# Design Document: Event-Driven Data Processing Pipeline

Version 2.0

Date: 2025-12-03

## 1. Core Philosophy: Decoupling and State Management

The fundamental principle of this design is to **decouple** the data preparation process from the core data processing task. These two modules operate independently, communicating only through a stateful **Manifest File**. This approach ensures that the core processing engine only triggers when all dependencies are verifiably ready, leading to a robust, reliable, and scalable system.

## 2. Directory Structure

A well-defined directory structure is established to manage the flow of data, scripts, and artifacts through the pipeline. Each directory has a specific, singular purpose.

```
C:\DATA_PIPELINE\
|
|--- 1_input\                <-- Raw source data arrives here.
|
|--- 2_preprocessing\       <-- Intermediate outputs (Partners, Units, For ex).
|
|--- 3_processing_hotfolder\ <-- The 'mailbox'; the Watcher monitors this folder.
|
|--- 4_archive\              <-- Successfully processed batches are archived here.
|
|--- 5_error\               <-- Failed batches are moved here for manual inspectio
n.
|
|--- 6_logs\                 <-- Stores all operational logs.
|
|--- 7_scripts\              <-- Location of all Python scripts for the pipeline.
```

## 3. The Manifest File (_MANIFEST.json)

This JSON file is the heart of the trigger mechanism. It is more than a simple flag; it is a rich metadata document that describes a complete, ready-to-process batch. It provides all necessary information to the processing engine, ensuring deterministic and repeatable runs.

```json
{
    "batch_id": "20251029153000",
    "creation_timestamp": "2025-10-29T15:30:00Z",
    "status": "READY_FOR_PROCESSING",
    "files": {
        "partners": "C:\...\Partner_Data_REPAIRED.csv",
        "units": "C:\...\merged_units_FINAL.csv",
        "forex": "C:\...\forex_rates_2025-10.csv"
    }
}
```

```
"raw_data": [
    "C:\...\CN_data.csv",
    "C:\...\US_data.csv"
]
```

## 4. Workflow and Script Responsibilities

The pipeline operates as a sequential, four-step process, orchestrated by three key scripts.

### *Step 1: Prepare Batch (`1_prepare_batch.py`)*

This is the manually initiated first step. Its sole responsibility is to run all prerequisite tasks (e.g., merging partners, units), gather all file paths, generate the `_MANIFEST.json` file, and place it into the `3_processing_hotfolder` to signal readiness.

### *Step 2: The Watcher (`2_watcher.py`)*

This is a persistent, background process. It continuously monitors the hot folder. Upon detecting a manifest file, it immediately 'locks' it by renaming it (e.g., to `_PROCESSING.json`) and then invokes the core processing script, passing the manifest file's path as an argument. This lock-and-execute mechanism prevents race conditions.

### *Step 3: Core Processing (`3_process_raw_to_meta.py`)*

This is the 'heavy-lifting' script. It reads the manifest file to understand its task, loads all specified data, and performs the complex transformation from 'rawdata' to 'metadata'. It is built with robust error handling (`try...except`) to manage potential failures gracefully.

### *Step 4: Finalization and Archiving*

This is the final stage, executed within the core processing script. Based on the outcome (success or failure), it updates the manifest's status and moves all associated files (raw data, preprocessed files, and the manifest itself) into the appropriate `4_archive` or `5_error` directory. This ensures the pipeline is clean and ready for the next batch.

## 5. Asynchronous Reference Data Acquisition (Exchange Rates)

Certain reference data, like monthly exchange rates, has its own release cadence and is not part of a standard batch. This requires a decoupled, asynchronous acquisition workflow that runs in parallel to the main pipeline.

In our data pipeline design, only exchange rates data changes frequently each month, requiring separate design for it. The IMF releases the previous month's data monthly, but based on previous testing, no data is often released at the beginning of the month. Therefore, when I ran the script on December 2, 2025, no data for November was retrieved because it hadn't been released yet. My idea is to have the script retrieve the previous month's exchange rate data periodically for a few days each month based on the system date, automatically calling the API to retrieve the data once the designated date arrives, ensuring timely and comprehensive automated data retrieval.

The script that automatically retrieves exchange rates is named "currency_last_month_data.py" and is stored in the "Pipline '(Trigger)' design" folder and the "Exchange_Rates" folder in OneDrive. The two files contain the same content.

### *5.1. Data Flow Architecture*

The architecture consists of two independent but intersecting workflows:

```
    Time-Driven Workflow (Background Task):
    [Schedule: 15th/22nd/28th of Month] -> [Run `currency_last_month_data.py`] -> 
[Check if file exists] -> [Fetch from API if needed] -> [Write `exchange_rates_YYY
Y_MM.csv` to `C:\DATA_PIPELINE\reference\`]

    Event-Driven Workflow (Main Pipeline):
    [`_MANIFEST.json` arrives] -> [Watcher locks & executes] -> [Core Processor ru
ns] -> [Read `exchange_rates_YYYY_MM.csv` from `C:\DATA_PIPELINE\reference\`] -> [
Apply currency conversion] -> [Final Output]
```

### *5.2. The Acquisition Step (Asynchronous)*

The `currency_last_month_data.py` script runs on a time-based schedule (e.g., via Windows Task Scheduler). Its sole responsibility is to ensure the latest monthly exchange rate CSV is present in a predefined directory. It is idempotent: if the file for the target month already exists, it does nothing.

### *5.3. The Consumption Step (Synchronous)*

This is the integration point. When the core processing script requires exchange rates, it synchronously reads the required file from the well-known reference directory. Because the acquisition workflow runs reliably in the background, the main script can assume the necessary file is available.