Automating Excel Workbook Data Refresh (Open■Source Solutions)

Automating the refresh of Excel workbooks (e.g., *WH Receipt FY25.xlsx*) that pull data from external sources can save time and reduce errors. This guide outlines **open■source, scriptable methods** to refresh Excel data—including **Power Query** queries and external connections to other Excel files on a shared drive—on a schedule (e.g., daily) and on■demand. We avoid proprietary/paid tools (like Power Automate), focusing on free solutions that work in a networked environment and provide logging for success/failure.

Challenges & Requirements

- **Power Query & External Connections:** Ensure Power Query queries and external data connections refresh. Non■Excel libraries (e.g., openpyxl) can't trigger these on their own.
- **Timing & Completion:** Power Query is async by default. Disable background refresh or wait until completion to avoid partial saves.
- **Environment (Shared Drive):** Run under an account that can access UNC paths (e.g., \\Server\Share\...). Avoid mapped letters if the scheduler may not have them.
- **Scheduling & Manual Runs:** Integrate with Windows Task Scheduler (or cron) and allow on■demand execution.
- **Logging & Alerts:** Capture outcomes in a text file, event log, or a workbook log sheet with timestamps and errors.

1) VBA Macro + Windows Task Scheduler

Overview: Use an Excel VBA macro to refresh the workbook, then call it on a schedule via Task Scheduler. 100% within Excel—no extra installs.

Setup – Embed a Refresh Macro:

```
Sub AutoRefresh()
    On Error GoTo Fail
    Application.DisplayAlerts = False
    Application.EnableCancelKey = xlDisabled

    ' Refresh all data connections / Power Query
    ThisWorkbook.RefreshAll

    ' Simple wait to allow async queries to finish (or disable background refresh per query)
    Application.Wait Now + TimeValue("00:00:30")

    ThisWorkbook.Save
    Application.Quit
    Exit Sub
Fail:
    ' Optional: write to a log sheet or text file here
    Application.Quit
End Sub
```

Schedule with Task Scheduler: Create a task → Trigger (e.g., daily 01:00) → Action: program path to EXCEL.EXE, arguments: the workbook path and optionally /mAutoRefresh to run a specific macro. Run under a user with network access; consider "Run with highest privileges".
Logging: Add VBA to append outcomes to a log file or a 'Log' sheet; include On Error handling.
Pros: Native, full compatibility. **Cons:** Windows■only; macro security; must manage hidden prompts.

2) Python Script with Excel COM (pywin32 / xlwings)

Overview: Use Python to drive Excel via COM. Open the file, refresh, wait until queries complete, save, close. Flexible logging and orchestration.

Setup – Install: pip install pywin32 xlwings

Python (pywin32) example:

```
import win32com.client as win32

excel = win32.DispatchEx("Excel.Application")  # new process
excel.Visible = False
excel.DisplayAlerts = False
try:
    wb = excel.Workbooks.Open(r"\\CompanyShare\Reports\WH Receipt FY25.xlsx")
    wb.RefreshAll()                                  # refresh all
    excel.CalculateUntilAsyncQueriesDone()           # wait for PQ to finish
    wb.Save()
    wb.Close(False)
    print("Refresh successful.")
except Exception as e:
    print(f"Refresh failed: {e}")
finally:
    excel.Quit()
```

Alternative (xlwings) – refresh each connection:

```
import xlwings as xw
app = xw.App(visible=False)
try:
    wb = xw.Book(r"\\CompanyShare\Reports\WH Receipt FY25.xlsx")
    for c in wb.connections:
        c.refresh()
    wb.save()
finally:
    wb.close()
    app.quit()
```

Scheduling: Use Task Scheduler → Program: python.exe; Arguments: path to script. **Logging:** Use Python's logging to write success/failure to a rotating log file; add email on exceptions if desired.
Pros: Powerful, multi■file orchestration. **Cons:** Requires Excel; Windows COM; handle hangs/alerts defensively.

3) PowerShell or VBScript Automation

Overview: Use built■in Windows scripting to automate Excel via COM; schedule with Task Scheduler.

PowerShell example:

```
$excel = New-Object -ComObject Excel.Application
$excel.Visible = $false
$excel.DisplayAlerts = $false
try {
    $wb = $excel.Workbooks.Open("\\CompanyShare\Reports\WH Receipt FY25.xlsx")
    $wb.RefreshAll()
    $excel.CalculateUntilAsyncQueriesDone()
    $wb.Save()
    $wb.Close($false)
}
catch {
    Write-Error "Refresh failed: $_"
}
finally {
    $excel.Quit()
}
```

VBScript example:

```
Dim ExcelApp, wb
Set ExcelApp = CreateObject("Excel.Application")
ExcelApp.Visible = False
ExcelApp.DisplayAlerts = False

On Error Resume Next
Set wb = ExcelApp.Workbooks.Open("\\CompanyShare\Reports\WH Receipt FY25.xlsx")
If Err.Number <> 0 Then
  WScript.Echo "ERROR opening workbook: " & Err.Description
  WScript.Quit 1
End If

wb.RefreshAll
ExcelApp.Application.Wait(Now + #0:0:30#)  ' simple wait
wb.Save
wb.Close False
ExcelApp.Quit
```

Scheduling: Task Scheduler → Action: powershell.exe -File ... or wscript .... **Logging:** Redirect output to file or Event Log (PowerShell).

Pros: No extra installs; native. **Cons:** Windows■only; VBScript limited; manage prompts/permissions.

4) Open■Source Tool: Power■Refresh

Overview: An MIT■licensed Excel■based toolkit (Reports Controller + Refresher) to manage and schedule refreshes for many workbooks without coding.

Key features:

- Controller workbook lists target reports, schedules, and parameters in one place (mission control).
- Refresher launches a clean Excel instance, opens target, runs RefreshAll, saves, closes.
- Handles multi■file scenarios; supports custom pre/post macros and query ordering.
- No admin rights required; logs runs/status within Excel; can be triggered by Task Scheduler via a small VBS.

Setup: Download and unpack; open the Controller to register your workbook(s) and desired cadence; use the included starter VBS or Task Scheduler to trigger runs.

Pros: Purpose■built, open■source, centralized logging. **Cons:** Extra controller workbook; still requires an always■on Windows host with Excel.

5) Other Considerations

Headless extraction (advanced): Recreate the refresh in code (e.g., Python/pandas to read sources, write values with openpyxl). This bypasses Power Query but achieves similar outputs for simple cases. **Pros:** No Excel required. **Cons:** You must re■implement all transformations.

Excel Online / Office Scripts: Cloud options exist but are proprietary; excluded per open■source requirement.

Comparison of Methods

Method	Requires Excel?	Scheduling	Logging	Pros	Cons
+ Task Scheduler	Yes	Task Scheduler	Custom in macro / TS	Native, full compatibility; no extra installs	Windows■only; macro secu
on (pywin32/xlwings)	Yes	Task Scheduler / cron	Python logging / email	Powerful orchestration; robust error handling	Needs Excel; Windows COM
erShell / VBScript	Yes	Task Scheduler	File or Event Log	Built■in Windows; lightweight	Windows■only; VBScript lim
er■Refresh (OSS)	Yes	Controller or Task Scheduler	Excel log	Open■source; multi■file mission control	Extra controller workbook; W
t data extraction (code)	No	Any	Custom	No Excel dependency; cross■platform	Rebuild queries in code; cor

Choosing an approach: For a single workbook, a simple **VBA + Task Scheduler** job is often enough. For multiple files and centralized control, consider the **Power■Refresh** toolkit. If you need fine■grained logic, retries, or notifications, a **Python COM** script provides the most flexibility. In every case, test with the actual service account, use UNC paths, disable background refresh for Power Query, and implement logging so failures are visible.

Sources (indicative)

- Microsoft Community / Stack Overflow threads on automating Excel refresh via COM and handling background refresh.
- Tutorials demonstrating xlwings/pywin32 refresh patterns and waiting for Power Query to complete.
- Open■source project: Power■Refresh (Reports Controller & Refresher) – GitHub README.