

SOMEBUSINESS Process Excel XLSM to Output XLS eBuilder

Flow Documentation:

Solution: SOMEBUSINESS prep for eBuilder files

Flow Name: Process Excel xslm to output xls

Created By: PAD Designer (Power Automate Desktop)

Version: 1.0.0.5

Purpose: Convert SOMEBUSINESS macro-enabled Excel files (.xslm) into clean .xls output files formatted for upload to eBuilder.

SUMMARY

In plain terms, this flow:

1. Reads a config file to know where everything is.
2. Scans a source folder for `.xslm` Excel files.
3. For each file, opens it, identifies the project name, then loops through each worksheet.
4. Depending on the worksheet type (Architect Equity, Architect Workforce, or Contractor Equity), reads the right data region, validates it against a spec, and saves it as a clean `.xls` file.
5. Deletes the source file after processing.
6. Throughout the run, displays a live status page in Edge so the user can watch progress.
7. At the end, writes an HTML results report and shows a completion dialog.

TRIGGER

This flow is a **manually triggered Desktop Flow** - it runs on demand on a local computer. There is no automated schedule or event-based trigger.

Required Input Parameter:

- **ConfigFilePath** (*Text - Required*): The full file path to the configuration CSV file `SOMEBUSINESS\_MasterValues.csv`. This file tells the flow where to find source files, templates, and mapping/spec files.
- **Default value:** `C:\...\setup eBuilder Files\SOMEBUSINESS\_Config\_Files\SOMEBUSINESS\_MasterValues.csv`

CONNECTIONS

This flow runs entirely **locally on the desktop computer** - it uses **no cloud connectors** (no SharePoint, no Dataverse, no HTTP calls, etc.).

The following local resources are accessed:

Resource Type	Description
Local File System	Reads source `.xslm` files from a configured source folder; writes output `.xls` files to a dated output folder
CSV Config Files	Reads `SOMEBUSINESS_MasterValues.csv` plus several mapping and spec CSV files to control how data is mapped and validated
Microsoft Excel (local)	Opens, reads, and closes Excel workbooks using the local Excel installation
Microsoft Edge Browser	Launches Edge locally to display a live status page while the flow runs
Note:	The `ConnectionReferences` list in the flow package is empty - confirming zero cloud connector dependencies.

ACTIONS

The flow is organized into **main body** and **9 sub-functions (subroutines)**. Here is what each one does:

Main Body

- 1. Initialize variables** - Calls `subSetVariables` to load all configuration, set up folder paths, and read all required CSV files.
- 2. Set status message** - Sets the current progress message to "Starting Processing."
- 3. Create output folder if missing**
 - Checks whether the output folder exists
 - if not, creates a new dated folder named `Processed\_YYYYMMDD` inside the master folder.
- 4. Get list of source files**
 - Scans the source folder for any `.xls` or `.xslm` files (using the wildcard `\*.xls?`), sorted by name.
- 5. Loop through each source file**
 - For each file found:
 - Updates the progress message with the current file name and count (e.g., "Processing file: ABC.xslm - 2 of 5").
 - Calls `subUXMessaging`**
 - to refresh the live status page in Edge.
 - Opens the Excel file in read-only mode using the existing Excel instance.
 - Calls `subGetProjectName`**
 - to find and read the project name from the file.
 - Calls `subGetWorksheetData`**
 - to process each worksheet in the workbook.
 - Closes the source Excel file.
 - Deletes the source file after processing.
 - If the file could not be deleted, appends a red warning to the status message.
- 6. Wrap up**
 - Calls `subEndMessages`**
 - to write the final HTML results report and show a completion dialog.

subSetVariables

Initialize All Variables and Config

- Captures the current date and time
 - formats them as `YYYYMMDD`, short time, and short date strings.
- Sets the new output folder name to `Processed\_YYYYMMDD`.
- Initializes counters and flags (file count, restart flag, process-complete flag).
- **Reads `SOMEBUSINESS\_MasterValues.csv`** (the main config file).
 - If the file can't be found at the given path, calls `subfileloadbrowse` to let the user pick it manually.
- Derives all key folder paths from the config:
 - **Source folder** (where input `.xslm` files live)
 - **Template folder** (Excel output templates)
 - **Config folder** (where mapping and spec CSVs are stored)
- Reads the **Sheet Map CSV**
 - defines which worksheet type maps to which processing route.
- Reads the **Architect Equity spec file(s)**
 - defines the expected columns and data types for Architect Equity worksheets.
- Reads the **Architect Workforce spec file** –
 - same but for Architect Workforce worksheets.
- Reads the **Contractor Equity spec file**
 - same but for Contractor Equity worksheets.
- Builds a list of valid range-header titles used to locate data regions within Architect Equity sheets.
- Sets the output file path for the HTML status/results message.

subfileloadbrowse

Manual Config File Picker (Fallback)

- Triggered when the config CSV cannot be found automatically.
- Opens a **file browse dialog** prompting the user to locate `SOMEBUSINESS\_MasterValues.csv`.
 - If the user selects the correct file, updates `ConfigFilePath` and logs a note to configure the path properly.
 - If the wrong file is selected (or none), shows an error dialog and calls `subCaptureErrorsAndClose` to exit gracefully.

subGetProjectName

Read Project Name from Excel

- Activates the first worksheet of the source Excel file.
- Searches the sheet for a cell containing the text "**Project Name**".
- Reads the cell **directly below** that found cell
 - that value is the actual project name.
- Appends the project name to the running status message.

subGetWorksheetData

Process Each Worksheet

- Gets the list of all worksheet names in the source workbook.

- **Loops through each worksheet** and checks it against the Sheet Map to determine what type of sheet it is:
 - **ArchitectEquitySheetName**
 - Searches for the data header row using a list of known possible header titles.
 - If found, calls `subArchitectEquityFlow`
 - then validates and saves output via `subValidateOutputData` and `subSaveNewWorksheet`.
 - If the header can't be found, logs a red warning and skips to the next sheet.
 - **ArchitectWorkforceSheetName**
 - Finds the data start row using the Architect Workforce header title. Reads the data range from that row to the first empty row.
 - validates and saves output via `subValidateOutputData` and `subSaveNewWorksheet`.
 - **ContractorEquitySheetName**
 - Same pattern as Architect Workforce, but uses the Contractor Equity spec file.
 - **Any other sheet name**
 - Skipped (no action).

subArchitectEquityFlow

Map and Load Architect Equity Data

- Reads the column headers from the Architect Equity data region (up to 20 columns).
- Cleans each header (strips special characters, trims whitespace).
- Loops through available **map CSV files** to find which one matches the current sheet's column layout (by comparing column title lists).
- Once a matching map is found, loads the corresponding **spec file** for that map version.
- Calls `SubArchitectEquityDataRetrieval` to actually read the data from the Excel sheet.

SubArchitectEquityDataRetrieval

Read Architect Equity Data from Excel

- Moves one row down from the header row to reach the data.
- Finds the last populated row in column 1 to determine data extent.
- Reads the full data range from Excel into a datatable.
- **Adds two calculated columns** to the data:
 - **CalcCompanyName:** Resolves the company name by checking nested sub-consultant columns (tier 1 through tier 3) in priority order.
 - **CalcSubConsultantTier:** Assigns a tier number (1-4) based on which level column the company name is found in (1 = prime, 2/3/4 = sub-tiers).

subValidateOutputData

Validate and Shape Output Data

- Creates a fresh output datatable with columns defined by the spec file.
- Adds two header rows: one with the human-readable field names, one with data type codes.
- Merges the raw Excel data into the output table (extra columns are ignored).
- Removes empty rows.
- **Loops through every cell** and enforces data types based on the spec:

- **Numeric:** Converts to a number; replaces blank with `0` (zero).
- **TextRequired:** If blank, marks the row for deletion.
- **DateOptional:** If not blank, formats as a short datetime;
 - if it's not a valid date, replaces with a blank.
- **Blank:** Forces the cell value to blank.
- Logs a summary line: worksheet name, original row count, and valid (kept) row count.
- Calls `subValidateOutputData\_RemoveRows` to delete the flagged rows.

subValidateOutputData_RemoveRows

Remove Invalid Rows

- Takes the list of rows flagged for deletion (e.g., TextRequired fields that were blank).
- Sorts, deduplicates, and reverses the list (so rows are deleted from the bottom up, preserving index positions).
- Deletes each flagged row from the output datatable.
- If any rows were removed, appends a red-highlighted note to the status message listing which original row numbers were dropped.

subSaveNewWorksheet

Create and Save the Output XLS File

- Cleans the project name (removes special characters; replaces with underscores).
- Optionally shortens the project name by cropping text between "Bond" and "CT" markers - to keep the total output file path under 250 characters.
- Generates the output filename in the format:
`ProjectName\_RCRowCount\_YYYYMMDD\_SheetType.xls`
- Launches a **new blank Excel workbook** (invisible).
- Writes the output datatable into the first cell of the first worksheet.
- Saves and closes the workbook as an `.xls` file in the output folder.
- Verifies the file was created
 - logs a success or failure message accordingly.

subCaptureErrorsAndClose

Error Handler

- Captures the most recent error message.
- Appends a red-highlighted error block to the status message (including the error text and a note to contact Developer).
- Attempts to close the source Excel instance if it's still open.
- Calls `subEndMessages` to write the results file and show the final dialog even when an error occurred.

subEndMessages

Write Results and Show Completion Dialog

- Marks the process-complete flag as `True`.
- If the restart flag was set (due to Excel not closing cleanly), adds a recommendation to restart the computer.
- Builds an **HTML results page** with a heading showing the run date and the full status log.

- If a results file from a previous run exists, **prepends the old content** so history is preserved in the same file.
- Writes the HTML file to the output folder.
- Closes the Edge browser status window (if open).
- Shows a **custom Adaptive Card dialog** with:
 - A "Process has completed" message.
 - A button to open the HTML results file in the browser.
 - A "Close" button to dismiss.
- Exits the flow with code `0` (Zero).

subUXMessaging

Update Live Status Page in Edge

- Builds an HTML status page showing a spinning animation and the current progress message.
- Writes the HTML to a status file in the template folder.
- If Edge is not yet open
 - launches it and points it at the status file.
- If Edge is already open
 - refreshes the page to show the latest message.

subArchitectEquityUnused

Archived/Unused Code

- This function contains **disabled and commented-out code** from an earlier version of the flow.
- It is not called anywhere in the active flow and exists purely as a reference/archive.