From File to Database: Automating Data Ingestion at Statistics Finland

Toni Kuronen, Statistics Finland, toni.kuronen@stat.fi

Abstract

To modernize the production of official statistics, it is important to be able to automate the processing of data files received from various data providers. The recent advances in file analysis techniques and metadata management systems (MMS) have made it possible to automatically detect file formats and generate database import configurations without manual preprocessing. In this paper, a complete framework for automating data file ingestion at Statistics Finland is proposed. The framework includes automatic detection of file encoding, field separators, and fixed-width structures, validation of field count consistency, generation of database format files for bulk loading operations, and integration with the metadata system for retrieving field headers, datatypes and for schema validation. The framework was evaluated with files from tax authorities, immigration services, education registries, and other sources. The results showed that the proposed methods are suitable for processing heterogeneous file formats with varying structures and quality issues. Moreover, the results demonstrated that significant reduction in manual preprocessing time can be achieved while maintaining complete audit trails for data governance requirements. The modular architecture using standard technologies makes the framework adaptable to other Nordic statistical organizations facing similar challenges with various data sources. With modifications, the framework can be utilized in other statistical production environments where automated file processing and validation are needed.

Keywords: automation, data ingestion, format detection, metadata, data governance

1. Introduction

The production of official statistics relies heavily on administrative data received from government agencies and other data providers. At Statistics Finland, data files are received from tax authorities, education registries, immigration services, healthcare providers, and numerous other sources. These files arrive as comma-separated values (CSV), semicolon-separated files, tab-separated files, fixed-width text, Excel workbooks, and other tabular structures. The heterogeneity of formats, character encodings, and structural conventions creates persistent friction at the boundary between sub-processes 4.4 (Finalise collection) and 5.1 (Integrate data) of the Generic Statistical Business Process Model (GSBPM), version 5.2 [2].

Previously, each data source required its own manually created format file and loading procedure or utilizing import wizard manually. When a delivery arrives in the same shape as the previous one the loading works smoothly using the stored configuration. Problems arise whenever a provider changes something: encoding, separators, line endings, or a header row is changed or removed. Developers then must diagnose the change, which consumes time and requires file-format expertise that does not scale as the number of sources grows. To address these challenges, an automation framework was developed that analyses each incoming file independently, detects its structural properties, validates compatibility with the schema, and generates the configuration required for database ingestion. The framework integrates with the Statistics Finland MMS so that processed files are checked against the expected data structure before loading.

This paper describes the architecture and algorithms of the framework, reports on its evaluation with administrative deliveries, and discusses its applicability to other national statistical institutes (NSIs) facing similar problems.

1.1 Related work

Automatic detection of tabular file properties has a substantial prior literature. The csv module in the Python standard library exposes a Sniffer class that infers delimiter and quoting conventions from a sample of lines; van den Burg et al. [5] showed that this approach fails on a third of real-world CSV files and proposed CleverCSV, a consistency-based detector with higher accuracy. Character encoding detection has

an established toolset in the Mozilla universalchardet family and its Python port chardet [6], which use byte statistics and language models to distinguish legacy 8-bit encodings. For delimited-file dialect inference more broadly, the Frictionless Data project [7] has standardised descriptors for tabular resources.

At the workflow level, open-source data integration platforms such as Apache NiFi [8] and Airbyte [9] provide ingestion primitives, but they are not tailored to the combination of strict schema governance and deterministic bulk loading into SQL Server that applies in the Finnish statistical production environment. Several NSIs have reported internal ingestion modernisation efforts; the framework described here is complementary to those efforts and focuses specifically on deterministic format detection and metadata-driven schema validation.

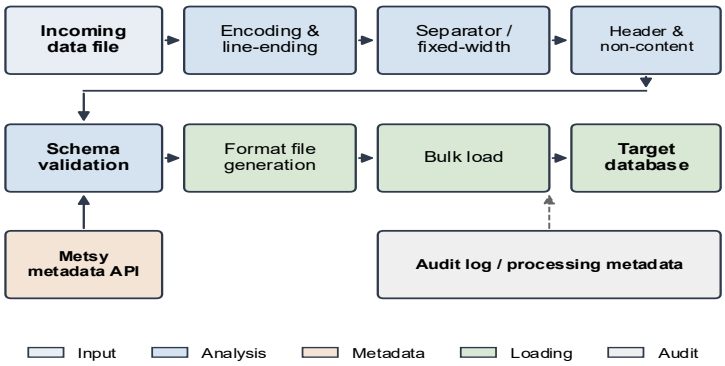
2. Methods

2.1 System architecture

The framework is implemented as a PowerShell module for integration with the existing Windows- and SQL-Server-based statistical production environment at Statistics Finland. PowerShell was selected because of its native integration with SQL Server, its wide availability in the enterprise Windows environment in use, and its support for modular, object-oriented design. The pipeline is shown in Figure 1. Concerns are separated into file analysis, metadata system integration, format file generation, database operations, and error handling, each implemented as a discrete module.

Ingestion operates in two modes. In standalone-mode the framework performs file analysis only: encoding, separator, and structure detection, non-content row(s) removal, and row counting, with results recorded to the audit table. In API-mode, the framework additionally retrieves the registered schema from the metadata API, validates detected file structure against it, generates an SQL Server XML format file for the BULK INSERT operation, and performs field-length and type-compatibility checks. Standalone-mode is used for sources whose schema is not yet registered, or for preliminary analysis before schema registration.

Figure 1 Data ingestion pipeline.



2.2 File format detection

The format detection starts with encoding detection first inspects the byte order mark BOM and when it is not present, the detector reads a prefix of the file and applies a two-stage test. First a UTF-8 validity checks and then other profiles. Following line-ending detection counts CR, LF, and CRLF occurrences in the sample and selects the dominant convention.

For delimited files, separator detection evaluates six candidates: [,;\t:$] on a sample of user defined data lines. Each candidate is scored on two signals: (i) field-count consistency; and (ii) type coherence per column. A candidate is selected when the combined score exceeds a threshold.

Fixed-width detection is attempted when no delimited candidate passes the threshold. The detector looks for columns of consistent whitespace across the sampled lines and verifies that the implied slicing produces type-coherent columns.

Real deliveries include files with a single header row, files with multiple header, placeholder, and metadata lines prefixed with a comment character. The detector classifies each of the file beginning rows using a rule set: a row is considered a candidate header if its values are short strings, they match the column-type profile of the remaining rows, and the row does not match the dominant data pattern below it.

2.3 Schema validation and format generation

Each data source is registered in the MMS under a stable business key that identifies the expected data structure, including field names, data types, allowed lengths, and

other constraints. When a file is processed, the detector's output is compared with this registered structure. Mismatches in field count, incompatible data types, or over-length values are reported as validation errors; warnings are produced for structural drift that does not block loading.

Database format files [1] are generated per delivery rather than maintained by hand. The format file specifies the field terminator, row terminator, encoding parameters, and the mapping between source columns and target columns. Type conversions that are known to be fragile, for example, Finnish-locale date strings are loaded as strings into a staging column and converted to date type downstream using SQL.

3. Results

3.1 Evaluation dataset

The framework was evaluated on 100 administrative data. Fifty files were processed in the API-mode, i.e. against a registered schema with format-file generation enabled. Other fifty were processed in standalone-mode. Formats in the evaluation set include CSV with comma, semicolon, and dollar separators, fixed-width text, and Excel workbooks converted to delimited form. File sizes range from 5 KB to 95 GB.

3.2 Format detection accuracy and processing efficiency

Table 1 reports detection accuracy and processing-time summaries for each mode. Across the evaluation set, encoding detection achieved 100% accuracy and separator detection 100%. These results should be interpreted in the context of the evaluation dataset: all files were real production deliveries and used a constrained set of encodings. Field terminators were limited to [,;:$], and row terminators to LF or CRLF. Header detection is the most error prone phase, with failures concentrated in deliveries that include multi-row or placeholder-style headers. Ninety-eight percent of files overall were processed fully unattended on first go, meaning detection was correct and schema validation passed without human involvement. However, also these cases could usually be fixed by setting the metadata information according to the new incoming file. In standalone-mode, where there is no schema validation, "fully unattended" denotes the complementary condition that detection produced no flags for review and the table fully forms to the database.

Table 1 Detection accuracy and end-to-end processing time by mode (n = 100 files). Fully unattended = detection correct, and, in API-mode, schema validation passed without human review. † In standalone-mode, "fully unattended" means detection produced no review flags; schema validation is not performed.

Processing mode	Files	Header correct	Fully unattended	Mean (s)	Max (s)	Bulk import Mean (s)	Bulk Import Max (s)
API-mode	50	50/50 (100%)	50/50 (100%)	170	560	243	3080
Standalone-mode	50	48/50 (96%)	50/50 (100%) †	94	828	652	3649
Total	**100**	**98/100 (98%)**	**100/100 (100%)**	**125**	**828**	**448**	**3365**

Processing time is linear in file size, dominated by the bulk-load phase; the analysis phase is nearly constant for files above a few megabytes because detection operates on a bounded sample defined in the configuration. Before automation, a non-conforming delivery that previously "just worked" typically required at least 30 to 90 minutes of developer investigation to identify the change and update the format file. With API-mode, such deliveries are either processed without intervention or flagged with a specific diagnosis, reducing the reviewer action to under five minutes.

3.3 Audit trail and traceability

Every processing activity writes to a metadata table that records file properties, detection results, validation outcomes, row counts loaded, and any warnings or errors. The table provides the traceability required by the European Statistics Code of Practice [3] and by the Quality Assurance Framework of the European Statistical System [4]. In the evaluation, 100% of processing runs produced a complete audit record, and the records were sufficient to reconstruct the loading decision for every file.

4. Discussion

The evaluation shows that automated ingestion of heterogeneous administrative files is feasible in a production statistical environment and that the bottleneck of manual format-change investigation can be reduced. The combination of deterministic format detection, schema validation against an authoritative metadata registry, and dynamic format file generation removes the three most common sources of friction, encoding drift, separator drift, and header drift, in a single pipeline.

Automation does not eliminate the need for human oversight. The framework is designed to manage routine cases autonomously and to produce specific, actionable diagnoses for the cases it cannot manage, so that reviewer attention is directed where it is needed rather than spent on diagnosis.

4.1 Limitations

Couple limitations are worth stating explicitly. First, header detection might degrade on files with multi-row or placeholder-style headers. Second, huge files are not fully analysed, and the bulk import process can fail on unseen error cases. Third, Excel workbooks are currently managed by conversion to CSV before analysis, which is acceptable for the tabular sheets seen in production.

4.2 Impact on statistical production and applicability to other Nordic NSIs

The reduction from hours of investigation to under five minutes of reviewer confirmation for non-conforming files corresponds to many developer-hours saved per. The framework also eliminates silent load failures from encoding or locale drift and accelerates onboarding of new sources, since the marginal cost per source becomes schema registration in the MMS rather than format-file development.

The problems addressed by the framework are common to all Nordic *NSIs*. The architecture and algorithms are portable: the detection logic depends only on byte- and row-level properties of input files, and the schema-validation logic depends only on the MMS exposing a lookup by business key.

5. Conclusions

An automated data ingestion framework was developed and evaluated at Statistics Finland. The framework detects file format properties deterministically, validates detected structure against a central metadata registry when one is available, generates SQL Server bulk-load configurations per delivery, and records a complete processing audit trail; for sources that are not yet registered it can run in standalone-mode. In an evaluation on hundred administrative files, all of deliveries were processed fully unattended, and investigation time for non-conforming deliveries fell by more than an order of magnitude. Future work includes extending support to JSON/XML filetypes and, integration with workflow for end-to-end pipelines.

References

[1] Microsoft. BULK INSERT (Transact-SQL). SQL Server Documentation, 2024. https://learn.microsoft.com/en-us/sql/t-sql/statements/bulk-insert-transact-sql

[2] UNECE. Generic Statistical Business Process Model (GSBPM), Version 5.2. United Nations Economic Commission for Europe, 2025.

[3] Eurostat. European Statistics Code of Practice. Publications Office of the European Union, 2023.

[4] Eurostat. Quality Assurance Framework of the European Statistical System. Publications Office of the European Union, 2019.

[5] van den Burg, G. J. J., Nazabal, A., Sutton, C. Wrangling messy CSV files by detecting row and type patterns. Data Mining and Knowledge Discovery, 33(6), 1799–1820, 2019.

[6] Li, S., Momoi, K. A composite approach to language/encoding detection. Proceedings of the 19th Unicode Conference, 2001. (Mozilla universalchardet; Python port: chardet.)

[7] Open Knowledge Foundation. Frictionless Data - Data Package and Table Schema specifications. https://frictionlessdata.io, accessed 2026.

[8] Apache Software Foundation. Apache NiFi. https://nifi.apache.org, accessed 2025.

[9] Airbyte. Airbyte — Open-Source Data Integration Platform. https://airbyte.com, accessed 2025.