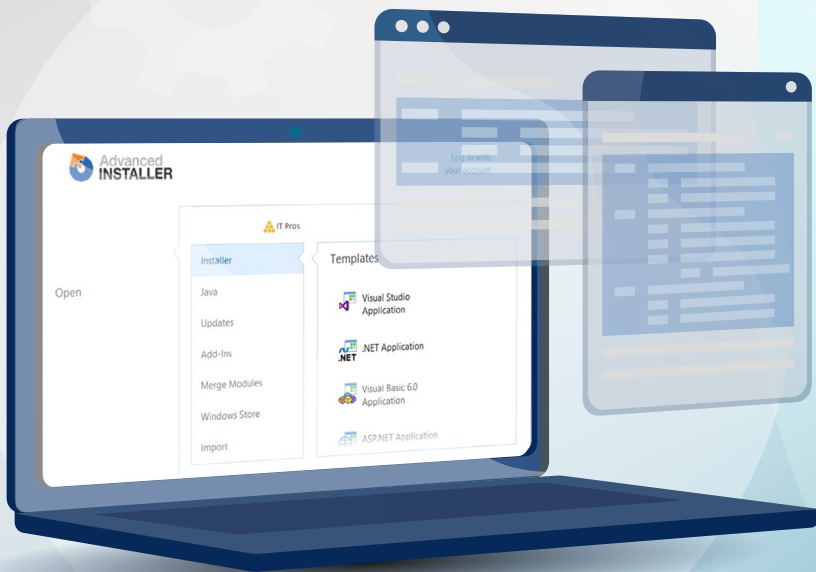




## 17 Best Practice Recommendations to

# Package Your Application for Enterprise Use.



*From the installer type to documentation guidelines, everything you need to consider when building your installer package to ensure flawless performance within enterprise environments.*

## Application Packaging for independent developers and ISVs

### The essentials for building your installer

1. MSI or EXE?
2. Support the command line switches for silent installation (and configuration).
3. Create an upgrade package to update to the new version
4. Make your installer 100% offline.

### For enterprise deployment

5. Try not to bundle multiple packages/MSIs into a single one.
6. Provide a command-line switch to disable automatic updates
7. Include building rollback custom actions.
8. Never modify system resources from immediate custom actions--only deferred ones.
9. Avoid triggering restarts during the installation.
10. Store your application data in the Roaming AppData folder.

### Testing

11. Check the package for UI blocking custom actions on the InstallExecute sequence.
12. Test your package installation under the SYSTEM account, using PsExec tools.
13. Test your package in VMs (virtual machines), on multiple systems, to cover all your users.

### Documentation

14. Document the command line switches/options in a dedicated download page (or PDF) for the IT pros.
15. Document all GUI user inputs (if any) as command line parameters, with examples.
16. Document the list of requirements: prerequisites, minimum OS required, database connection.
17. Keep logs to ease debugging when your custom actions fail.

### That's it!

# Application Packaging for independent developers and ISVs

If you are a developer in charge of building and maintaining the installer package for your product, this paper is for you. It is meant to help dev. teams in preparing their product installer for enterprise use. We hope you find this resource both helpful and useful.

By the end of it, you will have a deeper understanding of what IT Pros need from your installer package and the reasons why, by delivering a comprehensive installer, you will directly contribute to reducing customer support tickets and repackaging requests.


In practice, the installer is the very first direct interaction of your users with your application.

And a positive first interaction is essential for your success and a smooth, simple install experience is a step towards ensuring a higher rate of happy users.

However, dedicating extra resources to your installer could result in having less time to work and develop your main product.

This is a challenge many of our customers face until they make the decision to give the right attention to the installer building phase and include it in the product development plan right from the beginning.

Enterprise or B2B software products have a specific characteristic as they need to be prepared for eventual repackaging or transformation. We know it as the application packaging process, and it is a layer that deals with the installation package.

To find out more about the IT Pro's tasks and activities, we highly recommend reading [The End-to-end Application Packaging Process - Best Practices and Tips for Success.](#) 

At Advanced Installer, we are privileged to work both with developers and IT Professionals in enterprises, like you and your colleagues. To share a bit of what we've learned, we put in place a list of recommendations for you to follow when you prepare an installer to ensure a smooth implementation of your product.

**Recommendation:**

- Use the [Software Packaging Checklist App](#) to keep track of your installer building process.
- 🖨️ Print your checklist progress to use it in your status meetings.

To make better sense of use of the recommendations, we classified them in the following topics: **The Essentials**, **Enterprise Deployment**, **Testing**, and **Documentation**.

Let's go through them.

# The essentials for building your installer

## 1. MSI or EXE?

### What installer type is the best to deliver your product?

If your application will be used in an enterprise (e.g. Network Monitoring app), then you may want to create the package as an MSI installer because of the nature and processes of the implementation and use.

In an enterprise environment, the deployment of an application is made from central workstations and the sysadmin usually prefers MSI as it can be customized based on specific needs and abilities (e.g. removing shortcuts, disabling automatic updates, etc.).

The IT Pros in enterprises prefer working with MSIs because it will allow them to:

- Use silent deployment for all users throughout the network (using SCCM or similar tools).
- Configure the package using standard Windows Installer properties.
- Customize its contents to meet their internal company policies using MSI editing tools (such as: customizing shortcuts, adding company templates, disabling automatic updates, etc.).
- Leverage the Windows built-in support for system restore points and automatic repairs for broken installations.
- Generate verbose logging for debugging purposes, if needed.

**Note:** There might be cases where you need to make an EXE wrapper containing MSI package. In this scenario the IT Pro will usually need to extract the MSIs in order to access all the above listed benefits.

However, if your application is targeted towards the end-user, you may want to go with an EXE package type. The difference is mainly that you'll be able to get creative with your UI for the installation and overcome MSI limitations.



With **Advanced Installer**, you can have multiple builds in your project. Take advantage of the MSI build to support Enterprise customers and the EXE build with a premium UI for your product end-users.

**Advanced Installer** keeps both builds in sync--no need for special settings to be set.

## 2. Support the command line switches for silent installation (and configuration).

As a best practice, we recommend documenting any switches that the installer may use during the installation process e.g. if a particular action is executed based on a specific condition.

By default, all installers created with Advanced Installer support silent installation.

You can find the comandlines supported by Advanced Installer here:

<https://www.advancedinstaller.com/user-guide/msiexec.html> 

<https://www.advancedinstaller.com/user-guide/exe-setup-file.html> 

## 3. Create an upgrade package to update to the new version

While a patch is useful in many scenarios of the software packaging process, it only contains the differences between two versions of the same product. This means that through a patch you can only update a package that is already installed on the target machine.

For this reason, creating an upgrade package is a better approach. An upgrade package is a stand-alone installer that can perform a first-time installation when the old product is not installed on the target machine.

Some of the reasons why an enterprise sysadmin will always use the upgrade package are:

- IT Pros perform changes in the MSI packages and it's significantly easier to modify an upgrade package than a Microsoft Patch (MSP).

- Patch installations can fail. When IT Pros perform changes on a base MSI ( a common practice in managed infrastructures), there are high chances that the patch will fail due to internal Windows Installer validations.
- It could be a nightmare to administrate - patches need to reference their base applications so using patches in the enterprise infrastructure could turn complicated.
- On the vendor's side it takes more time to build a patch than an upgrade package.



#### 4. Make your installer 100% offline.

##### **Do not make Internet access mandatory at installation time.**

Installation processes may get frustrating for users with slow internet speed and dealing with large downloads. To avoid this, installers should contain all the resources inside, requiring no internet connection during the installation process.

Also, try to provide the offline version of the installer just for your app, without including any prerequisites if your default version is a web-based installer.

However, there are situations when the installer also appoints some runtimes that are required by the application (e.g. .Net Runtime or Java runtime, etc). This will add an extra payload to the size of the installer. If the number of users that do not have that runtime already installed on their machines is small, you may want to add those runtimes from their online location - to reduce the size of the installer.

**Note:** In case any runtime is required for installation, the installer will download them at installation time.

# For enterprise deployment

## 5. Try not to bundle multiple packages/MSIs into a single one.

**Provide them all separately and specify the order of installation, if such an order is required.**

In Enterprises, the installers are usually installed one by one. So, if you have other installers bundled, specify them and provide all details that may be needed such installation order, if any.

All this will be managed by the sysadmin using a configuration management tool.

## 6. Provide a command-line switch to disable automatic updates

**If you have this support included in your application.**

**All IT pros need this option.**

In the Enterprise environment, updates are handled by the sysadmins, on-demand, and after the update is verified and confirmed. That's why they should consider disabling the auto-updates in an application.

Not only does it avoid additional support inquiries, but it also allows transparency and the option of disabling automatic updates through dedicated switches.

Otherwise, the installer will be repackaged and packaged into a new installer where updates are disabled (e.g. if the update is done through a service, then the new installer will not include that service).

Even if you have the update buried in your application, the PCs of an enterprise network operate behind a firewall and will have restricted access to your servers to update the application.

## 7. Include building rollback custom actions.

**If you have deferred custom actions (the code that modifies the system), also build rollback custom actions (to revert the changes if something else crashes during the installation).**

The rollback custom actions are performed during the installation rollback and their purpose is to reverse a custom action that has made changes to the system, e.g. a deferred custom action.

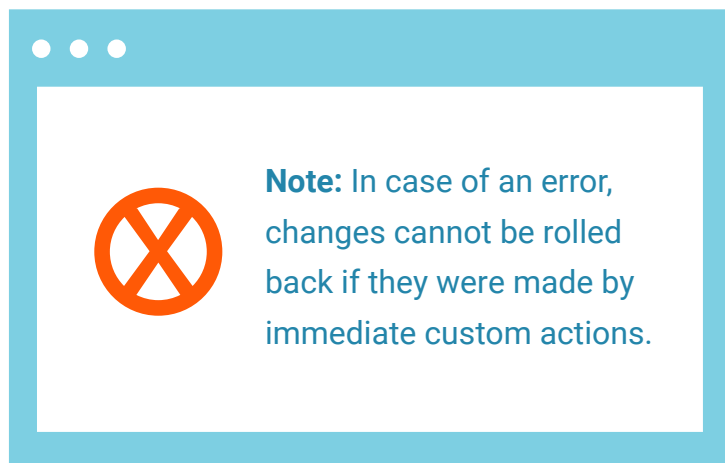
As you've probably noticed, a rollback custom action always precedes the deferred custom action as it reverses in the action sequence.

A rollback custom action should also handle cases where the deferred custom action gets interrupted in the middle of the execution. For example, if the user presses the [ Cancel ] button while the custom action is being executed.

## 8. Never modify system resources from immediate custom actions--only deferred ones.

Although it is convenient to make everything from an immediate custom action as it gives you access to the installer properties, the only changes that should be made are the ones that influence the installation process, such as setting and verifying properties.

**Don't Do It:** [Use Immediate CustomAction For Changing The System State \(advancedinstaller.com\)](https://www.advancedinstaller.com) 



## 9. Avoid triggering restarts during the installation.


The forced reboot during installation should be avoided as much as possible. Instead, you may want to instruct the user for a reboot at the end of the installatio

## 10. Store your application data in the Roaming AppData folder.

In an enterprise environment, users belong to a domain and there is a big chance they will use multiple devices to access the same account.



Information stored in the roaming appdata folder is synchronized between the devices, therefore the application user data will always be the same - avoiding the need of manually copying to multiple devices.

If you want to get more information about the reasoning behind this, check the following article: <https://www.advancedinstaller.com/appdata-localappdata-programdata.html> 

## Testing

### 11. Check the package for UI blocking custom actions on the InstallExecute sequence.

**And make sure you do not include error messages given by your custom actions.**

When the installer may be deployed from a central location (e.g. a server) there should not be any messages that require manual input. This is also available at both install and uninstall time.

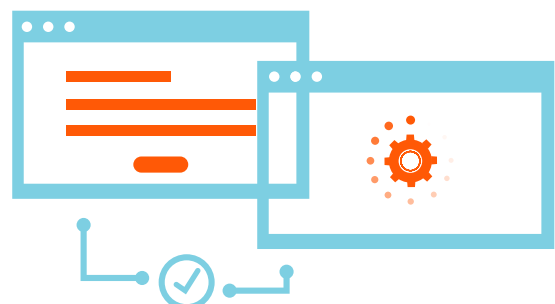
### 12. Test your package installation under the SYSTEM account, using PsExec tools.

In Enterprise, the installation needs to be successfully deployed under the SYSTEM account. LOCAL SYSTEM is the account performing the installation for most deployment tools e.g. GPO deployment, SCCM.

To ensure that your application is installed successfully, you need to act as the LOCAL SYSTEM account. To test this, use the PsExec.exe from Sysinternals which is the standard.

- Download PsTools from <https://download.sysinternals.com/files/PSTools.zip>
- Unzip the content and copy PsExec.exe to C:\Windows\System32
- Open a Command Prompt as admin
- Launch a new Command Prompt using PsExec.exe. By using PsExec.exe you will open the new Command Prompt in the System Context and the account doing all the operations will be the LOCAL SYSTEM account.

This is the best way to simulate how the installer is deployed to the machines in an enterprise environment.



```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>psexec.exe -s -i cmd.exe

PsExec v2.11 - Execute processes remotely
Copyright (C) 2001-2014 Mark Russinovich
Sysinternals - www.sysinternals.com

Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
nt authority\system

C:\Windows\system32>
```

### 13. Test your package in VMs (virtual machines), on multiple systems, to cover all your users.

Test your application under all supported environments. With Advanced Installer, you can use the Run in VM option to start the installation in a virtual machine, or you can simply copy-and-paste the installer into the virtual machine. The more Operating Systems are tested, the better.

## Documentation

### 14. Document the command line switches/options in a dedicated download page (or PDF) for the IT pros.

An installation manual is a best practice so that your users know how to handle the installation. Sooner or later, users will contact the support team for help, and having the information at hand can help quickly sort unnecessary support inquiries.

### 15. Document all GUI user inputs (if any) as command line parameters, with examples.

Provide documentation for all properties that need to be set. If gathering any user input, expose the properties that need to be set, with examples.

### 16. Document the list of requirements: prerequisites, minimum OS required, database connection.

If the application requires any runtimes (e.g. certain C++ version, .Net Framework version etc), it is convenient for the runtimes to be specified in the proper documentation.

The same applies for any database need. Documenting this type of information is mandatory in enterprise deployment.

## 17. Keep logs to ease debugging when your custom actions fail.

**Write to the MSI log from your custom actions or, even better, to the machine's event log, to make it easier to debug whenever one of your custom actions fails.**

When troubleshooting failed installations, the MSI log is the first one you should inspect. In the log file, the actions that fail will be highlighted.

It is useful to have more details of what happened there when troubleshooting failing custom actions.

If you're using PowerShell for custom actions, you can write to the MSI installation log by simply using the "Write-Output" cmdlet into your PowerShell custom action code.

For DTF C# custom action, use the session.log to write in the MSI log: `session.Log("Begin CustomAction1");`

If using VBScript, use the below code to write the event in the log file:

```
Function WriteToLog
Const MsgType = &H04000000
Set rec = Installer.CreateRecord(1)
rec.StringData(1) = CStr(Session.Property("AP-
PDIR"))
`rec.StringData(1) = CStr("Any type of message can
be wrote here")
Session.Message MsgType, rec
WriteToLog = 0
End Function
```

## That's it!

Ready to give it a try? Take advantage of [Software Packaging Checklist App](#). It will make your job easier, guide you through all the best practice checks and help you build a better installation package.



Following the basic rules and best practices for your application packaging software processes will ensure the quality of your product and give you fewer headaches regarding the support or possible bugs.

We hope this list gives you a clear overview of what you need to do to ensure smooth Enterprise use of your software product.

# **!Best Practices Validation Testing**

Advanced Installer generates builds in accordance with ICE Validation Standard and industry best practices brought together in over 15 years of constant contact with our Customers.

[Get Advanced Installer 30-Day Full-Feature FREE Trial](#) 

Authors:

**Bogdan Mitache**, VP of Product  
**Danut Ghiorghita**, Customer Support Lead