IFC-Bank of Italy Workshop on "Data Science in Central Banking: Applications and tools"

14-17 February 2022

Microdata utility – data loading with automated data parsing and data structure creation[1]

Marcus Jellinghaus,
BIS

[1] This contribution was prepared for the workshop. The views expressed are those of the authors and do not necessarily reflect the views of the Bank of Italy, the BIS, the IFC or the other central banks and institutions represented at the event.

Microdata utility

Data loading with automated data parsing and data structure creation

Marcus Jellinghaus, Bank for International Settlements

Abstract

Each year, the Bank for International Settlements (BIS) processes thousands of heterogeneous data files for economic research and analysis. This poses a challenge for the Information Technology team with the Monetary and Economic Department (MED IT team).

To address this challenge, the MED IT team has developed the 'microdata utility', facilitating the loading of millions of CSV and XML files into a SQL Server database with minimal configuration effort. The database structure is automatically generated based on the structure detected in the data files.

The BIS is using the microdata utility to process data purchased from commercial data providers, used for economic research, and the monitoring of financial markets.

The microdata utility will

- parse the specified data files,
- automatically detect the data structure within these files, and
- create tables in a database that match the structure of the data files.

By using this utility, the effort of setting up and regularly updating datasets can be significantly reduced, and technical resources can be used more efficiently.

Based on feedback, the BIS is open to sharing the utility with partners, and/or making the microdata utility available as open-source software via the BIS Open Tech Hub.

Keywords: Data loading, Automation, Database

JEL classification: C81, C88

1. Introduction

The microdata utility has been developed to load data from different commercial data providers into a relational database and offers a generic method for the loading of CSV files and XML files.

When processing CSV files, the utility will auto-detect the data types of each column and create the appropriate table structure within the database to fit the data. The utility can detect data type changes, new columns / structural changes in data files over time, and allow for the underlying database tables to be automatically altered. The utility allows users to load CSV files without a complex specification, (other than file names and table names).

When processing XML files, which may store the data in a more complex tree-like structure, the structure is automatically analysed and a relational data model is auto-generated.

The utility manages the status of each data file during the loading process and allows for the sequential or parallel processing of data. (To date, the BIS has used the utility to process millions of data files.) The Python- and Pandas-based utility includes options for loading data into SQL-Server using performant 'Bulk Insert'-operations and storing the data using columnstore indices[1]. This allows for fast data loading, optimised data storage, and performant query processing.

The microdata utility can process data files available on disk, or via external HTTP or FTP servers. Using a minimal specification, the utility can download data files on a defined update frequency, ensuring that databases are kept up to date.

The utility is already used to process more than 12 commercial datasets within the BIS production environment.

The remainder of this paper is organized as follows:

Section 2 describes the use case and the design objective. Section 3 introduces the data parsing and the structure creation for CSV files, section 4 provides additional information for XML files and their more complex data structures. Section 5 describes how the source files and the destination tables are configured, and section 6 introduces the status tracking for files and the mass processing of files. Section 7 gives an overview of different technical optimisations. Section 8 concludes with the current status and gives an outlook on possible next steps, including the possible sharing of the utility.

2. Microdata use case and design objective

What is microdata?

In MED IT, we define microdata as non-aggregated data at the entity level, eg on individual companies, instruments, investment funds, etc. The BIS is using microdata for economic research and to calculate aggregates for analysing financial markets.

[1] Columnstore indexes: Overview - SQL Server | Microsoft Docs

This data is purchased from commercial data providers. A dataset may include just a few files, in other cases, it can include more than one million files.

Also, the consumed disk storage space varies, from some megabytes to more than one terabyte.

Storing the data

The microdata is stored in a relational database, making it easily accessible for our end-users. So far, we have worked with Microsoft SQL-Server database servers.

The goal is that the microdata utility loads all data provided by the vendor. While the full data collection may not be required for a current research project, other parts of the data may be required for a future research project, hence the goal to ensure all data is successfully loaded to the database.

Depending on the structure of the data, the data delivered by a provider could be stored in a single table or may need to be spread across several tables within the database.

Data quality and data quality checks

The data is being purchased from commercial data providers. As a general rule, we assume that the data is provided in a logical structure, and the utility will load the data as provided.

The utility performs structural data checks on the data files but does not perform data quality/coherence/validation checks. We assume that data quality issues will be identified by research analysts as they develop research papers, etc. If the data would be used for trading decisions, or to look in more detail at individual institutions, a more rigorous data quality assessment might be required.

Why a generic utility?

In the past, we created one custom application for each dataset. While these applications often had a limited complexity, they were all different and offered different features depending on the dataset to be processed. We have now created a generic data loading utility that allows us to process any dataset.

The microdata utility has been developed in an agile and iterative manner, with each release, adding new features required to process the data files in our collection. Every new dataset provides new challenges, some features are simple to implement, for example, the usual separator in a CSV file is a comma, and we had to make the separator configurable. Others, however, are more complex, for example, the need to implement a new parser for XML files.

The migration away from the custom applications to the microdata utility has challenges. For each dataset, specific features and workflows were implemented, which work sometimes slightly differently in the microdata utility, these need to be reviewed on a case-by-case basis.

Looking at the big picture, we benefit from the generic utility, which allows us to load new and large datasets very quickly. The overall configuration and operations only take a few minutes to define, and the processing of large datasets, eg with more

than 700,000 XML files, are parsed and loaded in less than 100 hours, also based on the degree of data complexity and available resources for parallel data loadings.

Why develop a utility, and not use an existing solution?

While many ETL tools allow analysing the structure of source files, we are not aware of other tools that automatically create and adjust the structure of a database following the structure of the input data files. On the contrary, the analysis often happens at a design stage, and the data structure is then fixed for the runtime. Also, the automatic parsing and translation of XML files to tables (as described in section 3) is not something that we have encountered before. Last but not least, the mass file processing with file status tracking allows for parallelisation, and different technical optimisations allow for efficient usage of resources (see sections 6 and 7). With these advantages in mind, we decided to implement our utility instead of using existing software.

Basic functionality (like reading CSV files, HTTP & FTP downloads, working with data frames, connecting to SQL-Server) was not implemented by us. Instead, we benefit from powerful Python libraries, including Pandas and SQLAlchemy.

Goal: Provide a generic, efficient data processing utility requiring minimal configuration effort

The utility has the goal to minimise the configuration effort, eg we do not want to hardcode the specific column names or their data types. The utility parses the files and detects the columns, their names, and data types. For XML files, the table structure follows the received data structure. This enhances flexibility, reduces the configuration effort, and provides, based on our experience, very high quality of the data structure.

Since we need to process millions of files, the utility needs to be performant and efficient. The files are being received from external providers; their technical quality cannot be controlled. Therefore, the utility needs to be robust and needs to flag possible issues as they are encountered.

3. Processing of CSV files: Data parsing and structure creation, dynamic adjustment of tables

Basic process for parsing data, creating tables and loading the data into a database

The microdata utility can parse CSV files automatically for column names and column types (ie str, float, datetime, etc.). When reading CSV files, the Pandas library automatically detects the different columns. We have added a feature to detect the data type of each column, eg whether it is text, dates, or numbers.

The microdata utility creates tables in a database automatically with corresponding column names and types detected when parsing the data.

After creating the tables in the database, the utility loads the data into the database.

Dynamic adjustment of tables

Tables are being automatically adjusted in SQL-Server if additional files have a different data structure:

- **Creation of additional columns**: Let's say an additional data submission includes additional columns – eg because the data provider added further information over time. In that case, the utility will automatically create a new column.
- **Extension of string columns (to store wider strings)**: Let's say a column in the database has a certain width, eg to store a company name with 25 characters. In case a new file has a record with a company name of 30 characters, the database column will be automatically adjusted.
- **Adjustment of data types**: Let's say so far, a column only included integer numbers. If an additional data file includes a number that is not an integer but includes decimals, the data type would be automatically adjusted to store the complete number. Eg if a first file contains a "1", and a second file a "1.1", the column will be initially created as an integer column and will then be adjusted to column type "floating number".

The table definition is expanding. If an additional data file has fewer columns, the NULL values will be inserted for these missing columns.

4. Processing of XML files: Translation in relational table structures

The microdata utility also allows for the processing and storage of XML data files. XML files typically include more complex data structures above CSV data files, eg a tree of data items.

Translation to one table

Let's take a simple example. Imagine an XML file that includes data on several students and for each student several attributes like name, email, street, and city. This can be converted into a table with the different students as records, and the different attributes as columns.

Box 1

Example: Simple translation of XML file into one table

A simple XML data example can eg look like this:

```
<data>
    <student>
        <name>Alice</name>
        <email>alice@mail.com</email>
    </student>
    <student>
        <name>Bob</name>
        <email>bob@mail.com</email>
    </student>
</data>
```

The resulting table looks like this:

name	**email**
Alice	alice@mail.com
Bob	bob@mail.com

Translation to several tables

Let's take a more complex example: Again, we have students with attribute email. Now, we also have zero, one, or several subjects for each student and also the grades. In that case, we would create a second table "subject" and store the course name and the grade there. This table would also have a relational link to the table student.

The parsing of XML files has been used to process several datasets from different data providers. The resulting table structure and data model within the database provide a logical data structure for our analysts.

Further enhancements

The processing of XML files can be complex, we have identified further enhancements to the microdata utility which we believe will add business value:

- Trees with higher depth are being handled, more tables are being created. In some cases, the XML structure can be quite deep, therefore several tables are being created.
- Details that do not require extra tables get moved up into the main table.
- Unnecessary middle layers without extra data get removed.

  Data relations are being tracked. In the database, these are being stored as primary keys and foreign keys. Relations between tables are also being tracked and are stored in a specific table. Further functionality allows presenting the relations as a dynamic network graph.

Box 2

Example: Translation of XML file into several tables

XML data example with data on different levels:

```
<data>
    <student>
        <name>Alice</name>
        <email>alice@mail.com</email>
    </student>
    <student>
        <name>Bob</name>
        <email>bob@mail.com</email>
        <subjects>
            <subject name = "Math">
                <grade>C</grade>
            </subject>
            <subject name = "History">
                <grade>B</grade>
            </subject>
        </subjects>
    </student>
</data>
```

The resulting tables 'student' and 'subject 'look like this:

student_uuid	name	Email
5b765840	Alice	alice@mail.com
a81d2b18	Bob	bob@mail.com

name	grade	student_uuid
Math	C	a81d2b18
History	B	a81d2b18

The field student_uuid is a "universally unique identifier". For each student record, one value has been generated. This identifier is used also in table 'subject' and allows to link records between both tables.

5. Dataset specification via Excel

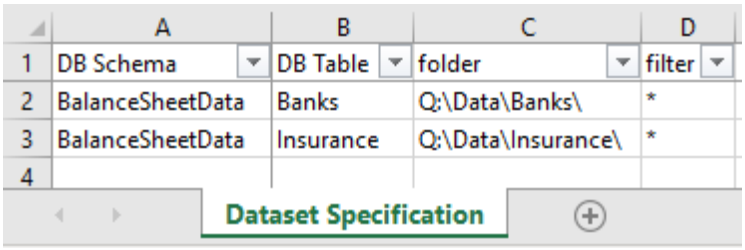
The microdata utility requires user input to identify which files to load, and into which table these files need to be stored; to do this an Excel template is used to provide the dataset specification. Only a few columns are required to provide basic information like the path and the filename of the source files as well as the target schema and table name. Each row can be used separately, this allows users to specify several source folders or destination tables.

Where a user wants to load many files into one table, the specification supports wildcards like asterisks (*) and question marks (?). It is also possible to specify whether subfolders should be searched for files.

Tables can be grouped in database schemas. This allows users to separate providers and creates a clean organisation of the data tables in the database.

Box 3

Example: Dataset specification



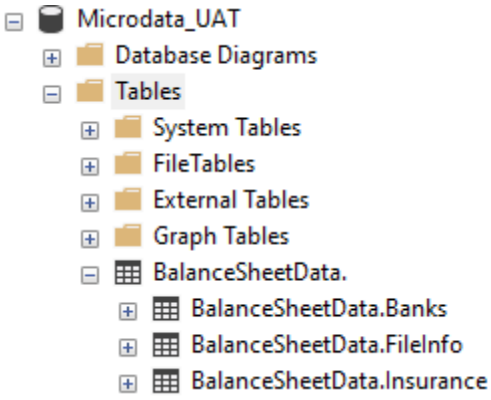
	A	B	C	D
1	DB Schema	DB Table	folder	filter
2	BalanceSheetData	Banks	Q:\Data\Banks\	*
3	BalanceSheetData	Insurance	Q:\Data\Insurance\	*
4				

Dataset Specification

In this example, the input files are defined in columns C and D. Column C defines from which folder the files should be read, and column D which files should be read.

Columns A and B define where the data should be written to. In Microsoft SQL-Server, the database schema allows to group tables.

The microdata utility automatically creates tables with the name as defined in column B; the column structure is derived from the data files.



Besides the two tables containing the data (in this example: Banks and Insurance), the utility also creates a table FileInfo which contains the names of all data files, and their loading status.

Specifications for updating a dataset

As previously mentioned, the microdata utility can download new data via FTP or HTTP, so that datasets can be kept up to date.

The microdata utility needs to know when to download a file. The dataset specification allows to define different parameters like update frequency (daily, business daily, monthly), point in period (eg end of month) as well as reporting gap and reporting frequency (ie data published after 1 day, or data published after 2 business days).

The utility requires the name of the files to be downloaded from an FTP server. When downloading files from an HTTP server, the name of the new data files must be

specified. For example, users would configure the system with the file naming schema "MyData_*.csv" and the date format "yyyyMMdd". If the date is the 22. September 2021, the file name "MyData_20210922.csv" is calculated.

Wildcards are also supported. This allows downloading several files for a specific date from an FTP server.

Zip files can also be specified, automatic unzipping is supported.

6. Mass file processing and status tracking

The microdata utility allows for the parallel processing of files, which dramatically reduces the time needed to load millions of data files. The status of each file is being tracked within the database. A database table "FileInfo" contains the path and filenames of all files, their processing status, and related information.

Several or even many agents can be used to load data and store it in the database. This allows to speed up the data loading significantly. A potential bottleneck is the database server, which depending on CPU and memory allocation, may only be able to handle a certain amount of data insertions concurrently.

Process for loading data

The standard process for loading a file consists of the following steps: Read Excel specification file, register file, analyse file, load file, mark file as successfully loaded.

In the first step, the dataset specification is read. This specification is used to find files. All found files are being registered in the FileInfo table in the database. In a further step, each file is being analysed for some high-level descriptive information (including the dates covered in the file). In the next step, each file is being read into memory and the data is being parsed. The data structure is being compared to the database. Where required, the database tables are being adjusted. Finally, the data is being inserted into the database table. In the case of very large CSV files, the files are processed in several steps. Once the data has been loaded successfully, the file gets the status "File loaded successfully".

Special cases are also supported, files can be ignored and deleted from the database based on a separate list. Different exceptions when loading files are being tracked inside the table FileInfo, including issues when reading the data, or when the data file is empty. Unexpected issues get logged.

A database view (Status View) allows users to have an overview of all files processed across different schemas. This Status View is also used to create a monitoring dashboard. At the BIS, we use a Tableau dashboard to see the summary statistics of the file loading status by dataset, and a second overview to see the file loading status of the files containing the data of the last 65 days.

7. Technical optimisations: SQL-Utility, Bulk Insert, ColumnStore, and Index Creation

The microdata utility is based on Python and uses libraries like Pandas, SQLAlchemy, and others to store data in SQL-Server.

To enhance the performance of the data loading processes, and to reduce the required resources, several technical optimisations have been implemented, including:

- A small SQL utility wraps different SQL commands into python functions.
- In the case of larger amounts of data, the "bulk insert[2]" method is used to transfer the data from the client running the microdata utility to the database server. This method speeds up the data inserts into the database.
- To reduce the amount of required storage, columnstore indices are being used. This special table storage method reduces data retrieval time and stores the data compressed on the storage volume. This is transparent to end users / data consumers, and SQL queries for querying the data do not need to be changed.
- Certain database indices are automatically created to increase query performance.

In summary, the microdata utility allows users to load large amounts of data quickly and efficiently.

8. Production status and considerations to share the utility

The microdata utility is used in production since 2020. It has already processed more than 12 datasets. In total, more than two million files have been loaded. All in all, more than 500 GB of compressed (!) data is being stored in the database, including data of various providers:

- different datasets provided by CryptoCompare,
- Fitch fundamental data on banks, data on issues and issuers,
- IHS Markit Iboxx data including constituents, indices, and mappings data
- IHS Markit Credit Indices
- Informa iMoneyNet data on mutual funds,
- iVolatility data
- Lipper Fund data
- Moody's CreditEdge
- MorningStar
- S&P Trucost data

[2] BULK INSERT (Transact-SQL) - SQL Server | Microsoft Docs

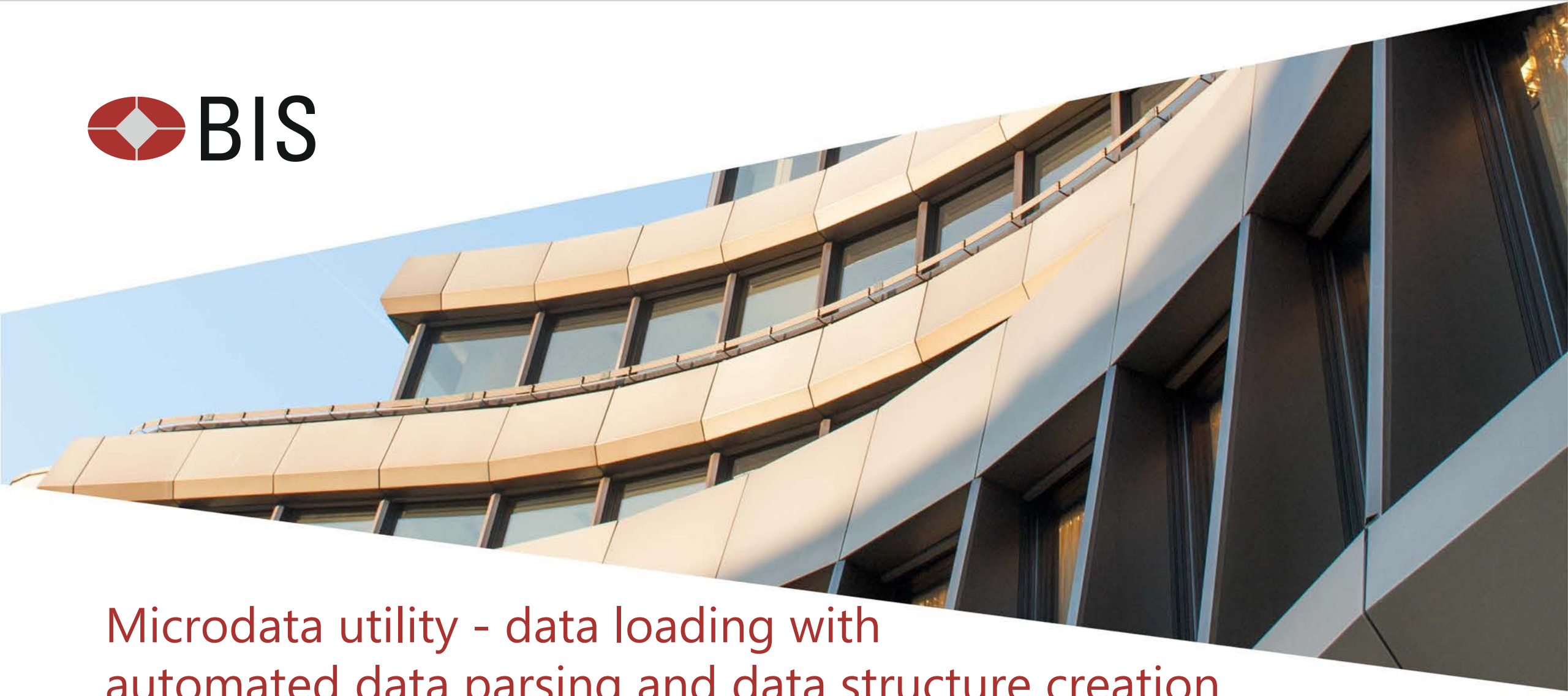
As mentioned above, only a small dataset specification is required to process these datasets quickly and efficiently. There is no need to define detailed table structures, these technical details are automatically generated based on the data. Based on XML files, sets of related database tables might be created. The utility can process millions of files, and dynamically adapt database tables based on developing data structures.

The microdata utility is continuously enhanced as we process additional datasets and migrate legacy dataset processing to use the utility.

Additional features are planned, eg to support hybrid usage on Linux / container stack and on windows servers, and to support a dimensional data model with slowly changing dimensions. Efficiency improvements related to the direct loading of data files from compressed zip files, and to storing GUID data more efficiently, are also envisaged.

Based on feedback, the BIS is open to sharing the utility with partners, and/or making the microdata utility available as open-source software via the BIS Open Tech Hub.

Please let us know whether you would be interested in collaborating in this space. Queries should be directed to Marcus.Jellinghaus@BIS.Org.



Microdata utility - data loading with automated data parsing and data structure creation

IFC and Bank of Italy Workshop on “Data Science in Central Banking”, part 2, 14-17 Feb 2022

Marcus Jellinghaus, Principal Data Scientist, MED-IT, BIS

Microdata utility - a generic data loading utility

- What is **Microdata**?

- Why a **generic** utility?

- Next slides:
  - Dataset parsing and database table creation for CSV and XML files
  - Dataset specification
  - Some technical features
  - Current status and possible next steps

Data parsing and structure creation, data loading, dynamic adjustments

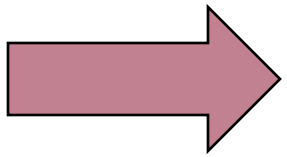
- Data parsing
- Data structure creation
- Data loading
- Dynamic adjustments

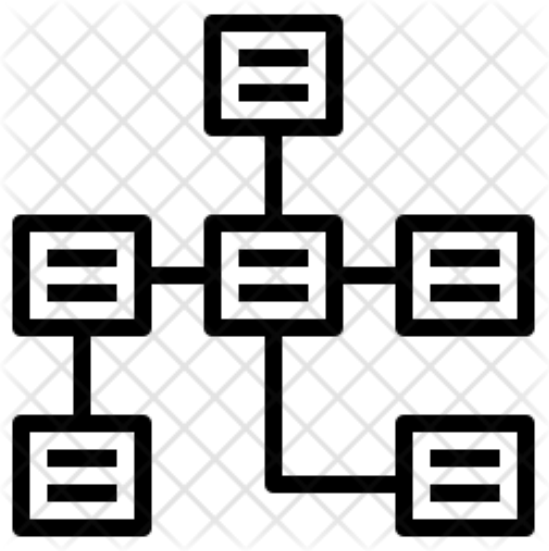
File parsing

Table creation

Data loading

Processing of XML files: Translation in relational table structures

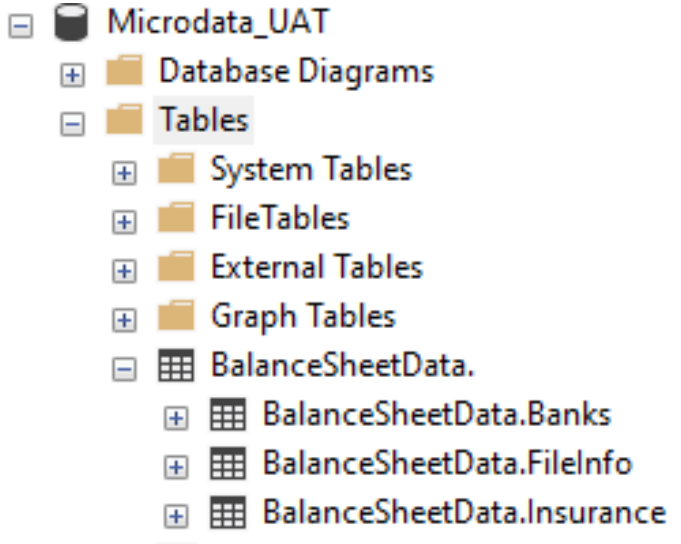




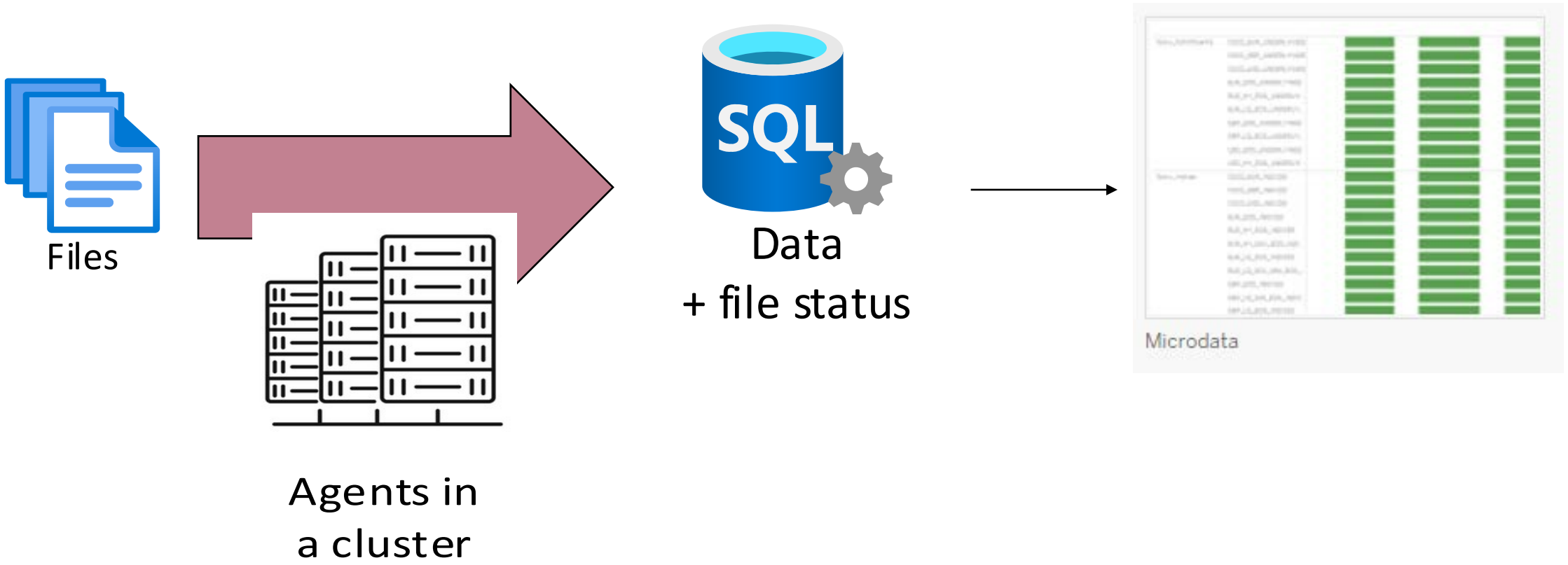
Dataset specification with a small excel sheet

	A	B	C	D
1	DB Schema	DB Table	folder	filter
2	BalanceSheetData	Banks	Q:\Data\Banks\	*
3	BalanceSheetData	Insurance	Q:\Data\Insurance\	*
4				

Dataset Specification



Mass file processing and status tracking



Technical platform

Bulk Inserts

Columnstore

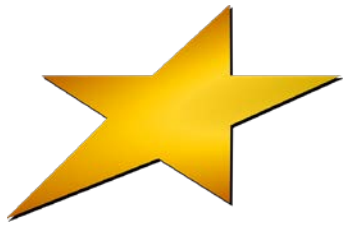
Python API for DB

pandas

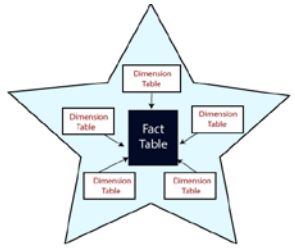
python™

➔ The tool allows to load large amounts of data very fast, and reduces the amount of required resources

Current status



New datasets



Star Schema

Used in production since 2020

- 25 datasets
- 1.4 million files
- 400 GB of compressed data

open source

Contact: Marcus.Jellinghaus@BIS.org