeros remain text. `acceptanceDateTime` remains text, including any suffix and fractional seconds: extraction makes no decision about

the time zone represented by that text. The other tables do not need separate raw versions because they do not contain acceptance timestamps.

The extractor saves checkpoints. By default it returns after about 150 seconds; repeating the call resumes the same extraction. The example uses `max_seconds=0` to continue until completion. Do not replace the source ZIP while resuming an associated checkpoint. Using a separate directory for each snapshot also prevents its companion tables from replacing those of another snapshot.

Establishing the timestamps

The unit to which the package attaches a corrected timestamp is the **accession number**. A filing may appear under more than one CIK, but those appearances should refer to the same acceptance instant. An accepted correction is therefore joined to every occurrence of that accession when producing the candidate file. A rule for an entire archived CIK file is not sufficient: its timestamps can have mixed interpretations.

The first update and later updates

Our original analysis had a useful starting point: the May 2024 archive used a consistent New York local-clock convention. The first update compared a Parquet reference derived from that archive with the ZIP downloaded on 5 September 2026, then used duplicate-accession comparisons, live JSON, and selected SGML headers to address the remaining cases. The package's `make_previous()` function supports that historical bootstrap, but the 2024 convention is not a rule for newer snapshots.

The second update started from the corrected September 5 Parquet and processed the September 30 ZIP. Readers need not repeat either historical preparation: the downloadable September 30 reference incorporates that work and can serve as the starting point for their next update. Thereafter, their own preceding corrected Parquet can provide the reference.

Create an initial candidate using the newly extracted data and available evidence:

```
sec.process()
```

With the default layout, `process()` uses the latest extracted `filings_raw.parquet` and the local `DATA_DIR/submissions/filings.parquet` as its predecessor when that file exists. Otherwise it uses the downloaded reference. It reuses the reference's observations and locally collected JSON/SGML evidence, writes a fresh numbered candidate such as `filings_candidate_0001.parquet`, and saves diagnostics beside it. It does not download new SEC evidence or replace the current release. Explicit inputs are available for controlled comparisons, as described in the workflow guide.

Only supported prior instants can act as anchors. For each accession, the usable prior rows must agree on one instant, and interpreting a new raw clock as either UTC or `America/New_York` must reproduce that instant. A previously unresolved Eastern fallback or provisional live-file inference is not an anchor. Its observed source clock may nevertheless supply one member of a new exact clock pair. The processor can use the preserved `acceptanceDateTime_raw` column or explicitly supplied historical raw files, rather than treating a converted provisional guess as another observation. Changes in non-timestamp fields come from the newly extracted archive.

Resolve differing clocks first

An accession appearing under several CIKs may have different raw clocks. Comparisons with the previous snapshot or live submissions JSON may also reveal a second clock for the same accession. The useful case has exactly two distinct valid clocks that resolve to one instant when the earlier clock is interpreted as `America/New_York` and the later as UTC. For example, 08:01:07 in New York on a December date and 13:01:07 UTC on that date describe the same instant.

The package tests that equality using the named time zones, accounting for daylight saving and changes of date. A difference of four or five hours alone is insufficient. Neither the newer snapshot nor the live JSON is assumed to supply the UTC member: either source may provide the later clock reading. Equal clocks provide no new timezone evidence, and unexplained differences remain available for investigation. This comparison does not require a filing to be outside ordinary EDGAR hours.

Use live-file evidence carefully

One live JSON response can provide observations for many accessions. Where a direct pair cannot resolve an accession, supported prior instants, resolved clock pairs, and SGML observations can anchor the interpretation of that response. If all available anchors agree, the package can apply the inferred convention to its other observed clocks.

A live response can contain both UTC and Eastern clocks, even when the available anchors suggest a common convention. The resulting proposals are therefore **provisional**, not confirmations. The package records the anchors, rejects contradictory file evidence, and never promotes a provisional proposal to confirming evidence in a subsequent update. It identifies a response version by its source URL and retrieval time: a later response from the same URL need not use the same convention. It does not apply a live-file convention indiscriminately to all rows in the ZIP or to future accessions not observed in that response.

An already established accession instant can usually be reused without fetching its JSON or SGML again, provided the new raw clock reproduces it. That is the principal saving in a subsequent update. It is different from assuming that all new filings for an earlier CIK must use the same time zone.

Collecting additional evidence

The initial candidate identifies what the existing observations cannot resolve. It is useful to screen unresolved and provisional recent filings, not only those outside ordinary EDGAR hours: a UTC clock can fall within that window and still be misinterpreted as Eastern. For example, collect live JSON for files containing unresolved records from 2024 onward, then process the data again:

```
from datetime import date

sec.collect_live_json(since_date=date(2024, 1, 1), workers=8, rate=5)
sec.process()
```

The collector freezes the selected set of source files, fetches matching raw accessions, and follows linked historical JSON when necessary. It includes already supported accessions in those files so that they can serve as anchors. Its Parquet observations preserve source URLs and retrieval times; a completed request queue does not imply that every accession was found or every timestamp resolved. The recent cutoff defines the collection scope, not a claim that older unsupported timestamps are correct. Workers overlap network waits, while `rate` limits total requests per second across those workers.

JSON comparisons should precede a large SGML queue because one response can resolve many timestamps. For remaining questions, the archived filing header's `<ACCEPTANCE-DATETIME>` field provides an accession-level reference, interpreted here as a New York local clock and converted using `America/New_York`. An exact SGML observation has priority over weaker file-level inference, while disagreements remain in the diagnostics.

`collect_sgml()` expects an explicit queue with `cik` and `accession_number` columns; it does not automatically select an anchor for every unresolved file. After preparing such a queue from the candidate and its coverage diagnostics, collect it and create another candidate:

```
sec.collect_sgml("sgml_queue.parquet", workers=8, rate=5,
                 index_fallback=True)
sec.process()
```

Possible selections include missing live matches, contradictory evidence, and live response versions without anchors. For the September 30 update, the follow-up used one accession for single-accession response versions and the first and last accessions for multi-accession versions. The queue contained 23,906 accessions; 23,905 supplied a timestamp tag and one returned 404. That historical anchoring design relied on a common-convention assumption. The current package retains such inferences as provisional; an independent audit addresses a different question.

An outside-hours screen remains useful for diagnosis, but an outside-hours time is not automatically an error. `EFFECT` is excluded from that screen, and the targeted anchor collection also excluded `CORRESP`, `UPLOAD`, and `DRSLTR`. Those forms

can still participate in exact clock-pair comparisons and the random audit. Their form names or a global date cutoff should not determine their time zone.

Successful SGML observations, missing timestamp tags, and request errors are retained separately. Transient request failures can be retried. Where an SGML tag is absent, filing-index fallback reads the labelled `Accepted` value and retains its source separately; an unsuccessful lookup remains unavailable evidence. Likewise, literal midnight is a useful diagnostic, not a sufficient rule for declaring a timestamp missing. Historical date-only values must remain distinguishable from observed acceptance times: midnight Eastern is a storage convention for those dates, not evidence of an intraday acceptance time. The collection functions support resumption with the same inputs and settings; see the workflow guide for those options.

The candidate and its provenance

Every processing run combines the evidence into one accession-level interpretation and writes a separate candidate with time-zone-aware `acceptanceDateTime` values. The package uses a temporary DuckDB database for this computation, exports the candidate and Parquet diagnostics, and removes the working database. Readers do not need to create or maintain a permanent database.

Rows without an accepted instant retain an Eastern-time interpretation labelled `unresolved_raw_as_eastern`. That is a compatibility fallback, not a verified timestamp. The output's `timestamp_provenance`, `timestamp_reference_id`, and `timestamp_interpretation` preserve the distinction between established instants, inferred conventions, and unresolved cases. For example, `sgml` denotes direct header evidence and `provisional_live_file_inferred` denotes a proposed live-file convention. The older `live_file_inferred` label must likewise be treated as provisional when reading the frozen September 30 release. Current diagnostics count supported, provisional, and unresolved rows separately.

The main Parquet therefore carries the accepted instants and their status forward. The separate observation files retain source clocks, failed attempts, retrieval versions, and evidence for accessions that may be absent from a later snapshot. Diagnostics record the inputs, support counts, rejected evidence, and disagreements. Check these alongside row counts and non-timestamp fields before releasing the candidate. Request completion, evidence-supported coverage, and an accuracy estimate are different things.

Checking quality with a sample

The checks used to find and correct problems do not estimate the frequency of errors in the released data. A collection concentrated on recent or outside-hours filings is not representative of all filing rows. For that purpose, draw a probability sample from a frozen candidate. Readers using the published data can inspect the results and downloadable audit artifacts without repeating this collection themselves.

Freeze predictions before fetching headers

The population assessed here is the rows of `filings.parquet`, including repeated appearances of an accession under different CIKs. An accession-level accuracy estimate answers a different question. All form types belong in this audit; a sample limited to unresolved cases cannot establish the accuracy of the whole file.

The current audit uses SGML first and the filing index's labelled `Accepted` value when the SGML tag is absent. Use the package to draw and audit a reproducible simple random sample:

```
sec.audit(sample_size=10_000, seed=202609304, workers=8, rate=5)
```

By default this selects the latest candidate, saves the sampled row keys and predicted instants before fetching SGML, and does not reuse an observation cache. It uses seeded reservoir sampling, retains repeated appearances of an accession in the sample, and deduplicates only the fetch queue. It never changes the candidate or its frozen predictions. The seed alone does not reproduce the published sample for another file; the candidate's identity and the saved identifiers matter too.

The output includes sample identifiers, frozen predictions, observation batches, row-level outcomes, and a summary. Record agreement, disagreement, missing timestamp tags, and request failures separately, retrying transient failures without changing sample membership. Do not replace sampled rows because their headers are unavailable. An audit

used to repair a candidate must be followed by a new independent assessment of the revised candidate. The processor does not automatically use a candidate's own audit observations as repair evidence; those observations can be retained for a subsequent update.

A fresh header lookup does not make every comparison independent of the original evidence. Where an SGML timestamp already supported the prediction, agreement checks application of that evidence rather than independently establishing the header's timezone semantics. Report outcomes by evidence group where sample sizes permit, as below.

What the sample can establish

If there are zero disagreements in n verifiable observations, the one-sided 95% binomial upper bound on the error rate is

$$p_U = 1 - 0.05^{1/n}.$$

For a small sampling fraction, this is a conservative approximation for sampling rows without replacement. It is close to $3/n$: zero errors in 600 verifiable observations gives a bound of about 0.5%, and zero in 3,000 gives about 0.1%. With observed errors, use an appropriate binomial interval rather than only the sample error percentage; NIST describes exact binomial confidence limits. The package calculates a one-sided exact bound using the optional audit dependencies installed above.

Unavailable timestamps limit the conclusion. A bound among verifiable rows does not establish whether rows without reference timestamps are correct, so it is not valid to call the entire file 99.9% accurate merely because 3,000 obtainable headers agree. Diagnostic samples of recent filings, correspondence forms, or suspicious clocks can be useful additions, but must remain separate from the overall estimate unless they form part of a probability design with appropriate weights.

Results for the September 30 update

The worked example's first ZIP, downloaded on 5 September 2026, contained 27,209,834 filing rows, with filings dated through 4 September. The September 30 ZIP was retrieved at 13:39 UTC, had an HTTP modification time of 12:34:49 UTC that day, and supplied 27,272,863 filing rows. The release manifest records its checksum, the released candidate's checksum, and the audit metadata.

The frozen September 30 file labels 25,584,390 timestamps as supported under the policy used for that release, alongside 1,688,473 explicitly unresolved Eastern fallbacks. That historical supported count includes live-file inference, which the current package treats as provisional. The following results describe the original SGML-only audit of that file. A fresh 10,000-row sample was frozen before new SGML lookup, using seed `202609304`. The audit found no disagreements among the 9,230 rows with verifiable headers. The exact one-sided 95% upper bound on the error rate among those verifiable rows is **0.0325%**.

Candidate evidence group	Sample rows	Verifiable	Disagreements
Trusted 2024 snapshot	6,702	6,695	0
ZIP/live-JSON clock pairs	1,446	1,445	0
Live-file inference	953	862	0
Cross-CIK accession pairs	173	99	0
Direct SGML evidence	129	129	0
Snapshot-pair inference	1	0	n/a
Unresolved Eastern fallback	596	0	n/a
Total	**10,000**	**9,230**	**0**

All 770 unavailable references lacked the SGML timestamp tag; none remained unavailable because of a failed request. They were filings dated 1994 through 2002, with no missing tags observed for 2003 onward in this sample. In particular, none of the sampled unresolved fallbacks was verifiable: the result does not validate those rows. The sample identifiers and frozen predictions and row-level outcomes make the assessment transparent without listing 10,000 records in the note. These results remain a dated record of that original assessment.

Subsequent findings

A follow-up obtained a filing-index reference for each of the 770 missing-tag rows while retaining the original sample and predictions. Seven supplied acceptance times that agreed exactly; the other 763 were historical date-only values whose dates agreed with the source. Of those date-only values, 52 had an incorrect midnight representation in the frozen file. This distinction matters: agreement on the date is not agreement on an observed acceptance instant.

An independent October 2 audit of a later pair-corrected candidate found two disagreements in 10,000 rows, with all rows verifiable using 9,234 SGML and 766 filing-index references. Both disagreements arose from a live-response-wide timezone convention, which led to the package's provisional treatment of such inferences. The observed disagreement rate for that candidate was 0.020%, with a one-sided 95% upper bound of 0.06294%.

Later targeted collection and checks against saved samples verified repairs, but those are regression checks rather than a new independent estimate for the repaired file. A newly frozen random audit remains the release gate for that candidate. The maintained audit report identifies each candidate, the original observations, the supplemental references, and the limits of the comparisons.

Releasing an update

After reviewing the candidate, diagnostics, and audit, install the candidate as the local current file:

```
sec.publish()
```

This writes `DATA_DIR/submissions/filings.parquet`. If a release already exists, the publisher first saves it as `filings_previous.parquet` and moves an older previous version into a dated `archive/` entry. It validates the copied bytes and preserves the existing current file's local identity. The in-place update is not atomic for other readers, so they should wait until it finishes. The package does not upload the data or perform storage-service-specific link checks.

For the public reference, I attach the validated files and a checksum manifest to a dated release in a separate data repository. All files are uploaded to a draft release and checked before it becomes public. The current-file link follows the latest release; a dated link identifies a fixed snapshot. The package's `fetch_reference()` resolves the current release once and downloads its files from that dated version, avoiding a mixture of releases.

For the next update, retain the preceding corrected Parquet and its supporting observations, then repeat extraction, comparison, bounded evidence collection, and an independent audit. The earlier ZIP is useful for reproducibility but need not be reprocessed to carry its established accession instants forward. The command-line equivalents of the functions shown here are documented in the package workflow guide.