FollowMe Printer Deployment Automation

Professional portfolio project summary

Amandio Pereira | UK Professional Standards Organisation | 2025

This case study explains how I used PowerShell and Intune to correct a failed printer deployment, repair the driver state on legacy hybrid PCs and provide self-service FollowMe installation for cloud-managed devices.

Organisation-specific identifiers, application names, internal paths and original script names have been generalised for public presentation; the core technologies, workflow and engineering decisions are retained.

Project objective	Correct the legacy printer deployment and deliver a reliable FollowMe queue across hybrid and Autopilot-managed PCs.
My role	Senior technical lead who analysed the existing printer deployment problems and designed, developed, tested and documented the PowerShell/Intune remediation, driver and self-service deployment paths for hybrid and Autopilot-managed devices.
Business value	Removed the legacy deployment, corrected the driver state on hybrid PCs and provided self-service FollowMe installation for cloud-managed devices.
Core technologies	PowerShell 5.1, Intune Win32 apps, Company Portal, Windows Print Management, Windows Scheduled Tasks and custom detection.

1. Executive Summary

This project addressed a print deployment problem in a managed Windows environment. “FollowMe” is a virtual print queue rather than a conventional network printer with its own IP address. The solution had to work across both system and user contexts for driver preparation, installation logic, and user access.

Legacy hybrid PCs added further complexity because, up to this point, they had been issued with a manual printer installation using incorrect drivers which had impeded previous attempts to automate deployment. Once the correct drivers were installed on the print server, legacy PCs still using the earlier incorrect driver could no longer print because of the mismatch. Reviewing the installation paths showed that these PCs only required the correct driver layer to be fixed, while a previous fix attempt needed to be removed and replaced with a working automated deployment.

In operational terms, the problem was multi-layered: removing a failed bulk deployment, correcting the driver on hybrid PCs, and automating the FollowMe printer in the user context while handling driver installation in the system context.

In simple terms

The solution followed two paths. Existing hybrid PCs needed the incorrect driver replaced so their printer connection could work again. Cloud-managed PCs needed a self-service workflow that prepared the driver and created the printer connection for the signed-in user. Cleanup removed the earlier incorrect local queue where present.

2. Solution overview

The script set contains specialised components for cleanup, driver preparation and user printer connection. The diagram shows their overall relationship; each device follows the stages required for its deployment path.

1. Remove legacy local printer	2. Install correct driver	3. Confirm driver exists	4. Create user printer connection	5. Confirm user connection exists

This staged approach allowed for a controlled deployment through a combination of an estate-wide cleanup script and targeted Win32 applications, so that remediation, automatic deployment, and self-service availability could be applied to the appropriate endpoint groups.

2.1 Deployment stages

Component	PS1 file(s)	Target and delivery	Logic
Estate-wide remediation	Remove-LegacyPrinter.ps1	All PCs; Intune PowerShell script	Removes an incorrect local “FollowMe” printer. Set to run only once rather than as a Proactive Remediation script. If this fails, the main self-service script includes its own logic to detect, remove, and reinstall the printer connection.
Automatic hybrid PC deployment	Install-PrinterDriver.ps1; Detect-PrinterDriver.ps1	Hybrid / domain-joined PCs; Win32 app, required assignment	Automatically applies the required Konica Minolta driver fix to affected PCs and uses custom detection to confirm that the correct driver is present.
Self-service deployment	Install-FollowMePrinter.ps1; Detect-FollowMePrinter.ps1	Company PCs / Autopilot group; Win32 app, available in Company Portal	Makes the FollowMe queue available on demand through the Company Portal and uses custom detection to confirm that the printer connection exists in the signed-in user profile.

Through this split, I sought the most practical and effective approach: estate-wide correction for a previous local printer deployment, an automated driver fix for legacy hybrid PCs, and a more involved self-service deployment that allowed users to install the printer on demand through Company Portal.

3. Key programming areas and rationale

These scripts are operational automation rather than software products, aimed at structured problem-solving.

3.1 Core script structure

Programming area	Meaning	Logic
Variables and configuration	Server names, driver names, log paths, task names and printer names are defined once and reused.	This improves readability and makes the scripts easier to maintain or adapt to another environment.
Functions	Reusable logging and utility functions avoid repeating the same code block many times.	Reusable logging and utility functions kept behaviour consistent across the script set.
Conditional logic	The scripts check whether a printer, driver, task or registry entry already exists before acting.	This creates safer, more predictable automation and avoids unnecessary changes.
Error handling	Try/catch blocks, exit codes and validation checks are used at important failure points.	This helps administrators understand what went wrong and supports controlled remediation through the log file.

3.2 Deployment-specific design

Programming area	Meaning	Logic
Context awareness	Some tasks run as SYSTEM and some in the signed-in USER context.	Execution scope, permissions and Windows profile behaviour differ between contexts.
Architecture checks	Scripts relaunch themselves in 64-bit PowerShell where needed.	This reduces compatibility issues when working with Windows management components.
State detection	Separate detection scripts verify that deployment completed successfully.	This supports reliable Intune deployments because install and detection are treated as distinct stages.
Task orchestration	The main self-service deployment script creates a scheduled task to bridge from system context into user context.	Because this Win32 package runs in SYSTEM context, the main script cannot simply create the final printer connection where it actually needs to live. It therefore uses a scheduled task as a controlled handoff into the signed-in user context. This keeps driver checks, cleanup, orchestration, and the final user-side printer connection within one managed workflow, rather than splitting the process into a separate user-side deployment with its own dependency chain.
Logging and auditability	Timestamped logs are written to user or system locations.	This improves supportability, troubleshooting and evidence of execution.

3.3 PowerShell 5.1 and 64-bit execution

The scripts in this deployment were developed in PowerShell 5.1, which was the default version available on the Windows 11 devices used in the deployment environment. This provided broad native compatibility and allowed the solution to be deployed without introducing an additional runtime. Within that design, however, 64-bit execution still had to be enforced explicitly in the Intune Win32 deployment path.

In an Intune Win32 app, calling `powershell.exe` directly from either the Install or Uninstall command launches 32-bit Windows PowerShell. The install command therefore uses `%SystemRoot%\Sysnative\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy Bypass -File .\filename.ps1` to invoke the 64-bit host. Because Intune does not expand environment variables in the Uninstall command, the equivalent explicit path `C:\Windows\Sysnative\WindowsPowerShell\v1.0\powershell.exe` is used instead.

As a further safeguard, the main scripts also include a small function titled **"Relaunch in 64-bit PowerShell if needed"**, which verifies the current process architecture and restarts the script in 64-bit PowerShell where necessary. This keeps installation, configuration, and detection aligned with the intended 64-bit Windows context, including the native registry path `HKEY_LOCAL_MACHINE\SOFTWARE` rather than the redirected 32-bit path under `HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node`.

An alternative approach would be to introduce PowerShell 7 as a deployment dependency, deploy it ahead of the package (silently with `/norestart`), and then call the 64-bit host explicitly by using `%ProgramFiles%\PowerShell\7\pwsh.exe -ExecutionPolicy Bypass -File .\filename.ps1`. Although the relaunch safeguard should no longer be required in this scenario, this was treated as a secondary option because it added an extra dependency and a more complex deployment sequence. A simpler approach was therefore preferred for a pressing deployment, using a reliable delivery path.

Dates and explanatory remarks were included throughout the scripts to improve readability during troubleshooting, facilitate knowledge sharing, and support function reuse in other projects where appropriate.

Rationale behind the design

The scripts are not intended to perform every task within a single file without clear separation of purpose. Separating cleanup, preparation, deployment and verification allowed each stage to be assigned, tested, detected and supported independently. It also kept the hybrid remediation path distinct from the user-facing self-service workflow.

4. Script-by-script explanation

4.1 Remove-LegacyPrinter.ps1 – Estate-wide cleanup script

This script runs in the user context and removes a wrongly deployed local printer called “FollowMe”. It is a corrective script intended to clean up the effects of a previous deployment method.

Purpose	This script looks for an incorrectly installed local printer and removes it. It first uses a standard removal method and, if that fails, falls back to an older Windows print command. It also writes a user log so that the action can be traced later.
Key inputs	Target printer name, user log path, and the current signed-in Windows profile.
Main output	Either the printer is removed successfully or the log records that it was not present or could not be removed.
Logic	Defensive automation: it checks first, tries a primary method, uses a fallback method, then verifies the result.

4.2 Install-PrinterDriver.ps1 – Win32 driver deployment script for hybrid PCs (domain-joined)

This script runs in system context and supports the hybrid PC deployment path by ensuring that the required Konica Minolta Universal PCL driver is installed and registered correctly in Windows Print Management.

Purpose	This is the preparation stage for the legacy hybrid path. It fixes the driver layer so that affected devices have the correct printer driver available, without needing to perform the full user-side printer deployment.
Key inputs	Driver name, INF package path, spooler service, and a system log directory.
Main output	A device that has the correct printer driver installed and a log showing what happened.
Logic	The script validates the environment, controls the spooler service, installs the driver conditionally and returns exit codes for Intune.

4.3 Detect-PrinterDriver.ps1 – Driver detection script for hybrid PCs

This is the companion detection script for the hybrid driver deployment. It is used by Intune to confirm that the required printer driver is present and to record the Win32 app status in the admin portal.

Purpose	This script does not install anything. It simply asks Windows whether the correct printer driver exists. If yes, it reports success; if no, it reports failure.
Key inputs	The expected driver name in Windows.
Main output	A simple detection result for Intune, used to record the Win32 app status in the admin portal.
Logic	Installation logic and detection logic are deliberately separated.

4.4 Install-FollowMePrinter.ps1 – Win32 self-service deployment script

This is the Win32 deployment script for the self-service path. It is more elaborate because, unlike the hybrid driver-fix package, it has to coordinate cleanup, driver checks, user-context execution, and creation of the actual FollowMe printer connection.

Purpose	Removes old printer connections, installs the driver if needed and launches a temporary scheduled task for the signed-in user. After a 15-second wait, it stops the task if still running and removes the task and helper script. The separate detection script checks whether the user's printer connection exists.
Key inputs	Printer server path, driver details, current interactive user, scheduled task name, and log locations.
Main output	A per-user connection to the FollowMe shared printer, set as the default printer when successfully created, together with system and user log entries.
Logic	Combines configuration management, context switching, temporary script generation, task scheduling, network checking, and cleanup into one controlled workflow. It also removes earlier queue variants before deployment and clears its own temporary scheduled task and helper script afterwards, preventing stale artefacts from being left behind.

4.5 Detect-FollowMePrinter.ps1 – Printer detection script for self-service deployment

This is the companion detection script for the self-service deployment. It confirms that the FollowMe printer connection exists for the active user and allows Intune to record the resulting Win32 app status in the admin portal.

Purpose	This script verifies that the printer really exists in the user profile, not just somewhere on the machine. That matters because user printer connections are stored differently from machine-wide components such as drivers.
Key inputs	The active user session, the user SID, the expected registry path, and the printer connection key.
Main output	A detection result that tells Intune whether the user-side printer deployment completed and records the resulting Win32 app status in the admin portal.
Logic	The detection script identifies the signed-in user and checks for the expected printer connection in that user's registry hive, rather than checking only whether the device has the driver installed.

4.6 Packaging design

For hybrid and self-service deployments, the required Konica Minolta driver files were packaged in the same source folder as the relevant installation script. This was a deliberate simplification for two small Intune Win32 packages, each containing one deployment script and driver files. Keeping the driver payload with each package made the deployment self-contained, easier to test, and less dependent on external sequencing.

An alternative design would have been to package the driver separately as its own deployment script and use it as a dependency for the printer deployment packages. While that would reduce duplication, it would also add dependency handling and make the overall deployment sequence more complex. Given the size and scope of the solution, bundling the driver files directly with each package was the more practical choice.

A cleaner package layout would place the driver payload in a dedicated `Drivers` subfolder and reference it from the script by using `$PSScriptRoot`. This keeps the Win32 package tidy while allowing all required files to be bundled and resolved reliably at runtime, for example:

```
$DriverINFPath = Join-Path -Path $PSScriptRoot -ChildPath 'Drivers\PrinterDriver.inf'
```

5. Outcome and Value Delivered

Taken together, the workflow replaced a mixed and partly failed printer configuration with a controlled deployment model appropriate to each device type. Affected hybrid PCs received the required driver correction automatically, while cloud-managed devices could install the FollowMe queue on demand through Company Portal. Separate detection scripts allowed Intune to verify the device-side driver and user-side printer connection independently, while logging and cleanup made the process easier to troubleshoot and support.

Engineering approach

I treated the problem as a set of distinct states rather than a single installation task: remove the failed legacy queue, correct the driver layer where required, then create and verify the user-scoped connection. Keeping remediation, deployment and detection separate allowed hybrid and Autopilot-managed devices to follow the appropriate path, while native PowerShell 5.1 and self-contained packages avoided unnecessary dependencies. The design favoured clear state checks, predictable cleanup and supportability over a more elaborate dependency chain.

End of case study