

M3 Data Migration Platform (v2.0)

Comprehensive User Manual & Technical Reference

1. Introduction

The M3 Data Migration Platform is an enterprise-grade ETL (Extract, Transform, Load) solution designed to automate the migration of legacy Movex data into Infor M3. Unlike standard Excel mapping sheets, this tool provides a robust Rules Engine, automated "Surgical" extraction for delta loads, and AI-assisted reverse engineering of legacy data patterns.

Key Features

- **Intelligent Rules Engine:** Supports Direct mapping, Constants, Lookup Tables, and Python Scripting.
- **Surgical Extraction:** "Load by ID" feature to extract specific business objects (e.g., Customer 1001) across multiple source files and merge them into a single load file.
- **Auto-Detect ("Magic Drop"):** Identifies unknown legacy files based on column signatures and automatically routes them to the correct migration pipeline.
- **Audit & Versioning:** Tracks every rule change by user and timestamp, with snapshot/restore capabilities.
- **Batch Processing:** Headless mode for processing high volumes of files during cutovers.

2. Installation & Setup

Prerequisites

- **OS:** Windows 10/11 (Required for Excel interaction).
- **Python:** Version 3.10 or higher.

Dependencies

Run the following command to install the required libraries:

```
pip install pandas openpyxl xlswriter colorama customtkinter scikit-learn
```

Directory Structure

Ensure your project folder looks like this:

- `/config` - Stores CSV maps and global settings.
- `/config/rules` - Stores the Excel-based Rule Configurations (`MMS200MI.xlsx`).

- `/config/sdt_templates` - Stores the blank M3 SDT Excel files.
- `/raw_data` - Default folder for legacy source files.
- `/output` - Destination for generated load files.
- `/modules` - Application source code.

Launching the Application

- **GUI Mode:** Run `python gui.py` (Recommended for daily use).
- **CLI Mode:** Run `python main.py` (Text-based menu).

3. The Rules Engine (Deep Dive)





Located in the Rules & Admin Hub.

The Rules Engine is the core of the platform. It determines how a value from the legacy system is transformed before being written to the M3 SDT template.

A. The Rule Editor Interface

The editor is split into two panels:

1. Field List (Left):

- **Search Bar:** Filter fields by Target Name or Description.
- **Scope Filter:** Toggle between `ALL` rules and specific Division scopes (e.g., `DIV_US`).
- **Status Indicators:**
 -  **Red:** `TODO` (Rule needs definition).
 -  **Blue:** `MAP` (Uses external lookup).
 -  **White:** Active/Valid Rule.
 -  **Gray:** `IGNORE` (Field will be skipped).

2. Rule Form (Right):

- Displays metadata (Data Type, Length) imported from the MCO.
- Allows editing the **Rule Type**, **Source Field**, and **Value/Logic**.

B. Rule Types & Logic

Rule Type	Description	Usage Example
DIRECT	Copies data 1-to-1 from the Legacy file.	Source: <code>MMITNO</code> → Target: <code>ITNO</code> .
CONST	Hardcodes a specific value for every row.	Value: <code>20</code> → Target: <code>STAT</code> .
MAP	Translates values using an external Excel/CSV file.	Value: <code>`maps/PaymentTerms.xlsx</code>

PYTHON	Executes custom Python logic (see below).	Value: <i>[Code Block]</i>
IGNORE	Explicitly skips the field.	-
TODO	Marker for fields requiring analysis.	-

C. Context & Scoping (Overrides)

The platform supports **Global vs. Local** rules.

- **GLOBAL:** The default rule applied to all records.
- **Scope Override:** A specific rule applied only when the user selects a specific Scope (e.g., `DIV_US`) during migration.

To Create an Override:

1. Select a rule in the Editor.
2. Click the `+ Override` button.
3. Select the target Business Unit from the popup.
4. The editor creates a new rule entry. Modify the logic and save.

D. Python Scripting Guide

When `RULE_TYPE` is set to `PYTHON`, the "Value / Logic" box accepts standard Python code.

- **Input Variables:**
 - `source` : The value of the column defined in "Source Field".
 - `row` : The entire row (pandas Series), allowing access to *other* columns.
- **Return Value:** The script must `return` a string or number.

Recipes:

1. Simple Case Statement (Status Translation)

```
val = str(source).strip()
if val == '10': return '20'
elif val == '90': return '99'
else: return '10' # Default
```

2. Cross-Column Conditional (If Facility = 300...)

```
# Use row.get() to safely access other columns
faci = str(row.get('FACI') or row.get('MMFACI') or '').strip()

if faci == '300':
```

```

    if str(source) == 'F3': return '03'
    if str(source) == 'F7': return '07'

return "" # Return blank if conditions not met

```

3. Date Formatting (YYYYMMDD to YYMMDD)

```

# Input might be 20231231 or 2023-12-31
val = str(source).replace('-', '').replace('/', '').strip()

if len(val) == 8:
    return val[2:] # Returns 231231
return val

```

4. Operation Modules (GUI Tabs)

Module 1: Run Migration

- **Standard:** The primary tool for bulk data loads. Selects a Rule Config and a Source File.
- **Auto-Detect:** Scans a file's headers against the MCO library to guess the API. Useful when you have a file named `Book1.xlsx` and don't know what it contains.
- **Load by ID (Surgical):**
 1. Select an Object (e.g., `ITEM`).
 2. Paste a list of IDs (`1001` , `1002`).
 3. The tool scans **all** configured source files (Item Master, Balances, Facilities) defined in `source_map.csv` .
 4. It extracts only those IDs and stitches them into a **single multi-tab SDT file**.
- **Batch:** Runs a sequence of jobs defined in an Excel manifest.

Module 2: Configuration

- **Import MCO:** Bootstraps new Rule Sets from functional specification documents.
 - *Force Overwrite:* Check this to reset rules. Leave unchecked to "Smart Merge" (keep your manual Python scripts while updating field descriptions).
- **Reverse Engineer:** An AI tool that compares a Legacy File against a filled M3 Template. It detects patterns (e.g., "Field X always equals Field Y" or "Field Z is always '20'") and generates a Draft Rule Config.
- **Map Editor:** A built-in grid to edit the system's CSV configuration files (`migration_map` , `source_map` , etc.) without opening Excel.

Module 3: Utilities

- **Copy Sheet:** Helper to copy data between Excel tabs, filtering out rows flagged as "NOK" (Not OK) by the M3 API.
- **Merge Files:** Combines multiple Excel files into one Master file. It uses smart deduplication (only appends rows that don't already exist in the Master).

5. Configuration Reference

To fully utilize the platform, you must maintain the CSV maps in the `/config` folder.

1. `migration_map.csv`

Links the functional object to the technical API.

- **MCO_SHEET:** The name of the tab in your Spec file (e.g., "Item Master").
- **API_NAME:** The name of your Rule File (e.g., `MMS200MI`).
- **SDT_TEMPLATE:** The blank template file (e.g., `MMS200MI_API.xlsx`).
- **TRANSACTION_SHEET:** Comma-separated list of tabs to populate (e.g., `AddItmBasic,AddItmWhs`).

2. `source_map.csv`

Used by the Surgical/Delta Loader to find raw data.

- **MCO_SHEET:** Matches the entry in `migration_map.csv` .
- **SOURCE_FILE:** Relative path to the raw data (e.g., `raw_data/MITMAS.xlsx`).
- **JOIN_KEY:** The column used to filter by ID (e.g., `MMITNO`).

3. `surgical_def.csv`

Groups multiple sheets into a single business object.

- **OBJECT_TYPE:** The dropdown category (e.g., `ITEM`).
- **MCO_SHEET:** The specific sheet to include.
- *Example:* An `ITEM` object might consist of 3 rows: `Item Master` , `Item Warehouse` , and `Item Facility` .

4. `business_units.csv`

Defines the valid scopes available in the Rules Editor dropdown.

- **UNIT:** The code (e.g., `DIV_US`).
- **DESCRIPTION:** Display name.

6. Troubleshooting & FAQ

Q: I saved a rule, but the History tab is blank. **A:** This usually happens if the Excel file was locked or if the `_Audit_Log` sheet was corrupted.

1. Go to Rules & Admin -> Tools.
2. Click Commit All Excel Edits.
3. If that fails, use Hard Reset System (Warning: Clears all history).

Q: The Surgical Extractor returns "No tasks generated". A: Check your `source_map.csv` .

1. Does the file path exist?
2. Does the `JOIN_KEY` column exist in that file? (Note: The tool handles case-sensitivity, but the column name must be spelled correctly).

Q: My Python rule is crashing. A: The system wraps all Python rules in a `try/except` block. If your code fails (e.g., dividing by zero), the tool catches the error and returns the original source value as a string. Check the **System Log** console in the GUI for specific error messages (e.g., `[RULE ERROR] STAT: division by zero`).