Scope Conversion Lifestance – Documentation

# Workflow Overview

1. Data Extraction: The main script (\_main\_1.py) reads the input Excel file and uses various helper scripts to extract fields like names, NPIs, specialties, etc.
2. Template Population: Extracted data is mapped to a template Excel file, creating a new output file with the required structure.
3. Dropdowns and Formulas: The script adds dropdowns and formulas for data validation and reference, using both built-in logic and helper scripts.
4. Sheet Management: Additional sheets (like ValidationAndReference and Location) are copied or generated, and further dropdowns/formulas are added.
5. Finalization: The output file is saved and opened for the user, ready for review or further editing.

# Main Python Files

## \_main\_1.py

Purpose:

This is the central orchestrator of the workflow. It coordinates the entire process, from reading the input Excel file to producing the final output. The script calls various helper modules to extract, clean, and transform data, then maps this data into a template structure. It also manages the addition of dropdowns, formulas, and the integration of auxiliary sheets, ensuring the output is ready for downstream use and validation.

Key Code Snippet:

# Extract name and gender data using Name.py  
extracted\_rows = extract\_name\_gender(input\_file)  
# Extract NPI data using Npi.py  
npi\_list = extract\_npi(input\_file)  
# ... (other extractors)  
# Call Location.py to generate the Location sheet  
subprocess.run(['python', 'Location.py'], check=True)  
# Add dropdowns and formulas  
apply\_provider\_dropdowns(output\_file, dropdown\_specs)  
apply\_provider\_formulas(output\_file, formula\_specs)

## Location.py

Purpose:

This script is responsible for generating the Location sheet in the output Excel file. It copies relevant sheets from the template, standardizes and cleans address data, and ensures that each location is properly categorized (e.g., splitting 'Both' into 'Virtual' and 'In Person'). The script also manages the transfer of zip codes and other location-specific fields, ensuring consistency and completeness in the final output.

Key Code Snippet:

# Duplicate rows for 'Both' location types  
for i, row in reversed(rows\_to\_duplicate):  
 ws\_location.delete\_rows(i)  
 # Insert two new rows: one with 'Virtual', one with 'In Person'  
 new\_row\_virtual = list(row)  
 new\_row\_virtual[location\_type\_idx] = 'Virtual'  
 new\_row\_inperson = list(row)  
 new\_row\_inperson[location\_type\_idx] = 'In Person'  
 ws\_location.insert\_rows(i)  
 for col\_idx, value in enumerate(new\_row\_inperson, start=1):  
 ws\_location.cell(row=i, column=col\_idx, value=value)  
 ws\_location.insert\_rows(i)  
 for col\_idx, value in enumerate(new\_row\_virtual, start=1):  
 ws\_location.cell(row=i, column=col\_idx, value=value)

## Name.py

Purpose:

Handles the extraction of provider names and gender from the input Excel file. It ensures that the gender field is standardized (e.g., mapping 'Prefer not to say' to 'Not Applicable') and that all relevant name fields are captured accurately. This module is crucial for maintaining data integrity and consistency in the provider records.

Key Code Snippet:

def extract\_name\_gender(input\_file):  
 # ...  
 for row in ws\_in.iter\_rows(min\_row=2, values\_only=True):  
 extracted = {col: row[input\_indices[col]] for col in ['First Name', 'Last Name', 'Gender']}  
 if extracted['Gender'] == 'Prefer not to say':  
 extracted['Gender'] = 'Not Applicable'  
 extracted\_rows.append(extracted)  
 return extracted\_rows

## Npi.py

Purpose:

Extracts National Provider Identifier (NPI) numbers from the input file. This script ensures that each provider’s unique NPI is accurately retrieved and mapped, which is essential for provider identification and compliance with healthcare data standards.

Key Code Snippet:

def extract\_npi(input\_file):  
 # ...  
 for row in ws\_in.iter\_rows(min\_row=2, values\_only=True):  
 npi\_list.append(row[npi\_idx])  
 return npi\_list

## Headshot.py

Purpose:

Responsible for extracting headshot URLs from the input data. This allows the output file to include direct links to provider photos, which can be used for display in downstream systems or directories.

Key Code Snippet:

def extract\_headshot(input\_file):  
 # ...  
 for row in ws\_in.iter\_rows(min\_row=2, values\_only=True):  
 headshot\_list.append(row[headshot\_idx])  
 return headshot\_list

## professional\_suffix.py

Purpose:

Extracts professional suffixes (such as MD, PhD, etc.) from the input and applies them to the output file. It also manages the addition of dropdowns for these suffixes, ensuring users can select from standardized options when editing the output.

Key Code Snippet:

def extract\_professional\_suffix(input\_file):  
 # ... (logic to extract suffixes)  
# Adds dropdowns for professional suffixes  
add\_professional\_suffix\_dropdowns(output\_file)

## Specialty.py

Purpose:

Extracts provider specialty information from the input file and manages the application of specialty dropdowns in the output. This ensures that specialty data is both accurate and validated against a reference list, supporting downstream reporting and analytics.

Key Code Snippet:

def extract\_specialty(input\_file):  
 # ...  
 for row in ws\_in.iter\_rows(min\_row=2, values\_only=True):  
 specialty\_list.append(row[specialty\_idx])  
 return specialty\_list

## PatientsAccepted.py

Purpose:

Extracts information about the types of patients accepted by each provider (e.g., Adult, Pediatric, Both) and sets up the corresponding dropdowns in the output file. This helps ensure that patient acceptance data is standardized and easy to update.

Key Code Snippet:

def extract\_patients\_accepted(input\_file):  
 # ... (extract logic)  
set\_patients\_accepted\_dropdown(output\_file)

## Education.py

Purpose:

Handles the extraction of education and school information for each provider. This module ensures that educational backgrounds are accurately captured and mapped, supporting credentialing and provider profiling.

Key Code Snippet:

def extract\_education(input\_file):  
 # ... (extract logic)  
 return education\_list

## Professional\_statement.py

Purpose:

Extracts provider bios or professional statements from the input file. This information is important for provider directories and public profiles, giving context about each provider’s background and philosophy.

Key Code Snippet:

def extract\_professional\_statement(input\_file):  
 # ...  
 for row in ws\_in.iter\_rows(min\_row=2, values\_only=True):  
 bio\_list.append(row[bio\_idx])  
 return bio\_list

## Board\_certification.py

Purpose:

Extracts board certification and subspecialty information for each provider. It also manages the addition of dropdowns for board certifications, ensuring that only valid certifications are selectable in the output.

Key Code Snippet:

def extract\_board\_certification(input\_file):  
 # ... (extract logic)  
set\_board\_certification\_dropdown(output\_file)

## optoutrating.py

Purpose:

Adds a dropdown for the 'Opt Out of Ratings' field in the output file. This allows providers to indicate whether they wish to be excluded from ratings, supporting privacy and compliance requirements.

Key Code Snippet:

def set\_opt\_out\_of\_ratings\_dropdown(output\_file):  
 # ... (dropdown logic)

## ESF.py

Purpose:

Adds a dropdown for the 'Enterprise Scheduling Flag' in the output file. This field is used to indicate whether a provider participates in enterprise-level scheduling, which can affect appointment availability and system integration.

Key Code Snippet:

def set\_enterprise\_scheduling\_flag\_dropdown(output\_file):  
 # ... (dropdown logic)

## Langauge.py

Purpose:

Extracts the languages spoken by each provider from the input file and sets up language dropdowns in the output. This ensures that language data is standardized and can be used for filtering or matching providers to patient needs.

Key Code Snippet:

def extract\_languages(input\_file):  
 # ...  
 for row in ws\_in.iter\_rows(min\_row=2, values\_only=True):  
 # ... (split and assign languages)  
 return lang1\_list, lang2\_list

## provider\_dropdowns.py

Purpose:

Applies a wide range of dropdowns and formulas to the Provider sheet in the output file, based on a list of specifications. This script centralizes the logic for data validation and formula application, making the output robust and user-friendly.

Key Code Snippet:

apply\_provider\_dropdowns(output\_file, dropdown\_specs)  
apply\_provider\_formulas(output\_file, formula\_specs)

## specialtydropdown.py

Purpose:

Adds specialty dropdowns to the output file using validation references. This ensures that specialty selections are always consistent with the reference data, reducing errors and improving data quality.

Key Code Snippet:

add\_specialty\_valref\_dropdowns(output\_file)

## \_status \_check.py

Purpose:

Checks the status and completeness of the output file by validating columns and grouping related fields. This script helps ensure that the final output meets all structural and data requirements before delivery or further processing.

Key Code Snippet:

# Group columns by base name (e.g., 'Specialty 1', 'Specialty 2' -> 'Specialty')  
for col\_idx, col\_name in enumerate(header):  
 base = re.sub(r'\s\*\d+$', '', str(col\_name))  
 if base not in col\_groups:  
 col\_groups[base] = []  
 col\_groups[base].append((col\_name, col\_idx))