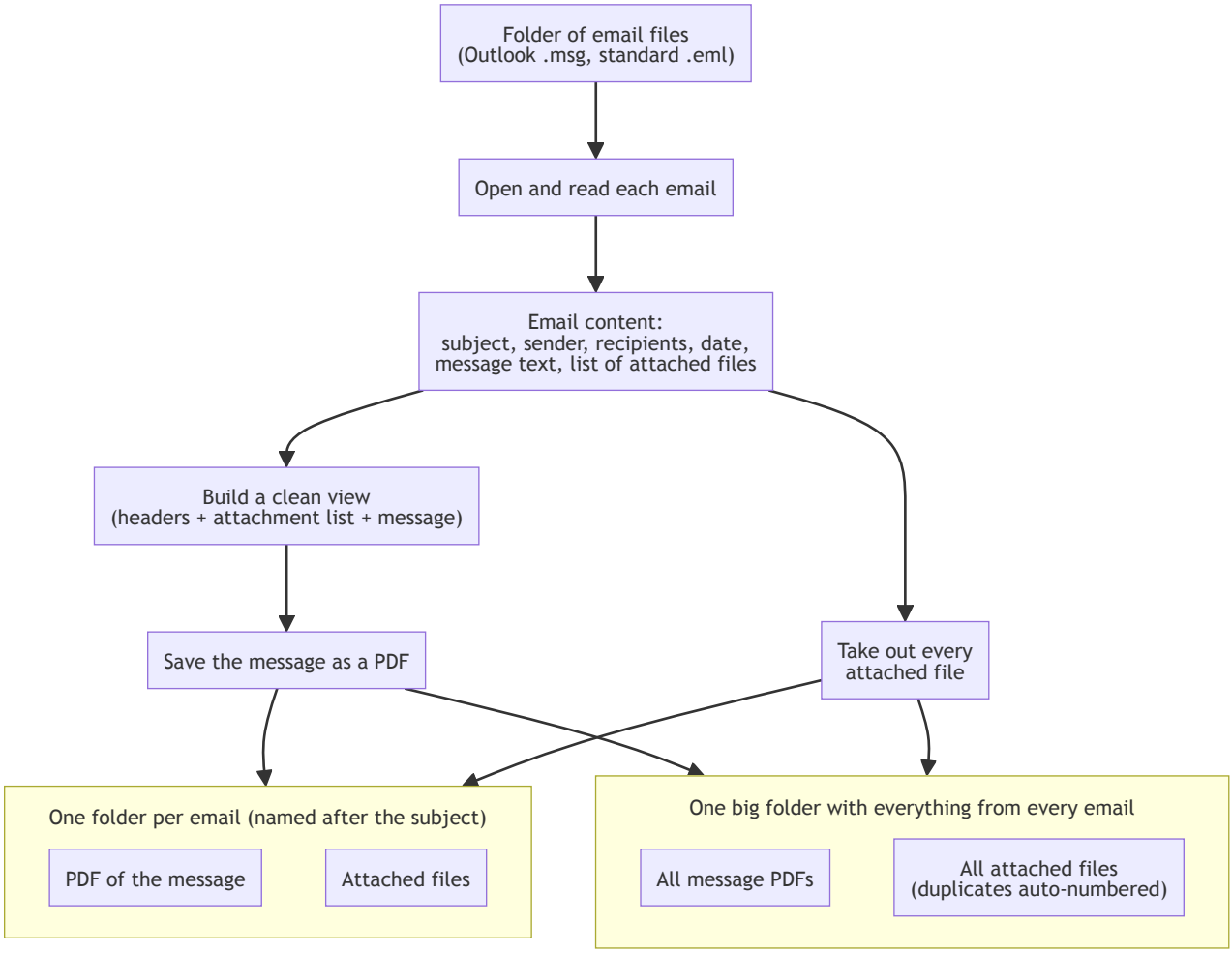
Email Extractor Tool

Prepared by: Lieu Vo · **Date prepared:** April 2026

A command-line tool that takes a folder of saved email files (`.msg`, `.eml`) and produces, for every email, a per-email subfolder containing the body as a PDF and copies of all attachments – plus a single consolidated `_All_Attachments/` folder gathering every email PDF and every attachment from the whole batch.

How It Works



Timing: Tool vs Manual

Per email, end-to-end:

Step	Tool	Manual (Outlook on Mac / Windows)
Open the email		~3 s
Create a per-email folder, type the subject as folder name		~10 s
Print / Save As → PDF for the email body		~12 s
Right-click each attachment → Save As → navigate → save		~10 s per attachment
Copy each saved file into a "consolidated" archive folder		~5 s per attachment
Total per email (2 attachments average)	**~0.3 s**	**~50–60 s**

Tool time is roughly **150× faster** than doing it by hand: a one-time ~1 s Chromium launch plus ~0.3 s per email regardless of size or attachment count. You also get identical, consistently-named files every time – no typos in folder names, no missed attachments, no inconsistent separators, no "invoice (1).pdf" copy-paste duplicates.

For a batch of 20 emails: tool ~7 s, manual ~17 minutes.

Output Files

For each email, the script writes back into the *same* folder you pointed it at:

```
<input folder>/
├─ <original .msg / .eml files>          ← never modified
├─ _All_Attachments/                     ← consolidated
│   ├─ _Email – <Subject 1>.pdf
│   ├─ _Email – <Subject 2>.pdf
│   ├─ <attachment 1>
│   └─ <attachment 2>
├─ <Subject 1>/                          ← one per email
│   ├─ _Email – <Subject 1>.pdf
│   └─ <attachment 1>
└─ <Subject 2>/
    ├─ _Email – <Subject 2>.pdf
    └─ <attachment 2>
```

File / folder	Source	Description
`<Subject>/`	One per parsed email	Folder named exactly like the email subject (filesystem-illegal characters replaced with _)
`_Email – <Subject>.pdf`	Headless Chromium render	Email body as A4 PDF – header block (From/To/Cc/Date/Subject + attachment list) above a divider, body below
`<attachment files>`	Pulled from the email payload	Every attachment from that email (auto-numbered on collision)
`_All_Attachments/`	Aggregated across the whole run	Every email body PDF and every attachment in one place, collisions auto-numbered

File-naming rules (`sanitize_name`):

- `< > : " / \ | ? *` and control characters are replaced with _.
- Whitespace is collapsed; trailing dots and spaces are trimmed.
- Names are capped at 150 characters; empty subjects fall back to the source `.eml`/`.msg` filename stem (or `no_subject`).
- Same-named outputs in the same folder get `(2)`, `(3)` suffixes – nothing is overwritten.

How Each Email Is Processed

1. **Detect format** by file extension (`.msg`, `.eml`). Anything else is skipped.
2. **Parse** through the matching backend:
   - `.msg` → `extract_msg.Message` (Outlook compound binary).
   - `.eml` → `email.parser.BytesParser` with `policy.default`.
3. **Normalise** to a `ParsedEmail` dataclass: `subject`, `sender`, `to`, `cc`, `date`, `html_body`, `text_body`, `attachments[]`. Nested embedded messages become `.eml`/`.msg` byte attachments.
4. **Build HTML** for the PDF: a header block (From, To, Cc, Date, Subject, `Attachments (n)` as a bulleted list of filenames), then `<hr/>`, then the email's HTML body (or `<pre>`-wrapped plain text if no HTML part). `<img src="cid:…">` references are replaced with `[inline image]` text – the actual image files are still extracted as attachments.
5. **Render** the HTML to A4 PDF via a single shared Chromium instance (Playwright sync API). One browser is launched per run and reused across every email.
6. **Sanitise** the subject for filesystem use, create `<Subject>/` next to the email file (auto-numbered `(2)`, `(3)` on collision).
7. **Write**:
   - `_Email – <Subject>.pdf` into the per-email folder.
   - Each attachment into the per-email folder (auto-numbered on collision).
   - A copy of every PDF and every attachment into `_All_Attachments/` (auto-numbered on collision).

Why Playwright for the PDF

The email body can be arbitrary HTML – tables, inline styles, web fonts, multi-column layouts. Pure-Python HTML→PDF libraries either need extra system libraries to install (`xhtml2pdf` pulls in `pycairo`; `weasyprint` needs `pango` / `cairo`) or render a degraded subset of HTML/CSS.

Playwright drives a real headless Chromium, so the rendered PDF matches what the email looks like in a browser. Side benefits:

- **Cross-platform, no extra system installs** – Playwright bundles its own Chromium download on both macOS and Windows; no Homebrew or MSYS2 needed.
- **One browser, many emails** – the script launches Chromium once and reuses the same `BrowserContext` across every email, opening a fresh `Page` per render. Avoids the ~1–2 s per-email startup cost of spawning a new Chromium each time.
- **Page setup**: A4, `print_background=True`, 2 cm margins all sides – matches the UK business document layout used elsewhere in this project.

Requirements

- Python 3.10+ (macOS or Windows)
- `pip install extract-msg playwright`
- `playwright install chromium` – one-time browser download:
  - macOS: `~/Library/Caches/ms-playwright`
  - Windows: `%LOCALAPPDATA%\ms-playwright`

Usage

```
python msg_extractor.py

================================================
  Email Extractor Tool
================================================

Enter folder path (containing .msg / .eml files): ~/Downloads/my-emails

Found 4 email file(s) in /Users/you/Downloads/my-emails
→ email_one.eml
→ email_two.msg
→ email_three.eml
→ email_four.msg

Done. 4 email(s) processed, 6 attachment(s) extracted.
Output: /Users/you/Downloads/my-emails/_All_Attachments
```

Input accepts:

- An interactive prompt (no arg given): typed path, or drag-and-drop a folder into the terminal (Finder on macOS, File Explorer on Windows). Surrounding quotes are stripped automatically.
- A CLI path argument for scripting: `python msg_extractor.py "/path/to/folder"` (macOS) or `python msg_extractor.py "C:\path\to\folder"` (Windows).

Re-running on the same folder is safe – existing per-email folders get a `(2)` suffix; nothing is overwritten.

Notes

- Subject sanitisation only fixes filesystem-illegal characters; non-ASCII characters (Cyrillic, CJK, accented Latin, etc.) round-trip unchanged.
- Inline `cid:` images in HTML bodies are replaced with `[inline image]` text in the PDF, but the underlying image files are still saved to the attachments folders (the email parser exposes them as ordinary attachments).
- Empty bodies (some emails are sent with only attachments and no message text) render as just the header block + divider + nothing below – that reflects the source email, not a bug.