Application Packaging

## Introduction

Application Packaging plays a crucial role in modern IT environments. Whether you're working in a large enterprise or a growing business, ensuring that software is installed and managed consistently across hundreds or thousands of machines is no easy feat. That’s where application packaging steps in.

With over a decade of experience in this field, I’ve seen how packaging, when done right, can eliminate chaos, reduce support costs, and make life easier for both users and IT staff. This guide will walk you through everything from industry trends to practical steps, best practices, and helpful tools.

## 1. What Is Application Packaging?

Application packaging is the process of creating a customized installation package for a software application. This allows the software to be installed silently, with predefined settings, across many systems.

Think of it like wrapping a gift: you're not just handing over the software—you’re preparing it in a neat, repeatable way so it's ready to be delivered and used without hassle.

## 2. Why Is It Important?

Here’s what well-executed application packaging offers:

* **Consistency**: Every machine gets the same version with the same settings.
* **Speed**: You can deploy software faster, especially during large rollouts.
* **Security**: Standardized installs reduce the risk of misconfigurations.
* **Support**: Easier to manage, troubleshoot, and update software later.

## 3. The Real-Life Expectations

In today’s job market, organizations expect a packager to handle the entire process—from gathering requirements to deployment. This can be demanding, especially as companies aim to cut costs and speed up software delivery.

Unfortunately, this also means finding skilled packagers is getting harder. So, having a deep understanding of the full packaging cycle is essential if you want to stand out.

## 4. The Packaging Process – Step by Step

### A. ****Application Discovery****

Before you create a package, you need to understand the app:

* What are its installation steps?
* Are there any prerequisites?
* Do different teams use it differently?

Good discovery prevents costly rework later.

### B. ****Packaging the Application****

Based on your findings, you create a package using tools like:

* **MSI**
* **App-V**
* **MSIX**

Each has its advantages. MSIX, for example, uses containerization, which isolates the app and reduces conflicts.

### C. ****User Acceptance Testing (UAT)****

Testing is vital. We typically use virtual machines that mirror the production environment to ensure:

* The app works correctly.
* It doesn’t interfere with other software.
* All settings are applied as expected.

## 5. Deployment

Once a package passes UAT, it’s time to deploy it. This is usually done through tools like:

* **Microsoft Intune**
* **Configuration Manager (SCCM)**

Deployment should be done in **phases**. Start small, then expand. If something goes wrong, it’s easier to manage on a small group than across the entire company.

## 6. Tools of the Trade

Besides the packaging tools, you might use automation and management tools like:

* **PacKit**: Helps manage and deploy packages.
* **PowerShell App Deployment Toolkit (PSADT)**: Automates installation scripts.
* **WinGet**: Pulls software from repositories.

These tools simplify the process and reduce manual errors.

## 7. Best Practices to Remember

Here are some tips every good packager follows:

* **Always document** the discovery and testing steps.
* **Avoid manual installs** whenever possible.
* **Use naming conventions** to stay organized.
* **Update apps regularly**, especially with Windows 10’s rapid update model.

## 8. Application Models vs. Package Models

In Configuration Manager, you can use either:

* **Package Model**: Simple, fast, but limited.
* **Application Model**: Supports dependencies, detection methods, and supersedence (auto-upgrading apps). Recommended for most use cases.

## 9. Managing Old Packages

Don’t let old packages pile up. Follow this cleanup process:

1. **Retire** the application (stop new deployments).
2. **Uninstall** it from devices.
3. **Remove** it from the system.

This keeps your environment clean and reduces maintenance.

## 10. Application Rationalization

This means evaluating all the apps in use and deciding:

* Which to keep
* Which to retire
* Which to replace or consolidate

It takes time and budget but leads to long-term savings.

## 11. Real-World Tips

* Test every change in a clean virtual environment.
* If a fix is needed, **repackage** instead of creating a separate patch.
* Keep separate packages for prerequisites—you’ll save time in the future.
* Use supersedence to handle upgrades cleanly.