ExeFoundry: Portable `BAT` → `EXE` Converter for Shareable Windows Utilities

Sudipta Kumar Das

Version 1.0.0

Abstract — We present *ExeFoundry*, a tiny Windows tool that converts a Batch script (`.bat`) into a single portable `.exe`. The converter builds a minimal C# launcher, preserves argument forwarding, optionally embeds an `.ico` icon, and can target either the console or GUI subsystem (no console window). The design requires no admin privileges, does not touch the registry or services, and is released with open, reproducible documentation.

1 Introduction & Background

Batch files remain a practical way to automate Windows tasks. However, distributing `.bat` scripts to non-technical users can be awkward: double-click behavior varies, icons look unpolished, and command-line quoting or dependencies often confuse end users. Packaging the same logic as a single `.exe` improves usability, branding, and shareability without rewriting the script.

2 Problem Statement & Objectives

Problem. Teams want to ship internal tools as one signed file with a clean icon and without invasive installers or admin rights.
Objectives.

- Convert `.bat` to a `.exe` with preserved argument semantics.
- Optional custom icon embedding; selectable Console/GUI mode.
- Zero system mutation (no registry/services), no admin elevation.
- Open, auditable, reproducible build with checksums and citation metadata.

3 Contributions

- **Minimal C# launcher.** Clear, auditable stub that invokes the original `.bat`.
- **Argument forwarding.** `MyTool.exe arg1 "two words"` behaves like the batch.
- **Icon & subsystem.** Embed `.ico`; build Console (default) or GUI (`-WinExe`).
- **Open access.** Tech note (PDF), `CITATION.cff`, release artifacts with SHA-256.

4 Expected Impact

For end users. Double-clickable, branded tools with no setup.
For IT/admin. No installs, no registry edits; optional code-signing for production.
For teaching/research. A small, reproducible example of packaging and software supply-chain hygiene (hashes, docs, provenance).

5 Method

ExeFoundry reads the input `.bat`, generates/compiles a tiny C# launcher, and emits a single `.exe`. The launcher runs the batch with proper quoting and forwards all arguments. The build optionally embeds an `.ico` and can switch to the Windows GUI subsystem.

Build invocation

```
# Console build (default)
.\ExeFoundry.exe -InputBat ".\scripts\hello.bat" -OutputExe ".\Hello.exe"

# With custom icon and GUI (no console window)
.\ExeFoundry.exe -InputBat ".\scripts\hello.bat" `
-OutputExe ".\Hello.exe" -Icon ".\scripts\icon\bat_to_exe.ico" -WinExe
```

Conceptual C# launcher (sketch)

```
using System; using System.Diagnostics;
class Entry {
       static int Main(string[] args) {
              var psi = new ProcessStartInfo("cmd.exe",
              "/c␣\"\"C:\\path\\to\\script.bat\"␣" + string.Join("␣", args) + "\"
                  ");
              psi.UseShellExecute = false;
              psi.CreateNoWindow = false; // GUI build sets true
              var p = Process.Start(psi);
              p.WaitForExit();
              return p.ExitCode;
       }
}
```

6 Trust & Security Model

ExeFoundry does *not* modify the registry, install services, or require admin elevation. It produces a transparent launcher that calls the batch file. For distribution, we recommend providing: (1) the original `.bat`, (2) `SHA256SUMS.txt` for all assets, and (3) a signed build for production deployments.

7 Reproducibility Checklist

- OS: Windows 10/11 (22H2/23H2).
- Tools: PowerShell; C# compiler (`csc`, often available via .NET SDK/Developer Pack).
- Inputs: `.bat` source; optional `.ico`.
- Outputs: single `.exe`; release ZIP with PDF, `CITATION.cff`, and `SHA256SUMS.txt`.
- Repository includes: tech note (LaTeX+PDF), example scripts, and build commands.

8 Usage

1. Prepare your script, e.g., `scripts\hello.bat`.
2. Build with ExeFoundry (examples above).
3. Distribute `.exe`. Users invoke it with the same arguments as the batch.

Minimal test batch

```
@echo off
echo Hello from BAT!
echo Args: %*
pause
```

9 Limitations

- A C# compiler must be present; most Win10/11 setups can install the .NET SDK.
- Antivirus engines may flag unsigned one-file tools; include hashes and consider code-signing.
- Paths containing spaces require quoting in both input and launcher contexts.
- Hiding the console (`-WinExe`) means background execution; add `pause` for demos.

10 Ethical/Operational Notes

Use only for packaging scripts you are authorized to distribute. ExeFoundry is a wrapper: the resulting `.exe` runs your batch with the same privileges as the user and does not bypass authentication, modify policy, or elevate privileges.

11 Funding Rationale & Deliverables

A small, time-boxed grant (e.g., RA/TA support) would harden ExeFoundry for campus-wide use:

- **D1.** Evaluation kit & report (launcher overhead, AV profile, quoting tests).
- **D2. Bundle Mode (folder → EXE)** with manifest, hash verification, cache/cleanup.
- **D3.** Signed portable builds, checksums, and reproducible scripts (CI job).
- **D4.** Admin/end-user guide (icons, GUI vs console, signing); reproducibility bundle with DOI.

12 Future Work: Bundle Mode (Folder → Single EXE)

Many real tools require a batch file *plus* helpers (configs, scripts, small tools). We will add a **Bundle Mode** that packages a folder into one EXE:

- **Design.** Embed a ZIP + manifest inside the launcher; on run, verify SHA-256, extract to `%LOCALAPPDATA%\ExeFoundry\{hash}` (user space only), and execute the entry `.bat` with the original arguments. No registry/services; no elevation.
- **Safety.** Path sanitization (no absolute/`..\`); optional cache; cleanup policy (`on-exit`/`never`); optional code-signing in Releases.
- **CLI.** `-InputDir`, `-EntryBat`, `-Icon`, `-WinExe`, `-CacheExtract`.

Evaluation. EXE size/launch overhead; hash integrity tests; AV profile; conversion success across public BAT repos; small student pilot.