

**ADVANCED  
TECHNIQUES IN**

# MSI

**PACKAGING**



ALEXANDRU  
MARIN



POWERED BY  
**Advanced  
INSTALLER**

<b>Introduction</b>	<b>6</b>
What is MSI packaging?	6
What is the structure of an MSI?	8
The Difference Between Application Packaging and Repackaging	9
Application Repackaging with Transforms	10
Application Repackaging via Snapshot Method	11
Application Repackaging via PowerShell App Deployment Toolkit	12
Advantages of Application Repackaging	16
<b>Preparing for MSI Packaging</b>	<b>19</b>
Understanding Application Dependencies	19
What are Application Dependencies?	19
Assessing Application Compatibility	20
Assessing the compatibility of the VLC Media Player application	21
Per-user versus Per-machine installations	22
What are per-user installations?	22
What are per-machine installations?	23
Way forward for your installer	23
<b>Repackaging the Application Using Repackager</b>	<b>28</b>
What is Application Repackaging	28
Preparing for Repackaging	30
Capturing an Application with Repackager	33
Introduction to Repackager	33
What is the SnapShot Method ?	35
What is Session Monitoring ?	36
Repackager settings	37
Capture cleanup	43
Practical repackaging example on VLC Media Player	44
<b>Further application customization</b>	<b>58</b>
Advertised Shortcuts	58
Significance of Advertised Shortcuts	58
What is Self-Healing	58
Properties in MSI Packaging	60
Custom vlc settings	62
Custom settings implementation in the package	64
Scenario one: Advertised shortcuts	69
Scenario two: Active Setup	73
Registry classes	76
"COM" Registry	77



Interfaces	78
Type Libraries	80
File Type Associations	81
Manipulating of registry classes with Advanced Installer	85
COM Page	87
COM Properties	88
COM ActiveX Properties	90
DCOM Properties	92
Interface Properties	93
Type Libraries	95
COM+	97
File Associations Page	98
<b>Advanced MSI Packaging Techniques</b>	<b>102</b>
Custom actions	102
Dehardcode within files	105
How to Discover Hard-coded Files?	105
Delete empty directory	113
Delete empty directories with Custom Actions	113
Delete with VBScript	114
Delete with PowerShell	116
Delete empty directories with Advanced Installer	117
Process handling	119
Terminate Process in Advanced Installer	119
Terminate Process with VBScript	120
Terminate Process with PowerShell	123
Particular Terminate Process Scenario	125
Detect Process in Advanced Installer	129
Detect process with VBScript	130
Detect Process with PowerShell	132
Wait for Process with VBScript	134
Wait for Process with PowerShell	135
Firewall	135
Firewall rules with VBScript	136
Install/uninstall driver	144
DPInst	144
PnPUtl	144
Installing drivers with VBScript	145
Installing drivers with PowerShell	152
Installing drivers with Advanced Installer	156



Install Unsigned Drivers	157
Create a digital certificate by using the MakeCert tool	158
Create a .cat (catalog) file for the driver	160
Sign the catalog file using SignTool	162
Export the certificate from certstore manually	163
Install the certificate to Root and TrustedPublisher	164
Build the MSI	165
Installing unsigned drivers with Advanced Installer	168
DLL/OCX register/unregister	170
What is the Regsvr32 tool?	170
How to Register DLL/OCX with VBscript?	172
How to Register DLL/OCX with PowerShell?	175
How to Register DLL/OCX with Advanced Installer?	178
Write line in hosts file	179
Write in hosts file with VBScript	180
Write in hosts file with PowerShell	181
Working with Conditional Statements	183
Component Conditions	184
Launch Conditions	190
System Launch Conditions	191
Software Launch Conditions	191
Custom Launch Conditions	193
Custom Actions as Conditional Statements	194
Working with Dependencies	197
Creating Transform Files	201
Click-Once Apps	202
What are the challenges of repackaging a ClickOnce application?	202
How to Repackage ClickOnce Applications?	203
How to adjust the package?	205
Introduction to Services	210
Understanding Services in MSI	210
Benefits of Using Services in MSI	211
Creating and Configuring Services in MSI	211
Creating and configuring services with Advanced Installer	212
Service Installation	213
Control and Configure Operations	215
Failure Operations	219
Service Example	221
<b>Introduction to MSI Upgrades</b>	<b>224</b>





What is an MSI Upgrade?	224
Patch vs upgrade	225
Example patch for VLC	227
<b>Package Deployment</b>	<b>235</b>
Command lines	235
MSI Command Lines	235
PowerShell Command Lines	237
VBScript Command Lines	238
Deploy with SCCM	241
Deploy MSI via SCCM	242
Deploy EXE/VBscript/PowerShell via SCCM	246
Deploy with Intune	257
Deploy MSI via LOBA	259
Deploy EXE/VBScript/PowerShell via Win32	266
<b>Final Words</b>	<b>282</b>
<b>About the Author</b>	<b>283</b>



# Introduction

Welcome to “MSI Packaging In-Depth,” the definitive guide to building robust and secure installation packages. This book builds upon the foundation laid by “[Application Packaging Essentials](#)” providing an in-depth exploration of the MSI packaging process and its many intricacies.

In this book, we delve into advanced topics such as custom actions, dependencies, and repackaging, providing practical insights and real-world examples to help you navigate the complexities of the MSI packaging process. With expert guidance and a wealth of knowledge at your fingertips, you’ll be well-equipped to tackle even the most challenging packaging scenarios.

To get started, download [Advanced Installer’s 30-day full-featured free trial](#).

Designed for IT professionals and experienced packagers, “MSI Packaging In-Depth” is an essential resource for anyone looking to master the art of installation package creation. Join us as we explore the full potential of MSI packaging and unlock its many secrets.

## What is MSI packaging?

MSI packaging plays an essential role in the application deployment and installation process, and IT professionals must understand this technology well in order to produce reliable, fast, and scalable installation packages.

As outlined in the [MSI Packaging Essentials](#), the process of creating an MSI package involves **capturing** the files, registry settings, and other components of an application and organizing them into a standardized format to be installed on target systems.

Quite often, the package may include customizations, such as configuring the application to run in a particular environment, adding features or functionality, or applying patches or updates.

However, creating a dependable and effective MSI package involves more than simply capturing the files and registry settings. IT professionals need to be knowledgeable with a variety of tools and technologies, including, but not limited to, the Windows Installer service, [MSI tables](#), command-line switches, and scripting languages.



When developing MSI packages, IT professionals must follow best practices and industry standards, such as thoroughly testing the packages on various system configurations, using standard syntax and scripting languages, and documenting the package.

Imagine, for a second, an application packaging industry without best practices and industry regulations. What would it look like?

We'd likely see a chaotic and inefficient industry where each application packager would use different tools, methods, formats, and standards to create installation packages for Windows applications.

It would become a risky and unreliable industry where the quality and compatibility of the packages would vary widely, leading to errors, failures, conflicts, and security breaches during installation and operation.

Imagine this costly and wasteful industry where the packagers would spend more time and resources on troubleshooting, fixing, and updating the packages than on innovating and improving them.

Moreover, we'd be faced with a fragmented and isolated industry where the packagers would have no common platform or mechanism to share, learn from, or collaborate with each other or with other stakeholders in the IT ecosystem.

In short, it would be an industry that would fail to meet the needs and expectations of its customers, users, and society at large.

That is why best practices and industry regulations are essential for the application packaging industry to thrive and deliver value in the digital era.

And that's precisely why we're embarking on this journey once again: to gather and organize all our accumulated knowledge—both standard regulations and best practices. Our aim is to create an extensive resource that all application packagers can turn to whenever they need guidance.

Furthermore, IT professionals must stay current on the latest trends and technologies in MSI packaging, such as the use of custom actions, launch conditions, and transform files, among other things. By staying current with these technologies, IT professionals may design tailored, efficient, and reliable installation packages that can be simply delivered to target systems.



# What is the structure of an MSI?

MSI (Microsoft Installer) is a Microsoft Windows software installation package format. It is intended to make installing, configuring, and removing software applications on Windows machines easier.

An MSI package is, at its core, a database that contains all of the information required to install and configure a software application. This information includes installation files, registry settings, environmental variables, and other configuration options. The Windows Installer service, which is in charge of managing software installation and removal, can read and process the MSI package thanks to its design.

The MSI database is made up of several different tables that are used to store the various package components:

- **The File Table:** contains information about the files that will be installed,
- **The Component Table:** describes the individual components of the application,
- **The Feature Table:** defines the features and options that will be available during the installation process.

The advantage of the MSI database format is its capacity to standardize and streamline the software installation and configuration process. Developers can ensure that their applications are installed and configured correctly across diverse Windows systems by using this common package format and structure. Additionally, the database format supports advanced features such as rollback and error handling, enhancing the smoothness of installations and facilitating the resolution of any arising issues.

## What is Msiexec.exe?

Msiexec.exe, a command-line tool that the Windows Installer service uses, is a crucial part of the Windows Installer technology. This tool is used to install, configure, and uninstall software on Windows systems. It is a key component of the Windows Installer technology, which enables standardized and consistent software installation and configuration.

- When an MSI package is executed, the msiexec.exe executable file reads and interprets the information stored in the MSI database before installing or uninstalling the application.
- When you run the msiexec.exe file, it reads the package header information from the MSI database to determine the basic properties of the package, such as the product name,



version number, and vendor. It then processes the database's various tables and components to determine whether the application should be installed or removed.

The first step in the installation process is to determine which application components must be installed. This is accomplished by inspecting the MSI database's Component table and determining which components are marked for installation. After identifying the required components, the msiexec.exe file copies the necessary files and registers any required DLLs or other system components.

During the installation process, the msiexec.exe file consults the MSI database's various tables and components to determine the appropriate configuration options and settings for the application. This information includes registry settings, environmental variables, and other database-specified configuration options.

The msiexec.exe file can be used to remove or repair existing installations in addition to installing applications. When an MSI package is launched with the /uninstall switch, for example, the msiexec.exe file will use the information in the MSI database to remove all of the application's components and files.

Overall, the MSI database format is a necessary part of modern Windows software installation and management. The MSI format ensures that applications are installed and configured correctly by providing a standardized and structured approach to software installation, which can help IT professionals and end users alike reduce issues and support costs.

The MSI structure is covered more in-depth in our first [MSI Packaging Essentials ebook](#).

## The Difference Between Application Packaging and Repackaging

Packaging and repackaging are two key principles in application deployment and installation. While both of these processes are necessary for developing reliable and efficient installation packages, they take different approaches and different factors into account.

**Application packaging** refers to creating an installation package from scratch, including the application files, registry settings, and other components required for the application to execute properly.



Typically, the application packaging process involves using a specialized tool like Advanced Installer to capture the required files and settings. These are then organized into a package format, such as MSI, and the installation procedure on the target system is specified.

Ensuring the package includes all necessary components, is compatible with the target system, and can be installed smoothly and without conflicts is of utmost importance during the packaging process.

In contrast, **application repackaging** involves modifying an existing installation package to meet specific criteria. This method is commonly used when an existing package needs to be updated, modified, or customized to align with the requirements of a particular environment or system configuration.

Repackaging encompasses various actions such as modifying packages to delete or add components, changing registry settings, and applying patches or updates to existing packages. Creating packaging from scratch is often less complex than application repackaging, as it requires a deep understanding of the existing package, its components, and the tools and technologies involved in its production.

Repackaging involves several procedures and considerations. One of the initial tasks is to analyze the existing package, understanding its components and dependencies. To inspect and make alterations to the contents of the package, you may need to utilize specialized tools like the Advanced Installer tool. During this phase, it is crucial to identify any potential issues or conflicts that may arise when updating the package. These could include missing dependencies or incompatibility with the target system.

Once the existing package has been analyzed, IT professionals can begin the re-packaging process, which can be separated into two areas:

- Repackaging via transform files
- Repackaging via Snapshot method

## Application Repackaging with Transforms

The practice of adapting or updating existing installation packages to fit specific criteria is known as application re-packaging. Transforms, which are adjustments to an existing package that are implemented during installation without affecting the original package itself, are one technique for re-packaging.



Transforms can be made with specialist tools like the Advanced Installer's built-in Transform Maker and can include changes to registry settings, file locations, or other package components.

After creating the transform file, it may be applied to the existing package during installation using the proper command-line switches or deployment tools.

To apply the transformation file to the existing package, you have a variety of techniques at your disposal, such as command-line tools, batch scripts, and deployment tools. It is critical to extensively test the changed package to ensure that it installs and works well on the target system and does not cause new problems or conflicts.

One of the benefits of employing transforms for application re-packaging is that they do not modify the original package; thus, it remains unchanged and can be updated or modified in the future.

We will dive into this topic a [few chapters later in the book](#).

## Application Repackaging via Snapshot Method

Another approach to application re-packaging is the snapshot method. This method involves producing a new installation package by capturing the changes made to an existing installation on a test machine.

The snapshot approach captures changes to the file system, registry settings, and other application components during the installation process. It then creates a new package that can be further customized or deployed on other systems.

To use the snapshot method for application re-packaging, we need to prepare a test system with the present installation package and any prerequisites or dependencies. The modifications made to the system throughout the installation process are then captured using a snapshot tool. When the snapshot is finished, the captured changes are used to create a new package, which can then be customized further or deployed on other systems.

One advantage of the snapshot method is that any modifications made to the system during installation are captured and can be included in the new package.



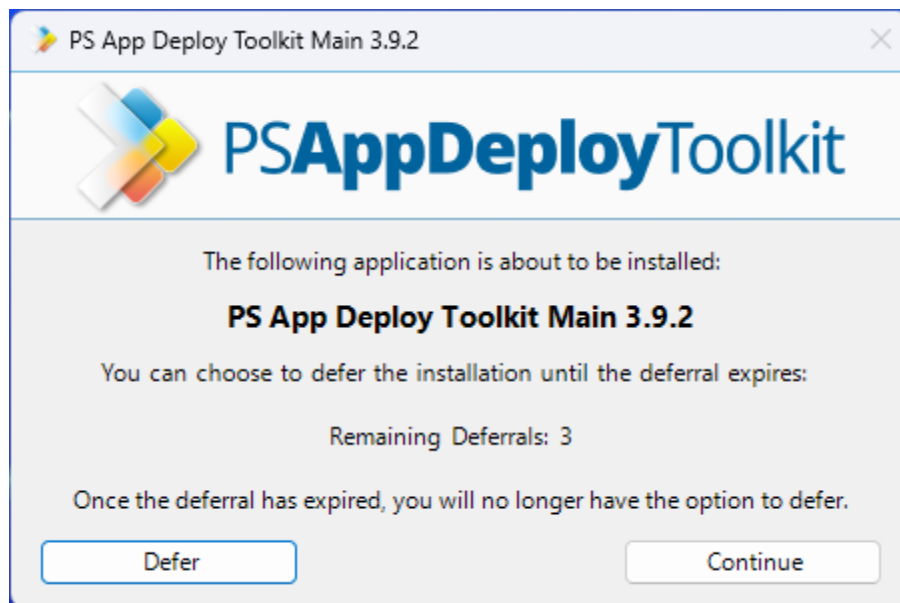
Furthermore, because the installation process can be captured and packaged, the snapshot method can be used to package applications that do not have an existing installation package.

Packaging and repackaging both necessitate a thorough understanding of the techniques and technology employed in the application packaging process.

A solid understanding of MSI, the Windows Installer service, and the tools used to create and modify packages is essential. Understanding the target system and the program's needs is crucial. It is equally important to consider how the application will be installed and configured on the target system.

## Application Repackaging via PowerShell App Deployment Toolkit

As mentioned in [MSI Packaging Essentials ebook](#), the PowerShell App Deployment Toolkit (PSADT) is a free and open-source framework designed to make enterprise application deployment tasks easier. It was created by Microsoft MVPs and is compatible with PowerShell 3.0 and higher.



The PowerShell App Deployment Toolkit includes a set of functions and tools for easily deploying and managing applications across multiple systems.





Administrators can use it to automate common application deployment tasks like installing, updating, and uninstalling applications.

The toolkit can deploy applications to traditional desktops, laptops, and servers, as well as virtualized environments like Citrix and VMware.

The PowerShell App Deployment Toolkit has a number of features that make it an effective application deployment tool. Among the key features are:

- **Easy to use scripting language:** PowerShell is a popular and powerful scripting language that is simple to learn and use. It is used in the toolkit which makes common application deployment tasks simple to automate.
- **Extensible framework:** The PowerShell App Deployment Toolkit is extensible, allowing administrators to customize and extend it to meet their specific requirements. As a result, it is a versatile tool that can be tailored to work with a wide range of applications and environments.
- **Application-specific functions:** The toolkit contains a set of functions that are specifically designed to work with popular applications like Adobe Reader, Google Chrome, and Microsoft Office. This simplifies the deployment and management of these applications across multiple systems.
- **Error handling and logging:** The PowerShell App Deployment Toolkit includes powerful error handling and logging capabilities.

Let's look at how to manipulate registry keys in PowerShell. While PowerShell provides straightforward cmdlets for modifying registry keys, there are a few things to keep in mind:

- Does the path to the registry key exist?
- Do we need to create a new registry key?
- Do we need to set a registry key?

The purpose of these questions is to assist you in developing your script. If the path to a specific registry key does not exist, you must create it. As a result, it's critical to test and handle this situation in your script.



```
$RegistryPath = 'HKCU:\Software\MySoftware\Scripts'
$Name      = 'Version'
$Value     = '2'
# Create the key if it does not exist
If (-NOT (Test-Path $RegistryPath)) {
    New-Item -Path $RegistryPath -Force | Out-Null
}
```

Following the creation of the registry path, we must determine whether the registry already exists or whether a new one is required. You have two options depending on the answer:

1. **If the registry already exists**, the [Set-Item cmdlet](#) can be used to set a specific registry value. As an example:

```
Set-Item -Path HKCU:\Software\MySoftware\Scripts\Version -Value "2"
```

2. **If the registry does not exist** and must be created, use the [New-Item cmdlet](#) to create a new registry item and its value. As an example:

```
New-Item -Path HKCU:\Software\MySoftware\Scripts\Version -Value "2"
```

The -Force parameter can be used to simplify the preceding steps, but it will make your script more complex with additional functions. The PowerShell App Deployment Toolkit can help with this. It includes custom cmdlets that simplify your script.

You can create or set a registry key using the Set-RegistryKey custom cmdlet that PSADT offers. Simply specify the registry key's exact location, and PSADT will handle the rest of the process for you.



PSADT has grown in popularity and is likely to be used in the majority of infrastructures at the moment, with its main advantage being that it allows IT professionals to customize a specific application installation without having to dive into the MST or repackaging areas.

For example, if you want to add small changes to your package, such as registry keys that disable automatic updates, files, or the EXE file can be installed silently, PSADT makes this much easier to accomplish, and you are essentially doing the same things as you would with repackaging via MST or Snapshot.

However, just like any other software tool, it has some potential weaknesses that IT professionals should be aware of, such as:

- **Limited Platform Support:** PowerShell App Deployment Toolkit is only compatible with Windows operating systems. This could be a problem for IT professionals who manage heterogeneous environments with a variety of operating systems.
- **Lack of Rollback Option:** PSADT, unlike MSI transforms, does not support rollback. This means that if something goes wrong during the deployment process, IT professionals may have to remove the application manually from each affected system.
- **Limited User Interface Customization:** While the PowerShell App Deployment Toolkit can be used to customize some user interface settings, it may not be as versatile as other tools for doing so.
- **Dependencies on PowerShell Versions:** To function properly, PowerShell App Deployment Toolkit requires PowerShell 3.0 or higher. IT professionals who manage systems using older PowerShell versions may need to upgrade their systems in order to use the PowerShell App Deployment Toolkit.

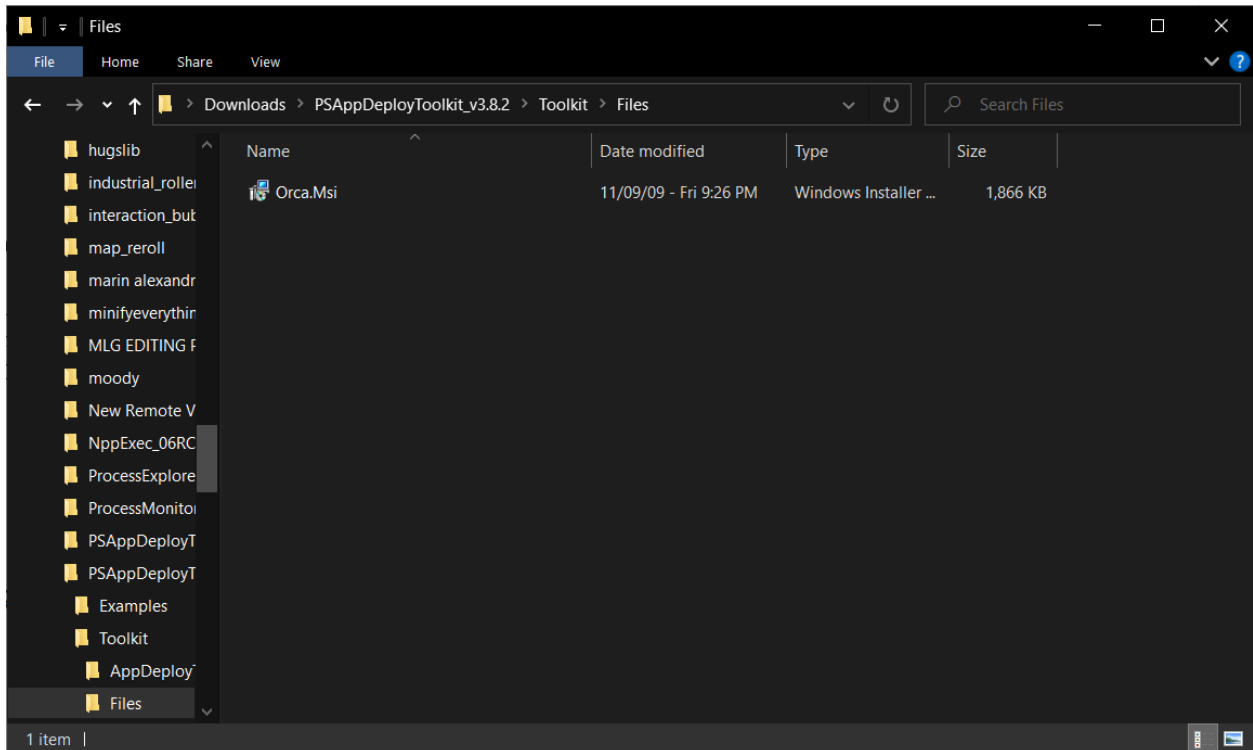
The PSADT structure is straightforward. In the root of the toolkit, you will find the following files:

- Deploy-Application.ps1
- Deploy-Application.exe
- Deploy-Application.exe.config

These are the files that you can run to begin the installation. The main PowerShell script that must be modified with the logical installation/uninstallation steps is Deploy-Application.ps1.

The Files folder will then hold all of your installation files, whether they are installers like MSI, MST, MSP, or other configuration files that you can copy later during installation.





The AppDeployToolkit includes not only the previously mentioned configuration files, but also the icons, banner, and main functions file.

If you want to add new functions to PSADT, you can either edit AppDeployToolkitMain.ps1 or create a new ps1 file and include it in the Deploy-Application.ps1.

The final folder is SupportFiles, where you can include any additional files that will be used in the main script. Technically, you can also use the [\\$dirSupportFiles](#) variable to run the installation of a specific file directly from the SupportFiles folder.

## Advantages of Application Repackaging

Application repackaging is mostly used to distribute programs in an enterprise environment, with many advantages over alternative methods. Repackaging involves capturing an application's data, registry settings, and other components and structuring them into an MSI package format that can be readily installed on target systems.



The application packaging process proves particularly useful when dealing with legacy applications or apps that were not originally designed for MSI packaging.

There are several advantages to using application repackaging as a method for deploying applications in an enterprise environment. These advantages include:

- Consistency
- Customization
- Scalability
- Standardization
- Efficiency

One of the primary advantages of application repackaging is **consistency**. IT experts can ensure that an application is installed consistently across all target systems by generating an MSI package for it. This decreases the possibility of problems and assures that the application works properly on all computers. When IT experts manually install software, the chance of human error increases, which can lead to inconsistency.

Another advantage of application repackaging is **customization**. IT professionals can change an application's installation package by repackaging it, such as setting the application to work in a specific environment, adding features or capabilities, or providing patches or updates. This level of customization is not possible when using off-the-shelf installation packages, which are often limited in terms of customization options.

Application repackaging also offers **scalability**. Repackaging an application allows IT professionals to deploy it on a big scale while maintaining control over the installation process and assuring the packages' dependability and efficiency. This makes it simple to deploy apps within an enterprise environment.

**Standardization** is another key advantage of application repackaging. IT professionals can use the MSI format to create standardized and industry-standard installation packages, ensuring that the packages can be deployed using a variety of tools and technologies, such as Microsoft System Center Configuration Manager (SCCM), Group Policy, or third-party deployment tools such as Advanced Installer. This makes it simple to deploy apps utilizing a wide range of tools and technologies without worrying about compatibility issues.

**Efficiency** is also an advantage of application repackaging. Repackaging an application can be a more efficient technique of distributing apps since it allows IT professionals to rapidly and simply construct installation packages without requiring access to the original installation media. When deploying apps, this can save a significant amount of time and resources, especially in big enterprise organizations.



Ultimately, application repackaging is an effective way for delivering apps in an enterprise setting. IT pros can use application repackaging to ensure that applications are installed consistently and reliably across all target platforms, while also modifying the installation package to fit unique needs.

Furthermore, application repackaging enables IT professionals to deploy applications on a large scale while maintaining control over the installation process and ensuring standardized and efficient packages.

Application repackaging offers IT professionals the benefits of consistency, personalization, scalability, standardization, and efficiency, making it a popular way of application deployment.



# Preparing for MSI Packaging

## Understanding Application Dependencies

Understanding application dependencies is an important part of the MSI packaging process because it allows IT professionals to ensure that the installation package contains all required components and can be installed and operated reliably on target computers.

Finding application dependencies can be hard and take a lot of time because you need to know a lot about the application and what it needs, as well as the technologies and tools that are used to find these dependencies and include them in the installation package.

### What are Application Dependencies?

Application dependencies encompass various components a program needs to function effectively. These include:

- **File Dependencies:** These are essential files, like DLLs, EXEs, or other resource files, that an application relies on. These files may be stored in various locations on the system and must be captured and included in the installation package in order for the application to execute correctly on target computers..
- **Registry Settings:** Many applications need specific registry keys or values set up to operate correctly. Gaining an understanding of an application's registry demands and ensuring these settings are part of the package is vital.
- **Prerequisite Software:** These are software components, such as database servers or runtime libraries, that must be present for the primary application to function. Their identification and incorporation into the package are necessary for the smooth running of the application on target systems.
- **Hardware Requirements:** Specific hardware prerequisites, like a particular processor type or amount of RAM, are necessary for some applications. Users must be aware of these requirements to ensure compatibility.

By understanding application dependencies and documenting them thoroughly, IT professionals can create installation packages that are reliable and efficient and can be installed easily on target systems.

Basically, application dependencies can be broken down into three areas:



- System dependencies
- Software Dependencies
- Other/Custom Dependencies

Advanced Installer offers a quick and easy way to declare dependencies into the MSI package and we go more in-depth on this topic in two chapters down below:

- [Working with Conditional Statements](#)
- [Working with Dependencies](#)

## Assessing Application Compatibility

Evaluating application compatibility is an important element of the MSI packaging process, especially for IT professionals in charge of big enterprise setups.

This process actively identifies and addresses compatibility challenges before application rollout. Such compatibility issues could lead to application malfunctions or failures, frustrating users and risking enterprise downtime.

One of the main reasons for evaluating application compatibility is to ensure that the application will work properly on the target system. This entails examining the program's system requirements in order to determine the minimum hardware and software needs for the application to work properly. To avoid compatibility concerns, IT professionals must ensure that the target system matches certain baseline criteria.

**Examining the application's dependencies** is another crucial component of determining application compatibility. This includes determining which third-party software components, such as runtimes or drivers, are required for the application to perform properly. Before installing the application, IT professionals must ensure that these dependencies are installed on the target system. Failure to install these dependencies may result in the program failing or functioning incorrectly.

Another important part of determining program compatibility is **compatibility testing**. This entails running the program on a variety of target systems in order to find any compatibility concerns. Prior to deployment, IT workers must test the program on various hardware and software configurations to discover and resolve potential issues.





## Assessing the compatibility of the VLC Media Player application

Let's take the example of assessing the compatibility of the VLC Media Player application. VLC is a popular media player that is widely used on Windows, Mac, and Linux operating systems.

Reviewing the application's system requirements is one of the first steps in determining VLC compatibility. VLC requires Windows 7 or later, macOS 10.7 or later, and a kernel version of 2.6.32 or later on Linux. To avoid compatibility concerns, IT professionals must ensure that the target system matches certain baseline criteria.

Another critical part of determining VLC compatibility is to examine the application's dependencies. To play various media kinds, VLC requires specific codecs and plugins, and IT professionals must guarantee that these dependencies are installed on the target machine. Failure to install these dependencies may result in the program failing or functioning incorrectly.

Examining the application's dependencies is also an important component in determining VLC compatibility. VLC requires appropriate codecs and plugins to play various media types, and IT professionals must ensure that these dependencies are installed on the target PC. If certain requirements are not installed, the software may fail or function poorly.

When evaluating VLC's compatibility, keep in mind the impact of any updates or patches that may be released for the application. These upgrades may have an influence on the application's compatibility with the target system, and

Consider the impact of any updates or patches that may be released for the application when evaluating VLC's compatibility. These updates may impact the application's compatibility with the target system.

Application compatibility testing has become much easier with time because nowadays organizations usually only run on a single OS version (unlike in the past where we had multiple OSes like XP, Vista, 7 in the same infrastructure) and patching/IPU (in place upgrades) have become more easy to manage and are released on a steady basis from Microsoft. Basically IT Professionals must test the application compatibility on a single system with 1 or 2 branches.



In conclusion, assessing application compatibility is a critical aspect of the MSI packaging process, particularly for IT professionals responsible for managing large enterprise environments.

IT workers can assure that an application will run appropriately on target systems by thoroughly testing application compatibility, lowering the risk of downtime and user displeasure. The use of tools and technology to automate the assessment and resolution of compatibility issues can improve efficiency and effectiveness.

It is critical to keep up with the current trends and technologies in the field of application compatibility to ensure that MSI packages are dependable, efficient, and simple to maintain.

## Per-user versus Per-machine installations

One of the most important decisions that IT professionals must make when deploying software on Windows machines is whether to use a per-user or per-machine installation.

Both installation methods have advantages and disadvantages, and selecting the correct method can have a significant impact on application performance, security, and user experience. In general, in managed environments, the per-machine installation is preferred.

### What are per-user installations?

**Per-user installations** are intended to install an application only for the current logged-in user. This means that the application will be available only to that user and will not be accessible to other users who log in to the same machine.

Per-user installations are commonly used for applications that aren't meant to be shared by multiple users or machines, such as personal productivity tools or small utilities.

A great mention in the per-user area in terms of installer technologies is MSIX. MSIX is a newer packaging format introduced by Microsoft in Windows 10 version 1709. Unlike traditional MSI packages, which support both per-user and per-machine installations, **MSIX is only intended for use with per-user installations.**

MSIX packages are specifically designed to support per-user installations, with an emphasis on providing end-users with a streamlined and reliable installation experience. The format makes use of a number of advanced features and technologies, such as containerization and



virtualization, to ensure that applications are correctly installed and configured, even in complex and diverse IT environments.

One of MSIX's primary advantages is that it provides a more secure and efficient method of installing and managing applications on Windows machines. MSIX packages are isolated from other applications on the system by using containerization technology, which helps to prevent conflicts and compatibility issues.

For more details about the MSIX technology we recommend you have a look over our [MSIX Packaging Fundamentals free ebook](#).

## What are per-machine installations?

In contrast, **per-machine installations** are intended to install an application for all users who log in to a specific machine. This means that the application will be accessible to all users of the machine, regardless of who installed it originally.

Per-machine installations are typically used for enterprise applications that require all users in an organization to have access to them, such as office suites or line-of-business applications.

Each installation type has advantages and disadvantages. Because they do not require administrative privileges and can be installed by the user without assistance from IT, per-user installations are frequently simpler and easier to manage.

Per-user installations, on the other hand, can be more difficult to manage in a large-scale environment because they necessitate individual installation and configuration for each user.

Per-machine installations are more complicated and necessitate administrative privileges. They are also less secure because they grant access to the application and its files to all users on the machine.

Per-machine installations are easier to manage in a large-scale environment because they can be deployed and configured once for all users.

## Way forward for your installer



Depending on the situation, there are a number of actions you can take when it comes to repackaging an application:

- **MSI:** If the application comes as an MSI, the way forward to further customize and deploy the application would be to create transform files (MST)
- **EXE:** There are three possibilities on how to advance with an EXE installer. The most popular one is to repackage it via the snapshot method, but there are cases where the EXE installer actually contains an embedded MSI which is extracted and then installed, and the final case is where the application is so complex that it's best to install it silently via a wrapper method

The MSI scenario is quite straightforward so we would go ahead and have a look over the EXE scenario.

The first step is to determine whether the EXE requires repackaging or if it contains an embedded MSI. There are several ways to determine whether an EXE installer contains an embedded MSI:

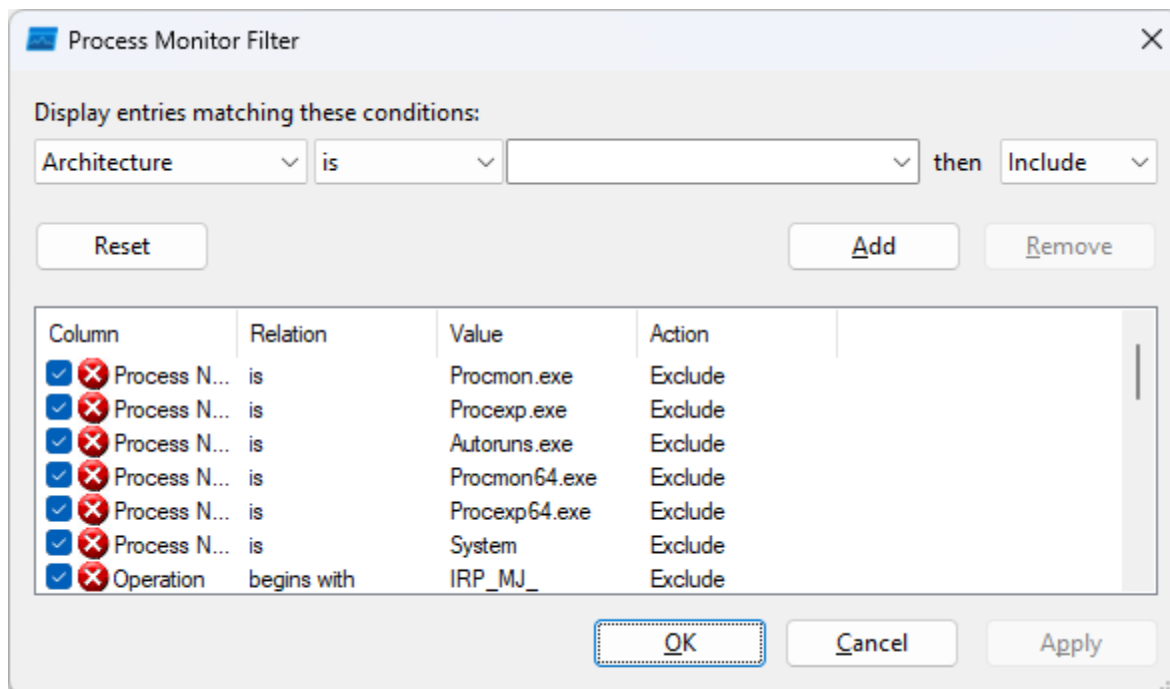
- Open Task Manager and check if the msixexec service appears in the list during the installation
- Check the [Uninstall registry](#) and see what uninstall command has appeared or if the key name resembles an MSI Product Code
- Use [Process Monitor](#)

Process Monitor has grown in popularity among IT professionals for not only detecting whether an MSI is embedded in another MSI, but also for detecting additional system changes performed by applications, debugging applications, and so on.

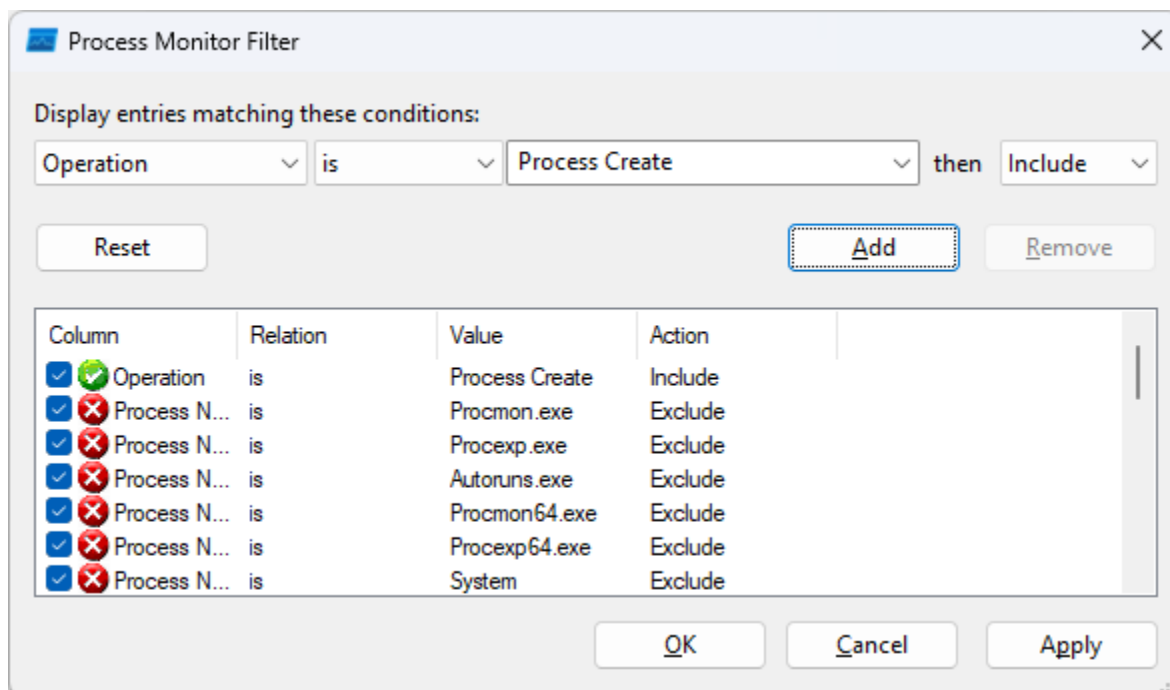
To detect if an EXE contains an embedded MSI with Process Monitor, follow these steps:

- Download Process Monitor: Visit the official website of Process Monitor (<https://learn.microsoft.com/en-us/sysinternals/downloads/procmon>) and download the latest version of the tool.
- Launch Process Monitor: Extract the archive and start Process Monitor
- Configure filters: Before monitoring the installation process, it's helpful to set up filters to narrow down the captured events and focus on the relevant activities. Click on the "Filter" menu, and then select "Filter..." to open the Filter dialog box.





Configure filter rules: In the Filter dialog box, specify the conditions to filter the events. To detect if an EXE installer contains an embedded MSI, you can set up the following filters: In the "Display entries matching these conditions" select "Operation", "is", "Process Create" and then "include". Click on Add.



These filters will record events related to the EXE installer and any MSI files used during its



execution.

- **Begin capturing events:** After configuring the filters, select "Start" from the "Capture" menu, or press the Ctrl + E shortcut to begin capturing events.
- **Run the EXE installer:** Launch the EXE installer you want to analyze. The Process Monitor will start capturing events in real-time.
- **Analyze captured events:** When the installation is finished, or when you want to stop capturing events, go to the "Capture" menu and select "Stop," or press the Ctrl + E shortcut. In the main window of Process Monitor, a list of captured events will be displayed.
- **Filter and analyze events:** To analyze the captured events, use the various columns and filter options in the Process Monitor window. Look for file system operations involving MSI files (with the extension ".msi") and registry operations involving MSI execution (ending with ".msiexec.exe"). These events indicate that an MSI is embedded within the EXE installer.

For example, if we install Tableau Reader and start the Process Monitor tool and follow the above steps, we will have the following operations captured:

[illegible]

We're most interested in the Path and Detail columns. Looking at what TableauReader.exe does, it first extracts the installer data into a temporary directory in the %temp% directory, then installs the Visual C++ Redistributables 2013, followed by the Tableau Reader. However, if we look closely, we can see that the msiexec.exe service is called, indicating that the exe file is extracting the file in that %temp% location.



This is an example where you will need to create a transform file (MST) to further customize the Tableau Reader MSI installation, but also to use the [dependencies](#) to declare that Visual C++ Redistributable 2013 is needed in order for the application to function properly.



# Repackaging the Application Using Repackager

## What is Application Repackaging

Repackaging plays an important role in organizations' software lifecycle management by enabling efficient deployment, customization, and maintenance of software applications.

Capturing an existing software installation, transforming it into an installer package, and customizing it to meet specific deployment requirements are all part of the process. While adhering to best practices and compliance standards, repackaging ensures consistent and reliable installations.

Some of the key concepts in Repackaging consist of:

- **Application Capture:** Monitoring the changes made to the system during the installation is part of the process of capturing an existing software installation. It records data such as file and registry changes, system settings, and dependencies. The collected data is used to create the installer package.
- **Transformation:** Once the application is captured, it needs to be transformed into an installer package format. This involves converting the captured data into a structured format compatible with industry-standard installation technologies such as Windows Installer (MSI).
- **Customization:** Customizations to the captured application can be applied through repackaging. Modifying installation settings, adding registry entries, configuring application parameters, defining file associations, and incorporating specific deployment requirements are all examples of this. Customizations ensure that the packaged application adheres to organizational policies as well as user preferences.
- **Quality Assurance:** Quality assurance is critical in repackaging, as it is in any software development process. It entails putting the packaged application through its paces to ensure its functionality, compatibility, and adherence to organizational standards. Installation testing, user acceptance testing, and compatibility checks with various operating systems and configurations are examples of quality assurance activities.
- **Version Control:** Version control of packaged applications is also involved in repackaging. Version control ensures that changes to installer packages are tracked and





documented properly over time. It allows organizations to revert to previous versions if problems arise, as well as keep a history of package changes for auditing and compliance purposes.

Although repackaging might take additional time to perform than to just take the MSI directly from the vendor website, there are many benefits on walking on the repackaging path:

- **Streamlined Deployment:** By providing standardized installer packages that can be easily distributed across multiple machines, repackaging simplifies the deployment process. It ensures consistent installations and saves time and effort on manual installations and configurations.
- **Customization and Configuration:** By incorporating specific customizations, configurations, or additional components, repackaging enables tailor-made installations. This allows organizations to meet specific deployment needs and deliver applications that are optimized for their environments.
- **Compatibility and Dependency Management:** Repackaging aids in the resolution of compatibility issues and the management of application dependencies. It ensures that applications run smoothly on target machines without conflicts or missing prerequisites by capturing and packaging the necessary components and dependencies.
- **Software Maintenance and Updates:** Repackaging makes software maintenance and updates easier. It enables organizations to efficiently package and distribute new versions or patches of applications across their infrastructure, ensuring consistent and timely updates.
- **Standardization and Compliance:** By adhering to established packaging practices and guidelines, repackaging promotes standardization. It ensures that applications are packaged consistently, in accordance with compliance standards and organizational policies. This aids in the maintenance of a secure and compliant IT environment.

As with any other area in IT, there are some best practices in repackaging that we suggest you follow:

- **Documentation:** Documenting the repackaging process is critical for keeping track of the steps taken, the customizations used, and any known issues or workarounds. Documentation is essential for troubleshooting, knowledge sharing, and ensuring consistent repackaging practices.
- **Testing and Validation:** It is critical to thoroughly test and validate packaged applications to ensure their functionality and compatibility. It is recommended that comprehensive testing be performed on various operating systems, configurations, and deployment scenarios before deployment to identify and address any potential issues.
- **Version Control and Change Management:** The use of version control and change management processes aids in the tracking and management of changes made to



packaged applications. It ensures proper documentation, promotes collaboration, and allows organizations to revert to previous versions if necessary.

- **Packaging Standards and Templates:** Setting packaging standards and using predefined templates can help to speed up the repackaging process. Standardized practices ensure consistency and efficiency while lowering the likelihood of errors or variations in the final packages.
- **Collaboration and Knowledge Sharing:** Collaborating and sharing knowledge among repackagers within the organization promotes best practices and fosters continuous improvement. Sharing knowledge, experiences, and troubleshooting methods can improve the repackaging process and ensure better results.

We have covered the best practices of repackaging more in-depth in our first [MSI Packaging Essentials Ebook](#).

Repackaging is a critical component of software lifecycle management, allowing organizations to deploy and manage applications more efficiently. Capturing, transforming, and customizing software installations to meet specific deployment requirements is involved. Organizations can streamline the repackaging process, ensure quality and compliance, and achieve consistent and reliable software deployments across their IT infrastructure by following best practices.

## Preparing for Repackaging

It is critical to adequately prepare before beginning the repackaging process to ensure a smooth and successful outcome. Understanding the software to be packaged, gathering the necessary resources, and laying a solid foundation for the repackaging project are all part of proper preparation. This chapter delves into key considerations and best practices for repackaging preparation.

Probably the most important part of software repackaging is to properly understand the application which you are going to recapture. In this process you can have a look at:

- **Application Documentation:** Begin by compiling detailed documentation for the software application to be repackaged. This includes installation guides, user manuals, release notes, and any other documentation that is relevant. Understanding the functionality, dependencies, and requirements of the application will make the repackaging process easier.



- **Application Dependencies:** Determine the application's dependencies, which may include runtime libraries, database connectors, frameworks, or any other components required for proper functionality. Make a list of these dependencies and include them in the repackaging process.
- **Licensing and Legal Considerations:** Examine the software's licensing agreements and ensure that licensing requirements are met during repackaging. Recognize any restrictions, limitations, or specific guidelines that may apply to redistributing the software. If necessary, seek the advice of legal and licensing experts to ensure compliance with licensing requirements.

The next step is to make sure that you have all the necessary resources for the repackaging operation:

- **Repackaging Tools:** Locate and purchase the necessary repackaging tools. Advanced Installer, AdminStudio, Wise Package Studio, and Orca are all popular repackaging tools. Examine the features, compatibility, and ease of use of these tools to find the best fit for your repackaging needs.
- **Virtual Machines or Test Environments:** To perform the repackaging process, set up virtual machines or dedicated test environments. These environments enable isolated testing and validation of repackaged applications while not interfering with the production environment. Make sure the test environment closely resembles the target deployment environment.

Working in a clean and controlled environment is one of the most important aspects of a successful application repackaging process. A clean environment improves the accuracy, reliability, and repeatability of the repackaging process. This section discusses the importance of a clean environment for repackaging and provides guidelines for creating one:

- **Isolation and Control:** Working in a clean environment allows you to isolate and control the repackaging process. You reduce the risk of unintended consequences, conflicts with existing software, or interference with critical system settings by separating the repackaging activities from the production environment. It allows you to concentrate solely on the repackaging task without being distracted by outside factors.
- **Avoiding Interference:** Interference from existing software installations, system configurations, or conflicting components is avoided by repackaging applications in a clean environment. It ensures that the repackaged application remains independent of the host system and does not inherit any unwanted dependencies or settings. This isolation improves the repackaging process's dependability and predictability.
- **Minimizing Variability:** Variability caused by different system configurations or inconsistent installation states can be reduced in a clean environment. Repackaging in a controlled environment ensures consistency across multiple repackaging sessions,



lowering the possibility of errors, inconsistencies, or unexpected behavior in the repackaged application.

- **Eliminating Conflicting Dependencies:** A clean environment enables you to eliminate potentially conflicting dependencies in the production environment. By starting with a clean system or virtual machine, you can install only the components required for the repackaged application. This method creates a clean slate and ensures that the repackaged application is self-contained and conflict-free.
- **Reproducibility and Troubleshooting:** A clean environment improves reproducibility and makes troubleshooting easier. Working in a clean environment allows you to identify and isolate the specific factors causing the problem if problems arise during the repackaging process. Without the complexities introduced by an existing system, it becomes easier to troubleshoot, diagnose, and resolve issues.

More about clean images can be found in our first [MSI Packaging Essentials Ebook](#).

To set up a clean environment, some steps must be considered:

- **Virtual Machines:** Consider using virtual machines (VMs) to create clean and isolated repackaging environments. Virtual machines enable you to create snapshots of pristine system states, revert to a clean state as needed, and easily clone environments for parallel repackaging efforts. Each VM can be dedicated to a specific repackaging project, ensuring that each application runs in a clean and controlled environment.
- **Clean System Image:** Create a clean system image or baseline configuration that includes the bare minimum of repackaging components. This image can be used to kick off repackaging efforts. You can maintain a controlled environment throughout the repackaging process by keeping the baseline system clean and avoiding unnecessary installations or customizations.
- **Repackaging Workstations:** Set aside specific workstations for repackaging activities. These workstations should be kept separate from machines used in production or on a daily basis. By repackaging on dedicated hardware, you reduce the risk of unintended interference from unrelated activities or software installations.
- **Sandbox or Container Solutions:** To create isolated environments for repackaging, consider using sandbox or container solutions. These solutions provide a virtualized environment that isolates the repackaging process from the host system. Sandboxing tools such as Sandboxie and container technologies such as Docker provide a controlled and isolated environment for repackaging applications.

Advanced Installer supports [repackaging in Docker Images](#).



Starting with version 20.7, Advanced Installer supports repackaging in Windows Sandbox.

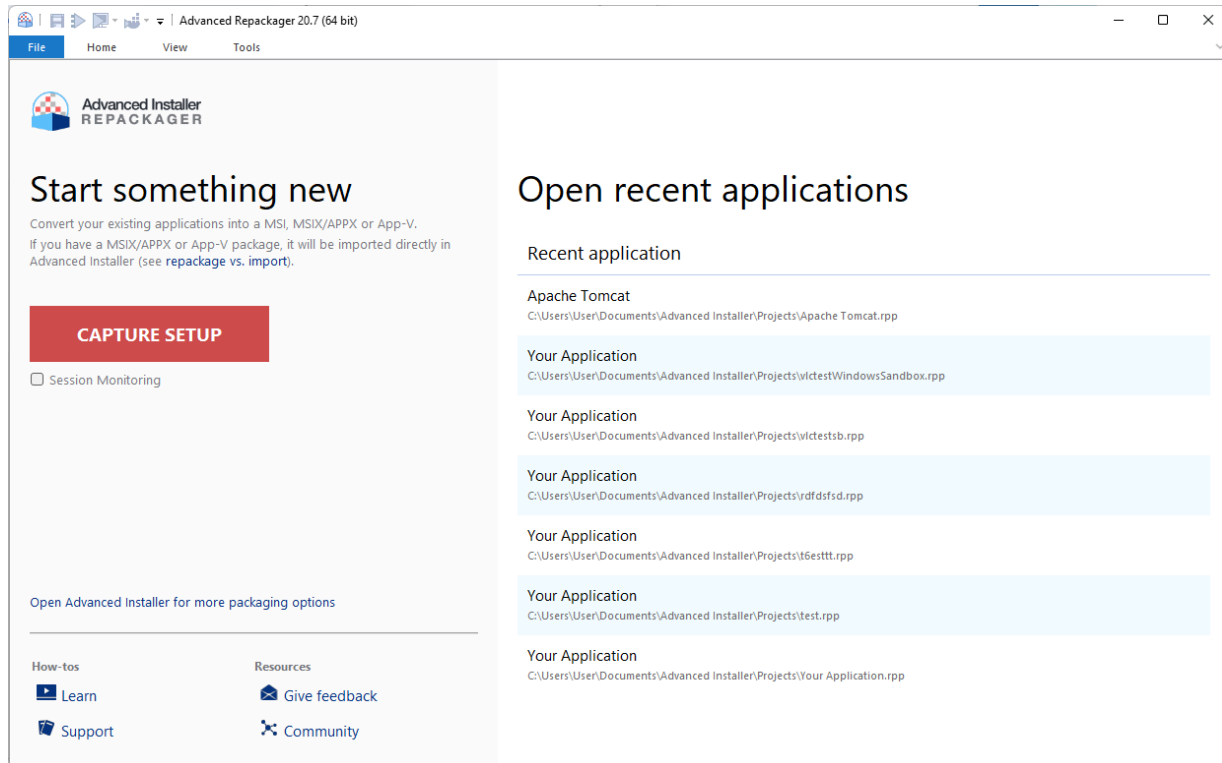
By creating a clean environment for repackaging, you improve the process's accuracy, reliability, and repeatability. A clean environment reduces interference, eliminates conflicts, and creates a safe environment for repackaging activities. Investing in a clean environment, whether through virtual machines, clean system images, or sandboxing solutions, ensures the success of your repackaging projects.

# Capturing an Application with Repackager

## Introduction to Repackager

Advanced Installer includes a powerful repackaging solution for converting traditional installations into MSI packages. Advanced Installer Repackager's user-friendly interface and extensive feature set make it simple to create dependable and professional MSI packages.





## Key Features of Advanced Installer Repackager:

- **Automated Repackaging:** The Repackager tool in Advanced Installer automatically captures changes made during application installation and generates an MSI package.
- **Snapshot Technology:** The Repackager monitors system changes during the installation process using advanced snapshot technology. It records file and registry changes, COM registrations, shortcuts, services, and other activities.
- **Intelligent Conflict Resolution:** Advanced Installer handles installation conflicts and automatically resolves issues such as file clashes, registry conflicts, and component dependencies.
- **Customization and Configuration:** Using Advanced Installer's extensive feature set, repackaged packages can be customized. Custom actions, installation conditions, installation sequences, and configurations can all be added.
- **Compatibility and Validation:** Advanced Installer ensures Windows compatibility and validates repackaged packages to ensure compliance with best practices and industry standards.

We provide the ability to connect Advanced Repackager to multiple VM platforms for the VM option:

- [VMWare machine](#)



- [Hyper-V machine](#)
- [Oracle VM VirtualBox machine](#)
- [vSpehere](#)
- Windows Sandbox
- [Docker](#)

## What is the SnapShot Method ?

The snapshot method is a popular technique for creating installer packages in application repackaging. Advanced Repackager, an Advanced Installer feature, makes repackaging easier by providing an intuitive interface and powerful capabilities. Let's take a closer look at the snapshot method and how it's used in Advanced Repackager.

The snapshot method involves capturing changes made to a system during application installation. These modifications are then packaged into an installer, which can be distributed and installed on other machines. This method allows you to create custom installer packages without requiring extensive manual configuration.

Advanced Repackager is a feature-rich tool that automates the process of capturing changes and generating installer packages, making the snapshot method easier to use. Here's a rundown of the main steps in using Advanced Repackager:

- **Preparing the Repackaging Environment:** Make sure your system environment is clean before beginning the repackaging process. This includes removing any previous installations or remnants of the repackaged application. This step aids in the capture of a clean snapshot.
- **Starting the Snapshot Process:** Select the "Create New Project" option in Advanced Repackager. To begin the snapshot process, select the "Snapshot" method. This will create a backup of your system prior to installing the target application.
- **Installing the Application:** Install the application that you want to package in the usual way. The changes made during the installation process will be monitored and recorded by Advanced Repackager.
- **Capturing the Changes:** Once the installation is complete, Advanced Repackager will compare the current state of the system to the earlier baseline snapshot. It will recognize and record all changes made, including file and registry changes, system settings, services, and more.
- **Reviewing and Customizing the Snapshot:** Advanced Repackager generates a detailed report of the changes that were captured. You can go over the captured data and make any necessary changes or customizations. Excluding unwanted files or registry entries,



configuring shortcuts, defining installation prerequisites, and other tasks may be included.

- **Generating the Installer Package:** Advanced Repackager generates an installer package based on the modifications after reviewing and customizing the captured changes. Additional installation options, such as installation location, user prompts, and uninstallation options, are available. The installer package that results is ready for distribution and deployment.

## What is Session Monitoring ?

Session monitoring is a feature in Advanced Repackager that allows you to observe and track changes made to the system without the need for an application installer. It records system changes like file and registry changes, environment variable updates, service installations, and more, providing detailed information about the installation activity.

The session monitoring feature is critical in the repackaging process because it enables you to understand how the application interacts with the system and identifies the changes needed to create a clean and reliable installer package. You can capture all of the necessary changes made by the application and ensure that they are accurately included in the final package by monitoring the customization session.

Here are some key aspects of session monitoring in Advanced Repackager:

- **Real-time monitoring:** Advanced Repackager monitors the system in real-time while an application is being modified, capturing all changes made by the application. It tracks the changes and compiles them into a detailed report that can be analyzed later.
- **Detailed change tracking:** Session monitoring records a variety of system changes, such as file additions, modifications, and deletions, registry changes, environment variable updates, service installations, and more. It provides a detailed breakdown of each change, allowing you to comprehend the installer's impact on the system.
- **Customizable monitoring filters:** During the monitoring process, Advanced Repackager allows you to define filters to focus on specific types of changes. You can choose to exclude specific file types, folders, or registry keys from monitoring, giving you more control over the results.
- **Post-monitoring analysis:** After the installation session is complete, you can review the captured changes and assess their relevance to the repackaged application. Advanced Repackager has an easy-to-use interface for navigating through the captured changes, allowing you to accept or reject specific modifications based on their significance.





## Repackager settings

Advanced Repackager provides a number of customization options for the repackaging operation, allowing you to tailor the process to your specific requirements. You can improve the efficiency and effectiveness of your repackaging workflow by selecting the appropriate actions, scanning options, and utilizing system data. Here are a few Repackager options:

### Actions:

- Prompt and wait before installing packages: Before the installation process can begin, this option requires user input. To proceed, you must press the ENTER key in the CLI window.
- Prompt and wait after installing packages: After the installation process is complete, this option requires user input. To proceed, press the ENTER key in the CLI window, as you did with the previous option.
- Detect issues that may interfere with the scanning operation: The Check Machine State dialog will be displayed before beginning the repackaging operation. This dialog provides information about potential problems that may arise during the scanning process. It makes suggestions for resolving these issues.
- Automate package installation by invoking UI controls to walk-through the installation steps: This option enables the automation of UI controls to automatically navigate through the installation steps. It is important to note that relying solely on this option is not recommended for more complex installations.

### Scanning Options:

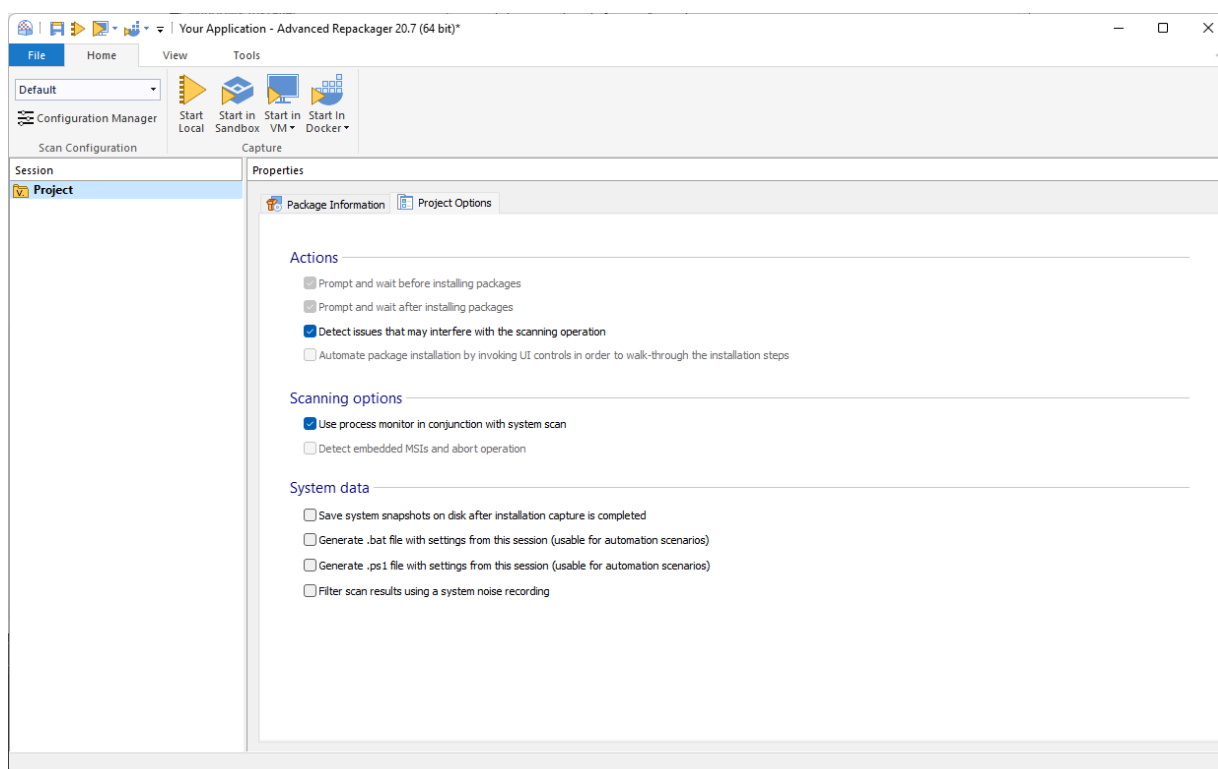
- Use process monitor in conjunction with system scan: This option combines the system scan with limited process monitor functionality. While it can improve scanning, it may lengthen the overall operation time.
- Detect embedded MSIs and abort operation: The repackaging operation will be aborted if it encounters embedded MSIs within the package. This contributes to a clean and accurate repackaging process.

### System Data:

- Save system snapshots on disk after installation capture is completed: After the installation capture is complete, this option saves an initial system snapshot to disk. The saved snapshot can be used in subsequent repackaging operations, removing the need for the Repackager to generate a new initial snapshot. This significantly reduces the time required to complete future repackaging processes.



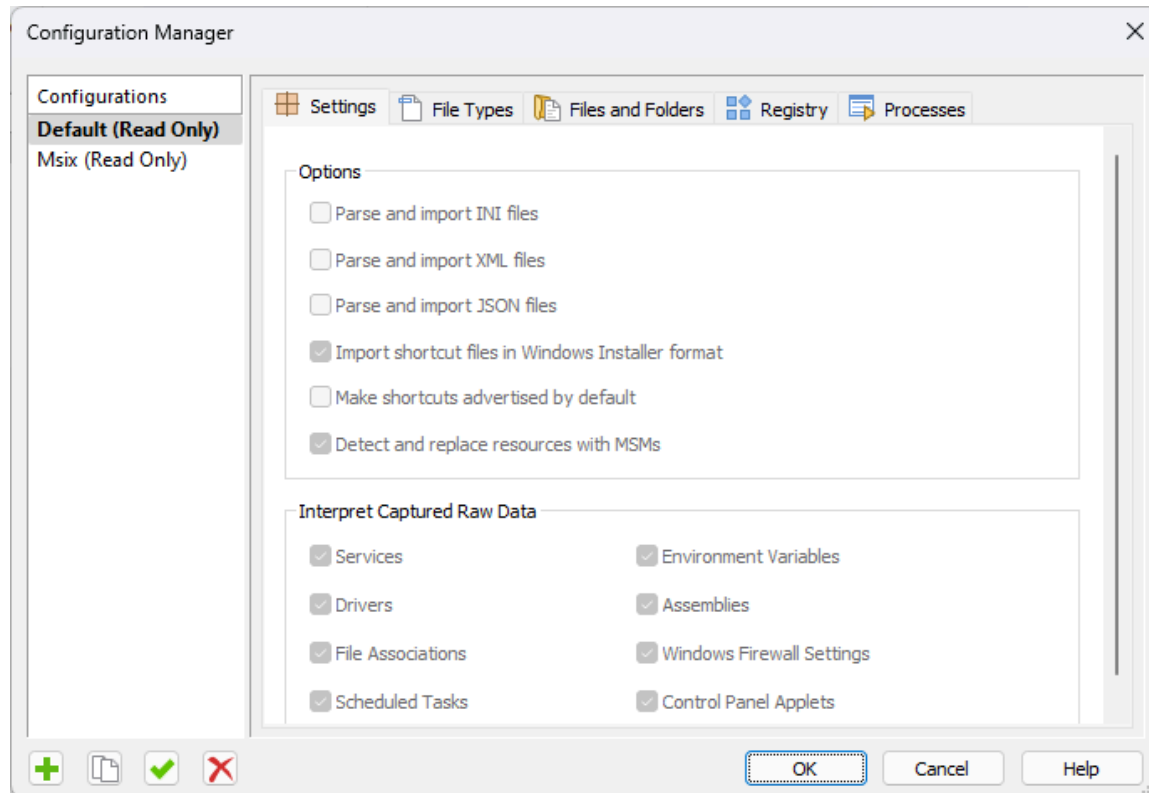
- Generate a .bat file with settings from the session: Enabling this option creates a .bat file in the repackaging operation's output folder. The .bat file automates the repackaging process, making it simple to replicate the same settings in future repackaging sessions.
- Filter scan results using a system noise recording: This option aids in identifying and documenting changes made by the system or third-party applications that may interfere with the repackaging process. The recorded noise is saved as a rpknnoise file, which can be accessed later.
- Perform a system noise scan before repackaging: This option performs a noise scan prior to the repackaging operation to identify system changes.
- Use a previously generated noise recording: By selecting this option and browsing for an existing noise recording file, you can load and utilize a previously saved noise recording.



Advanced repackager also includes a comprehensive set of filter settings that allow you to precisely and precisely fine-tune your installation packages. In this article, we will look at the various aspects of filter settings and how they can be used to improve your packaging process.



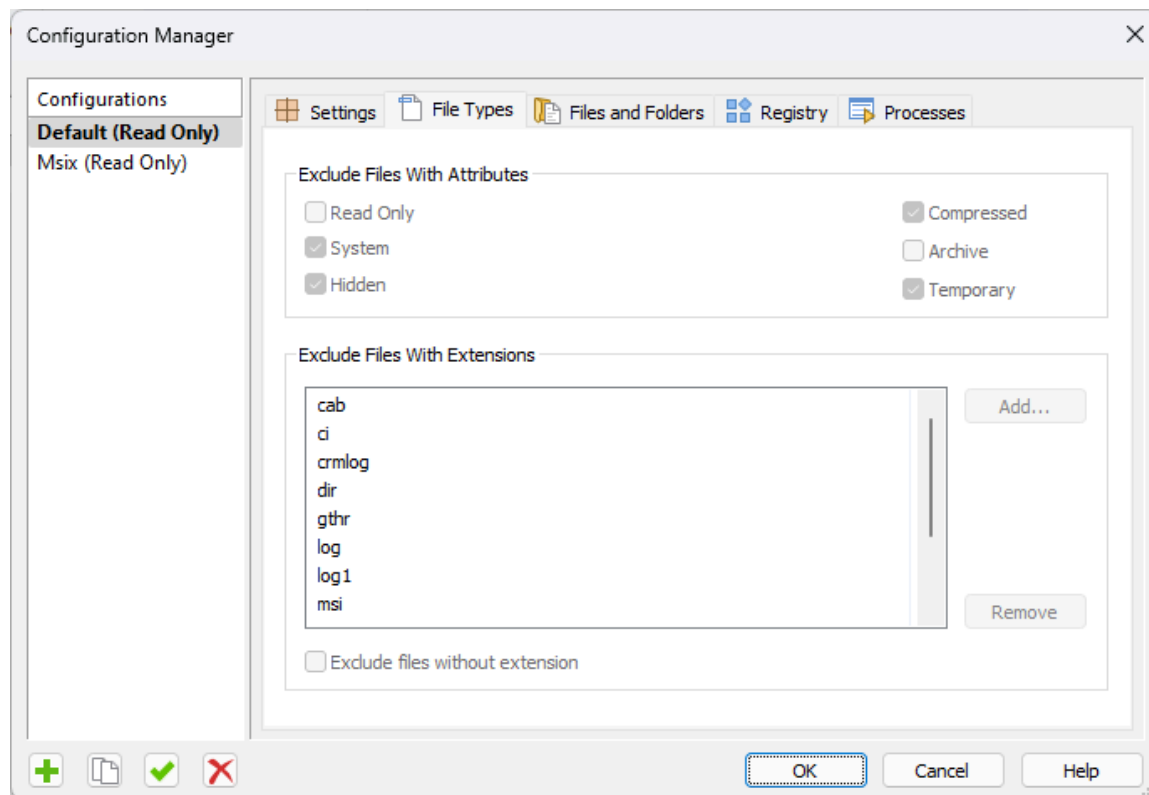
The Edit Filter Settings page is where you can configure filters for different elements in your installation package. It has an easy-to-use interface for managing filters for files, file folders, registry entries, and processes.



You can use filter settings to include or exclude specific elements based on a variety of criteria. During the installation process, you can define filters to include or exclude files, file folders, registry entries, and processes. This allows you to tailor the installation to your specific needs.

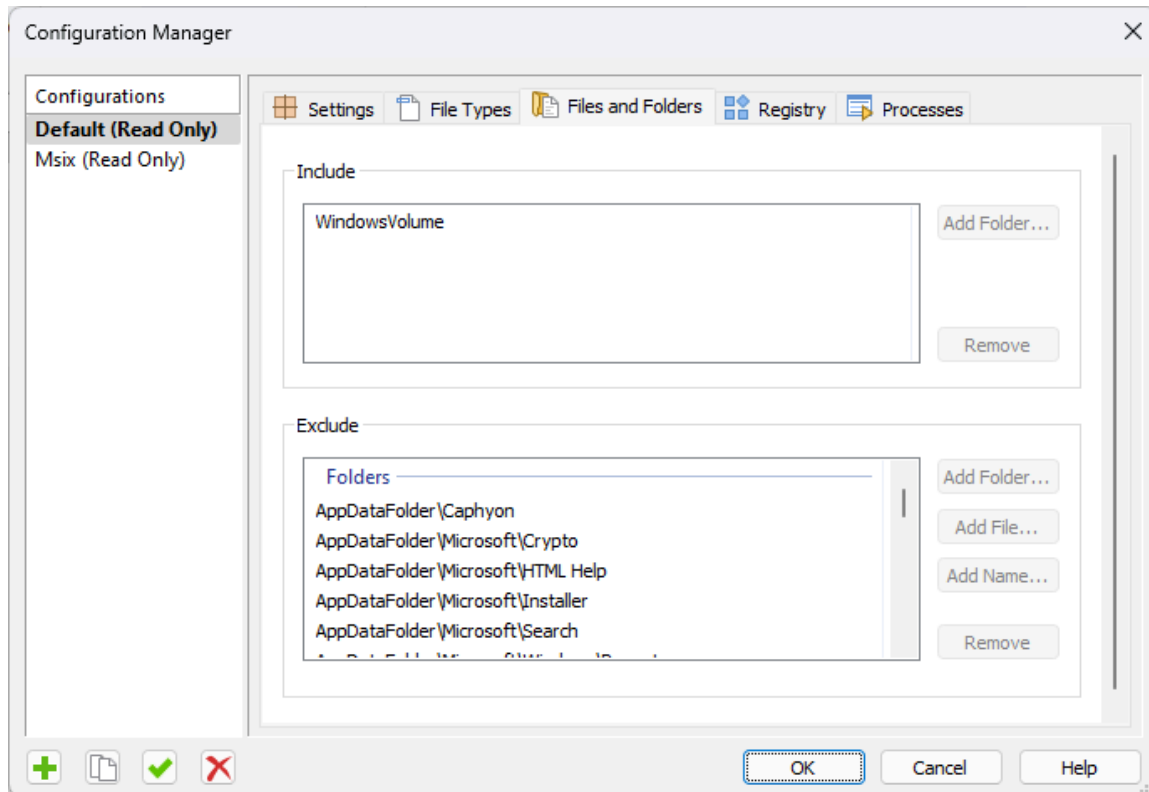
File Type Filters allow you to narrow down your file selection based on their type. To include or exclude specific file types from the installation, you can specify file extensions or patterns. This ensures that the package contains only relevant files.





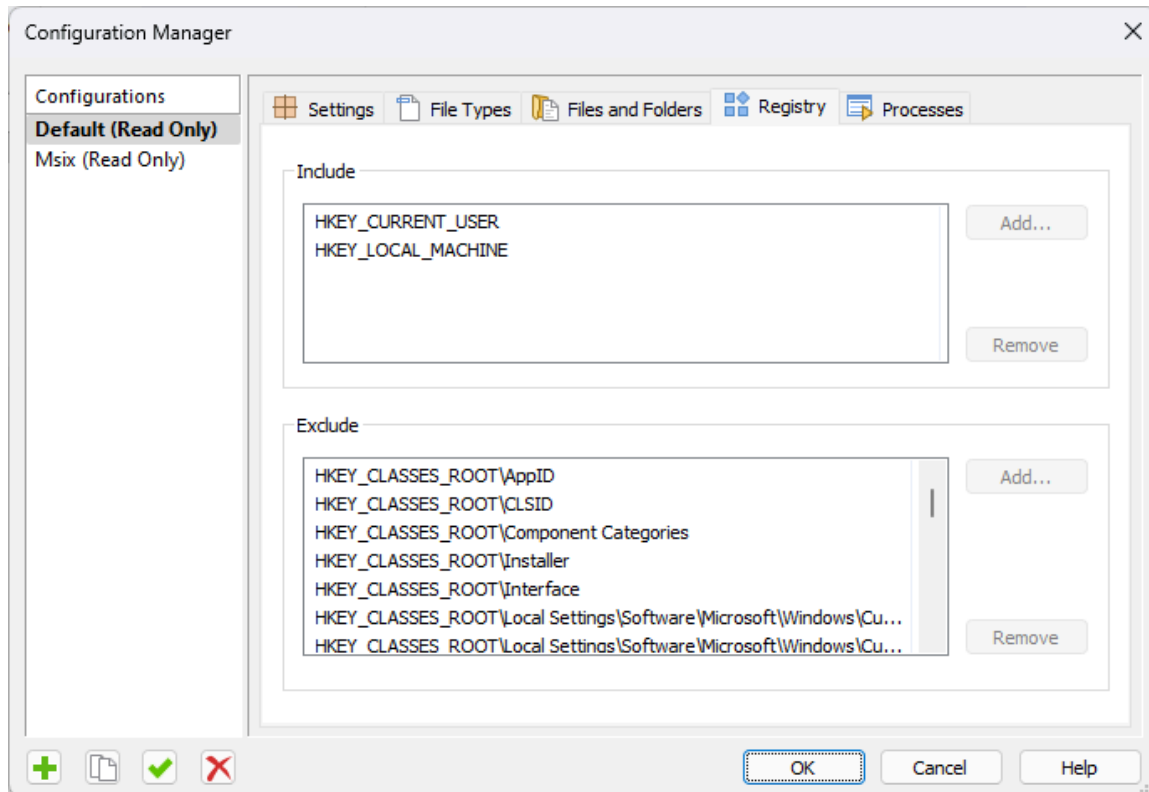
Files and Folders Filters enable you to include or exclude specific folders from the installation. You can define folder filters based on their names or patterns, allowing you to precisely control which folders are included or excluded.





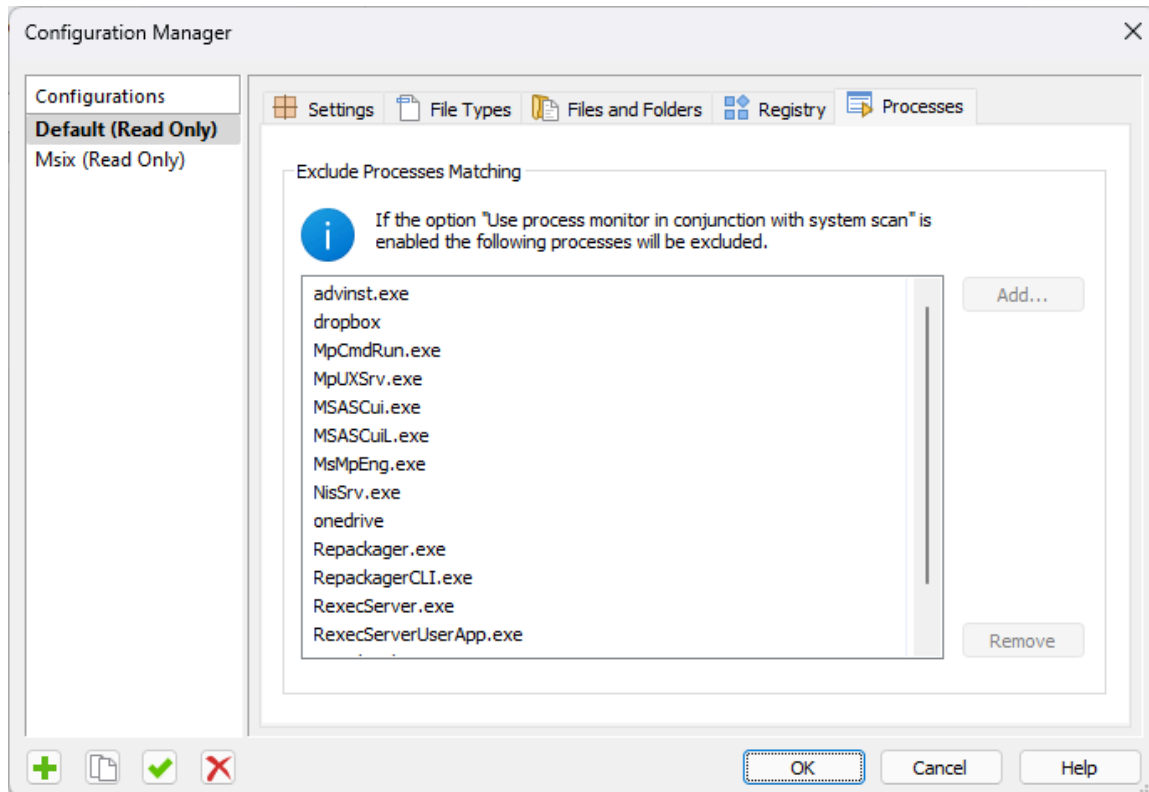
Registry Filters give you the ability to include or exclude specific registry entries from the installation. By defining registry filters based on key paths or patterns, you can ensure that only the required registry settings are included in the package.





Process Filters allow you to include or exclude specific processes during the installation. You can define process filters based on process names or patterns, ensuring that the installation process is not affected by unwanted processes.





While it may appear to be a bit overwhelming, especially for inexperienced IT professionals, the default settings provided by Advanced Repackager are more than sufficient to handle the majority of the repackaging operations that are required. The additional settings are for experienced IT Professionals who know exactly how to fine-tune the product to maximize their scenarios and achieve the best results possible.

## Capture cleanup

Capture cleanup is a critical step in the application repackaging process that involves removing any unnecessary or unwanted changes that were captured during the snapshot phase. It ensures that the resulting installer package contains only relevant and necessary changes, improving the overall efficiency and reliability of the deployment.

Advanced Repackager captures various system changes during the capture phase, such as file and registry modifications, system settings, services, and more. However, not all of these changes are required for the application to function properly. Some changes captured may be unrelated to the application or may cause conflicts with other software on target machines despite our efforts to continuously improve the noise captured by our tool. Keep in mind that we are improving the reliability of our tool with each release and we also improve our best practices implementations in the tool to help you get the best repackaging output on the market.



Before generating the final installer package, the capture cleanup process allows you to review and modify the captured changes. It entails locating and deleting any unnecessary files, registry entries, or settings that are not required for the application's installation and functionality.

Here are some important factors to consider and best practices for performing capture cleanup:

- Thoroughly review captured changes: Examine the captured changes carefully using the Advanced Repackager interface. Examine each file, registry entry, and setting to determine its relevance to the repackaged application.
- Exclude unnecessary files: Determine whether any files captured during the snapshot are unrelated to the application or can be obtained separately during the installation process. Exclude these files from the final installer package to reduce its size and complexity.
- Remove conflicting or redundant registry entries: Examine the captured registry entries for any that may conflict with existing system configurations or other software. To ensure a clean and error-free installation, remove any redundant or conflicting entries.
- Customize application settings: If the application's default settings are incompatible with your intended deployment, make the necessary changes during the capture cleanup. This may include configuring default options, disabling unnecessary features, or specifying custom settings that correspond to your deployment needs.
- Test thoroughly: After completing the capture cleanup, test the resulting installer package thoroughly on a clean system or virtual machine. Check that the application installs properly and works as expected. Testing identifies any remaining issues or conflicts that may require additional cleanup or customization.

You can optimize the resulting installer package for a smooth and reliable deployment by carefully reviewing and cleaning up the captured changes. It enables you to streamline the installation process, reduce potential conflicts or errors, and ensure that the installed application runs consistently and reliably across target machines.

## Practical repackaging example on VLC Media Player

Let's take an example when we repackage VLC Media Player from VideoLAN. Launch Advanced Repackager and select Capture Setup. A file prompt will appear in which you need to select the VLC executable.

Once the executable has been chosen you have the possibility to start the repackaging session either:

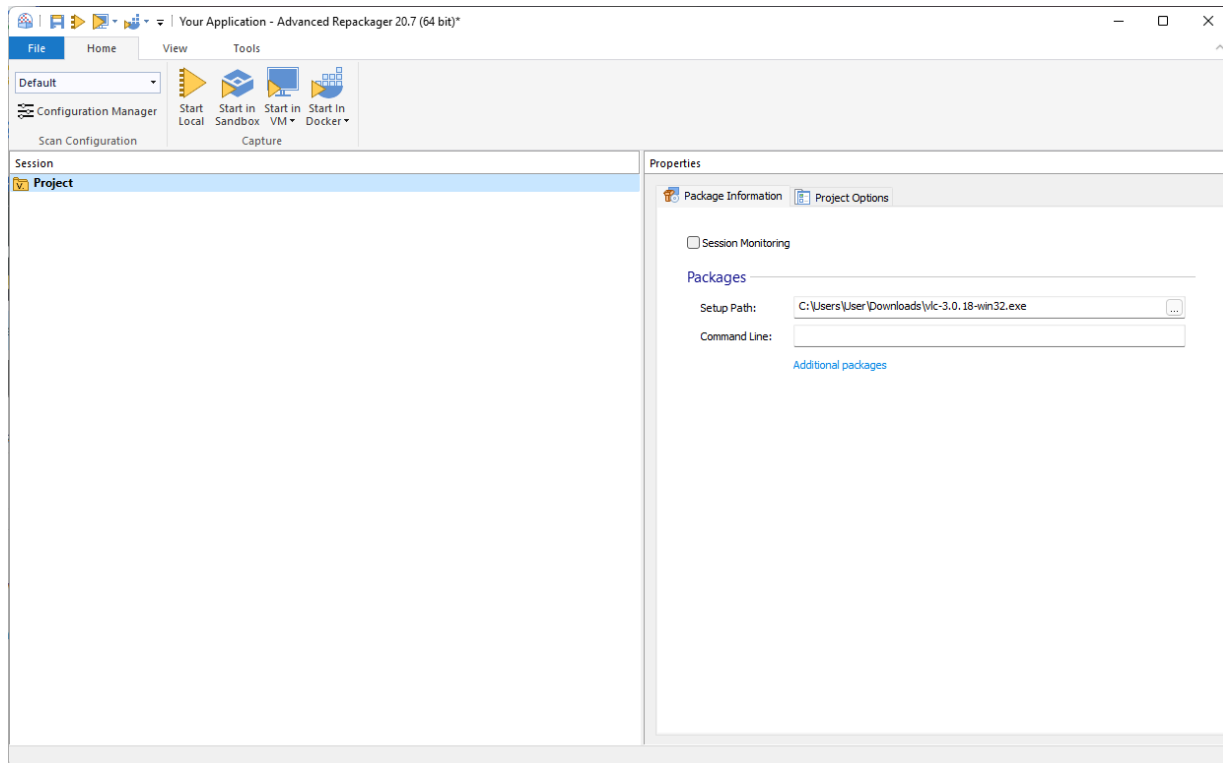
- Local
- Windows Sandbox





- VM
- Docker

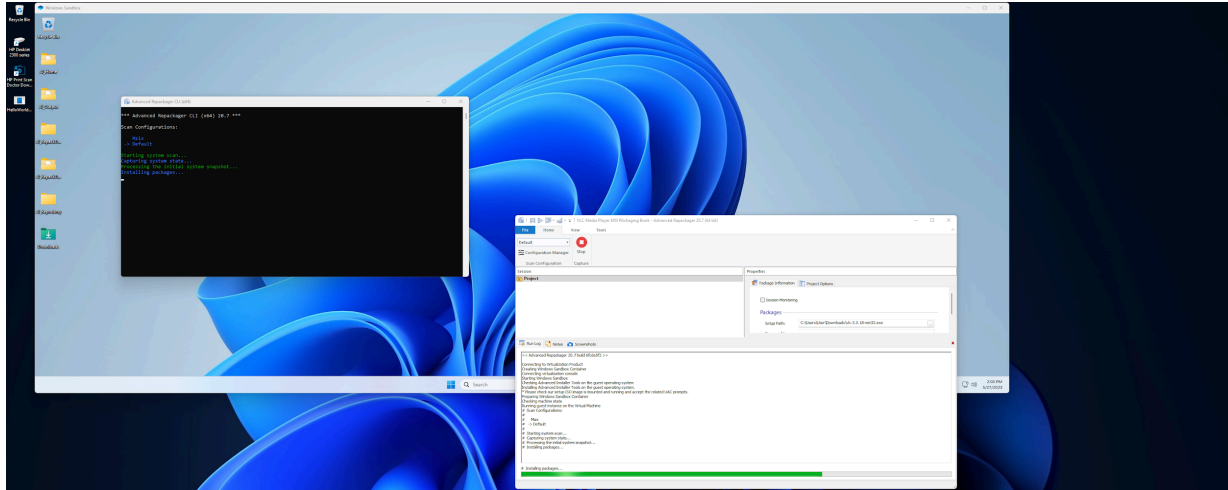
For the purpose of this tutorial we will use the Windows Sandbox option.



Once you click on the desired snapshot method, a prompt will appear to save the project. Once you save the project, Advanced Repackager handles the rest and:

- Starts the Windows Sandbox
- Installs the Advanced Repackager tools in order to control the operation within the Sandbox
- Starts the initial system snapshot
- Install the application. In this step you will have to manually perform the installation or you can pass any parameters you desire before you start the repackaging operation if you desire this to be silent
- Takes second system snapshot
- Outputs the RPK Project

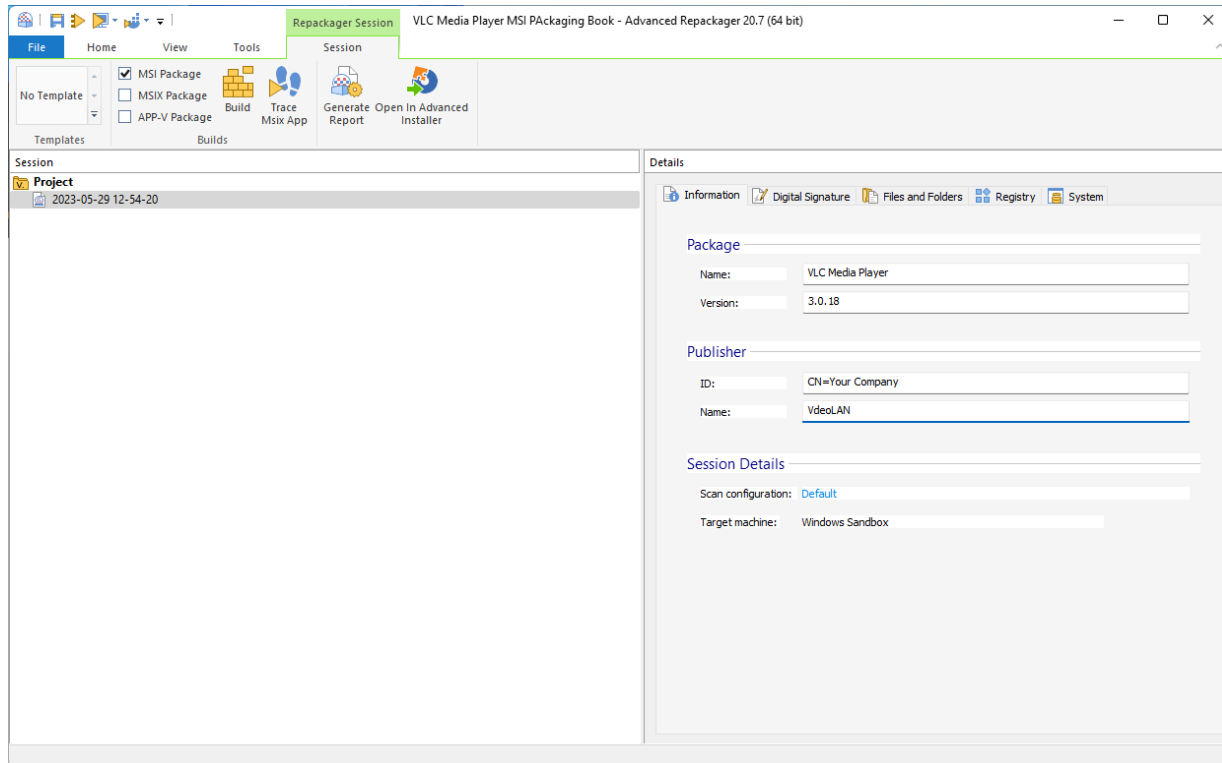




Once everything is done you now have an RPK project with all the captured modifications in the sandbox.

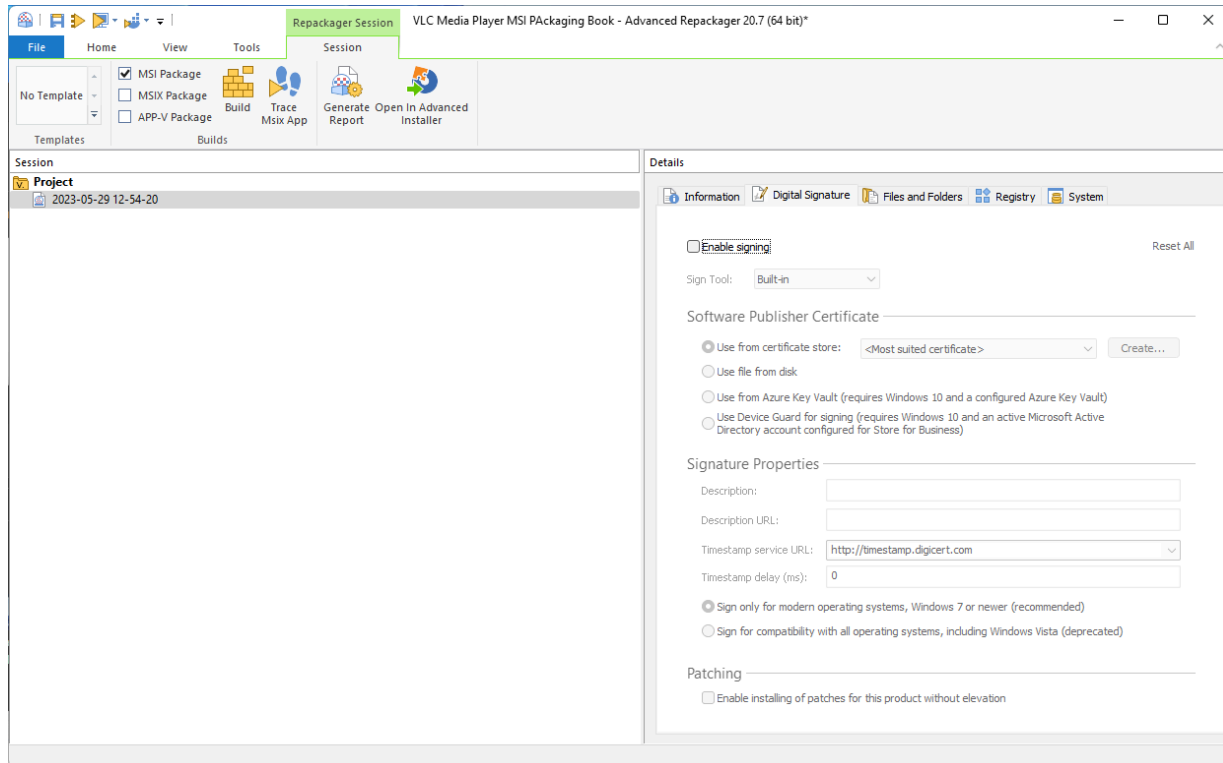
As mentioned earlier, this is where the cleanup part of the capture comes in. In the first tab you have the possibility to add the general information of your package, such as the application name, version, publisher. In our case the application name is VLC Media Player, the version is 3.0.18 and the publisher is VideoLAN.





The next tab refers to digital signatures, a topic which is coming more and more important even to MSI packages due to the [newly implemented SAC security by Microsoft](#). For the purposes of this tutorial we will skip this tab, but keep in mind that if you intend to deploy MSIX packages, digital signatures are not optional.

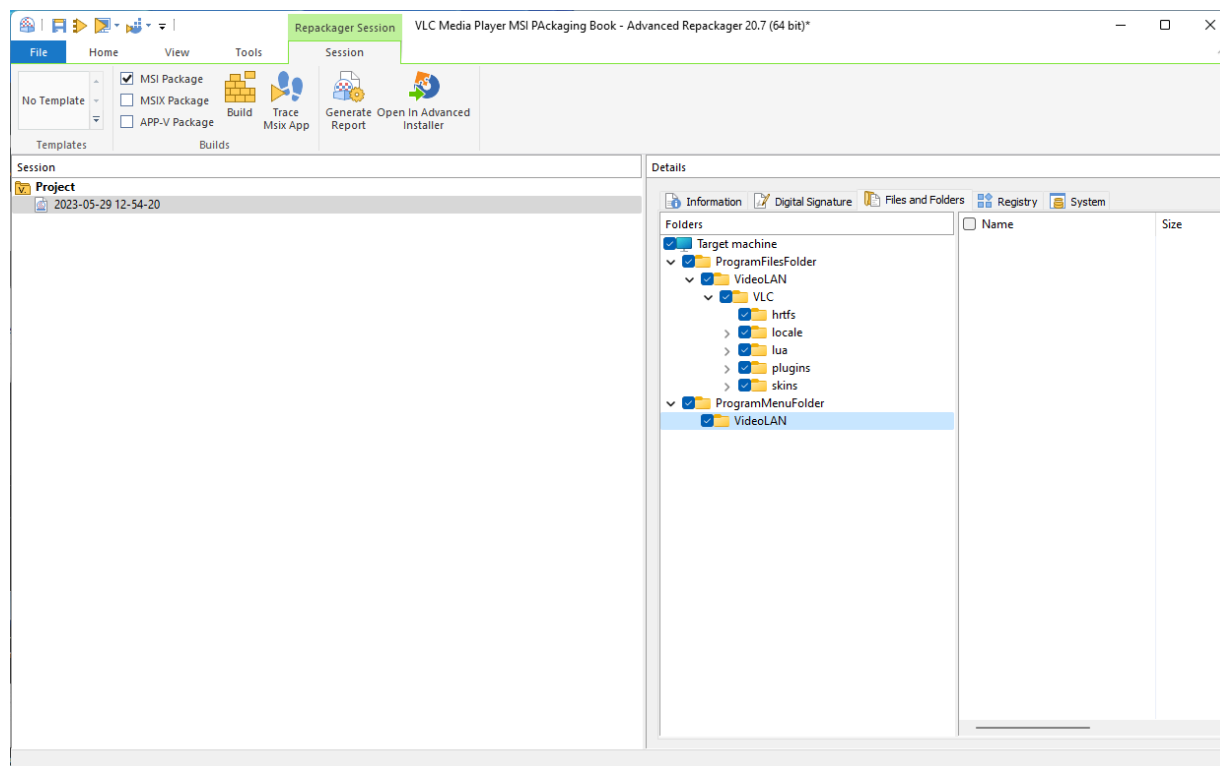




Moving on to the next tab, files and folders, it is critical to only check the folders that are related to your application. Unfortunately, there is no automated method for correctly identifying such changes, and you, as an IT Professional, must use your understanding and best knowledge of the application to determine the appropriate resources.

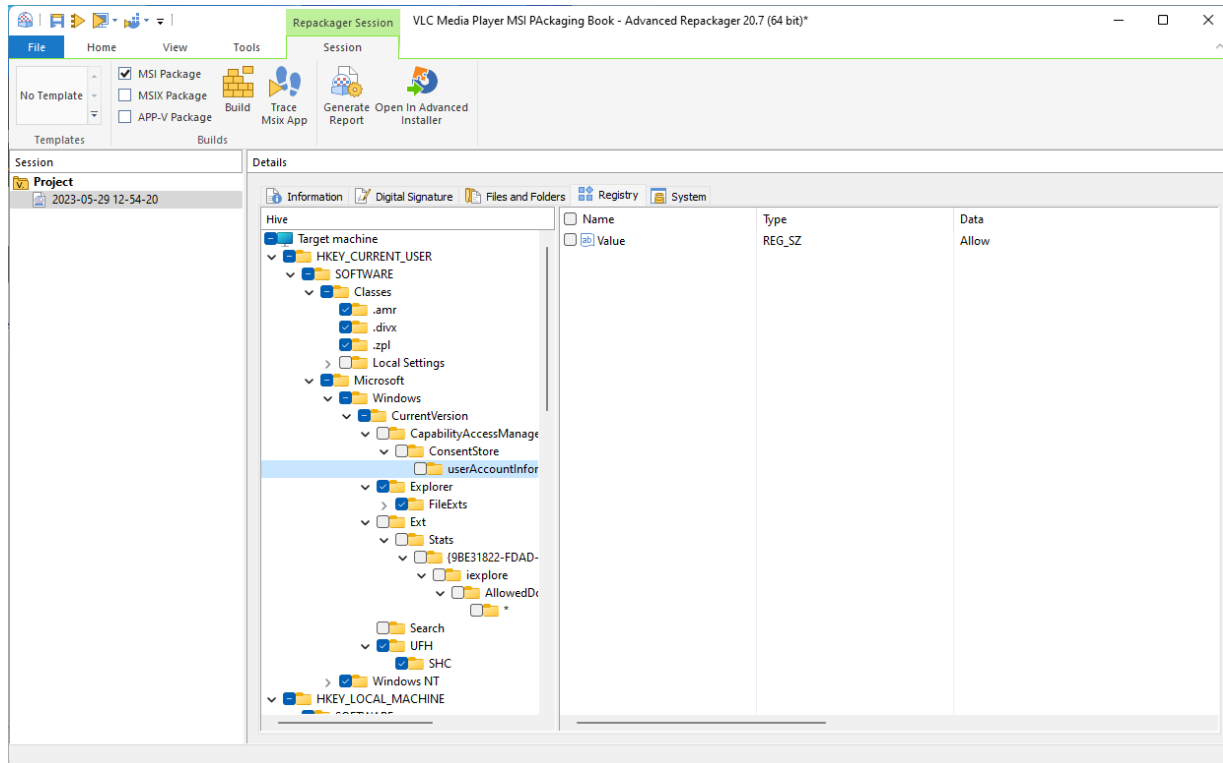
During this run, our repackager was successful in identifying only files and folders that are related to VLC, so we don't need to do any extra scrubbing.





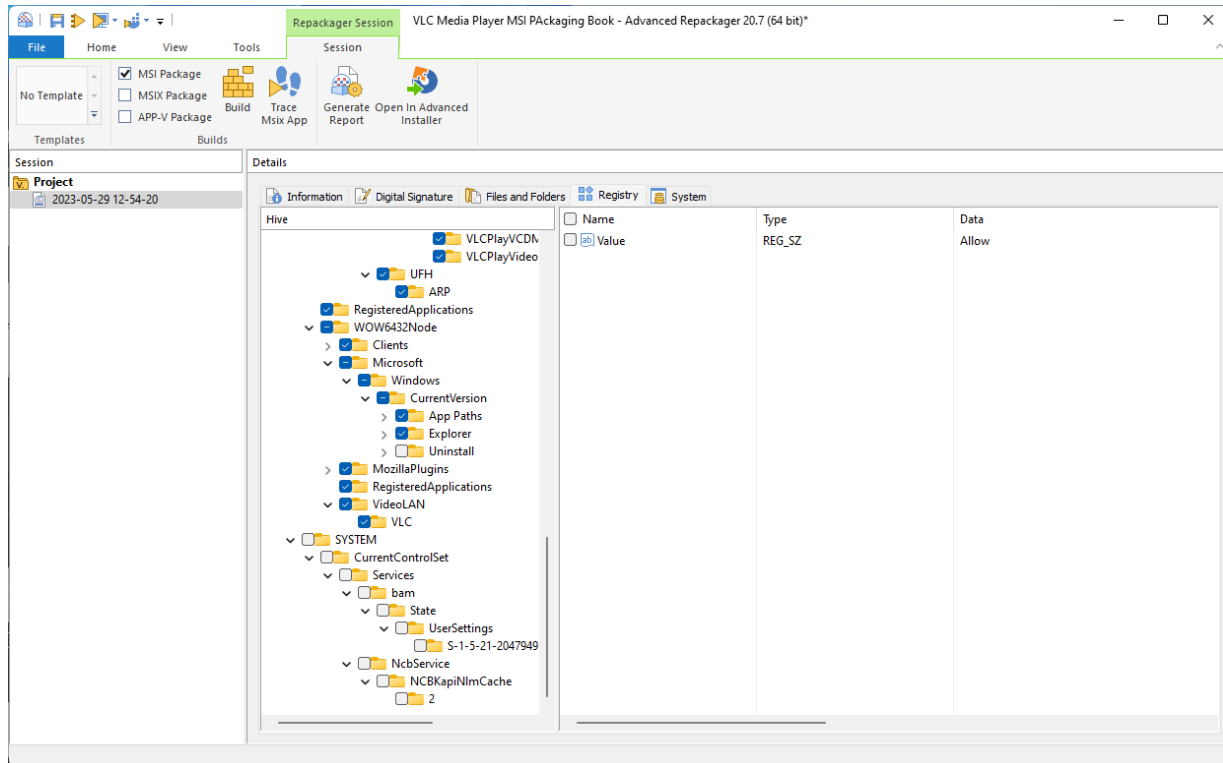
We then proceed to the registry tab, where we must perform some cleanups. Looking in the HKEY CURRENT USER hive, we see that we have the classes associated with VLC, as well as some file extensions for VLC, but we also have some captured registry that is clearly unrelated to our application, so those registry keys will be removed.





The HKEY\_LOCAL\_MACHINE hive seems to be properly captured with only minor information which is not needed, but during the setup some services were captured. We know for sure that VLC doesn't provide any services so these registry keys will be removed.





Because the application is recaptured, the uninstall registry is also captured. It is critical to remove the registry associated with Uninstall information when repackaging; otherwise, two entries will appear in the Add/Remove Programs section of the OS and the original one will not work. The following are the locations of uninstall information:

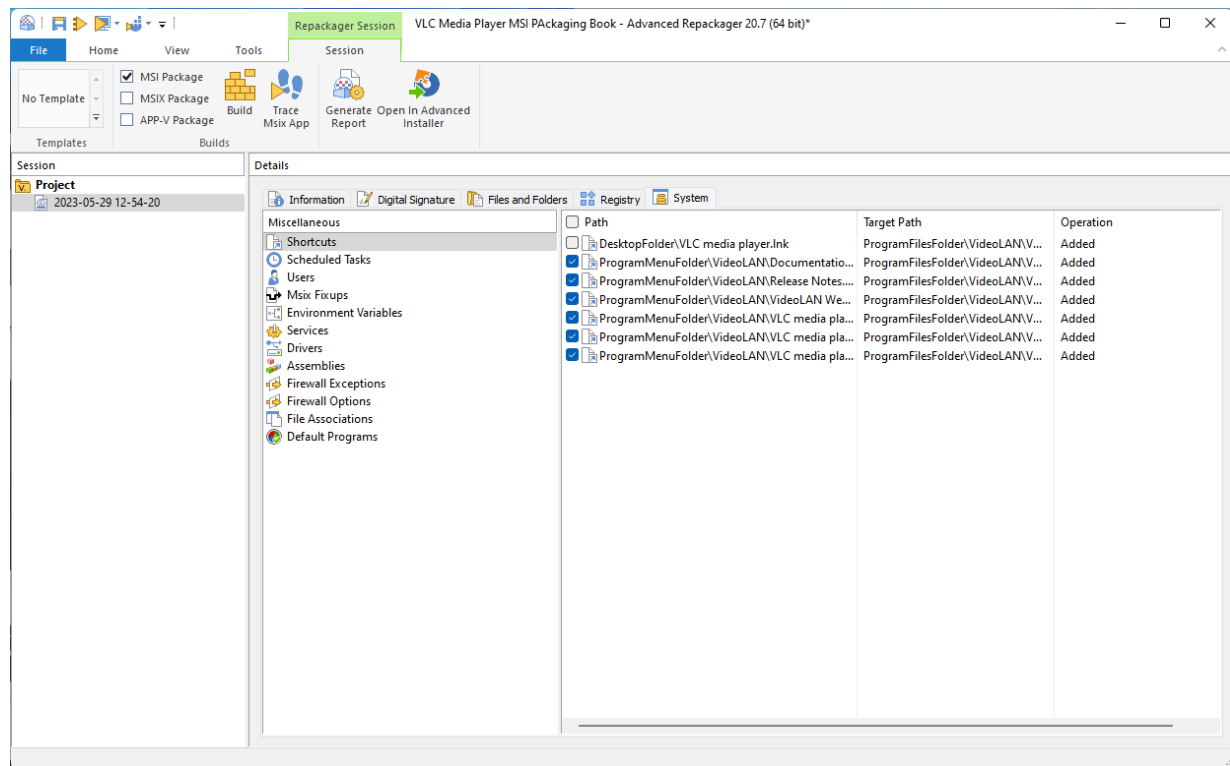
HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall  
HKEY\_LOCAL\_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall

Jumping to the system tab, we have captured information in the following sections:

- Shortcuts
- File associations
- Default programs

Depending on the customer's request, you can remove any of the above mentioned items. Typically, the best practice and most requested modification of the packages that has been seen in the industry is to remove the application's desktop shortcut, so we will do the same in our case.

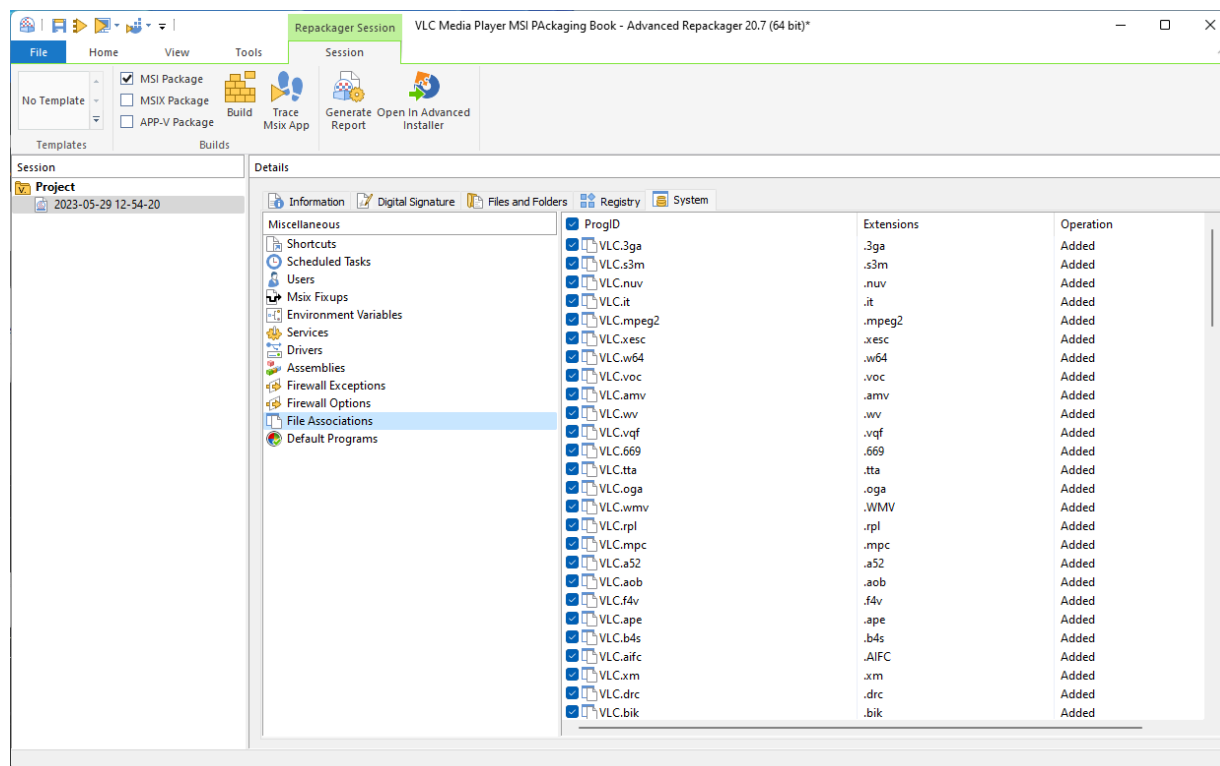




Now is the time to define other file associations for which VLC is now marked, as well as other default programs.





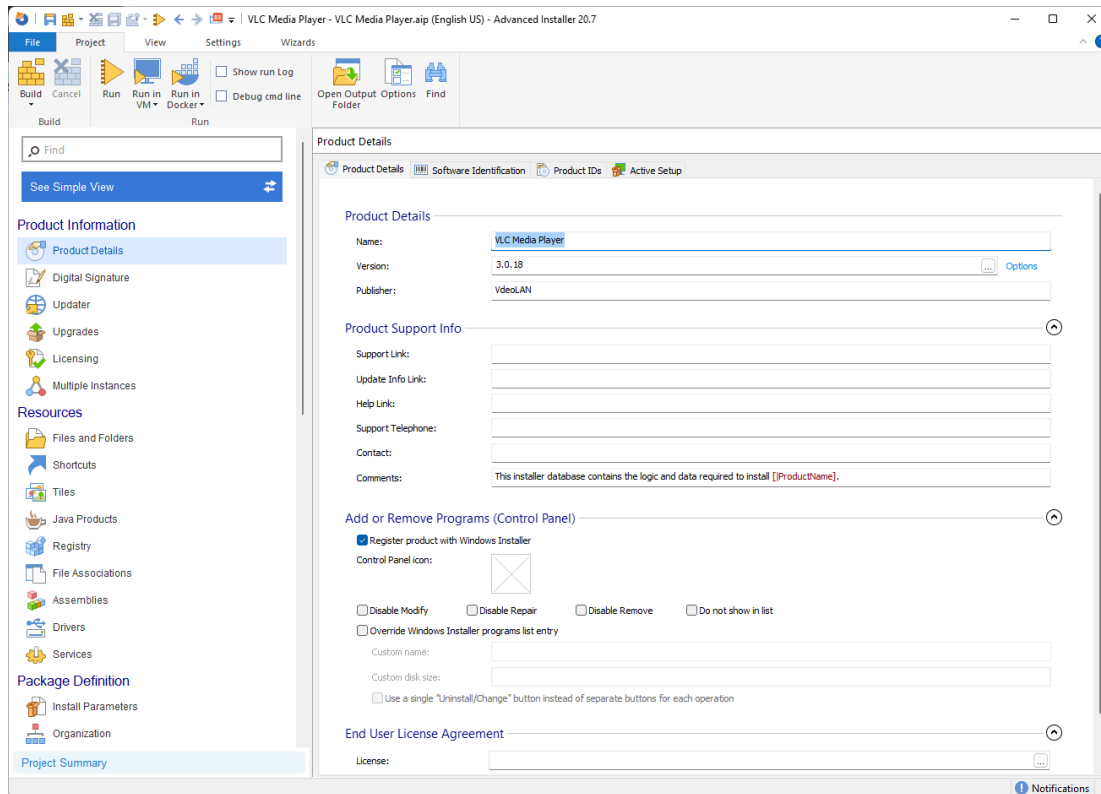


After you've cleaned everything up, you can open the resulting project in Advanced Installer by clicking the button in the upper tab.

If you look closely, there are three types of builds available (MSI, MSIX, APP-V). For the purposes of this tutorial we will only leave MSI selected, but check out the next chapter to learn more about this feature.

Advanced Installer will gather all of the necessary resources and generate a new AIP project for you to further customize or build the MSI.





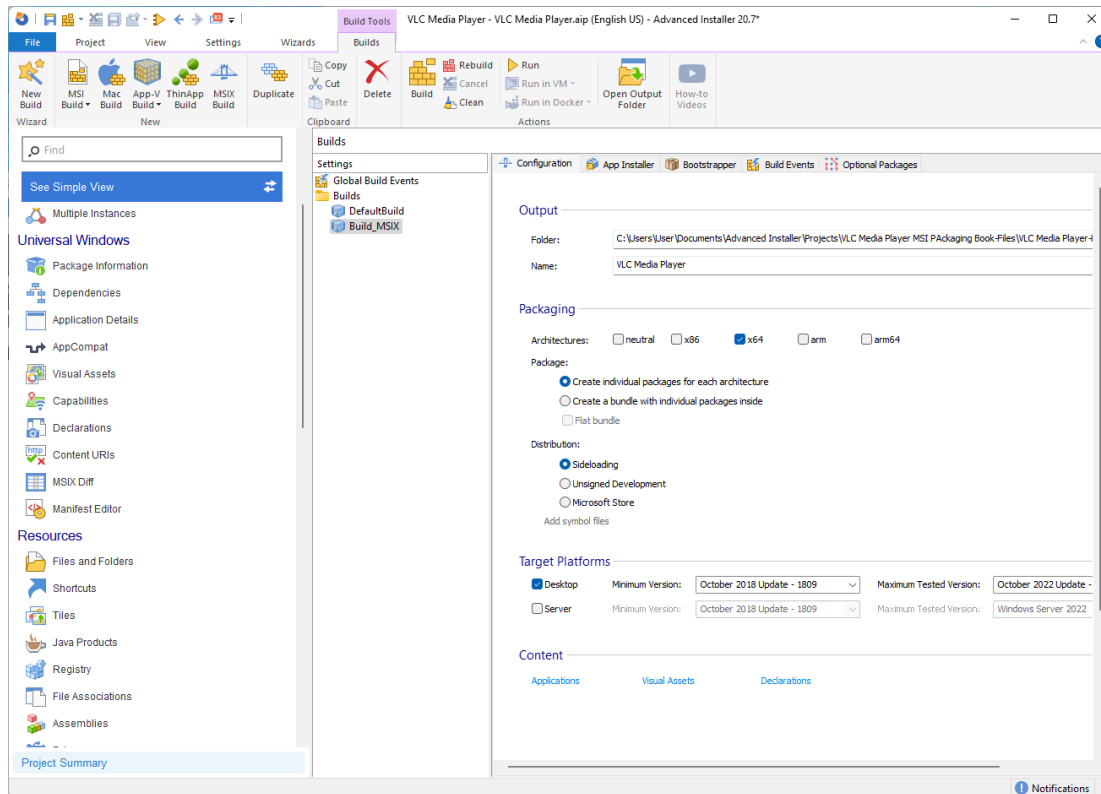
## Multiple Builds

As mentioned earlier, once a repackaging operation is finished you can choose one or multiple builds. Of course, you can create any additional builds in the AIP project. But what does that mean?

If you plan to deploy the application via multiple methods or wish to [upload the MSIX package to the Microsoft Store](#), with Advanced Installer you don't need to take multiple captures or create multiple projects in order to achieve multiple builds of the same installation package, it's all in one project.

Taking that our VLC project is already opened with Advanced Installer, all you need to do is navigate to the Builds page and click on MSIX Build on top. In the builds area a new Build\_MSIX will appear where you can perform further customizations.

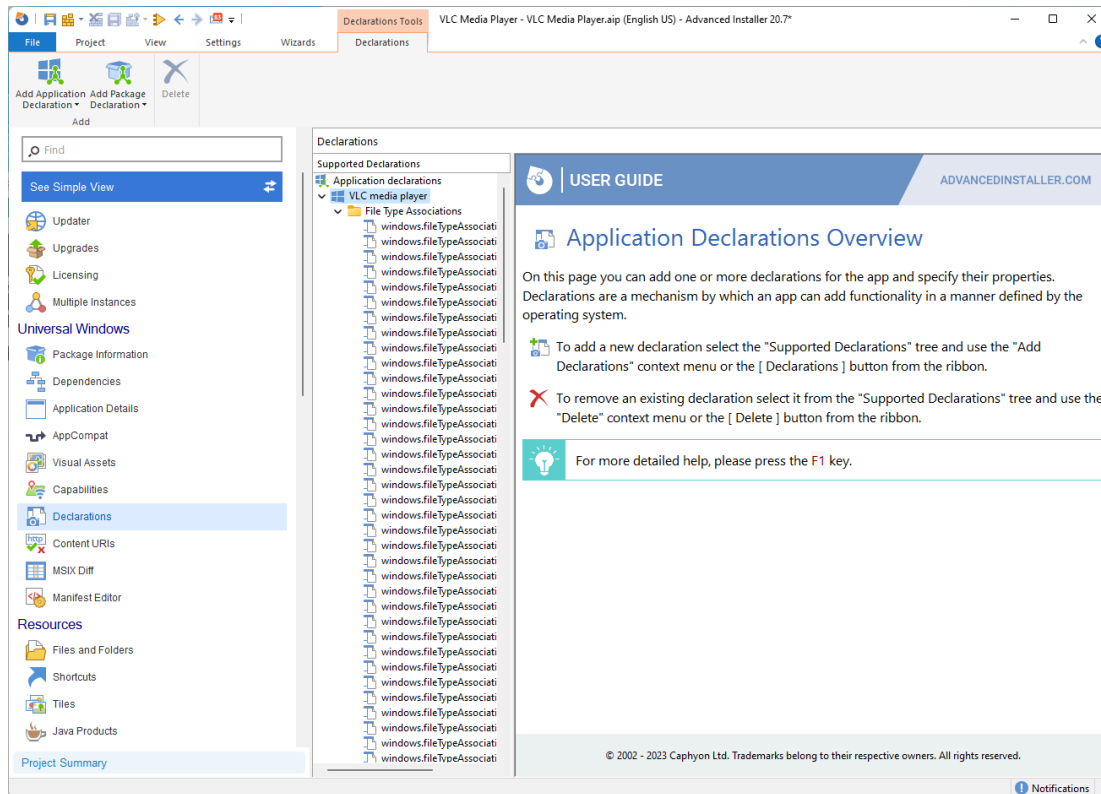




You can choose the architecture of your MSIX package, choose if you want a bundle for multiple architectures or independent packages, select your distribution method, but also target platforms and other important information such as Applications (shortcuts), visual assets and declarations.

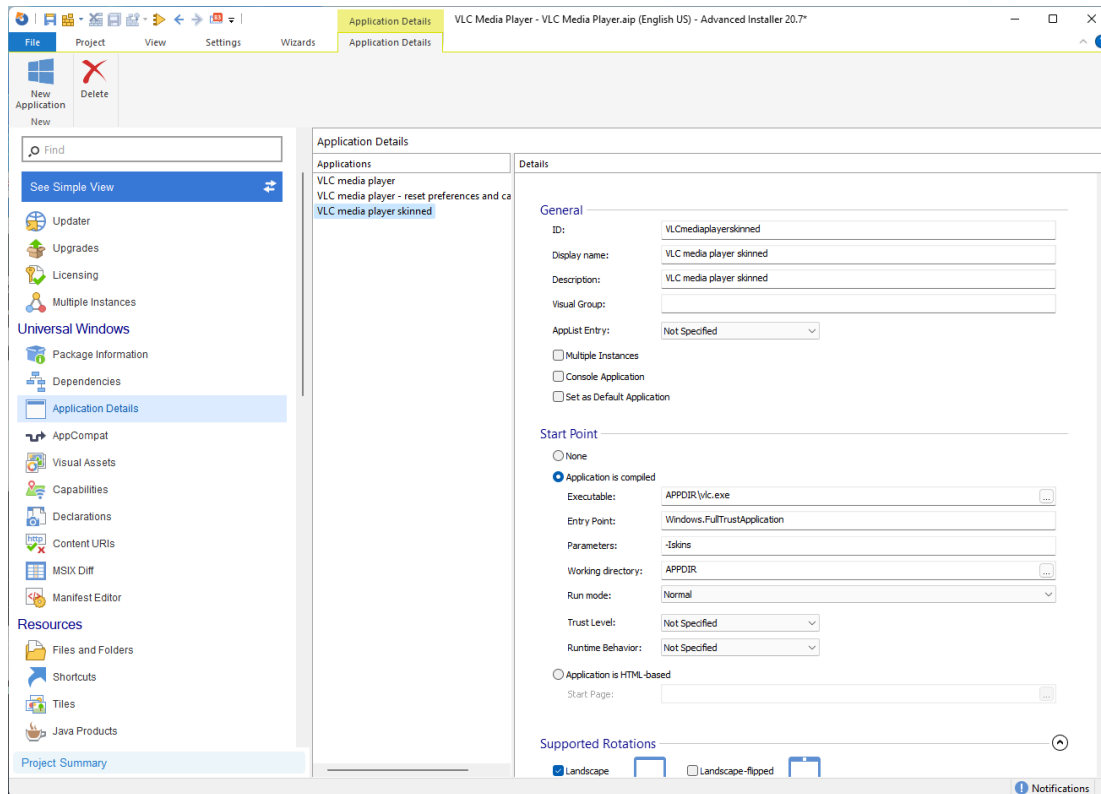
For example, Advanced Installer automatically detected that file type associations are included in this package and everything has been included in the AppxManifest file for the MSIX application. Clicking on the Declarations button which appears in the above screenshot, you will be redirected to the Declarations page and all the FTAs will appear in the list.





Advanced Installer also detected the classical shortcuts present in the project for the MSI built and automatically created the Applications for them.





Keep in mind that [MSIX migration limitations](#) exist and in some cases further workarounds must be applied or the solution might not work (as in case of drivers for example).



# Further application customization

## Advertised Shortcuts

### Significance of Advertised Shortcuts

An advertised shortcut is a pointer to a file or folder that is installed by an MSI package. When running an advertised shortcut, Windows Installer first checks that all the components of the respective feature are installed before running the file. If any of the components are missing, Windows Installer repairs the installation by installing the missing components. This ensures that the application remains consistent and that all of its features are available when required.

The best is to keep in mind the following about advertised shortcuts:

- When running an advertised shortcut, Windows Installer first checks that all the components **of the respective feature** are installed ( before running the file).
- The target of the shortcut must be present in the package.

What is to take is that if your application contains multiple features, but the shortcut is available only on one feature, then during the self-healing process only that feature will be checked if all the components are properly installed on the machine.

For more information regarding shortcuts, check out our first [MSI Packaging Essentials ebook](#).

## What is Self-Healing

The MSI (Microsoft Installer) self-healing mechanism is a powerful feature that ensures the integrity and availability of installed applications. It is intended to detect and repair any missing or corrupted files, registry entries, or other components of an application, preserving the application's functionality and preventing potential problems.

When an MSI-based application is installed, Windows Installer creates a comprehensive installation database known as the installation package or MSI package. This package contains all of the resources and information needed to install and maintain the application. Files,



registry entries, shortcuts, components, and other configuration settings are all included. This small MSI is also known as the “cached MSI” and can be found in the %systemroot%\installer folder.

**Important mention:** manual interventions in the Installer folder should never be done. For more details about the Installer folder, check out the [Do not delete your Windows Installer folder](#) article.

Specific events, such as the launch of an application, the opening of a document associated with the application, or the invocation of a Windows Installer action such as repair or modify, initiate the self-healing process. When any of these events occur, Windows Installer validates the application's integrity by comparing the recorded installation information to the current state of the system.

When the self-healing mechanism detects that a file, registry entry, or other component associated with the application is missing or damaged, it begins the repair process automatically. Locating the original installation source (usually the installation media or a network location) and restoring the missing or corrupted components is the first step in this process.

Windows Installer follows a predefined sequence of steps during the repair process:

1. **Verification:** The digital signature of the installation package is checked by Windows Installer to ensure that it is authentic and has not been tampered with.
2. **Component Detection:** By comparing the recorded installation information with the current system state, Windows Installer determines which components are missing or damaged.
3. **Source Resolution:** The installation source is attempted to be located by Windows Installer by referring to the original installation media or other specified locations.
4. **File and Registry Restoration:** Windows Installer uses the installation package's source files to replace missing or damaged files, registry entries, and other components.
5. **Reconfiguration:** Windows Installer reconfigures the repaired components, restoring any necessary settings or dependencies.

When the repair process is finished, the application is returned to its original state, with all of the necessary files, registry entries, and settings in place. This ensures that the application works as expected and that any missing or damaged components are completely restored.



**Important note:** the self-healing mechanism in MSI is dependent on the original installation source being available. If the installation source is not available, the repair process may fail, requiring manual intervention to restore the application.

MSI's self-healing mechanism offers numerous advantages:

- **Maintenance:** It makes application maintenance easier by automatically repairing any issues that may arise as a result of file corruption, accidental deletion, or system changes.
- **Data Integrity:** It contributes to data integrity protection by ensuring that applications have access to the necessary files and resources. This prevents application crashes, data loss, and other problems caused by missing or corrupted components.
- **User Experience:** It improves the user experience by restoring application functionality without the need for manual intervention or reinstallation.

However, to ensure that the self-healing mechanism works properly, the installation package must be properly configured and tested. This entails precisely specifying the dependencies, files, and registry entries that are required for the application to function.

In summary, MSI's self-healing mechanism is a critical feature that automatically detects and repairs missing or corrupted application components. It ensures smooth operation and helps prevent potential issues caused by component-related problems by maintaining the integrity and availability of applications.

## Properties in MSI Packaging

Properties are used in MSI packaging to define and customize various aspects of the installation process, such as installation location, product key, and installation options. MSI properties are classified as public or private based on whether or not end users can access them.

**Public properties** are those that users can access and modify during the installation process. These properties are usually defined in the MSI database's Property table and are used to store values like the installation directory, product name, and version number. When launching an installation, public properties are frequently specified on the command line, or they can be set using a [transform file \(.mst\)](#).





**Private properties**, on the other hand, are used internally by the installer and should not be changed by end users. These properties are typically used to control the installer's behavior and are hidden from users during the installation process. Private properties, which are typically defined in the MSI database's CustomAction table, are used to control the execution of custom actions, install sequences, or logging behavior.

MSI properties are typically stored as string values rather than integers or Boolean values when they are defined. Values are thus stored as text strings and must be converted to the appropriate data type as needed. A property that stores a version number, for example, could be defined as a string value but would need to be converted to a numerical value before performing arithmetic or comparison operations.

MSI comes with a plethora of built-in properties for defining various aspects of the installation process, including directory properties. The following are some of the most frequently used properties in MSI:

- ALLUSERS: Specifies whether the installation is for all [users of the machine](#) (1) or only the current user (2).
- INSTALLDIR: Specifies the default installation directory for the application.
- INSTALLLEVEL: Specifies the installation level, which controls which features are installed during the installation process.
- REBOOT: Specifies whether a reboot is required after installation (force, suppress, or prompt).
- TARGETDIR: Specifies the target installation directory for the application.
- USERNAME: Specifies the name of the user performing the installation.
- USERPROFILE: Specifies the user profile directory for the current user.

MSI also supports the creation of custom properties that can be used to define other aspects of the installation process in addition to these built-in properties. For example, you could define a custom property to specify a product's license key or a custom installation path.

When working with directory properties, it is critical to understand that MSI organizes files and resources using a hierarchical directory structure. The INSTALLDIR directory is at the top of the directory structure and is where the application is installed by default. Other directory properties, such as TARGETDIR and APPDATA, are used to specify specific directories within a larger directory structure.

It is important to understand these base concepts, as we will dive deeper into this topic [later down the road](#).



For more details about Properties, check out our first [MSI Packaging Essentials ebook](#).

## Custom vlc settings

Now that we've captured and repackaged VLC Media Player as an MSI, we need to figure out how to detect and implement any custom settings that may be required for this application.

In this example, we will look at the number one rule when it comes to repackaging applications in an enterprise environment: always disable automatic updates.

There are multiple ways in which we can detect where the settings of the application are stored, such as:

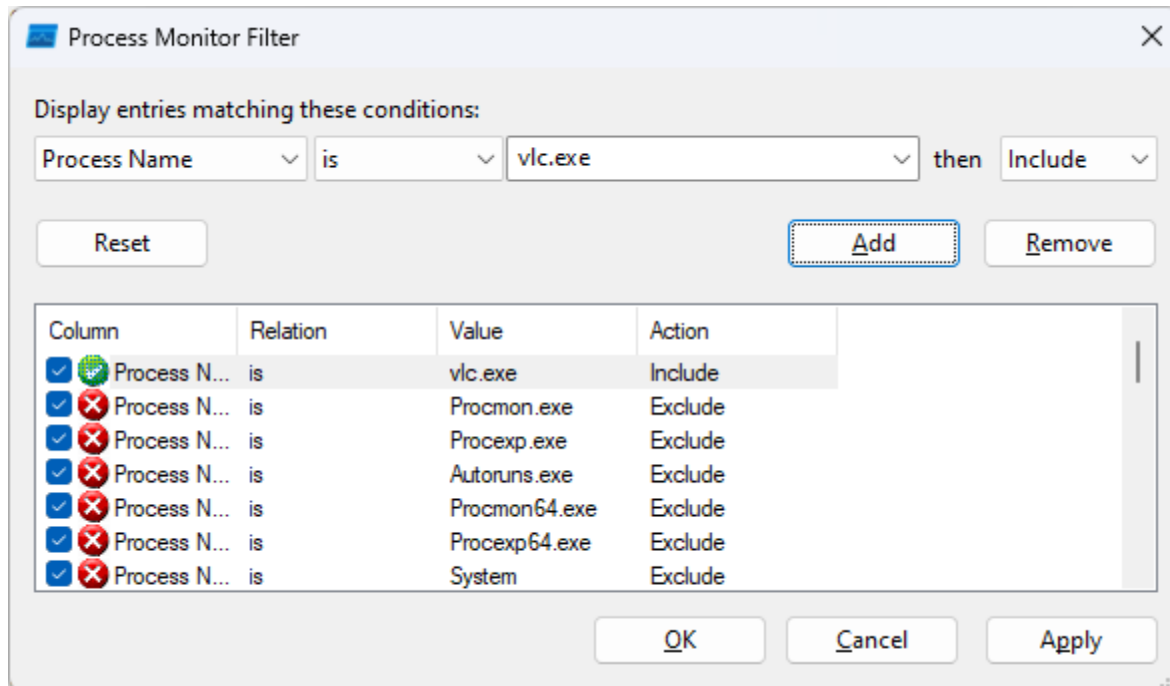
- Using Advanced Repackager with Session Monitoring
- Using Process Monitor
- Using other 3rd-party tools, such as Systracer

We have already covered in the MSI Packaging Essentials book how you can [detect the settings of VLC Media Player with Systracer](#). This basically acts exactly like Advanced Repackager and captures two states of the system and displays the difference between them.

We already looked at how to use Process Monitor to detect if an EXE contains an embedded MSI; we can do the same to detect where VLC Media Player writes its settings, but some additional settings must be performed in the filter section:

- Launch Process Monitor: Open Process Monitor from the Start menu or the shortcut on your desktop.
- Configure filters: Before monitoring the installation process, it's helpful to set up filters to narrow down the captured events and focus on the relevant activities. Click on the "Filter" menu, and then select "Filter..." to open the Filter dialog box.
- Under "Display entries matching these conditions" select "Process Name", "is", "vlc.exe", then "include". Click on Add

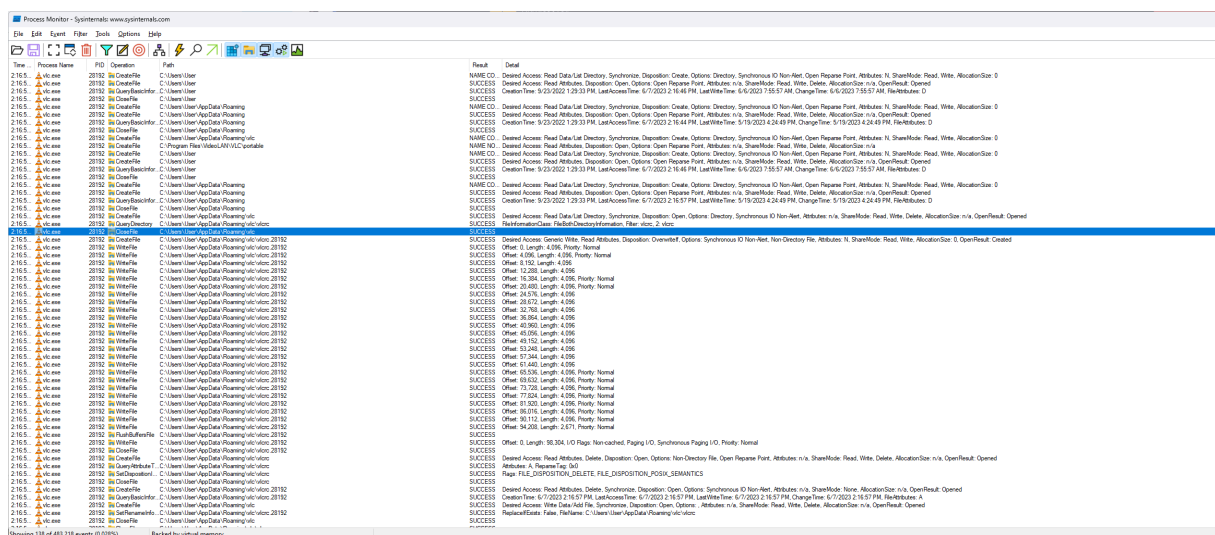




Click OK and start the capture process

- Open VLC Media Player and go to Tools > Preferences and implement the desired configurations. Close the preferences and close VLC Media Player

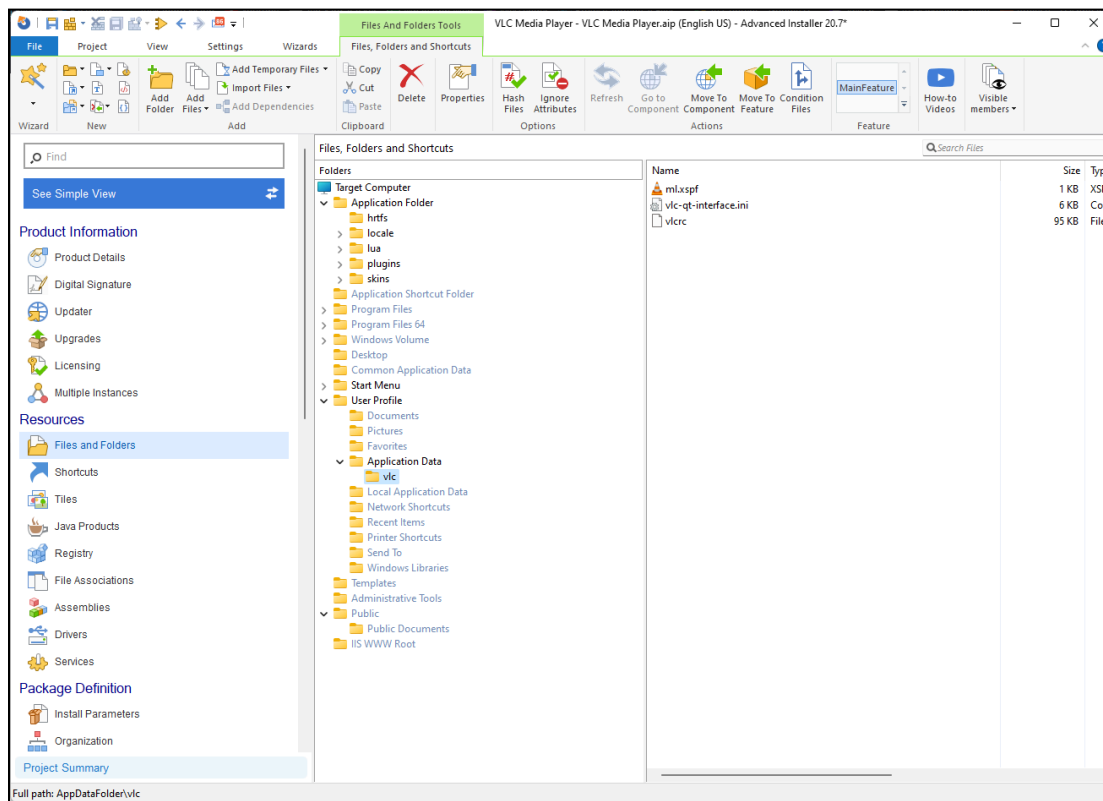
After we have completed all of these steps, we can see at the bottom of the Process Monitor capture that VLC Media Player created and modified the file vlcrc, which can be found in %appdata%\vlc.



If you want to use Advanced Repackager, follow the same steps as when we started the VLC Media Player repackaging process, but instead of selecting an executable, check the Session Monitoring checkbox. At the end, Advanced Repackager will identify the modified vlcrc.

## Custom settings implementation in the package

Now that we know which file needs to be added to the package, we can simplify our repackaged project [that we did earlier](#), navigate to the [Files and Folders page](#) and under the Application Data folder, create a new directory called vlc and add the vlcrc file. You can also include the vlc-qt-interface.ini and ml.xspf, and the project should look something like this:



When it comes to installers, however, working with user files or the registry is more difficult. When an application is deployed in an infrastructure using any infrastructure management tools, the NT System\Administrator account is used. This is the most privileged account available.

One of the golden rules when it comes to installation testing as an IT Professional is to always test your applications using the PSEXec utility. We have a more in-detail look over this topic in the MSI Packaging Essentials book and you can check it [here](#), but if we would leave the installer as it is right now, the settings won't reach the logged on user or any user that is using that particular computer, because the settings will only be populated within the NT System\Administrator account files.

This is where [advertised shortcuts](#) come into play. When working with Windows Installer, one important concept to understand is the concept of [advertised shortcuts](#).

Advertised shortcuts are a feature provided by Windows Installer that allows for the dynamic installation and repair of applications. In the following, we will explore the concept of advertised shortcuts, how they work, and best practices for utilizing them effectively.

Advertised shortcuts serve two main purposes: advertisement and resilience. Let's take a closer look at each of these points.

When an advertised shortcut is launched, Windows Installer validates that all the components included in the same feature as the shortcut feature are installed on the device. This is accomplished by determining whether their keypaths, which represent the critical resources for each component, are present. If a component is missing, Windows Installer launches the installation package and reinstalls all required resources from the.msi file. This ensures that the application has been completely installed before being launched.

To create an advertised shortcut, the resource it points to in the Component table must be designated as a KeyPath. A KeyPath is a resource that represents the feature's core component. Once a resource has been designated as a KeyPath, it cannot be moved to another component, and no other resource can be designated as a KeyPath for that component. This ensures that the advertised shortcut works properly and, if necessary, initiates the installation process.

Another advantage of advertised shortcuts is their resiliency. When the user clicks on the advertised shortcut, Windows Installer automatically triggers a repair of the application if any of the keypaths associated with the components are missing or corrupted. This ensures that the application will continue to function even if critical resources are compromised.

To take full advantage of both advertisement and self-healing, it is important to adhere to certain best practices:

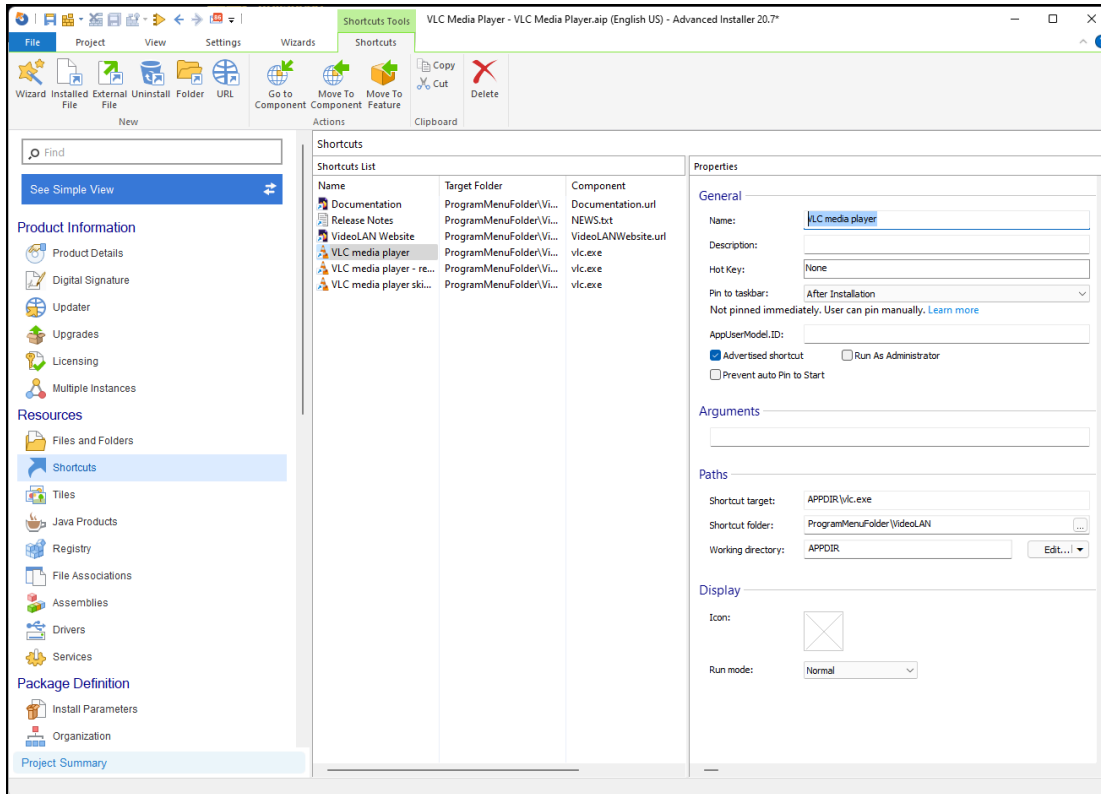


- Set all features' states to Advertise: It is recommended that all features be set to Advertise. While having only one feature is preferable, if multiple features are required, make sure that all of them are set to Advertise. This ensures that the required resources are correctly installed when the application is launched.
- Advertise all shortcuts: All shortcuts should be advertised in order to enable the dynamic installation and repair process. This is accomplished by checking the "Advertised shortcut" checkbox in the Shortcut Properties view during the shortcut's creation.
- Assign a keypath to all components: Each application component should have its own keypath. The keypath represents the component's critical resource and assists Windows Installer in validating the installation and initiating repairs if necessary. To ensure proper functionality, it is recommended that you follow Microsoft's general rules for organizing resources into components.

By following these guidelines, you can harness the power of advertised shortcuts to provide your users with a dynamic and resilient installation experience. Advertised shortcuts make application deployment and maintenance easier by ensuring that all necessary resources are installed correctly and that repairs are triggered when necessary.

Because all the files used by the shortcuts are included in the MSI package, we can navigate to the [Shortcuts Page](#) and for each shortcut available in the package we must enable the Advertised Shortcut checkbox.





When the package is installed and a user opens a shortcut, the MSI will perform a self-healing action and copy the previously placed files to the currently logged in user profile.

However, while this solution may work, it does not represent industry best practice, and we must consider some other issues that may arise if we place user data in this manner.

## Best practices for user data

Because of the way MSI packages are cached in the OS once they are installed, the above solution may not be the best suited to cover all cases when it comes to user data in the form of files.

If you've ever looked through your Windows system directory, you might have noticed a folder called "Installer" in the C:\Windows directory. This folder, which is frequently overlooked, plays an important role in Windows Installer technology by storing cached copies of MSI files.

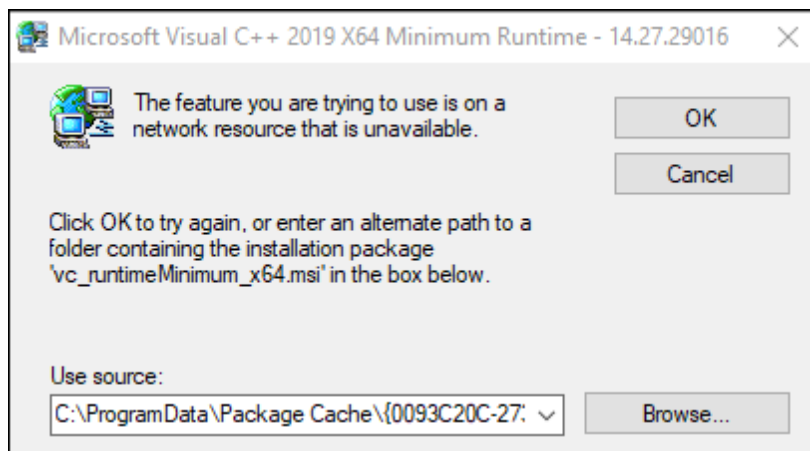
The cached MSI files in the **C:\Windows\Installer** folder serve several important purposes:



- **Repair and Maintenance:** When an installed application has problems or becomes corrupted, Windows Installer uses the cached MSI files to repair it. This ensures the application's proper operation by restoring missing or damaged files.
- **Uninstallation:** The cached MSI files are used to completely remove the application from the system during the uninstallation process. To reverse the installation and remove all associated components, Windows Installer accesses the original installation package stored in the Installer folder.
- **Patching:** When updating or patching an installed application, Windows Installer uses the cached MSI files to identify the installed version and apply the necessary changes. Patch files, which are used for incremental updates to installed applications, are also stored in the Installer folder.

However, an MSI is not fully cached into the C:\Windows\Installer directory; instead, the files are removed during the caching process to save hard drive space.

If we use the above solution and delete the MSI from where it was originally installed, this operation will fail and the user will not have the VLC Media Player preferences copied in his profile when he opens the shortcut and the self-healing mechanism starts.



If the user data files are not correctly implemented, the only way for the self-healing mechanism to work is to browse to the MSI, which means copying it again on the computer.

There are several approaches we can take in this situation, but the following two will usually suffice:

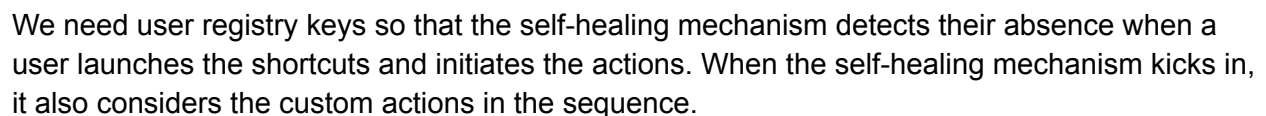
- Move the files to a per-machine location and copy them to the user profile using a custom action, keep the advertised shortcuts enabled, and add a dummy registry key to HKEY CURRENT USER.



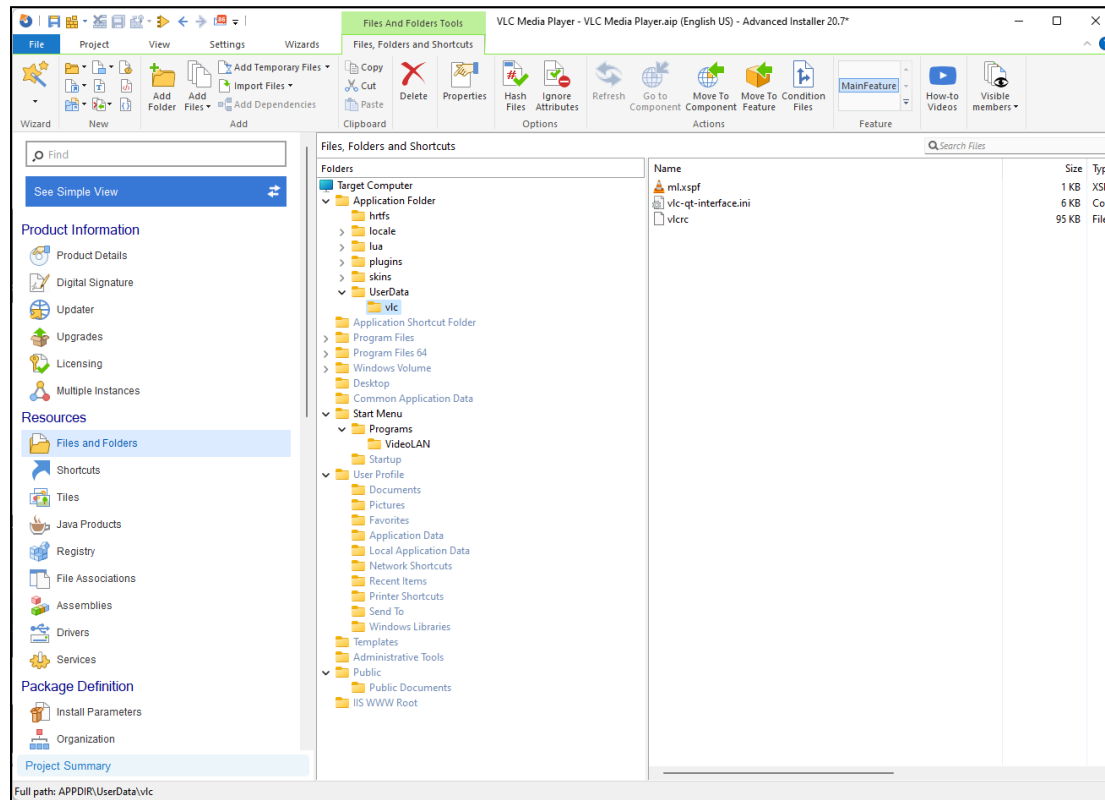


- Both methods rely on a per-machine location for the user files and a script that copies them. Let's look at each scenario and see how we can configure it.

We already have the advertised shortcut option on all of the shortcuts, so we'll keep it that way. Let's start with the simple part of this implementation: add a random HKEY CURRENT USER registry key if it isn't already present in the package. In our case, VLC contains the following user registry keys:



the previously created vlc folder under the Application Data and drop it under the newly created UserData. At the end, it should look like this:



We can now be certain that the files that must be copied on each user profile are always present on a per-machine basis, and that the files will be available even if the MSI is deleted. The final step is to create a custom action that copies the files from the UserData folder to the %appdata% folder.

As usual, we can do this either with VBScript or PowerShell. The code for the VBScript:

```
Dim objFSO, objShell
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objShell = CreateObject("WScript.Shell")

Dim sourceFolder, destinationFolder, programFilesPath, appDataPath

programFilesPath = objShell.SpecialFolders("ProgramW6432")
appDataPath = objShell.SpecialFolders("APPDATA")
```



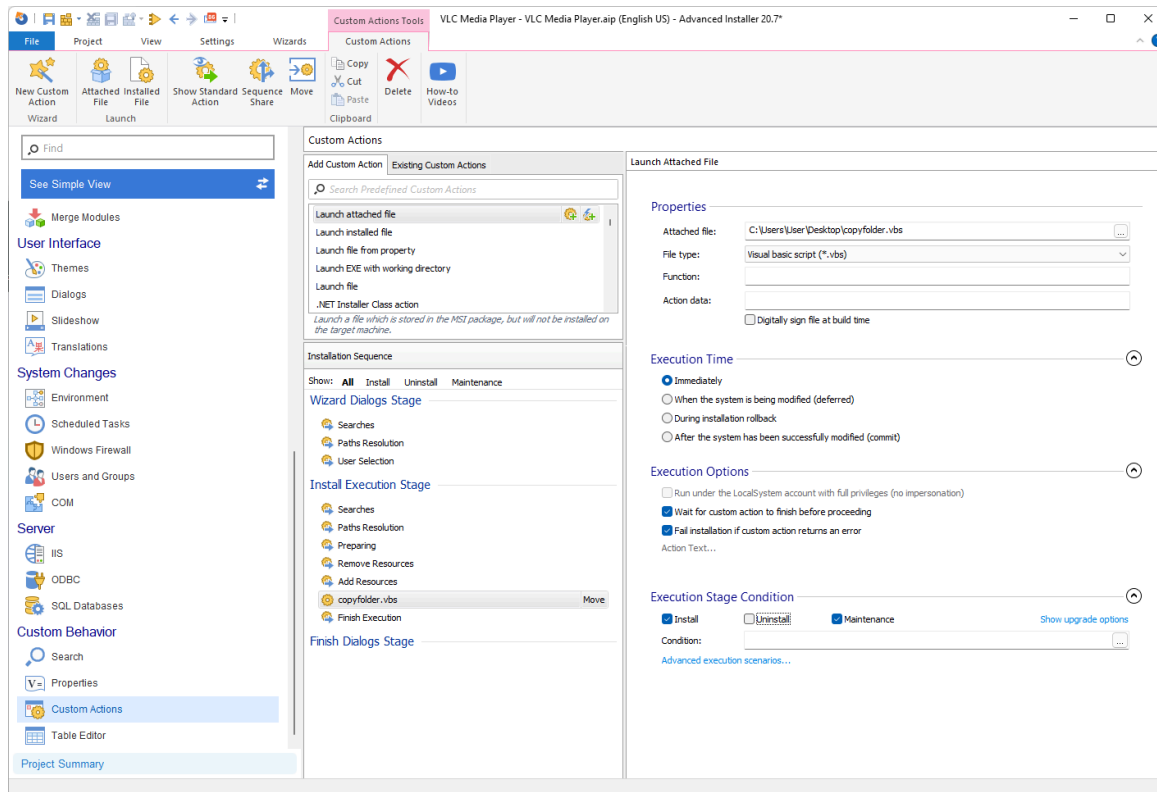
```
sourceFolder = programFilesPath & "\VideoLAN\VLC\UserData"  
destinationFolder = appDataPath & "\VLC"  
  
    ' Copy the folder and its contents  
    objFSO.CopyFolder sourceFolder, destinationFolder, True  
  
Set objFSO = Nothing
```

The following actions are carried out by the code snippet:

- Creates objects for the FileSystemObject and WScript.Shell to access file system operations and special folder locations.
- Retrieves the paths for the Program Files folder (ProgramW6432) and the AppData folder (APPDATA) using the SpecialFolders property of the WScript.Shell object.
- Constructs the source folder path by appending the desired subfolder path (in this case, "\VideoLAN\VLC\UserData") to the Program Files path.
- Constructs the destination folder path by appending the desired subfolder path (in this case, "\VLC") to the AppData path.
- Uses the CopyFolder method of the FileSystemObject to copy the entire contents of the source folder to the destination folder. The third parameter (True) indicates that the operation should overwrite existing files if necessary.
- Releases the resources by setting the FileSystemObject to Nothing.

Then, launch Advanced Installer and go to the Custom Actions Page. Look for the Launch attached file and select the VBScript's location. Then, as shown below, configure the custom action to execute:





The code for the PowerShell script:

```
$programfiles = $env:ProgramW6432
$appdata = $env:APPDATA
$sourcePath = $programfiles + "\VideoLAN\VLC\UserData"
$destinationPath = $appdata + "\VLC"

# Copy the folder and its contents recursively
Copy-Item -Path $sourcePath -Destination $destinationPath -Recurse -Force
```

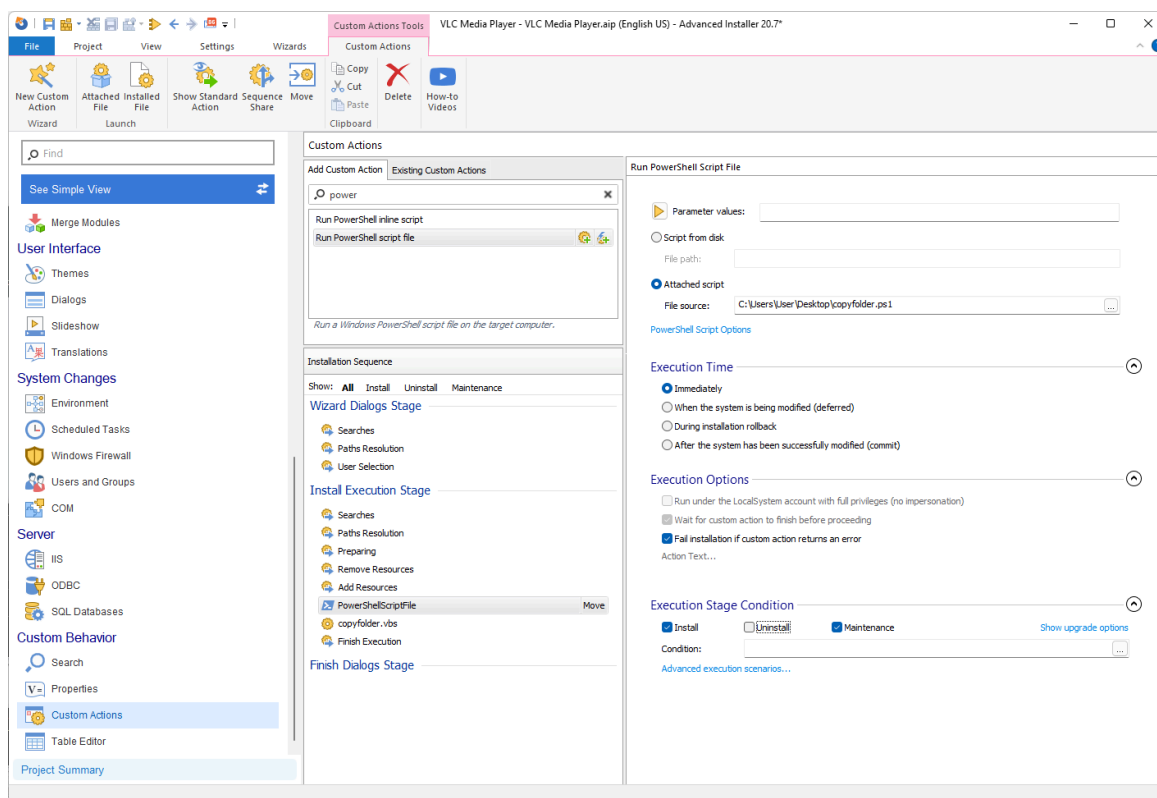
The following actions are carried out by the code snippet:

- Retrieves the values of the Program Files directory and the AppData directory using environment variables (\$env:ProgramW6432 and \$env:APPDATA).
- Constructs the source folder path by appending the desired subfolder path ("VideoLAN\VLC\UserData") to the Program Files directory.
- Constructs the destination folder path by appending the desired subfolder path ("VLC") to the AppData directory.



- Uses the Copy-Item cmdlet in PowerShell to recursively copy the contents of the source folder to the destination folder. The -Recurse parameter ensures that all files and subdirectories are copied, and the -Force parameter overrides any restrictions that would prevent the copying operation.
- The files and directories from the source path are copied to the destination path.

Then, launch Advanced Installer and go to the Custom Actions Page. Search for the Run PowerShell script file and select the PowerShell script's location. Then, as shown below, configure the custom action to execute:

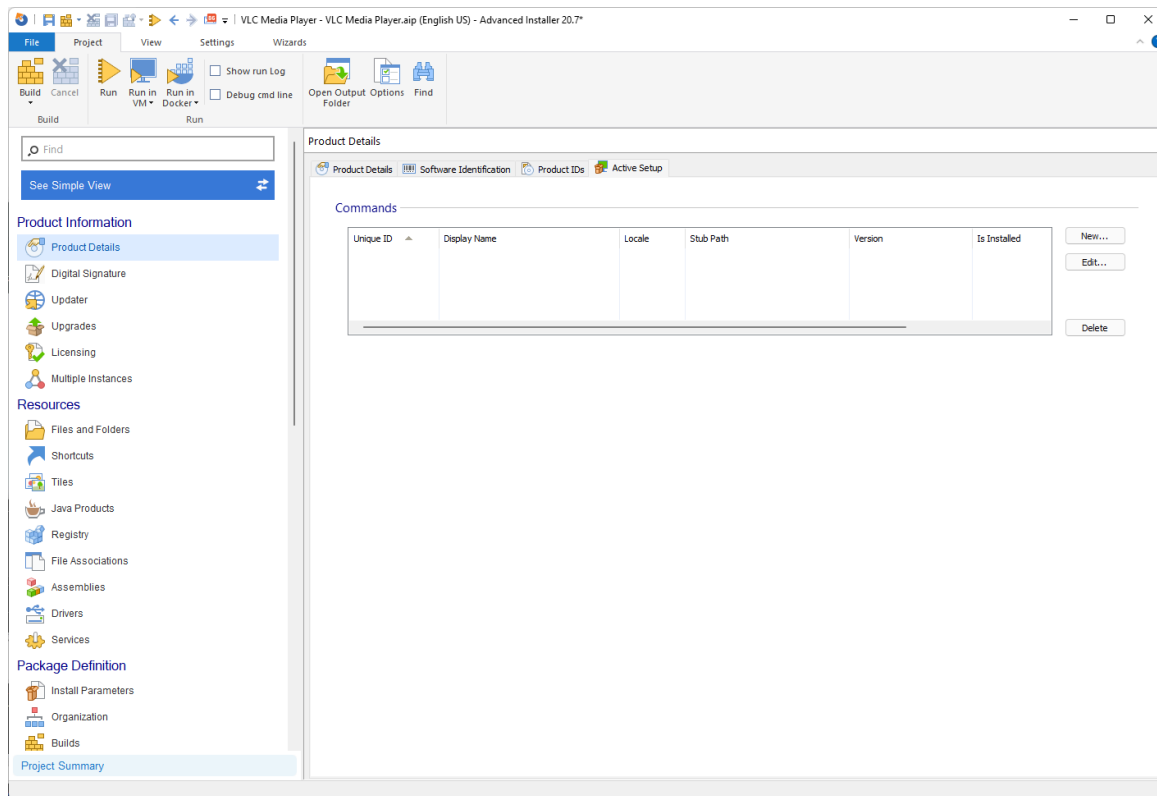


## Scenario two: Active Setup

The only difference between the two scenarios is that in the active setup case, we disable the advertised shortcuts and do not require any other HKCU registry keys.



After you've disabled the advertised shortcuts, go to the Product Details page and look for the Active Setup tab.



In here, click on New and you can leave everything as default in the new window that appears.



Active Setup Command

General

ID:

Display Name:

Stub Path:

Version:

Locale:

Is Installed:

Condition:

Help OK Cancel

We have discussed more in-depth about the [Active Setup mechanism in our first book](#), but if we check what the above settings mean, we end up with the following:

- ID: The ID which will appear in the Active Setup registry. This can be basically anything because the Active Setup mechanism will check if on the current logged in user the registry for Active Setup which is present under HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Active Setup is also present under HKEY\_CURRENT\_USER\SOFTWARE\Microsoft\Active Setup
- Display Name: The purpose of the "DisplayName" value is to provide a meaningful name or description for the Active Setup component so that users can understand its purpose or functionality. It helps users identify the component when they view it in registry editors or other software that displays registry information.
- Stub Path: By setting the "StubPath" value, software developers can define the action or task that should be executed when the Active Setup component runs. It allows them to perform various installation or configuration operations during user logon, ensuring that specific tasks are completed for each user. In our case we are triggering a repair of the installed MSI with /fou.
- Locale: The purpose of the "Locale" value is to define the language or regional settings that should be used when executing the Active Setup component. It allows software developers to provide localized versions of their installation or configuration tasks, ensuring that the appropriate language or regional settings are applied for each user.



- **Is Installed:** Software developers can use the "IsInstalled" registry key to check the installation status of their Active Setup component. During the Active Setup process, the component can set or update the "IsInstalled" value based on the successful completion of its installation or configuration tasks. This allows subsequent logons by the user to skip the installation process if the component is already installed.

So the active setup is doing exactly what we need, and that is to trigger a repair of the MSI when the user logs in again.

And that is it, in both cases after the application is installed we have ensured that even after the deletion of the MSI the users will still be able to receive the user data files in their profile. The difference between the two scenarios are the following:

- **If we are using the advertised shortcuts route**, every time a HKCU registry is missing the self-healing will start again. Also, the self-healing will start immediately after the user starts any shortcuts, meaning that after the application gets installed the user will also have the configuration once he starts the application
- **If we are using the active setup route**, this will only be executed once. Another downside of this method is that the user must log off/log on again in order for the Active Setup mechanism to start running and find the differences

## Registry classes

Classes in the Windows Registry enable the organization and management of file type associations and other related settings. Each file type association in the registry is represented by a class that defines the association's properties and behaviors. Classes can be used to define other system objects such as ActiveX controls, fonts, and COM objects in addition to file type associations.

The registry's classes are organized in a hierarchical structure that starts with the root key, HKEY CLASSES ROOT. Subkeys in this key represent file extensions, COM objects, and other objects that can be associated with classes. Each HKEY CLASSES ROOT subkey represents a specific class and contains values that define the class's properties and behaviors.

The class defines which application should be used to open files with a specific extension, as well as any command-line arguments or other settings required to open and handle the file





properly. Other behaviors defined by the class include whether the application should open the file in a new window or in an existing instance of the application.

Classes can be modified manually with the Registry Editor or other registry editing tools, or automatically with tools like Group Policy. IT professionals may need to modify classes to ensure that files are correctly opened and handled on their systems, or to prevent specific applications from opening specific file types.

Before we have a look at how to [manipulate the file extensions that VLC](#) offers support for, let us first understand what types of registry can be found on machines:

- COM
- Interfaces
- Type Libraries
- File Type Associations (FTA)
- COM+

## “COM” Registry

**Microsoft's COM (Component Object Model)** is a binary interface standard that allows software components to communicate with one another. Regardless of the programming language used to create the components, COM provides a standard way for applications to interact with one another and with the operating system.

COM is represented in the Windows Registry by a set of registry keys and values that define the properties and behaviors of COM components. These keys and values are used to register COM components on the system, specify which interfaces a component supports, and provide other configuration information to the operating system and applications.

COM components are represented in the Windows Registry by a series of keys and values that define the component's properties and behaviors. Among these keys and values are the following:

- CLSID: This key represents the component and includes subkeys for each version that is installed on the system. Each subkey contains values that specify the component's name, description, and other properties.
- InprocServer32: This key denotes the dynamic link library (DLL) that implements the component and contains values that specify the DLL's path and other properties.



- **Interface:** This key represents the interfaces exposed by the component and includes subkeys for each. Each subkey contains values that specify the interface's name, GUID, and other properties.

C++, Visual Basic, and .NET are just a few of the programming languages that can be used to create COM components. A COM component, once created, can be used by any application that supports COM, including those written in different programming languages. As a result, COM is a powerful and adaptable technology for developing software components and applications.

One of the key benefits of COM is the ability to enable late binding, which allows applications to call methods on a component without having to understand how the component works. This is accomplished by utilizing interfaces, which provide a standardized means for components to expose their functionality to other components. When an application calls a method on an interface, regardless of the programming language used to create the component, the COM runtime resolves the call to the correct implementation of the method.

Another benefit of COM is its versioning support, which allows components to be updated or replaced without breaking existing code that relies on them. This is accomplished by utilizing interfaces and versioning information stored in the Windows Registry. Developers can modify the implementation of a component without affecting its interface by defining interfaces for components and tracking their versions, ensuring that existing code continues to work as expected.

The following is an example of a registry entry for a COM component:

```
HKEY_CLASSES_ROOT\CLSID\{9BA05972-F6A8-11CF-A442-00A0C90A8F39}
(Default) = "Microsoft Windows Media Player"
InprocServer32 = "%ProgramFiles%\Windows Media Player\wmp.dll"
```

The above registry entry registers the Windows Media Player COM component on the system, specifying the default name of the component and the location of its DLL file.

## Interfaces

Interfaces are a fundamental component of COM and are used to define a component's methods and properties that are accessible to other components. Interfaces are represented in the registry by a series of keys and values that define the interface's properties and behaviors, including the methods and properties that are available to other components.



Interfaces are represented in the Windows Registry by a series of keys and values that define the interface's properties and behaviors. Among these keys and values are the following:

- **Interface:** This key represents the interface and contains values that specify the interface's name, GUID, and other properties.
- **Methods:** This key represents the interface's exposed methods and contains subkeys for each method. Each subkey contains values that specify the method's name, ID, and other properties.
- **Properties:** This key represents the interface's exposed properties and contains subkeys for each property. Each subkey contains values that specify the property's name, ID, and other properties.

COM components use interfaces to define their functionality and provide a standardized way for other components to interact with them. Interfaces are language-agnostic, which means that components written in different programming languages can communicate with one another via interfaces.

In COM, interfaces are also used to provide a mechanism for component versioning. Developers can change the implementation of a component without affecting its interface by defining interfaces for components. This means that components can be updated or replaced without interfering with existing code that relies on them.

Assume you have a COM component that provides email sending functionality. The component may define an `IMailSender` interface that exposes methods for sending email messages. Other components or applications can use this interface to interact with the email-sending component without having to understand its internal workings.

Here is an example of a registry entry for a COM interface:

```
HKEY_CLASSES_ROOT\Interface\{00020400-0000-0000-C000-000000000046}  
(Default) = "IDispatch"
```

The `IDispatch` interface, defined in the preceding registry entry, is used to provide automation support for COM components. It specifies the interface's default name and GUID.



## Type Libraries

Type libraries, which are used to define the data types and interfaces that a component supports, are another important aspect of COM. Type libraries are represented in the registry by a series of keys and values that define the properties and behaviors of the type library, including the interfaces and data types that are supported.

Type libraries are represented in the Windows Registry by a series of keys and values that define the type library's properties and behaviors. Among these keys and values are the following:

- **TypeLib:** This key represents the type library and includes subkeys for each version that is installed on the system. Each subkey contains values that specify the type library's name, description, and other properties.
- **Version:** This key denotes a particular version of the type library and includes subkeys for each interface defined in the type library. Each subkey contains values that specify the interface's name, GUID, and other properties.
- **Interface:** This key in the type library represents a specific interface and contains subkeys for each method and property defined in the interface. Each subkey contains values that specify the method or property's name, ID, and other properties.

Type libraries are commonly used by COM-compliant development tools and programming languages such as Microsoft Visual Basic, Microsoft Visual C++, and Microsoft.NET Framework. These tools use the type library information to generate code that can be used to call methods and properties on the COM component.

For example, if you have a COM component that exposes an interface for retrieving data from a database, you could generate code that calls the interface's methods and properties using a development tool like Visual Basic. The development tool would generate code based on the information in the type library, ensuring that the correct data types and method signatures are used.

Here is an example of a registry entry for a COM type library:

```
HKEY_CLASSES_ROOT\TypeLib\{1F2D0E65-8DFE-4BAF-A07E-4A4D4C4B1E9D}\1.0
(Default) = "Microsoft Excel 2010 Object Library"
```



The registry entry above defines the Microsoft Excel 2010 Object Library type library, which contains Excel's interfaces and data types. It specifies the type library's default name and GUID.

## File Type Associations

In the Windows Registry, file type associations represent the link between a file extension and the application that is used to open that file. When a user double-clicks on a file with a specific extension, the operating system searches the registry for the file type association to determine which application should be launched to open the file.

Each registry file type association contains several pieces of information, including the file extension, the application associated with the extension, and the command-line arguments passed to the application when the file is opened. This data is saved in a specific location in the registry, where it can be accessed and modified using tools like the Registry Editor or PowerShell.

Users or applications can change file type associations, and changes to these associations can affect how files are opened and handled on a given system. If an application that claims to handle a specific file type is installed, it may modify the file type association in the registry to ensure that it is launched when a file with that extension is opened. Basically, a file type association is a mapping between a file extension, such as .txt or .docx, and the application that should be used to open files with that extension, such as Notepad or Microsoft Word.

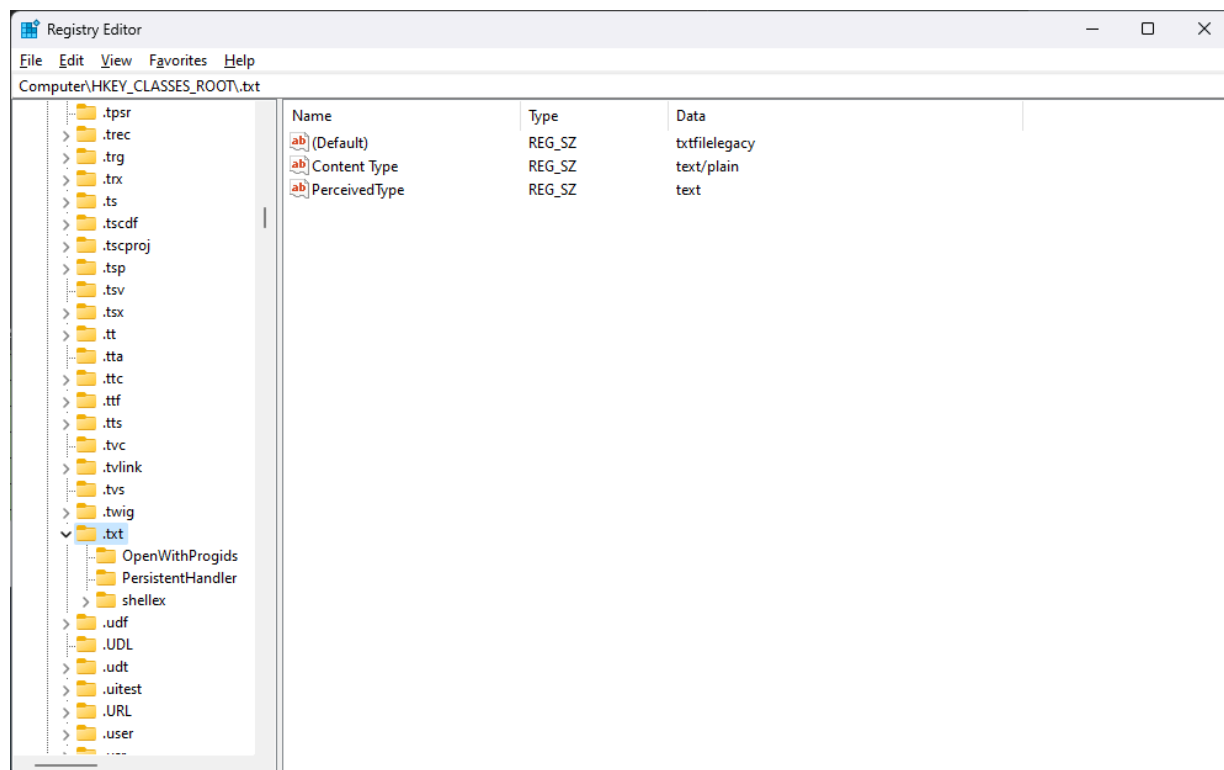
File type associations are represented in the Windows Registry by a series of keys and values that define the properties and behaviors of the association. Among these keys and values are the following:

- **File type:** This key represents the file type and includes subkeys for each file extension associated with it. Each subkey contains values that specify the file type's name, description, and other properties.
- **Shell:** This key represents the applications associated with the file type and includes subkeys for each. Each subkey contains values that specify the application's name, path, and other properties.
- **DefaultIcon:** This key contains values that specify the path to the icon file and represents the icon that is displayed for files with the associated file type.



If we take the .txt file extension as an example, HKEY\_CLASSES\_ROOT\.txt key represents the file extension ".txt" and contains values that define the properties and behaviors of the associated file type. Here are some examples of values that could be included:

- (Default): The name of the file type associated with the extension is specified by this value. The value in this case could be "txtfile."
- Content Type: This value specifies the file type's MIME type. The value for a text file could be "text/plain."
- PerceivedType: This value specifies the file type's perceived type. The value for a text file could be "text."
- Shell: This key represents the applications associated with the file type and includes subkeys for each. For example, there could be a "edit" subkey that specifies the application to be used to edit text files.
- DefaultIcon: This key represents the icon that is displayed for files with the associated file type and contains values that specify the path to the icon file.



## COM+

COM+ is a component services technology that extends the functionality of COM by providing additional features such as transactions, security, and queuing.

In the registry, COM+ is represented by a series of keys and values that define the properties and behaviors of the COM+ components and services, including the configuration information that is used by the operating system and applications.

COM+ has several features that make it useful for developing distributed applications, including:

- **Transaction support:** COM+ includes transactional support, allowing developers to create applications that can perform multiple actions in a single transaction. This is critical in distributed applications to ensure data consistency and reliability.
- **Object pooling:** Object pooling is a feature of COM+ that allows developers to reuse object instances across multiple client requests. This can boost application performance while lowering the overhead associated with creating and destroying object instances.
- **Automatic activation:** COM+ supports automatic activation, allowing objects to be created and destroyed as needed. This can improve application performance while reducing the overhead associated with object instance management.
- **Just-in-time activation:** Just-in-time activation in COM+ allows objects to be created and initialized only when they are required. This can boost application performance by lowering the overhead associated with creating and initializing objects that may never be used.

COM+ components are represented in the Windows Registry by a series of keys and values that define the component's properties and behaviors. Among these keys and values are the following:

- **CLSID:** This key represents the component and includes subkeys for each version that is installed on the system. Each subkey contains values that specify the component's name, description, and other properties.
- **InprocServer32:** This key denotes the dynamic link library (DLL) that implements the component and contains values that specify the DLL's path and other properties.



- **Interface:** This key represents the interfaces exposed by the component and includes subkeys for each. Each subkey contains values that specify the interface's name, GUID, and other properties.

COM+ also includes tools and utilities for managing and configuring distributed applications, such as the Component Services MMC snap-in, which allows developers to configure transaction settings, object pooling, and other COM+ component features.

Here is an example of a registry entry for a COM+ component:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\AppID\{00000000-0000-0000-0000-00000000  
0000}  
(Default) = "DefaultAppID"
```

Overall, the Windows Registry plays a critical role in managing and configuring COM components and related technologies such as interfaces, type libraries, and COM+.

By providing a standardized way to register and configure these components, the registry ensures that they are properly integrated into the system and available for use by applications and other components. IT professionals who work with COM and related technologies must have a thorough understanding of the registry and its structure in order to effectively manage and configure these components on their systems.

Let's take a look at how these classes communicate and interact with one another:

- **COM (Component Object Model):** In Windows, COM is the foundational technology for component-based development. It defines how software components communicate and interact with one another. COM allows components to be used across multiple applications and systems by facilitating inter-process communication. It specifies a set of rules and protocols that govern the instantiation, access, and management of components.
- **Interfaces:** Interfaces define the relationship between a component and the clients who use it. They define a set of methods, properties, and events that other components can access and use. Interfaces define how components interact with one another in a consistent and standardized manner. Components communicate in COM by invoking methods defined in interfaces.
- **Type Libraries:** Type Libraries provide a structured and standardized representation of interfaces, data types, and other components' elements. Type Libraries contain information about the interfaces that a component supports, as well as their methods,





properties, and parameters. They facilitate component communication and interoperability by providing a clear understanding of their capabilities and how they can be used.

- **COM+ (Component Services):** COM+ is a COM extension that adds features and capabilities for developing enterprise-level applications. It provides transaction management, object pooling, and distributed transactions among other services. COM+ components are intended to operate in a managed environment, enhancing reliability, scalability, and security.

The following are the relationships between these classes:

- Interfaces are the primary means of communication between components in COM. Interfaces exposed by components define the methods and properties they provide. Clients of a component use these interfaces to access the component's functionality.
- Type Libraries allow you to define and document a component's interfaces and other elements. They serve as a repository for information about the capabilities of the component, making it easier for clients to understand and interact with it.
- COM+ enhances COM's capabilities by providing additional services and features. COM+ components can communicate with other components via COM interfaces while leveraging COM+'s enhanced features for advanced functionality such as transaction management or object pooling.

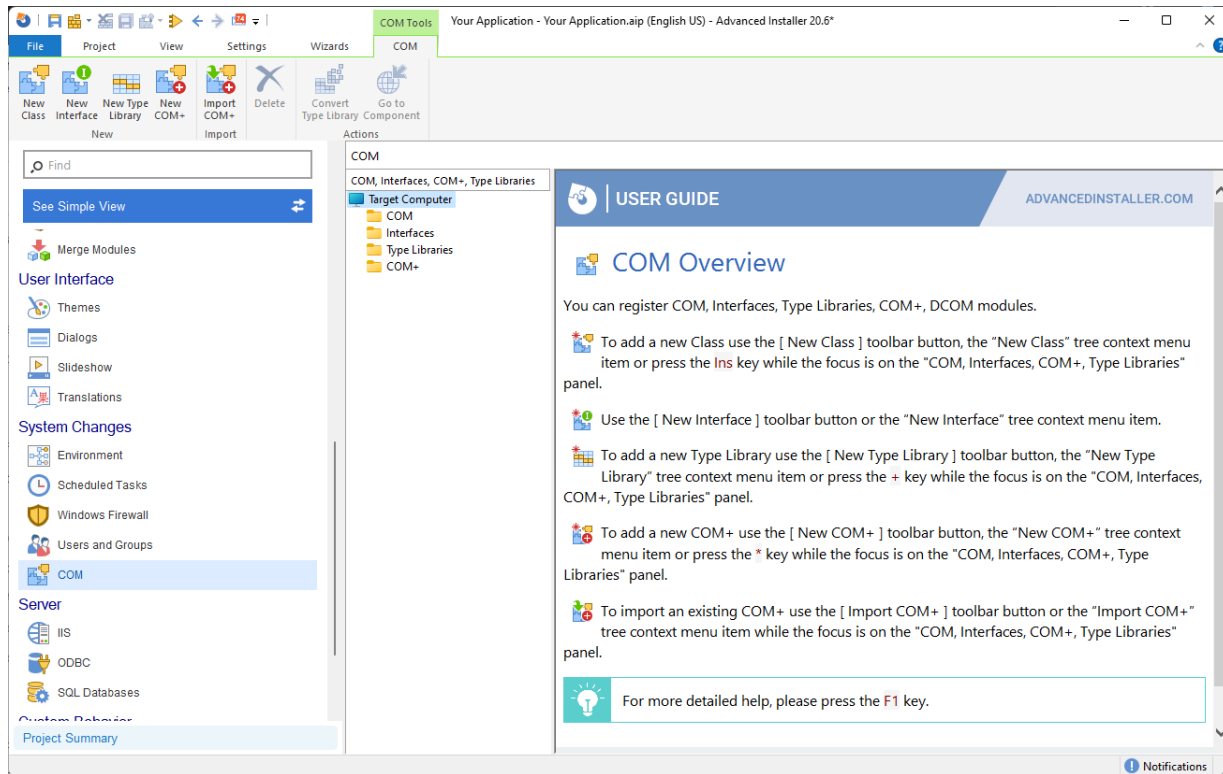
## Manipulating of registry classes with Advanced Installer

Advanced Installer supports working with COM components extensively, including the ability to define custom interfaces, type libraries, and classes. The Advanced Installer GUI can be used to define and register these components.

One of the primary advantages of using Advanced Installer for COM component work is its ability to generate registry entries and other configuration data automatically based on the information defined in the installation package. This can greatly simplify the process of creating and managing COM components, particularly in complex applications requiring multiple components and interfaces.



Advanced Installer offers two separate pages for the users to work with different types of classes. When it comes to COM components, Interfaces, Type Libraries and COM+ components, you can define this information directly into the GUI by navigating to the [COM Page](#).



When it comes to File Type Associations, Advanced Installer offers a separate GUI where you can define these and this can be found in the [File Associations Page](#).



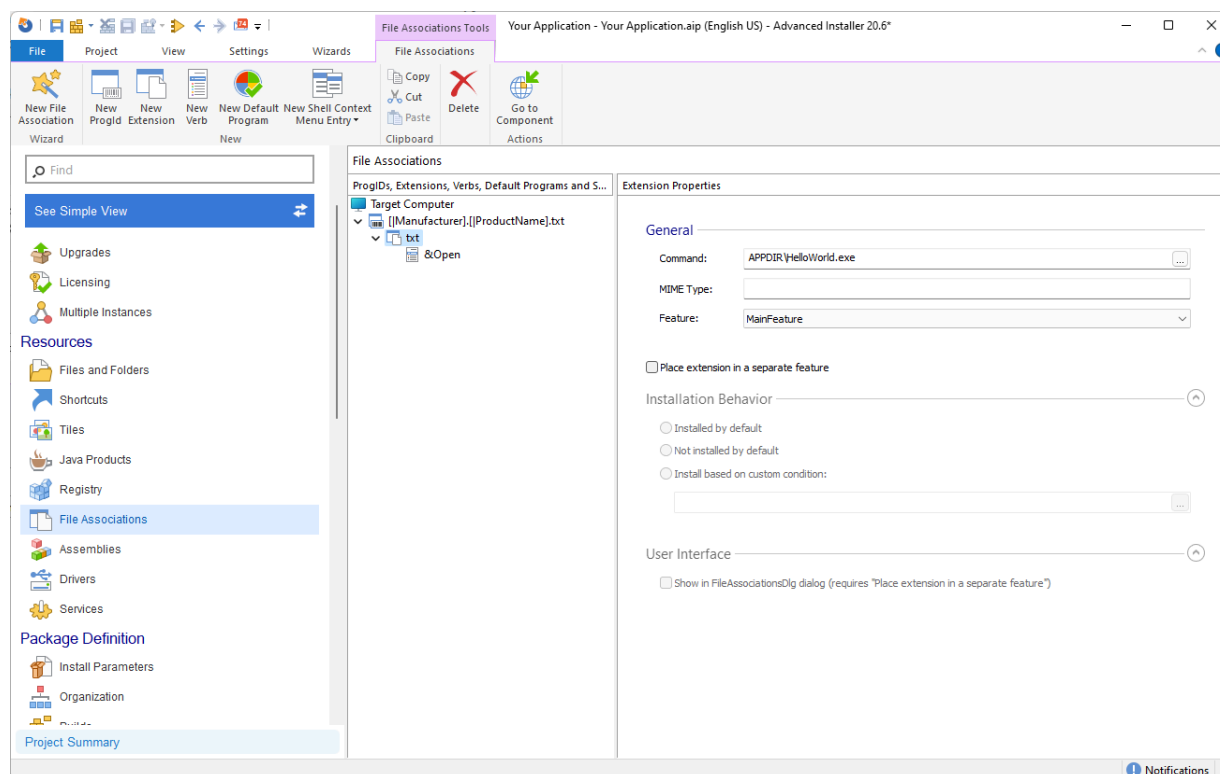


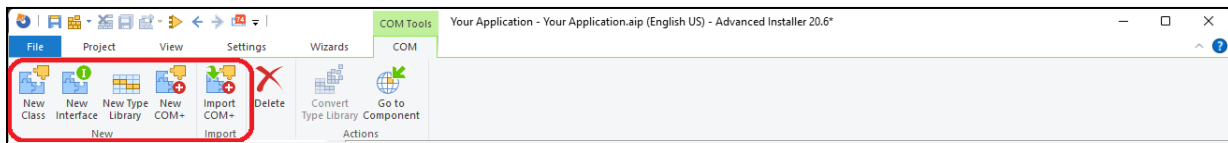
image-id	ai-file-associations-page.png
image-title	
image-alt-text	

## COM Page

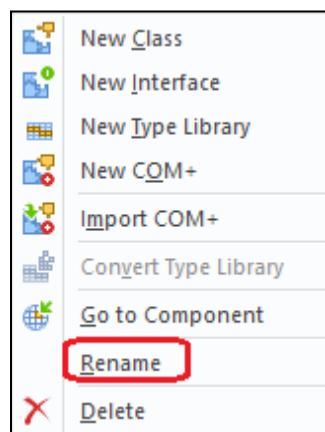
Advanced Installer includes a number of useful tools for working with COM components, such as the ability to create new classes, interfaces, and type libraries. These tools are accessible via the Advanced Installer interface's "COM, Interfaces, COM+, Type Libraries" panel, which allows developers and system administrators to easily create, edit, and manage COM components for use in their applications.

Simply click the corresponding toolbar button, tree context menu item, or keyboard shortcut to add a new class, interface, or type library. To add a new class, for example, while the "COM, Interfaces, COM+, Type Libraries" panel is focused, click the "New Class" button or press the Insert key. Similarly, use the "New Interface" or "New Type Library" toolbar buttons to add a new interface or type library.





Advanced Installer includes tools for editing and deleting existing components in addition to adding new ones. Use the "Rename" tree context menu item or press the F2 key while the element is selected to rename it. Use the "Delete" toolbar button, the "Delete" tree context menu item, or the Delete key while the element is selected to delete it.

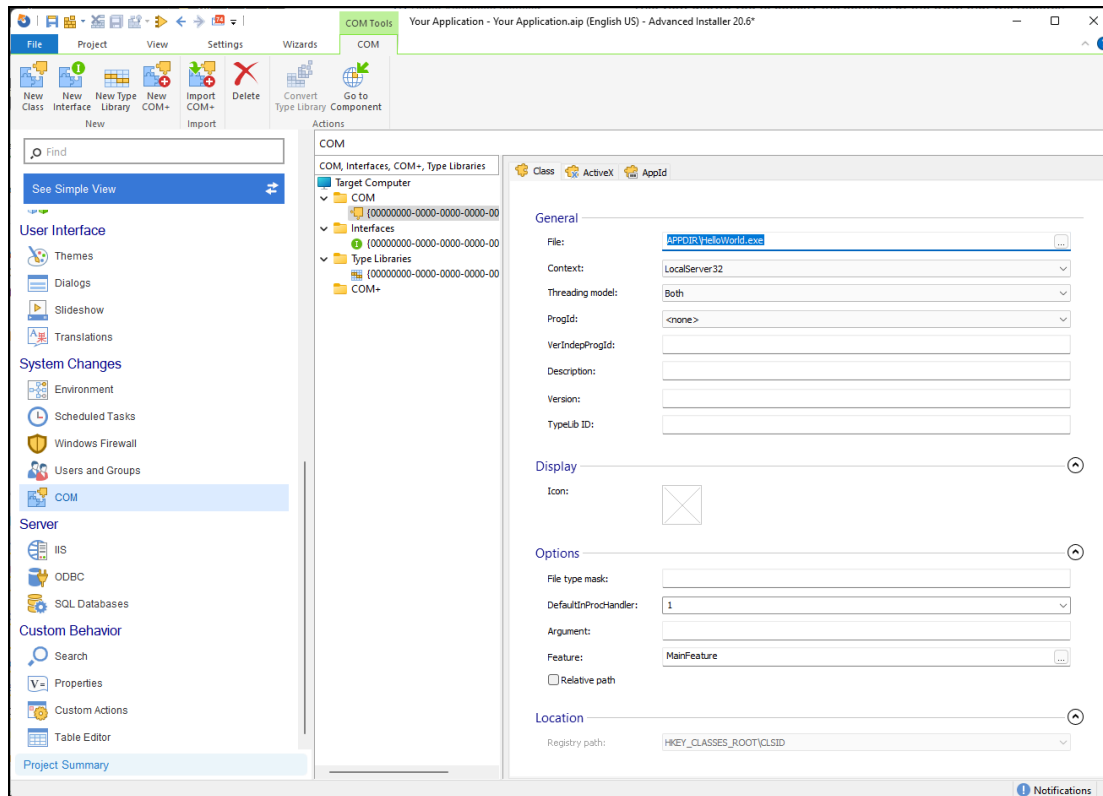


Advanced Installer also includes tools for importing COM+ components and converting type libraries for more advanced tasks. Use the "Import COM+" toolbar button or context menu item to import a COM+ component. Use the "Convert Type Library" toolbar button or tree context menu item to convert a type library. When converting a type library, the COM registration registries contained within the file are extracted and silently imported into the Registry page, bypassing the TypeLib table.

## COM Properties

Once a COM component has been added to the project, the properties tab view allows you to specify the [settings of that particular COM](#).





The interface in the view is simple and straightforward, allowing IT professionals to easily configure the settings for their installation package. When registering COM files with Advanced Installer, you have several options:

- **General Settings:** The "General" tab displays some general information about the COM file that is being registered. You can specify the file to be registered, the server context, and the threading model here. You can also give the COM file a description and a version number.
- **ProgId and VerIndepProgId:** You can specify the Program IDs associated with the Class ID using the "ProgId" and "VerIndepProgId" options. Advanced Installer's File Associations page is where program IDs are defined. You can also specify a version-independent Program ID in the Installer Project's File Associations or Registry page.
- **TypeLib Id:** You can enter the ID of the TypeLibrary that describes the COM here. This option is especially useful when dealing with a large number of COM files.
- **Display and Options:** The "Display" and "Options" tabs allow you to specify various settings related to the appearance and behavior of the COM file. You can select an icon

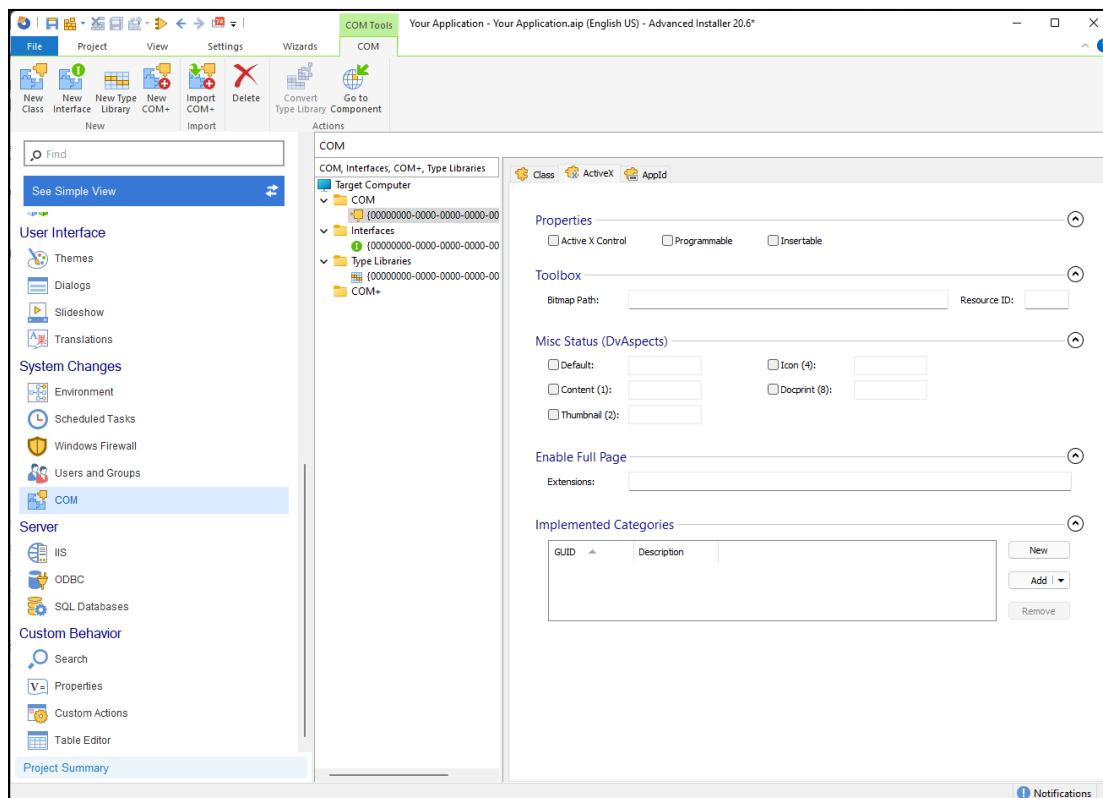


that will be associated with the CLSID, specify the file type mask, and select the default in-process handler for the server context.

- **Argument and Feature:** The "Argument" and "Feature" options are related to the CLSID keys LocalServer or LocalServer32. The "Argument" option allows you to register a text as the server's argument, which is used by COM to invoke the server. The "Feature" option lets you choose which feature provides the COM server.
- **Relative Path and Location:** Finally, you can specify whether the register path is relative or absolute, as well as the registry location where the COM is registered, using the "Relative Path" and "Location" options. The registry location for COMs defined through the MSI Class table is read-only.

## COM ActiveX Properties

If the COM has any ActiveX settings, you can easily configure them in the [ActiveX Properties Tab](#).



ActiveX is a subset of COM that is designed specifically for use in web browsers, whereas COM is a more general-purpose technology used to create software components that can be used by any application.

ActiveX controls are essentially a type of COM component that is specifically designed to work within a web browser. Typically, these controls are used to add functionality to a website or web application, such as displaying multimedia content or interacting with user input. They are built with the same technologies and techniques as other COM components, but with some web-specific features and restrictions.

When looking over the ActiveX view, you have the following options:

- **ActiveX Controls:** ActiveX controls are a type of COM component that can be used to add interactive features to web pages or desktop applications. When registering an ActiveX control, it is important to mark it as such in the properties.
- **Programmable COMs:** Programmable COMs are COM components that can be used in programming languages such as Visual Basic and C++. Registering a programmable COM requires specifying it as such in the properties.
- **Insertable Objects:** Insertable objects are a type of COM component that can be inserted into other applications, such as a Word document. When registering an insertable object, it is important to indicate that objects of this class should appear in the Insert Object dialog box list box when used by COM container applications.

**Additional Properties:** Aside from these main properties, there are several other properties that can be defined when registering a COM. These include:

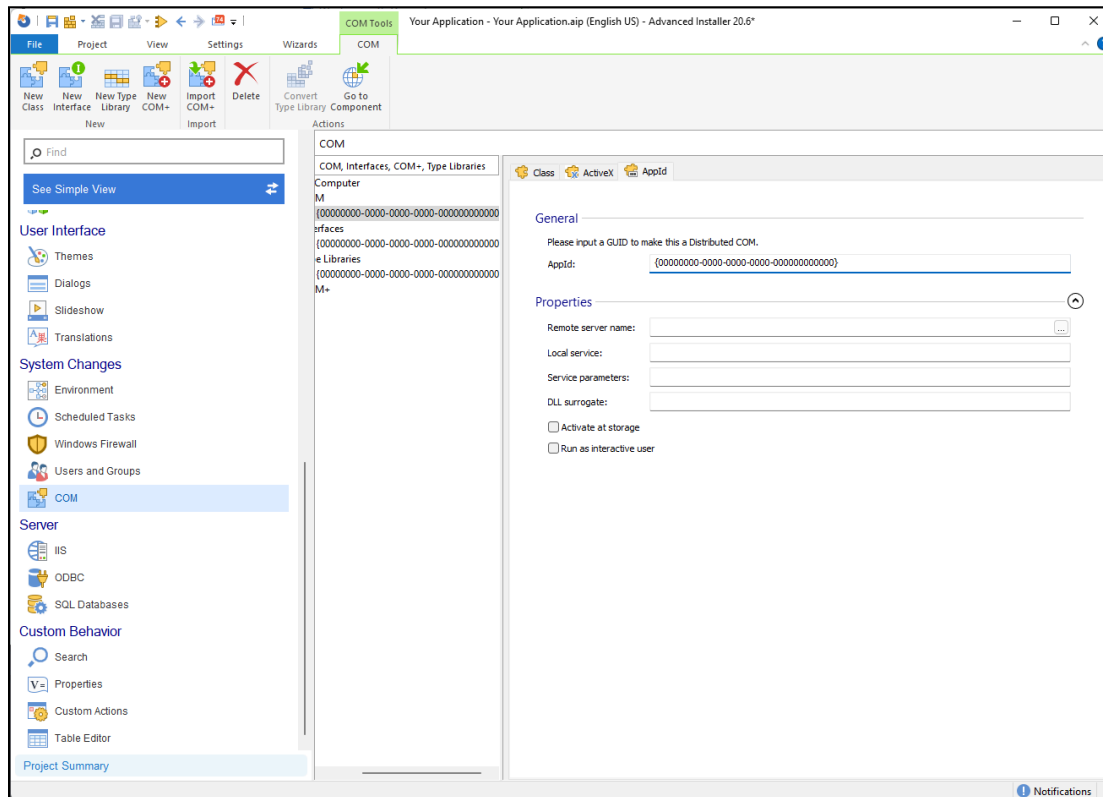
- **Toolbox:** This specifies the 16x16 bitmap to use for the face of a toolbar or toolbox button.
- **Misc Status:** This property specifies how to create and display an object, the desired data or view aspect of the object when drawing or getting data.
- **Extensions:** This property is used to register the control as the viewer for specified extensions.
- **Implemented Categories:** This property is used to specify the categories this COM implements.

To define new Implemented Categories, you can use the "New" button, the "Add" combobox, or the "Add" context menu. There are also predefined implemented categories, such as "Implemented in .NET", "Safe for scripting", and "Safe for initializing".



## DCOM Properties

If you wish to configure and register DCOM servers, you can easily input a GUID to transform it into a Distributed COM. This can be done in the [AppID Tab](#).



DCOM is a Microsoft technology that allows software components to communicate with one another directly over a network. It is a Component Object Model (COM) extension that provides the infrastructure for developing distributed applications. DCOM enables remote access to components over a network and allows developers to create applications that are distributed across multiple machines.

DCOM follows a client-server model, in which the client sends a request to the server, which processes the request and returns a response to the client. The client and server can communicate using a variety of protocols, including TCP/IP, UDP, HTTP, and HTTPS.

DCOM is commonly used in enterprise settings where applications must be distributed across multiple servers and workstations. It enables developers to create distributed applications that can be easily scaled to meet business requirements. However, installing and configuring DCOM can be difficult, requiring a thorough understanding of the underlying network infrastructure.



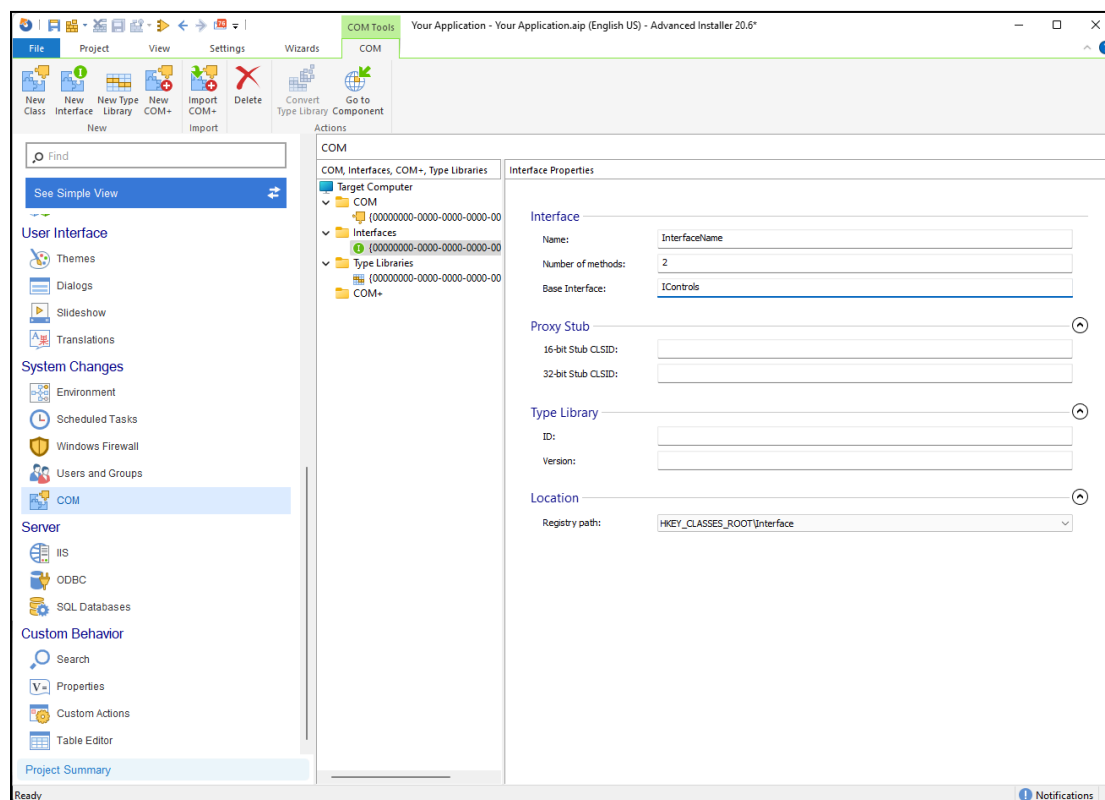


The AppId value can be specified in the General section of the AppId Properties tab. This value is written to the CLSID and generates the AppId GUID key in HKCR\AppId.

The AppId Properties tab's Properties section contains several fields that allow you to configure the DCOM server. The Remote Server Name field is a Formatted Type field that will be written under HKCR\AppId{AppId}. The Local Service field specifies the local service that will be stored in the HKCR\AppId{AppId} field. The Service Parameters field specifies which service parameters will be stored under HKCR\AppId{AppId}. The Dll Surrogate field specifies the Dll surrogate that will be stored in the HKCR\AppId{AppId} folder, and this field is usually left blank.

The Active At Storage checkbox is used to enable the "ActivateAtStorage"="Y" value, which is stored in HKCR\AppId{AppId}. The Run As Interactive User checkbox allows you to tell the DCOM server to run as an interactive user. This value will be written as "RunAs"="Interactive User" under HKCR\AppId{AppId}.

## Interface Properties



Advanced Installer has a dedicated view where you can specify the settings needed to create a new COM Interface. This view provides a comprehensive set of properties for customizing the interface and its components which includes the following fields:

- **Name:** This field allows you to specify the name of the interface. Choose a descriptive name that reflects the purpose and functionality of the interface.
- **Number of methods:** Here, you can specify the number of methods declared in the interface. This information helps in accurately defining the interface's behavior and functionality.
- **Base Interface:** Specify the base interface that the current interface extends or inherits from. This is useful when creating derived interfaces that build upon existing interfaces.

The Proxy Stub Properties section contains the following fields:

- **Proxy Stub:** This field allows you to provide marshaling support for your interface. Marshaling is a process that facilitates communication between different processes or systems by converting data between different formats or representations.
- **16-bit Stub CLSID:** Specify the CLSID (Class ID) of the 16-bit proxy/stub DLL associated with the interface. This DLL handles the marshaling process for the interface in 16-bit environments.
- **32-bit Stub CLSID:** Similarly, specify the CLSID of the 32-bit proxy/stub DLL associated with the interface. This DLL is responsible for marshaling the interface in 32-bit environments.

The Type Library Properties section includes the following fields:

- **ID:** Specify the ID of the type library associated with the interface. The type library contains information about the interface's structure, methods, properties, and other relevant details.
- **Version:** Define the version of the type library. This helps in managing compatibility and versioning of the interface.

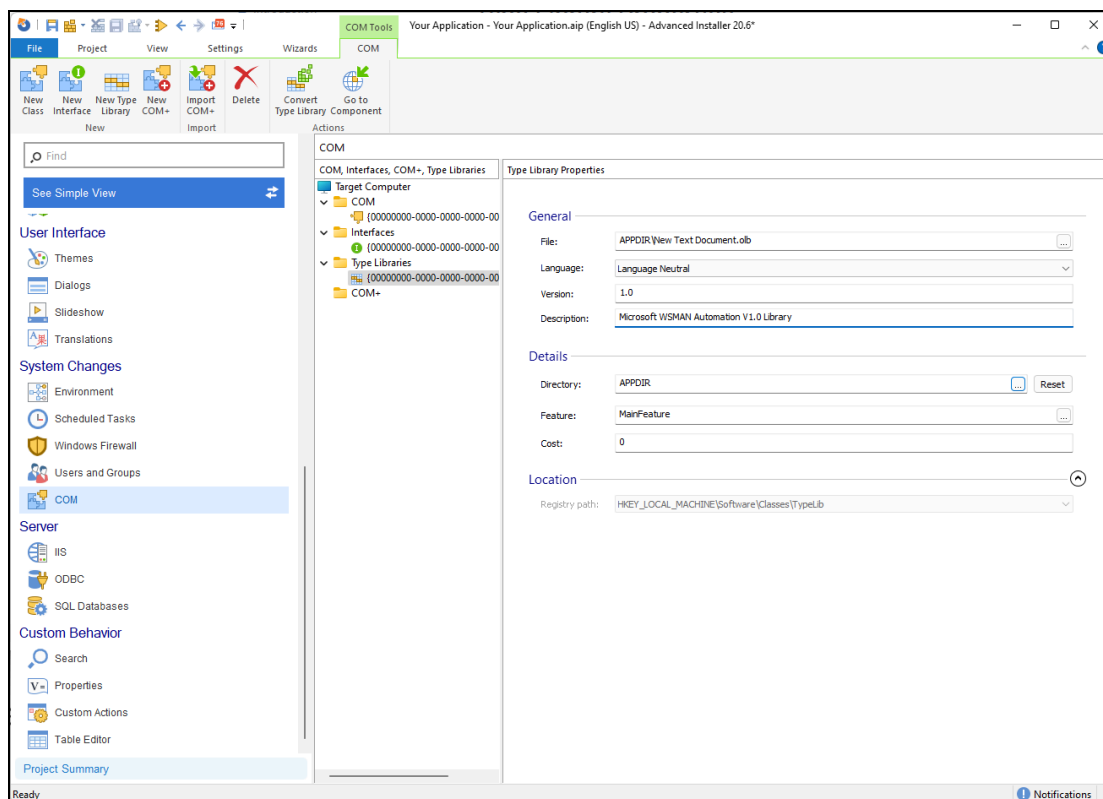
The Location section specifies the registry path where the COM Interface is registered. This path indicates the location in the Windows Registry where the necessary information about the interface is stored.



By configuring these properties in Advanced Installer, you can effectively define and customize COM Interfaces for your applications. These interfaces facilitate communication and interoperability between software components, enabling seamless integration and interaction within your applications.

Proper configuration of COM Interfaces ensures consistency, compatibility, and efficient communication between different software components. Advanced Installer simplifies the process, providing a user-friendly interface for specifying the required settings and ensuring a smooth integration of COM Interfaces into your applications.

## Type Libraries



The Type Library Properties view in Advanced Installer allows you to specify the information that will be stored in the registry during the Type Library's registration process. This allows for the Type Library to be seamlessly integrated and accessible within your applications.

**The General section** includes the following properties:



- **File:** This field displays the file that contains the Type Library. You can use the [ ... ] button to browse and select a different file if needed.
- **Language:** Specify the language of the Type Library. You can select the appropriate language from the drop-down list provided.
- **Version:** Define the version of the Type Library. The version follows the format of "major version dot minor version," allowing you to specify the specific version of the Type Library.
- **Description:** Provide a description for the Type Library. This description helps to provide additional information about the Type Library's purpose and functionality.

The **Details section** includes the following properties:

- **Directory:** Specify the directory that contains the Help file for the Type Library. You can use the [ ... ] and [ Reset ] buttons to navigate and select the appropriate directory.
- **Feature:** Select the feature that must be installed in order for the Type Library to be operational. This ensures that the necessary components are installed along with the Type Library to support its functionality.
- **Cost:** Specify the cost associated with the Type Library properties in bytes. This helps to manage the resource allocation and prioritize the installation of Type Libraries based on their cost.

The Location section specifies the registry path where the Type Library is registered. This path indicates the location in the Windows Registry where the Type Library's registration information is stored.

Note that for Type Libraries defined through the MSI TypeLib table, this location is read-only and determined by the installation package.

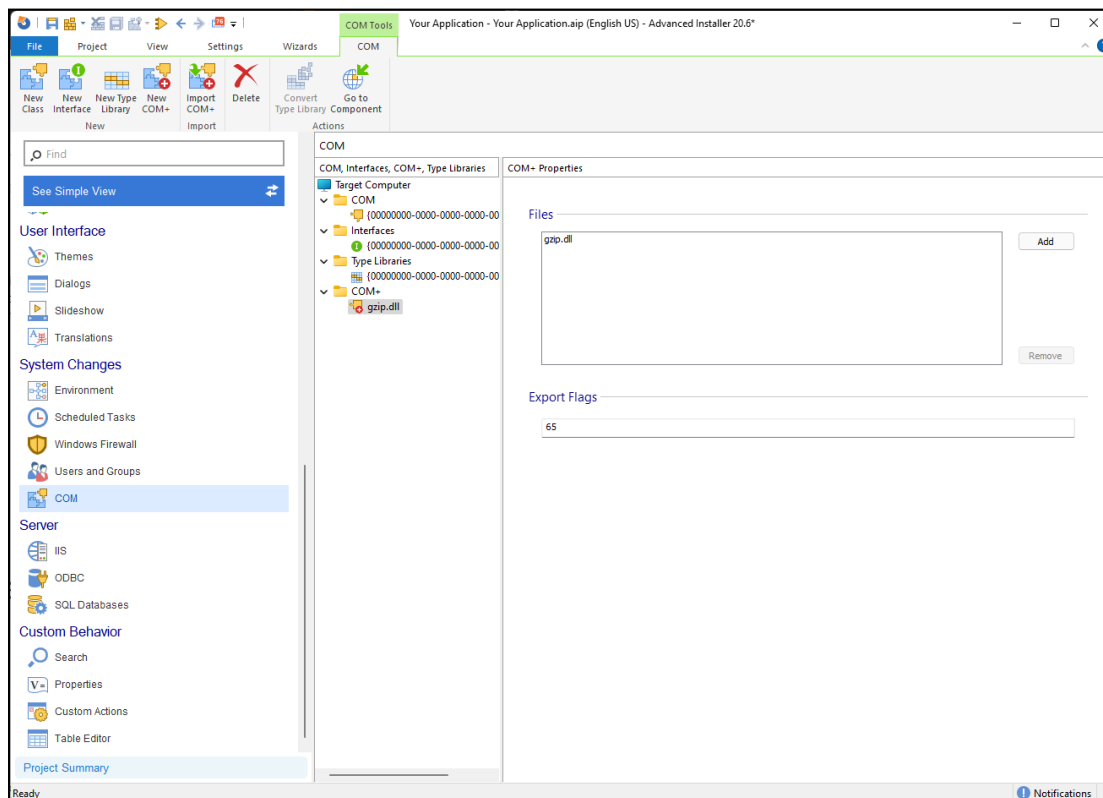
By configuring these properties in Advanced Installer, you can ensure that the Type Library is registered correctly and can be accessed by your applications. The provided information, such as file, language, version, and description, contributes to the seamless integration and functionality of the Type Library.

Proper configuration of Type Library properties is essential for maintaining consistency, compatibility, and accessibility of Type Libraries within your software solutions. With Advanced



Installer's user-friendly interface, you can easily specify the required information and streamline the registration process.

## COM+



The Advanced Installer COM+ Properties view provides a comprehensive set of options for specifying the information required for registering a COM+ component. This view allows you to define the files associated with the COM+ as well as the flags used during the MSI file creation process.

The Files section enables you to select the files that make up the COM+ component.

It is important to note that all the files belonging to a specific COM+ component must be placed within the same component in Advanced Installer. This ensures proper organization and management of the COM+ files.



To add files to the COM+ component, click the [ Add ] button and browse for the desired files. Conversely, if you need to remove any files from the COM+ component, use the [ Remove ] button. By managing the files within the same component, you ensure that all necessary dependencies and resources are properly registered and packaged during the installation process.

The Flags section allows you to specify the flags used when creating the MSI file for the COM+ component. Flags provide additional information and instructions to Windows Installer during the installation process. These flags can control various aspects of the installation, such as installation options, behavior, or special requirements.

When configuring the flags, it is important to understand the specific requirements and functionality of the COM+ component. The flags can be set according to the specific needs and characteristics of the component to ensure that it is installed and configured correctly.

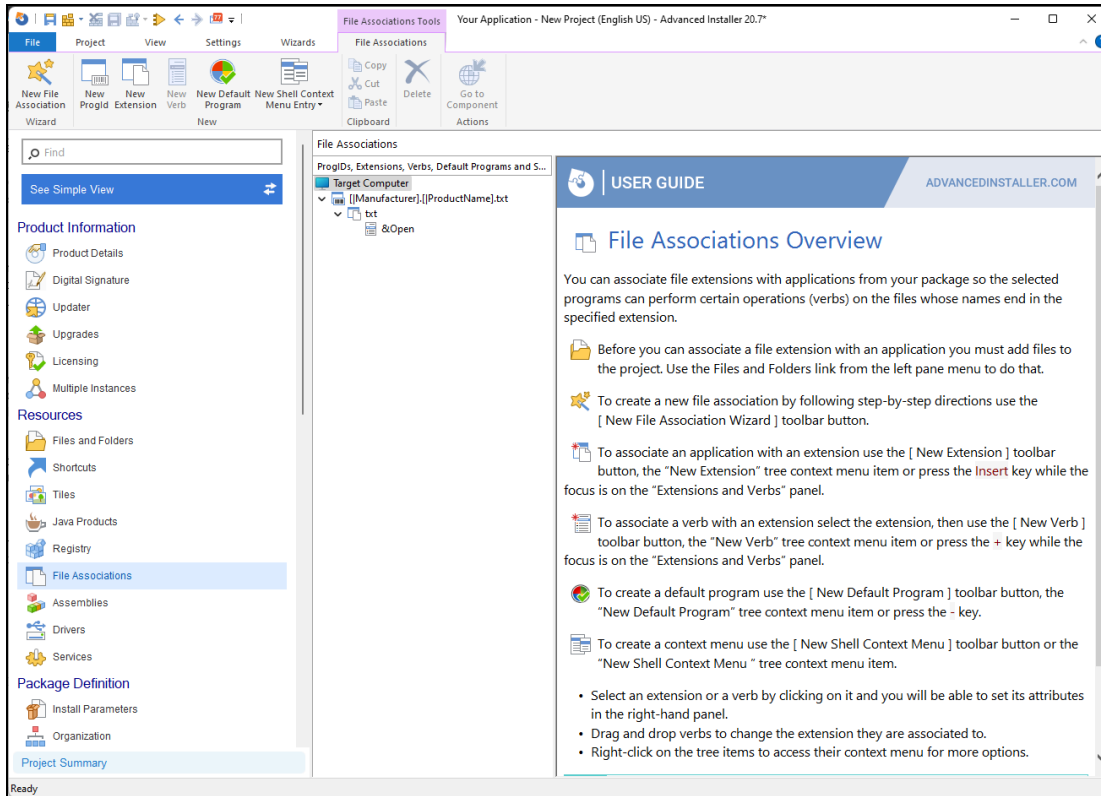
By carefully configuring the files and flags in Advanced Installer's COM+ Properties view, you can ensure that the COM+ component is properly registered and packaged within the MSI file. This guarantees seamless integration and functioning of the COM+ component within your application.

It's worth noting that Advanced Installer makes configuring COM+ properties easier by providing an intuitive interface that walks you through the steps. This enables IT professionals to manage COM+ components effectively and ensure their successful installation and operation.

## File Associations Page

You can associate specific file extensions with applications in your package, allowing the selected program to perform specific operations (verbs) on files with those extensions. Advanced Installer includes a page dedicated to creating and managing file associations, making it simple to define relationships between ProgIDs, extensions, and verbs.





In the left-side pane of the File Associations view, you'll find a tree with all the ProgidIDs defined at the top level. Each ProgidID can have multiple associated extensions, with each extension capable of defining multiple verbs.

Advanced Installer includes the New File Association Wizard, which allows you to quickly and easily create a new file association. Simply click the [New File Association Wizard] toolbar button and follow the on-screen instructions to set up the association.

You have several options for creating a new ProgidID. While the "ProgidID, Extensions, and Verbs" panel is focused, use the [New ProgidID] toolbar button, the "New ProgidID" tree context menu item, or press the \* key. When you create a ProgidID, you can associate it with an extension or a Component Object Model (COM) in the COM page. It should be noted that unless a ProgidID is associated with an extension or a COM, it will not be created during installation.

A ProgidID will not be created at install time unless it is associated with an extension or a COM.

Only one extension should be associated with each ProgidID.



Use the [New Extension] toolbar button, the "New Extension" tree context menu item, or the Insert key while the "ProgID, Extensions, and Verbs" panel is focused to create new extensions. This action generates a ProgID as well as an extension with default properties. If you want to add an extension to an existing ProgID, make sure to select it (or one of its children) before clicking [New Extension].

Use the [New Verb] toolbar button, the "New Verb" tree context menu item, or the + key while the "ProgID, Extensions, and Verbs" panel is focused to create new verbs. This action will generate a verb with the selected extension's default properties.

To establish a new default program, utilize the [ New Default Program ] toolbar button, the "New Default Program" tree context menu item, or press the - key while the "ProgID, Extensions, Verbs, and Default Programs" panel is focused. This will create a default program with default properties for the selected extension.

The Default Programs feature applies only when the package is installed on Windows Vista or later. For lower systems, it's ignored.

The Default Programs feature does not apply on Windows 8 and newer operating systems. Due to their design, these operating systems don't allow setting the default programs programmatically.


Use the [New Shell Context Menu] toolbar button to create a new shell context menu, and then select the type of context menu you want to create: Files Context Menu, Directory Context Menu, Background Context Menu, or a subentry for the currently selected shell context menu.





Context Menu Properties

☐ Windows 11 Context Menu


A Sparse Package is required to configure this option. Advanced Installer will automatically generate and include it into your setup package. All Sparse Packages must be digitally signed.

Display

Name:

Icon:

General

Type: Background

Command:

Advanced Installer also offers the option to define Context Menus for Windows 11. A Sparse Package is required to configure this option. It will be automatically generated and included into your setup package.

Digital signature is mandatory for this option. All Sparse Packages have to be digitally signed, thus the above Sparse Package will be signed with the digital signature configured in your project.

Simply use the "Rename" tree context menu item or press the F2 key while the element in the left-side panel is selected to rename ProgIDs, extensions, verbs, default programs, and context menus.

Use the "Go To Component" tree context menu item or press the F8 key while an element from the "ProgID, Extensions, and Verbs" panel is selected to locate the attached component for an extension. This command brings up the Organization page, with the appropriate component selected in the left tree control.



# Advanced MSI Packaging Techniques

## Custom actions

The most popular scripting language for MSI custom actions is VBScript as this is also natively supported and interpreted by the Windows Installer.

Microsoft's Visual Basic Scripting Edition (VBScript) is a lightweight scripting language. It was designed primarily to automate administrative tasks, boost productivity, and enable scripting capabilities in Windows environments.

VBScript is a procedural scripting language with syntax similar to Visual Basic. The Windows Script Host (WSH), a component built into Windows operating systems, interprets and executes it. VBScript files, which typically have the ".vbs" extension, can be run directly or through other programs.

VBScript executes code line by line and stores scripts in plain text files. It does not require explicit variable declarations and supports a wide range of data types such as strings, numbers, booleans, arrays, and objects. Statements end with a newline or a colon, and comments are added with apostrophes or the "Rem" keyword.

Some of VBScript's core features:

- **Variables and Constants:** VBScript uses variables and constants to store and manipulate data. Variables can have values assigned to them and be changed throughout the script, whereas constants have fixed values.
- **Operators:** VBScript includes a number of operators that allow data manipulation and decision-making within scripts, including arithmetic, comparison, logical, concatenation, and assignment operators.
- **Control Structures:** To control the flow of execution based on certain conditions, VBScript provides control structures such as conditional statements (If-Then-Else, Select Case) and loops (For-Next, Do-While, Do-Until).
- **Functions and Procedures:** VBScript allows you to create custom functions and subroutines to encapsulate reusable code blocks. Functions can return values, whereas subroutines execute code without returning a value.



VBScript includes a number of built-in objects and libraries for interacting with the environment, such as the FileSystemObject for file operations, the WScript object for script control and user interaction, and ADODB objects for database connectivity.

In any scripting language, error handling is critical, and VBScript provides mechanisms for structured error handling via the "On Error" statement. It enables developers to handle runtime errors gracefully, log exceptions, and take appropriate actions.

**Best Practices:**

- Use explicit variable declarations to improve code readability and avoid unexpected behavior.
- Comment your code adequately to enhance maintainability and readability.
- Implement error handling routines to gracefully handle runtime errors and provide meaningful feedback.
- Modularize your code by using functions and subroutines to promote code reusability and maintainability.
- Follow established coding conventions and style guidelines to ensure consistency and ease of collaboration.

As you will see in the examples below, VBScript accepts a wide variety of commands. If you are a beginner or want to remember some commands, I recommend you check out the [SS64 documentation of VBScript](#).

Unlike VBScript, PowerShell has emerged as a robust and versatile scripting language for modern IT environments. PowerShell combines the power of scripting with command-line functionality, providing administrators and developers with a powerful toolset to automate tasks, manage systems, and streamline workflows.

PowerShell is a task-based, object-oriented scripting language that runs on the .NET framework. It includes a command-line interface (CLI) and a scripting environment, which allow users to run commands interactively or write scripts for automation. PowerShell scripts are usually saved with the ".ps1" file extension.

PowerShell scripts are made up of cmdlets, which are small commands that perform specific tasks. These cmdlets can be combined and tailored to produce powerful scripts. PowerShell, like VBScript, supports variables, operators, loops, conditional statements, functions, and error handling. PowerShell's syntax, on the other hand, is based on a verb-noun naming convention for cmdlets, making it more intuitive and readable.



PowerShell treats everything as an object, making it simple to manipulate and access system resources like files, services, registry entries, and more. This object-oriented approach unifies the management and interaction with various data sources, making PowerShell a versatile and adaptable scripting language.

PowerShell's pipeline feature is one of its key strengths. The pipeline enables powerful data manipulation and processing by allowing the output of one cmdlet to be directly used as input for another. PowerShell also supports output formatting, filtering, sorting, and grouping, giving users complete control over data presentation.

PowerShell has a large number of modules and cmdlets that extend its functionality. These modules enable users to interact with a variety of technologies such as Active Directory, Azure, SQL Server, SharePoint, Exchange, and others. Furthermore, PowerShell allows developers to create custom modules, allowing them to encapsulate their code for reuse and distribution.

If we compare PowerShell with VBScript we can touch on a few points:

- **Syntax and Readability:** The syntax of PowerShell, which is based on natural language commands, is often considered more readable and intuitive than the syntax of VBScript, which is similar to Visual Basic. The verb-noun naming convention in PowerShell makes it easier to understand the purpose of commands.
- **Object-Oriented Approach:** PowerShell syntax, which is based on natural language commands, is often considered more readable and intuitive than VBScript syntax, which is similar to Visual Basic. PowerShell's verb-noun naming convention makes it easier to understand the purpose of commands.
- **Extensive Module Support:** PowerShell's vast module ecosystem gives it access to a diverse set of technologies, enhancing its capabilities and allowing for seamless integration with a variety of platforms and services. This level of extensive module support is not available in VBScript.
- **Pipeline and Output Processing:** The powerful pipeline feature in PowerShell allows for efficient data manipulation and processing. Because VBScript lacks this native capability, developers must write additional code to achieve similar functionality.
- **Integration with .NET:** PowerShell makes use of the .NET framework, which provides access to a wide range of system APIs and libraries. This integration enhances PowerShell's capabilities and makes it suitable for modern IT environments.

With its rich feature set, object-oriented approach, and extensive module support, PowerShell has transformed the world of scripting and automation. It is a versatile and powerful scripting



language for managing and automating complex IT tasks due to its intuitive syntax, pipeline functionality, and integration with the .NET framework.

While VBScript was once a popular scripting language, PowerShell now provides enhanced capabilities, improved readability, and a vast ecosystem of modules and cmdlets. Using PowerShell allows administrators and developers to automate tasks, streamline workflows, and realize the full potential of their IT environments.

## Dehardcode within files

Often, throughout the repackaging process, it's common to come across files featuring straightforward paths, such as: *C:\Program Files*.

Allowing users to modify the installation path of their application is a common practice. However, this practice comes with one downside: the need to change the path written in those particular files during the installation.

In this article, we'll see how we can replace hard coded paths or variables inside files using custom actions.

### How to Discover Hard-coded Files?

Before we actually replace the hard coded paths, we first need to find out which files contain hard-coded paths. There are various ways to get this information, but the most straightforward one is to use Notepad++.

For instance, let's imagine we have an application installed in *C:\Program Files\MyApp*. As a general rule, when repackaging, we always search for the **installation directory inside all of the application files**.

In this scenario, the installer sets the [ProgramFilesFolder](#) property to the full path of the Program Files folder. So we don't need to search for the full *C:\Program Files\MyApp* string, only for the part that we can use as a variable - in our case, the *C:\Program Files\* string.

Keep in mind that "Program Files" is an internal property known by the MSI.

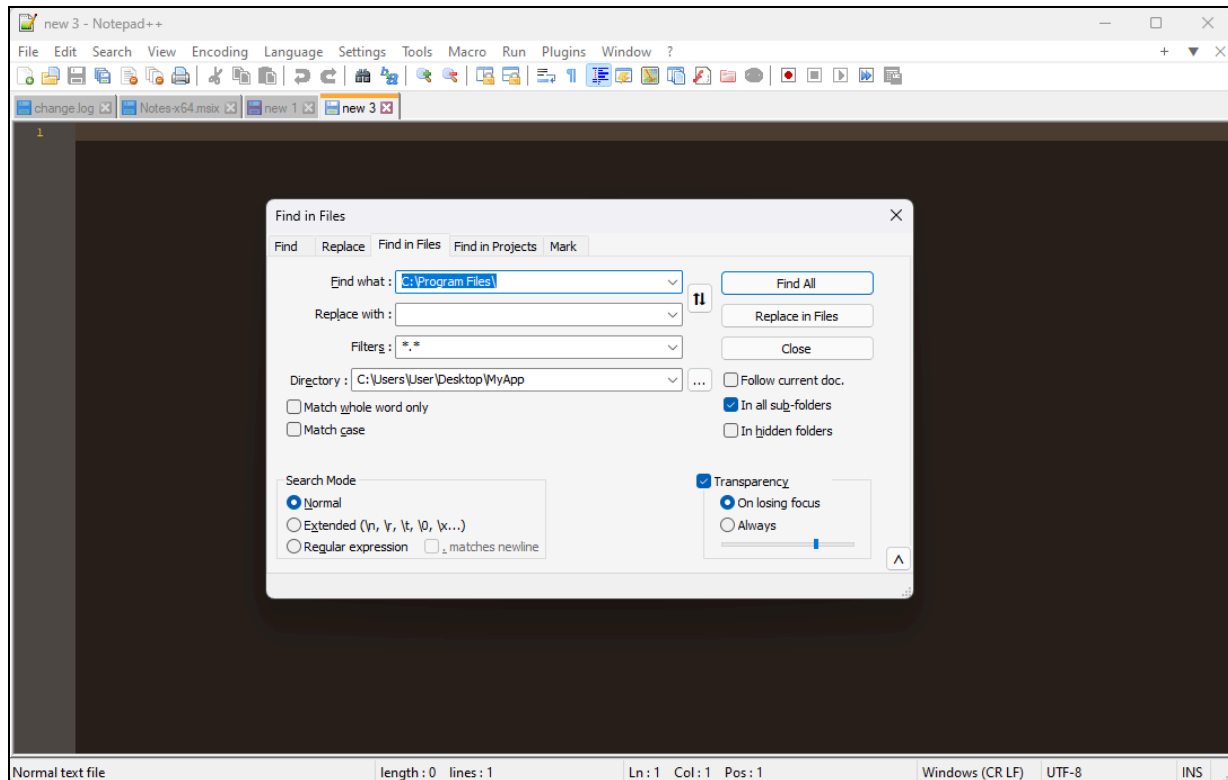
Other properties include [AppDataFolder](#), [CommonAppDataFolder](#), [CommonFiles64Folder](#), [CommonFilesFolder](#), [DesktopFolder](#), [FontsFolder](#), [LocalAppDataFolder](#), [MyPicturesFolder](#),



[PersonalFolder](#), [PrimaryVolumePath](#), [ProgramFiles64Folder](#), [ProgramMenuFolder](#), [StartMenuFolder](#), [System64Folder](#), [SystemFolder](#), [TempFolder](#), [WindowsFolder](#) and [WindowsVolume](#). By using one of these properties, we can de-hardcode part of the installation directory as a best practice.

To see which files contain the C:\Program Files\path, follow these steps:

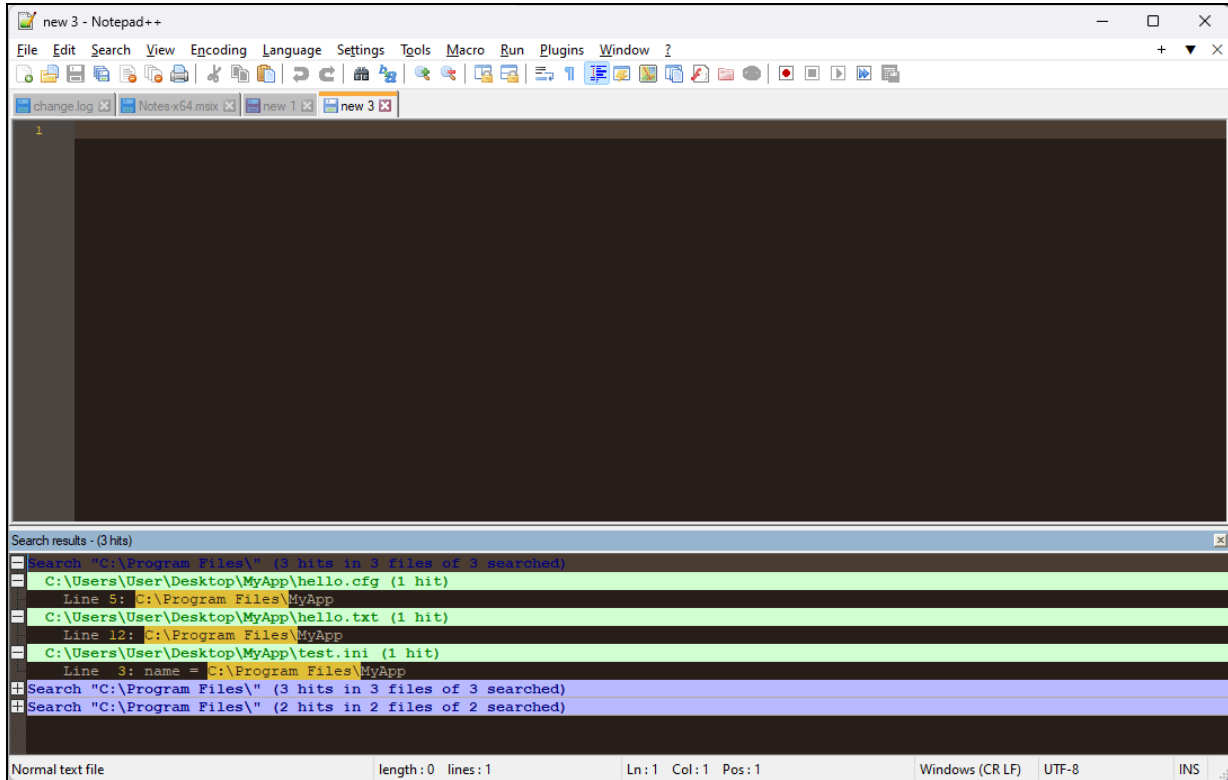
1. Open Notepad++ and go to **Search > Find in Files**.



2. In the **Find What** field, input the INSTALLDIR of your application, in our case: C:\Program Files\.
3. In the **Directory** field, input the location of your captured files.
4. Then, click on **Find All**.

If there are any matches found, Notepad++ will show you at what **file and line** you can find the string.





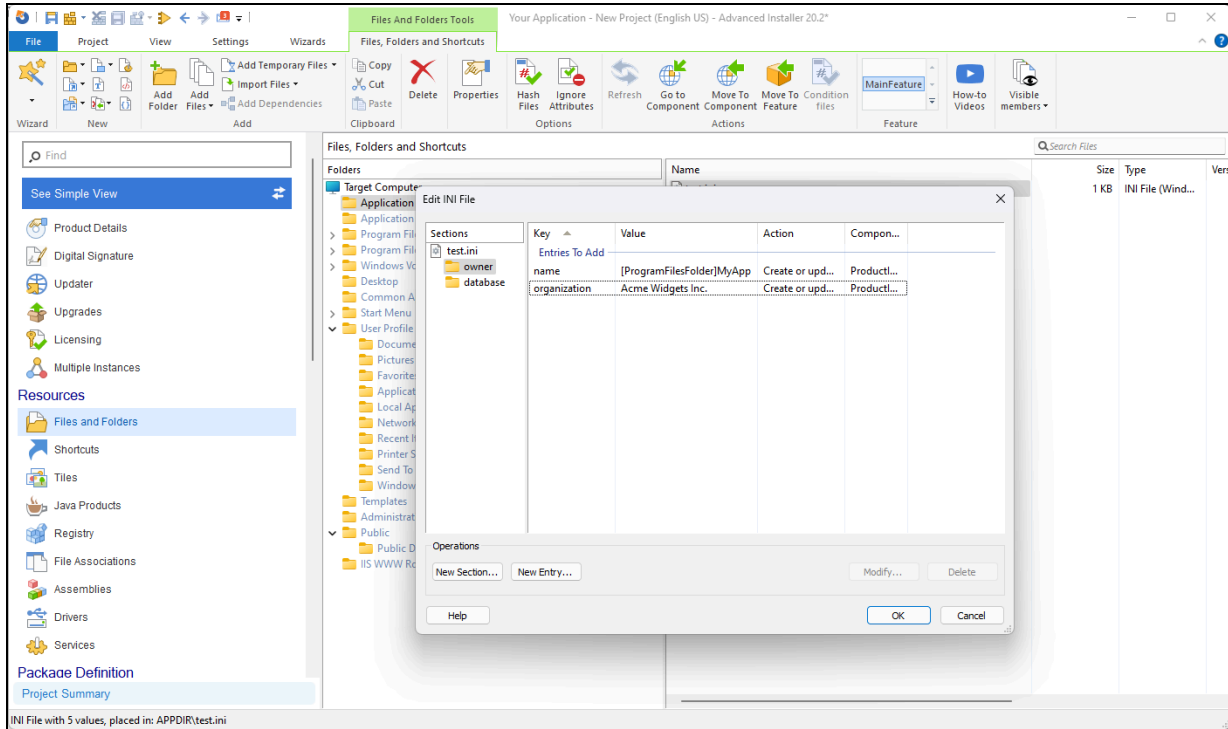
### How Does Automatic INI De-hardcoding Work?

As you can see from above, Notepad++ found our location inside an INI file. However, working with INI files becomes a straightforward process when using Advanced Installer. During the repackaging process, the Advanced Installer imports all INI files into the project and **de-hardcodes them automatically**.

Additionally, when you add a new INI file while editing your project, Advanced Installer easily identifies potential variables and automatically de-hardcodes the file.

Want to see Advanced Installer in action? Check it out through our [30-day full featured free trial](#) and elevate your installation process today!





## De-hardcoding Files Using VBScript Custom Actions

When it comes to other types of files, we must perform additional steps to de-harden the locations we need. In this case, we'll use the MSI's custom actions capability.

For **VBScript Custom Actions**, you can use the following script:

```
'On Error Resume Next
'Option Explicit

Const ForReading = 1, ForWriting = 2, ForAppending = 8
Dim strFilePath, strToReplace, strNewValue, path1, path2
strArgs = Session.Property("CustomActionData")
arrArgs = Split(strArgs, ";", -1, 1)

path1 = arrArgs(0)
path2 = arrArgs(1)

Function ReplaceInFile(strToReplace, strNewValue, strFilePath)

    Dim objFSO, objFile, strText, re
```





```

Set objFSO = CreateObject("Scripting.FileSystemObject")
if objFSO.FileExists(strFilePath) Then
    Set objFile = objFSO.OpenTextFile(strFilePath, ForReading,
True)
        strText = objFile.ReadAll
    objFile.Close
    Set objFile = Nothing

    strText = Replace(strText, strToReplace, strNewValue, 1, -1,
0)

    Set objFile = objFSO.CreateTextFile(strFilePath, True)
    objFile.Write strText
    objFile.Close
    Set objFile = Nothing

End If

Set objFSO = Nothing

End Function

strToReplace = "C:\Program Files\"
strNewValue = path1
ReplaceInFile strToReplace, strNewValue, path2 & "hello.cfg"

```

The script is quite long, so let's try to understand what it does.

Up to line 8, we are using **Session.Property("CustomActionData")**; this allows VBScript to catch any arguments passed to it.

Check out our article and learn [How to set an installer property using custom actions](#).

At line 9, the ";" separates the parameters, and **path1** represents the first argument. The **path2** represents the second argument that we pass.

Next, we can see the **ReplaceInFile** function, which takes the following arguments:

- **strToReplace** - the string we want to replace.
- **strNewValue** - the new value we want to add.
- **strFilePath** - the file path where we want to do the replacement.



The function opens the file mentioned in the **strFilePath** and parses all the text via **objFile.ReadAll**.

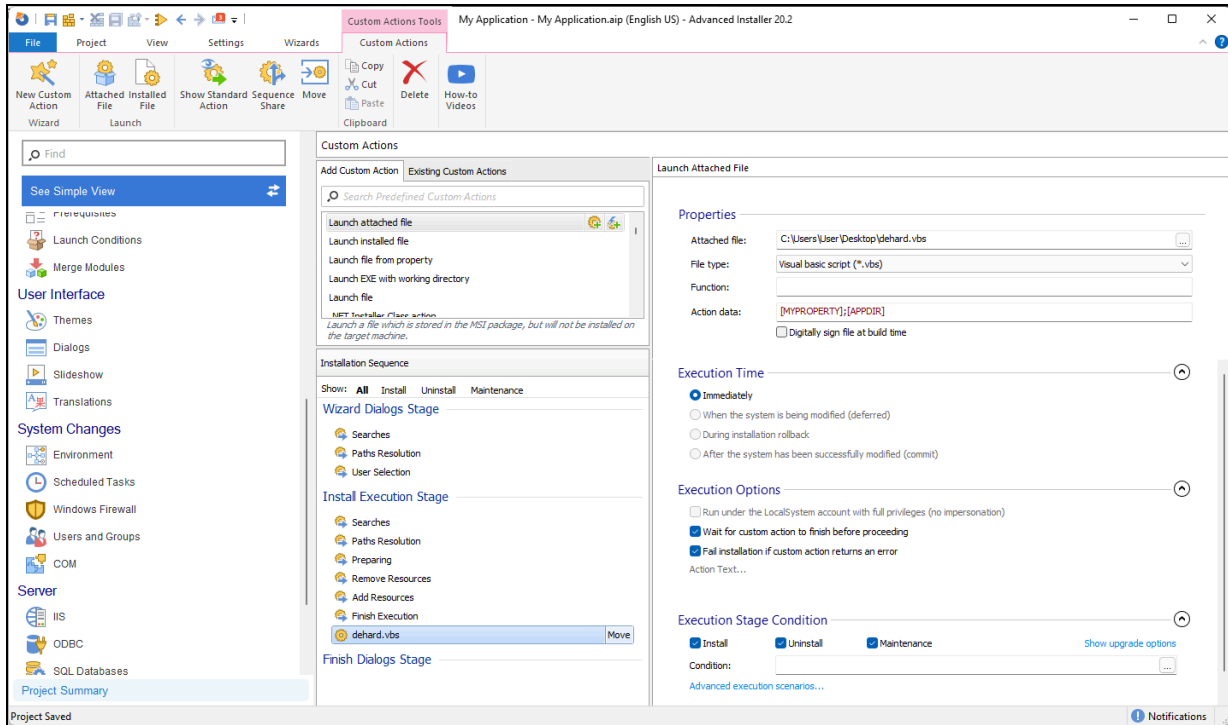
Then, we use the [Replace](#) function to replace the string. We can write it in the file with **objFile.Write strText** and close the file with **objFile.Close**. After that, we can dispose of the object with **Set objFile = Nothing**.

The values for **strToReplace** and **strNewValue** are defined below the function. Lastly, we call the function to perform the actions with **ReplaceInFile strToReplace, strNewValue, path2 & "hello.cfg"**.

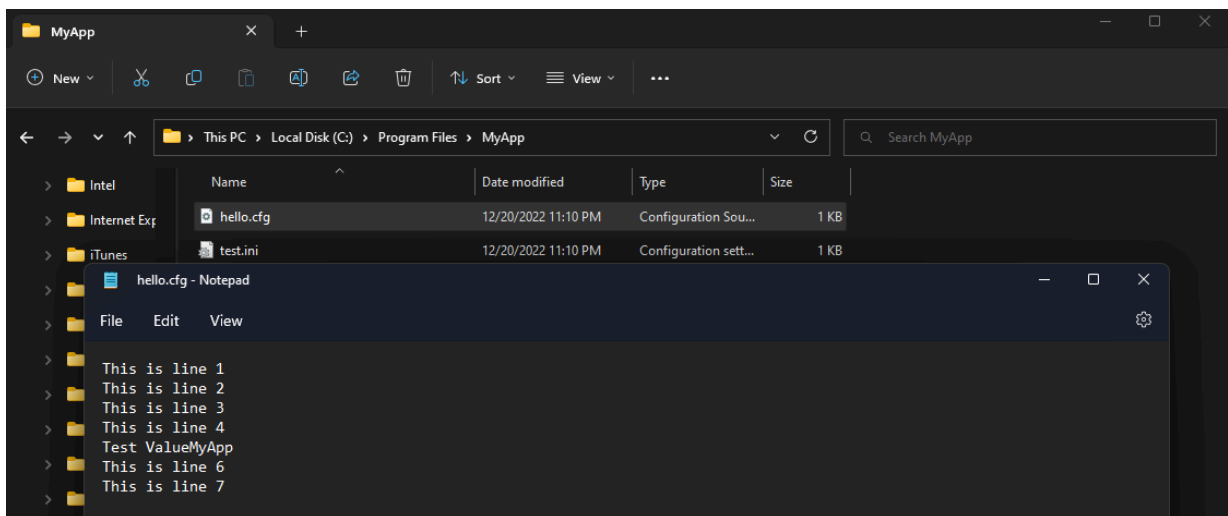
Now that we have our script ready, let's **create the Custom Action** in Advanced Installer and run it. For this, follow the next steps:

1. Navigate to the *Custom Actions Page*.
2. Search for the *Launch attached file* custom action and add it to the sequence.
3. Select the *VBScript* that we created previously.
4. In the *Action Data* field, we need to add the proper variables. Remember, the first one is the string we want to replace **C:\Program Files\** with, and the second one is the location where the **hello.cfg** file can be found. For this example, we created an additional property called MYPROPERTY which only contains a string.
5. Since the action is set to *Immediately* as the Execution Time, we will place it after the *Finish Execution* stage.
6. Build and install your package.





After the package is installed, if we check the **hello.cfg** file in the **INSTALLDIR**, we can see that it has been successfully de-hardcoded.



## De-hardcoding Files using PowerShell Custom Actions

With **PowerShell**, the code is much cleaner, shorter, and easier to understand:



```
$propValue = AI_GetMsiProperty MYPROPERTY
$propDir = AI_GetMsiProperty APPDIR
$fileDir = $propDir + "hello.cfg"

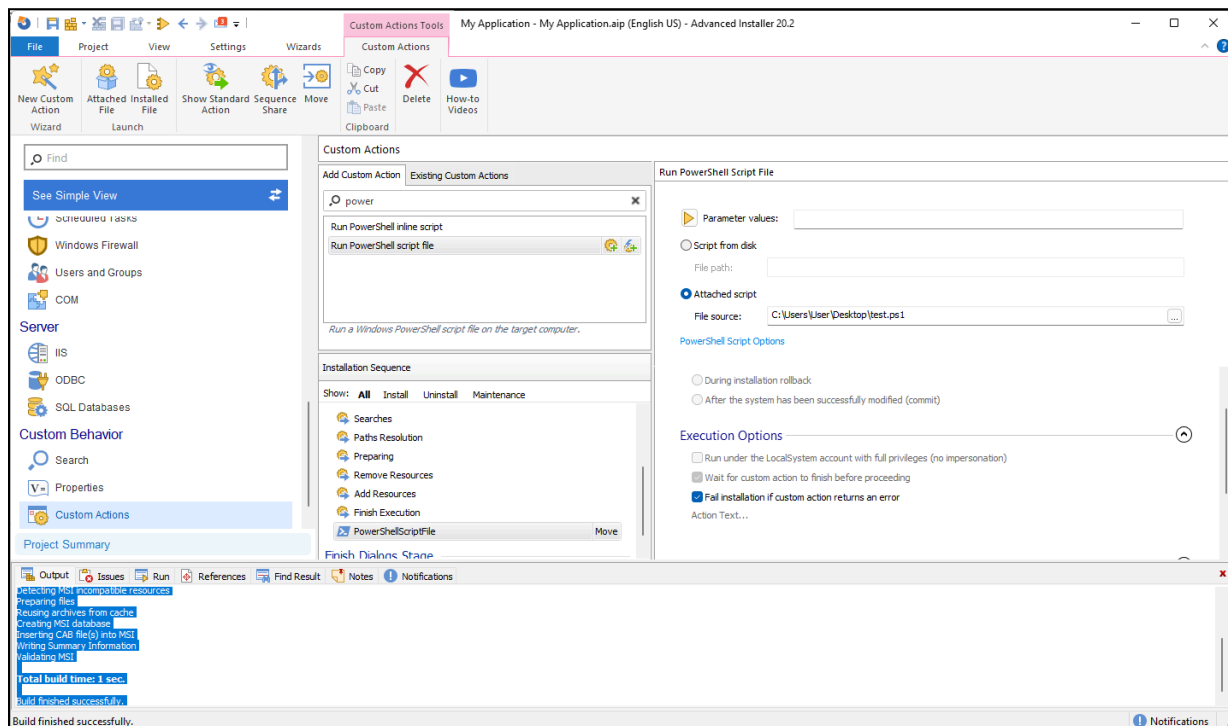
$content = [System.IO.File]::ReadAllText($fileDir).Replace("C:\Program Files\",$propValue)
[System.IO.File]::WriteAllText($fileDir, $content)
```

First, we use [AI\\_GetMsiProperty](#) to retrieve our two properties that we used for VBScript, and define the file directory.

Next, we read all the files and replace everything we find as *C:\Program Files\* in it with our property value, then write all the text back in.

To add the PowerShell script in the Advanced Installer:

1. Navigate to the *Custom Actions Page*.
2. Search for the **Run PowerShell script file** and add it to the sequence.
3. Click **Attached script** and select the file created above.
4. Place the sequence last after the **Finish Execution**.
5. Build and run the installation.



The behavior is the same as with VBScript, and the **hello.cfg** file is correctly de-hardcoded.



## Delete empty directory

Technically, during the uninstallation process, based on what directories are defined in the Directory table, the MSI usually handles the deletion of all directories and assures a clean machine. However, the rules for erasing directories are:

- The directory must be empty
- The directory must appear in the Directory table

Looking at the requirements, it might be the case that the application creates another directory that isn't present in the Directory table, and when the MSI is uninstalled, the directories will remain behind.

As discussed in the MSI Packaging Essentials book, you have the RemoveFile table. To ensure that a directory is deleted during MSI uninstallation, you need to modify the RemoveFile table in the MSI. Here's how you can achieve this:

- Open the MSI file using a compatible MSI editing tool like Orca, Wise Package Studio, or Advanced Installer.
- Locate and open the RemoveFile table within the MSI editing tool. This table specifies files and directories to be removed during uninstallation.
- Identify the directory you want to delete during uninstallation.
- Add a new row to the RemoveFile table with the following information:
  - Component\_: Enter the component identifier for the component that contains the directory.
  - FileName: Specify the name of the directory (not the full path).
  - DirProperty: Enter the directory property associated with the directory.
- Save the modified MSI file.

By adding an entry in the RemoveFile table, you instruct the Windows Installer to remove the specified directory during the uninstallation process. The Windows Installer engine will automatically remove the directory if it exists and is empty. If the directory is not empty, the uninstallation will fail unless you have custom actions or scripts in place to handle the removal of files within the directory.

### Delete empty directories with Custom Actions



## Delete with VBScript

When it comes to removing empty directories, you can go in two ways with VBScript:

- Use the native functions VBScript provides
- Use the [RD utility](#) inside the OS

When it comes to native VBScript functions, we can use the following script:

```
Dim objFSO, objFolder
Dim folderPath

folderPath = "C:\Path\to\folder"

Set objFSO = CreateObject("Scripting.FileSystemObject")

' Check if the folder exists
If objFSO.FolderExists(folderPath) Then
    Set objFolder = objFSO.GetFolder(folderPath)

    ' Check if the folder is empty
    If objFolder.Files.Count = 0 And objFolder.SubFolders.Count = 0
Then
        ' Delete the empty folder
        objFolder.Delete
        WScript.Echo "Folder deleted successfully."
    Else
        WScript.Echo "Folder is not empty."
    End If
Else
    WScript.Echo "Folder does not exist."
End If

Set objFolder = Nothing
Set objFSO = Nothing
```

Replace "C:Pathtofolder" with the actual path to the folder you want to inspect. This script checks the folder's existence, counts the files and subfolders within it, and deletes the folder if it is empty. Let's look at how the script works:

- The script begins by declaring variables: objFSO, objFolder, and folderPath.



- folderPath is set to the path of the folder you want to check for emptiness and delete if empty.
- CreateObject("Scripting.FileSystemObject") creates an instance of the FileSystemObject to work with file system objects.
- objFSO.FolderExists(folderPath) checks if the folder specified by folderPath exists.
- If the folder exists, objFolder is set to represent the folder using objFSO.GetFolder(folderPath).
- The script then checks if the folder is empty by verifying that both objFolder.Files.Count (number of files) and objFolder.SubFolders.Count (number of subfolders) are zero.
- If the folder is empty, objFolder.Delete is called to delete the folder.
- If the folder is not empty or if it doesn't exist, appropriate messages are echoed using WScript.Echo.
- Finally, the script releases the object references by setting objFolder and objFSO to Nothing.

If we want to use the RD utility together with VBScript, the following code can be used:

```
Dim objShell
Dim folderPath

folderPath = "C:\Path\to\folder"

Set objShell = CreateObject("WScript.Shell")

' Check if the folder exists
If objShell.FileSystemObject.FolderExists(folderPath) Then
    ' Execute the rd command to delete the folder
    objShell.Run "cmd /c rd /s /q "" & folderPath & """, 0, True
    WScript.Echo "Folder deleted successfully."
Else
    WScript.Echo "Folder does not exist."
End If

Set objShell = Nothing
```

Here's how the script works:

- The script begins by declaring variables: objShell and folderPath.
- folderPath is set to the path of the folder you want to delete.
- CreateObject("WScript.Shell") creates an instance of the WScript.Shell object, which allows us to execute shell commands.



- The script checks if the folder specified by `folderPath` exists using `objShell.FileSystemObject.FolderExists`.
- If the folder exists, `objShell.Run` is used to execute the `rd` command with the appropriate arguments (`/s` to delete all files and subdirectories, and `/q` to perform the deletion silently without prompts).
- The folder is enclosed in double quotes to handle any spaces or special characters in the folder path.
- The third argument of `objShell.Run` is set to `True`, which specifies that the script should wait for the command to complete before continuing.
- After the folder is deleted, a success message is echoed using `WScript.Echo`.
- If the folder does not exist, an appropriate message is echoed.
- Finally, the script releases the object reference by setting `objShell` to `Nothing`.

### Delete with PowerShell

For PowerShell, we can use the following script:

```
$folderPath = "C:\Path\to\folder"

# Check if the folder exists
if (Test-Path $folderPath) {
    # Check if the folder is empty
    if ((Get-ChildItem $folderPath | Measure-Object).Count -eq 0) {
        # Delete the folder
        Remove-Item $folderPath -Force -Recurse
        Write-Host "Folder deleted successfully."
    } else {
        Write-Host "Folder is not empty."
    }
} else {
    Write-Host "Folder does not exist."
}
```

Replace `"C:\Path\to\folder"` with the actual path of the folder you want to check. Here's how the script works:

- The script begins by setting the variable `$folderPath` to the path of the folder you want to delete.
- The script uses the `Test-Path` cmdlet to check if the folder specified by `$folderPath` exists.

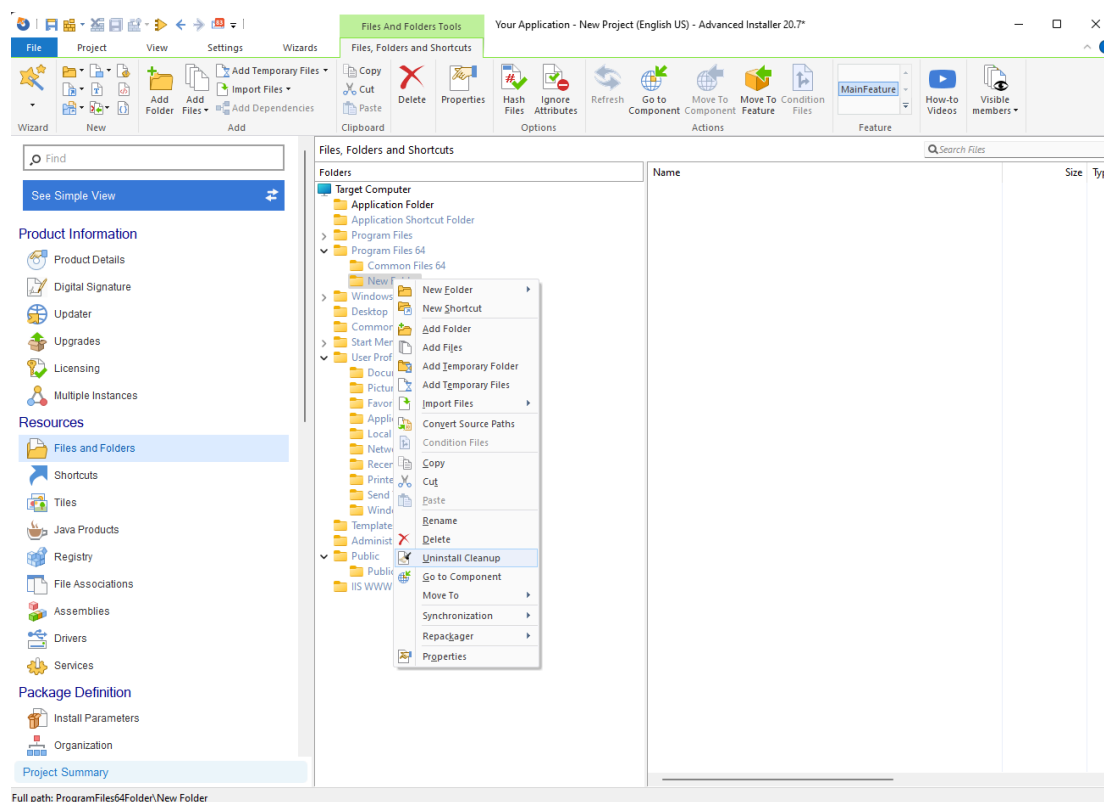




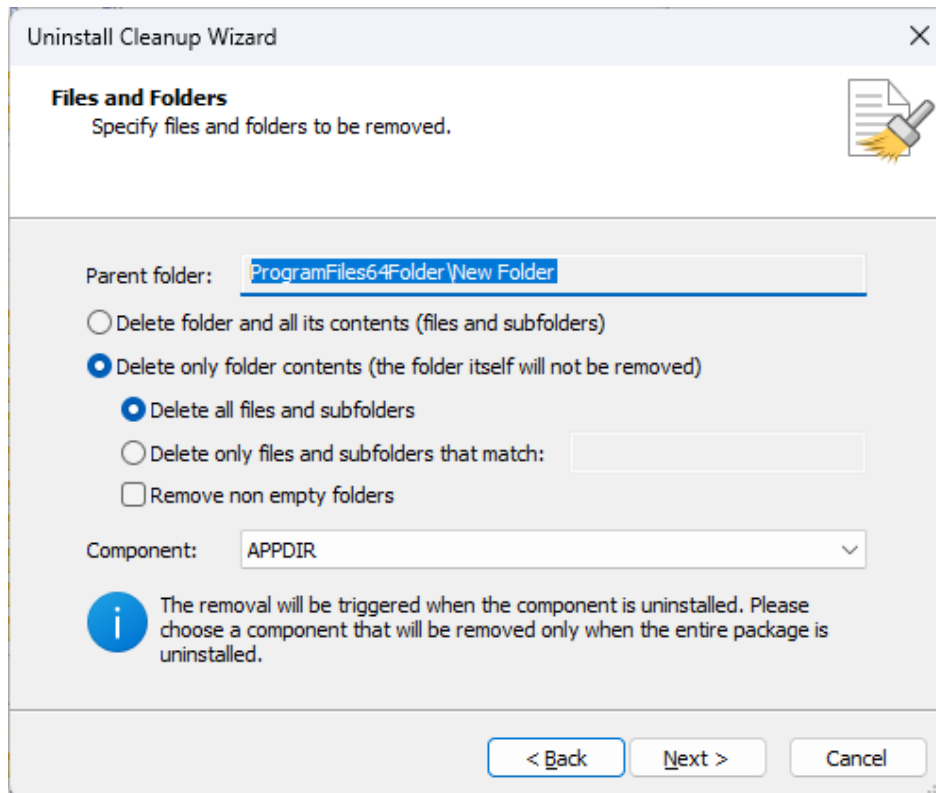
- If the folder exists, the script uses the Get-ChildItem cmdlet to get all items (files and subfolders) within the folder.
- The Measure-Object cmdlet is used to count the number of items. If the count is equal to zero, it means the folder is empty.
- If the folder is empty, the script uses the Remove-Item cmdlet to delete the folder.
- The -Force parameter is used to force the deletion without prompting for confirmation.
- The -Recurse parameter is used to delete the folder and its contents recursively.
- After deleting the folder, the script writes a success message using Write-Host.
- If the folder is not empty, the script writes a message indicating that the folder is not empty.
- If the folder does not exist, the script writes a message indicating that the folder does not exist.

## Delete empty directories with Advanced Installer

Advanced Installer offers a much simpler method to remove folders generated by the application. If you navigate to the Files and Folders page and right-click on a desired folder, you will see the Uninstall Cleanup option:

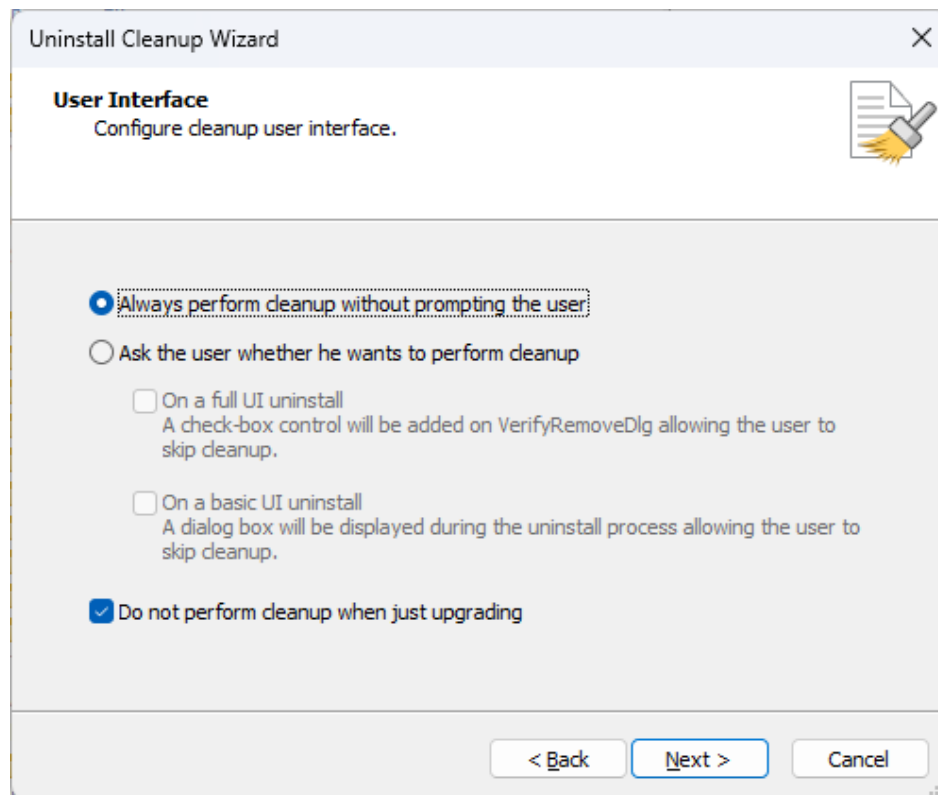


The uninstall cleanup wizard launches, allowing you to specify which files and folders created by your application will be removed when you uninstall it. To launch this wizard, select the parent folder, the directory from which you want to delete files and folders, and then choose "Uninstall Cleanup..." from the context menu. Following the wizard's launch, simply follow the on-screen instructions to select the files you want to delete.



As you can see, you can delete the desired folder and all of its subfolders, or just the contents of the folder, including non-empty directories.





Furthermore, Advanced Installer allows you to ask the user if the specified folders should be deleted when the cleanup is performed, or you can perform the cleanup silently.

## Process handling

The [custom actions](#) topic is one that most beginner IT professionals tend to avoid and it's to be understood because [MSI technology is a very complex topic](#), not to mention the [best practices](#) that were developed during the years and somehow expected for the uninitiated to implement in their installers.

So let's start an article series where we touch on nine of the most popular custom actions that are used in the industry.

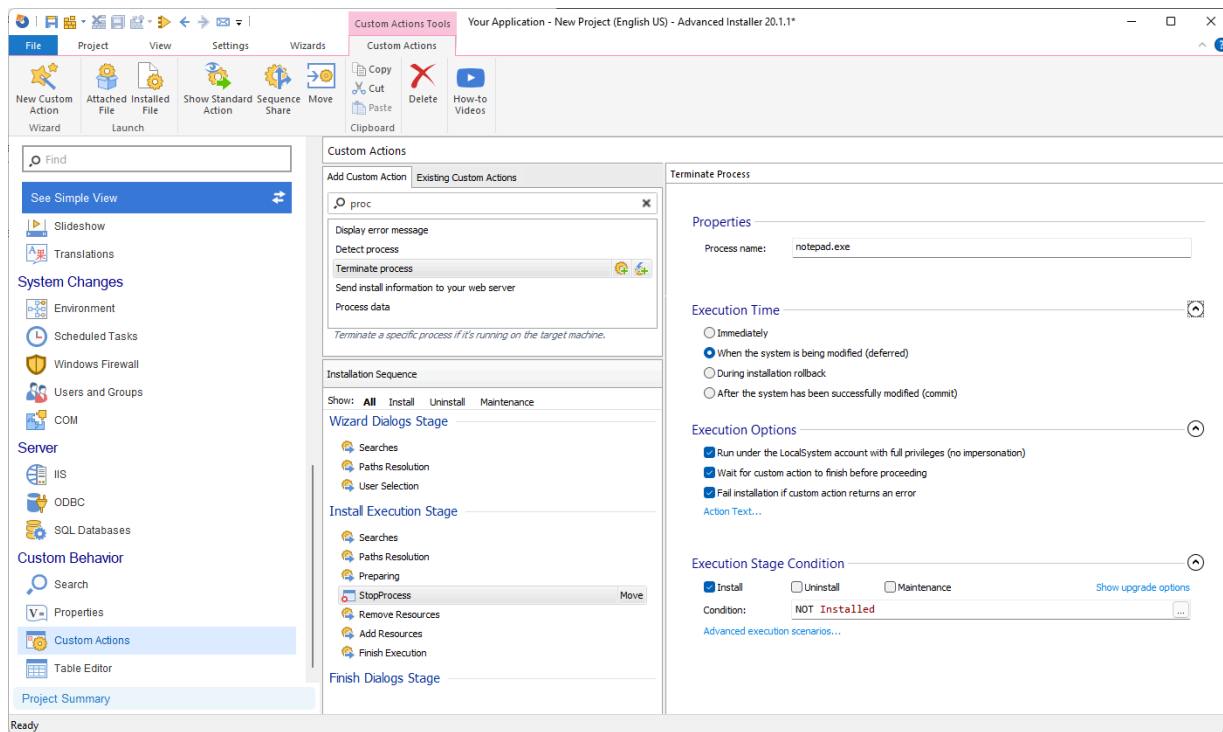
### Terminate Process in Advanced Installer

The terminate process custom action is something very often used when it comes to installers because you would like to close any running processes from your application before you start the installation or uninstallation. This ensures that no other files are in use during the installation/uninstallation operation and ensures a higher success rate for your installation.



For example, let's say I want to terminate the notepad.exe process. With Advanced Installer its quite easy:

1. Navigate to the Custom Actions Page
2. Search for "Terminate Process" custom action and add it in sequence
3. Type the process name (in our case notepad.exe)



4. Build and run the installation

The above example is configured to run only during the installation. If you want to run the same action during the uninstall sequence check the "Uninstall" checkbox and under condition modify to the following:

NOT Installed OR REMOVE~=ALL

Terminate Process with VBScript



If you want to use VBScript to close a specific process, this is quite easy to accomplish. There are two ways to do this:

1. Using the taskkill command available in cmd

```
Dim oSH
Dim returnVal
Dim shellCommand

Set oSH = CreateObject("WScript.Shell")

shellCommand = "cmd.exe /c taskkill /f /fi " & Chr(34) & "notepad.exe" & Chr(34) & " /t"
returnVal = osh.Run (shellCommand, 0, true)

Set oSH = nothing
```

This VBScript code is used to terminate all instances of the "Notepad.exe" process running on a Windows computer. Here's a breakdown of what each line does:

- Dim oSH: Declares a variable oSH to hold a reference to the Windows Script Host object.
- Dim returnVal: Declares a variable returnVal to hold the return value of the Run method.
- Dim shellCommand: Declares a variable shellCommand to store the command that will be executed in the command prompt.
- Set oSH = CreateObject("WScript.Shell"): Creates an instance of the WScript.Shell object and assigns it to the variable oSH. This object provides access to the Windows command prompt.
- shellCommand = "cmd.exe /c taskkill /f /fi " & Chr(34) & "notepad.exe" & Chr(34) & " /t": Sets the shellCommand variable to a command that will be executed in the command prompt. The command uses taskkill to forcefully terminate any process with the name "notepad.exe".
- returnVal = osh.Run(shellCommand, 0, true): Executes the shellCommand in the command prompt. The Run method launches a new process and waits for it to complete before continuing (true argument). The return value of the Run method is stored in the returnVal variable.
- Set oSH = nothing: Releases the reference to the WScript.Shell object.

To learn more parameters for the taskkill utility type taskkill.exe /? In



## 2. Using the Win32\_Process via WMI query

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:" _
    & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")

Set colProcessList = objWMIService.ExecQuery _
    ("Select * from Win32_Process Where Name = 'Notepad.exe'")

For Each objProcess in colProcessList
    objProcess.Terminate()
Next
```

This VBScript code is used to terminate all instances of the "Notepad.exe" process running on a local computer. Here's a breakdown of what each line does:

- `strComputer = "."`: Sets the `strComputer` variable to the local computer.
- `Set objWMIService = GetObject("winmgmts:{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")`: Establishes a connection to the Windows Management Instrumentation (WMI) service on the local computer. It uses the `GetObject` method to retrieve the WMI service object, specifying the impersonation level as "impersonate" to ensure the script runs with the necessary permissions.
- `Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process Where Name = 'Notepad.exe'")`: Executes a WMI query to retrieve all instances of the "Notepad.exe" process running on the local computer. The results are stored in the `colProcessList` collection.
- `For Each objProcess in colProcessList`: Loops through each process in the `colProcessList` collection.
- `objProcess.Terminate()`: Terminates the process represented by the current `objProcess` object.
- `Next`: Moves to the next process in the `colProcessList` collection and repeats the loop.

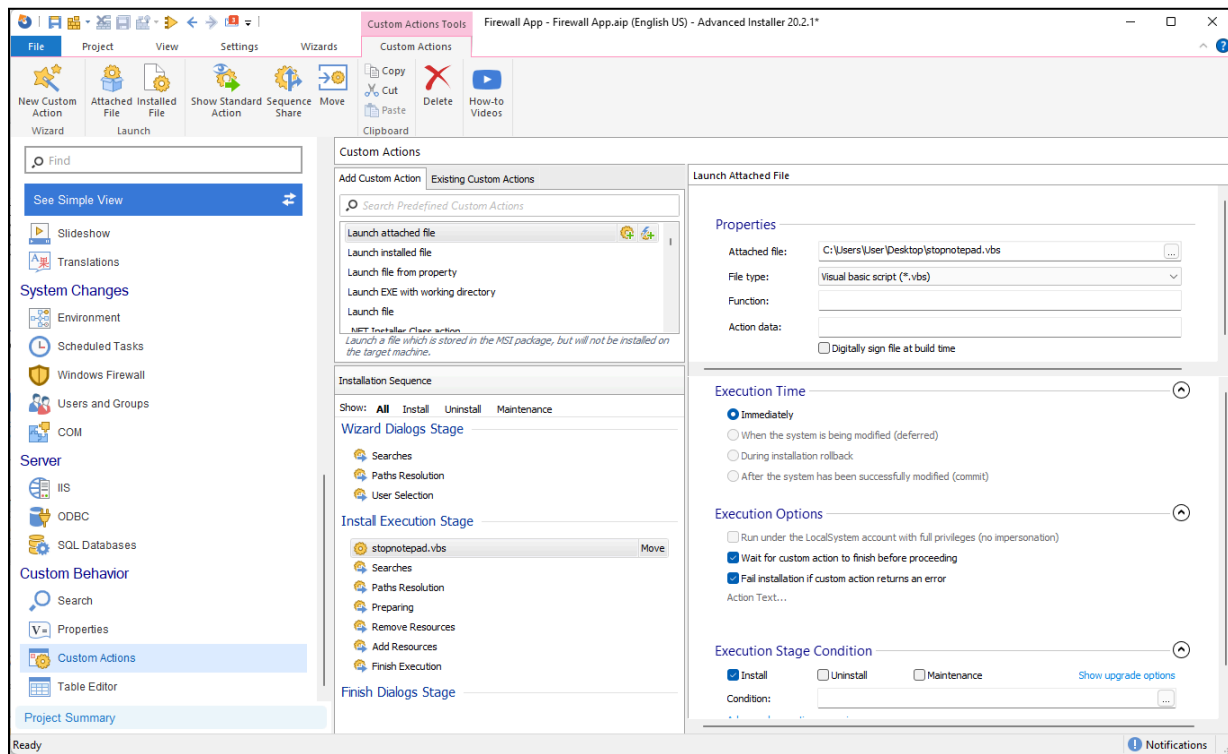
In both cases you will get the same result with `notepad.exe` being stopped, however for more complex operations the `Win32_Process` route is preferred. We will touch on the `Win32_Process` a bit later in this article when it comes to a specific type of process closure.

Once you have your VBScript ready, open Advanced Installer and perform the following steps:

### 1. Navigate to the Custom Actions Page



2. Search for “Launch Attached File” and add it to the sequence
3. A window will appear to chose the previously created VBScript



4. Build and run the installer

## Terminate Process with PowerShell

Similar to VBScript, there are two ways in which you are able to terminate a process with PowerShell.

1. Using the taskkill command

Same as VBScript, but with PowerShell it's even easier. All you need to type in a PowerShell script is:

```
taskkill /f /im notepad.exe /t
```

2. Using the [Stop-Process](#) cmdlet



`Stop-process -name notepad -Force`

Both the TASKKILL and Stop-Process allow you to kill a process forcefully with a PID or name. The difference in Stop-Process is that you can define a process object (a variable or command), but you can't define other objects such as system name, username, or password, as you would in the TASKKILL command.

However, Stop-Process helps you create an autonomous task with scripting powers. For example, the “-passthru” parameter allows you to return objects from commands, which you can later use for scripting. The Stop-Process also includes two risk mitigation parameters (-WhatIf) and (-Confirm) to avoid the Stop-Process from unwanted changes to the system.

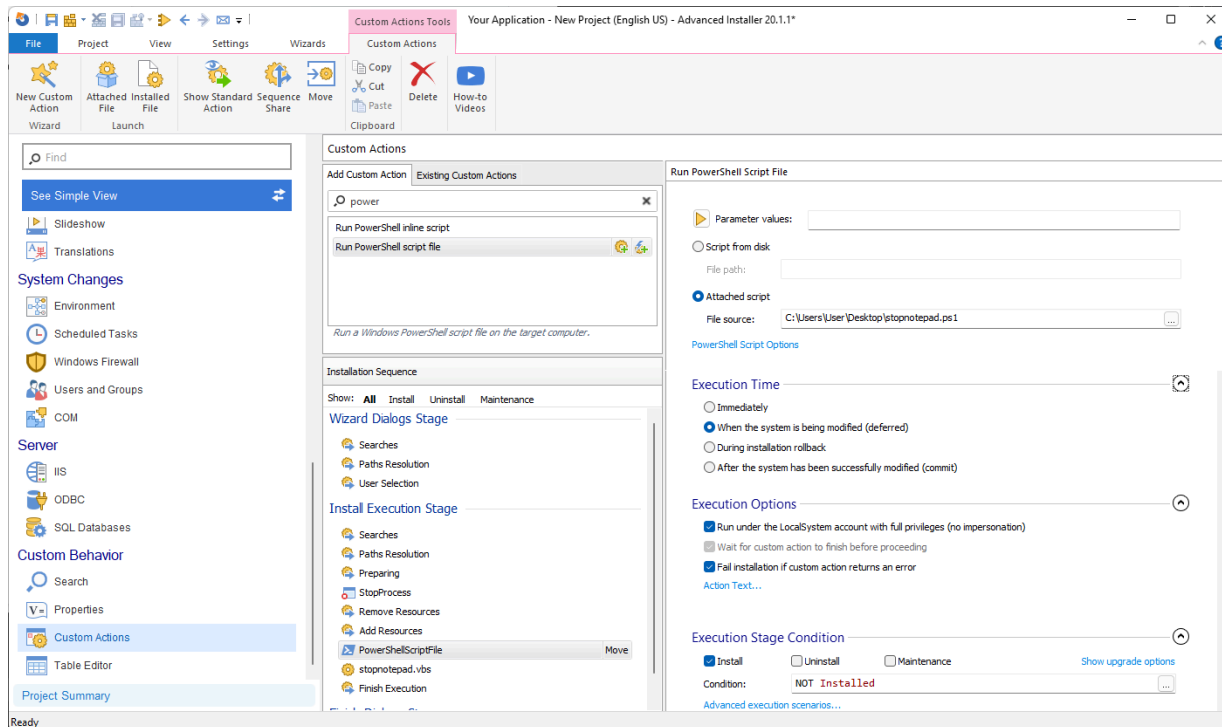
As you can see, the difference between the number of lines needed for VBScript and PowerShell is quite serious. Again, both of the above commands achieve the same result; it's up to you to decide which is best for you.

Once you have your PowerShell script ready, open Advanced Installer and perform the following steps:

1. Navigate to the Custom Actions Page
2. Search for “Run PowerShell script file” and add it to the sequence
3. Select “Attached Script”
4. A window will appear to choose the previously created PowerShell script







## 5. Build and run the Installer

### Particular Terminate Process Scenario

While the above examples cover most of the cases when it comes to process closure, there are specific scenarios that require a more complex scripting approach. One of these cases can be seen with Java applications.

If you have multiple Java applications opened and check the Task Manager, you will see that you actually have multiple Java.exe processes running.



Name	PID	Status	User name	CPU	Memory (ac...	Architec...	Description
java.exe	3848	Running	theje	00	1,136 K	x64	Java(TM) Platt...
java.exe	12284	Running	theje	00	12,444 K	x64	OpenJDK Platf...
java.exe	2320	Running	theje	00	1,132 K	x64	Java(TM) Platf...
java.exe	8488	Running	theje	00	12,452 K	x64	OpenJDK Platf...
jused.exe	16276	Running	theje	00	224 K	x86	Java Update S...
KinoniSvc.exe	4408	Running	SYSTEM	00	2,000 K	x86	KinoniSvc.exe
kinonitray.exe	3644	Running	theje	00	668 K	x86	kinonitray.exe
LockApp.exe	12608	Suspended	theje	00	0 K	x64	LockApp.exe
lsass.exe	984	Running	SYSTEM	00	5,720 K	x64	Local Security ...
mDNSResponder.exe	4308	Running	SYSTEM	00	412 K	x64	Bonjour Service
Microsoft.Photos.exe	13700	Suspended	theje	00	0 K	x64	Microsoft.Phot...
MsMpEng.exe	4912	Running	SYSTEM	00	91,440 K	x64	Antimalware S...
MsMpEngCP.exe	11872	Running	SYSTEM	00	47,580 K	x64	Antimalware S...
mspaint.exe	11856	Running	theje	00	6,332 K	x64	Paint
NisSrv.exe	8436	Running	LOCAL SER...	00	2,244 K	x64	Microsoft Net...
notepad++.exe	4116	Running	theje	00	5,292 K	x64	Notepad++ : a...
nvcontainer.exe	4580	Running	SYSTEM	00	2,564 K	x64	NVIDIA Contai...
nvcontainer.exe	8772	Running	theje	00	5,512 K	x64	NVIDIA Contai...
NVDIspay.Container....	2760	Running	SYSTEM	00	1,132 K	x64	NVIDIA Contai...
NVDIspay.Container....	8076	Running	SYSTEM	00	2,268 K	x64	NVIDIA Contai...
NVIDIA Web Helper....	7044	Running	theje	00	2,524 K	x86	NVIDIA Web H...
OfficeClickToRun.exe	4432	Running	SYSTEM	00	2,816 K	x64	Microsoft Offi...
OneDrive.exe	14696	Running	theje	00	26,580 K	x86	Microsoft One...

As you might imagine, if you are using the above techniques you will close all the Java processes which is not something we are aiming for. What we need is a way to identify each Java process for which application it is. The best way to do this is to find the command line for each Java process and find out which command line corresponds to your application. For full details on how to get the command line [check out this article](#).

For example, let's assume we have a Java application called Demo. If we search for the command line of each process we should find something like:

"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" Demo

All we have to do is modify our script to search through all processes and find the exact one which has the specific string in it, in our case "Demo".

For VBScript we can use the following:



```

On error Resume Next
Dim objWMIService, objProcess, colProcess, Linie, strComputer, strList

strComputer = "."
Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\\" &
strComputer & "\root\cimv2")
Set colProcess = objWMIService.ExecQuery _
("Select * from Win32_Process")

For Each objProcess in colProcess
if (objProcess.CommandLine <> "") Then
Linie = objProcess.CommandLine
if (InStr(Linie,"Demo")) Then
objProcess.Terminate
end if
end if
Next

Set objWMIService = Nothing
Set colProcess = Nothing

```

This VBScript code retrieves a list of running processes on a local computer and terminates any process whose command line contains the word "Demo". Here's a breakdown of what each line does:

- On Error Resume Next: Instructs the script to continue executing even if an error occurs.
- Dim objWMIService, objProcess, colProcess, Linie, strComputer, strList: Declares variables to hold references to WMI service, process objects, command line text, computer name, and process list.
- strComputer = ".": Sets the strComputer variable to represent the local computer.
- Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\\" & strComputer & "\root\cimv2"): Establishes a connection to the WMI service on the local computer.
- Set colProcess = objWMIService.ExecQuery("Select \* from Win32\_Process"): Retrieves a collection of all running processes on the local computer using the ExecQuery method.
- For Each objProcess in colProcess: Loops through each process in the collection.
- If (objProcess.CommandLine <> "") Then: Checks if the process has a non-empty command line.



- `Linie = objProcess.CommandLine`: Assigns the command line text of the process to the `Linie` variable.
- `If (InStr(Linie,"Demo")) Then`: Checks if the command line text contains the substring "Demo".
- `objProcess.Terminate`: Terminates the process if the "Demo" substring is found in the command line.
- `Next`: Moves to the next process in the collection.
- `Set objWMIService = Nothing` and `Set colProcess = Nothing`: Releases the references to the WMI service and process collection, respectively.

For PowerShell we can [use the following](#):

```
$CommandLines = Get-CimInstance Win32_Process

foreach ($command in $CommandLines)
{
    If ($command.CommandLine -like "*Demo*"){
        write-host $command.processId

        Stop-Process -id $command.processId
    }
}
```

This PowerShell script retrieves a list of running processes using the `Get-CimInstance` cmdlet from the `Win32_Process` WMI class. It then loops through each process and checks if the command line of the process contains the word "Demo". If a match is found, it writes the process ID to the console using `Write-Host` and terminates the process using the `Stop-Process` cmdlet. Here's a breakdown of what each line does:

- `$CommandLines = Get-CimInstance Win32_Process`: Retrieves a collection of running processes using the `Get-CimInstance` cmdlet and querying the `Win32_Process` WMI class.
- `foreach ($command in $CommandLines)`: Begins a loop to iterate through each process



in the \$CommandLines collection.

- If (\$command.CommandLine -like "\*Demo\*"): Checks if the command line of the process contains the substring "Demo".
- Write-Host \$Command.processId: Writes the process ID to the console. This line is used for displaying information and can be removed if not needed.
- Stop-Process -id \$Command.processId: Terminates the process using the Stop-Process cmdlet and the process ID obtained from \$Command.processId.
- }: Closes the if statement.
- }: Closes the foreach loop.

Once you have the script edited as desired, follow the above steps to add it in Advanced Installer and test the installer.

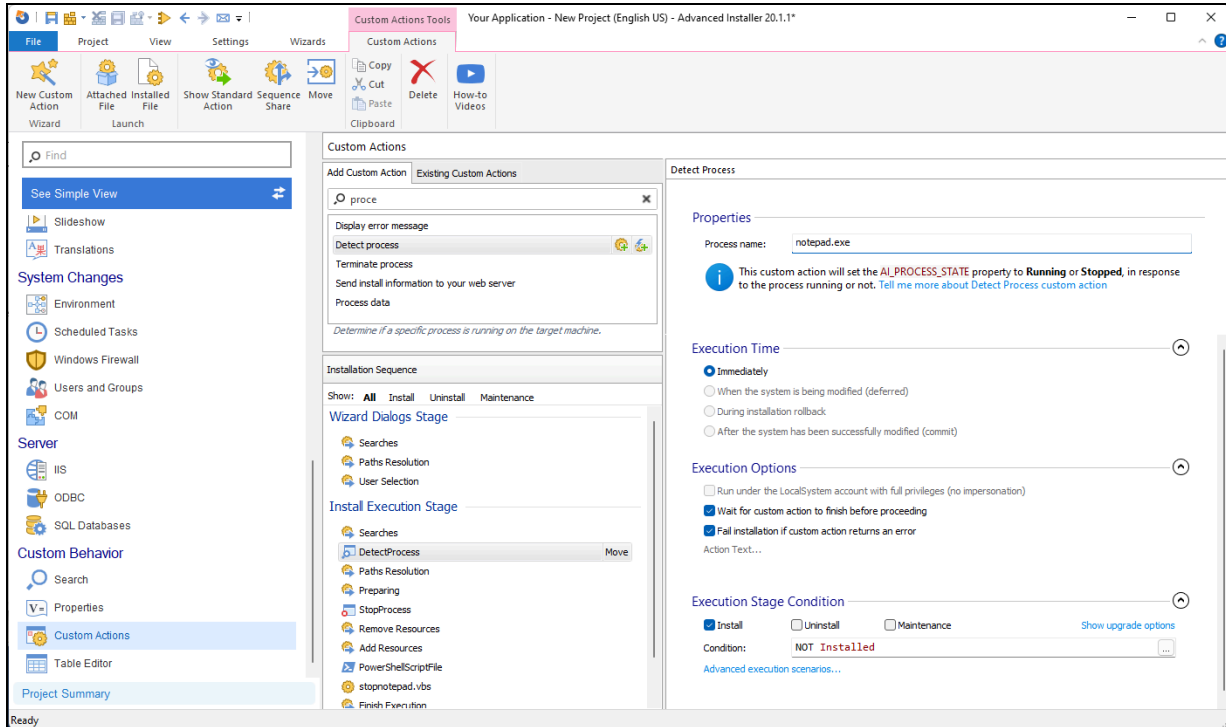
## Detect Process in Advanced Installer

There might be cases where you want to search if a different process is available on the machine before starting the installation or you want to use this knowledge in order to close other processes with the above mentioned methods.

Advanced Installer offers a quick and easy way to detect if a certain process is running.

1. Navigate to the Custom Actions Page
2. Search for "Detect Process" custom action and add it in sequence
3. Type the process name (in our case notepad.exe)





#### 4. Build and run the installation

However, this custom action only sets the `AI_PROCESS_STATE` property which you can later on use throughout your installer. For more information about it, [check out this article](#).

### Detect process with VBScript

To detect a process with VBScript we are going to use the `Win32_Process` WMI which we earlier used to terminate a process. If we want to detect if notepad is opened, we can use the following:

```
On error Resume Next
Dim strComputer
Dim objWMIService
Dim colProcessList
Dim objProcess
strComputer = "."
```



```

Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\" &
strComputer & "\root\cimv2")
Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process Where Name =
'notepad.exe'")
If colProcessList.Count > 0 Then
    MsgBox "Notepad is opened"
End If
Set objWMIService = Nothing
Set colProcessList = Nothing

```

This will open up a message box which states that “notepad is opened”. You can consider what you want to do if a certain process is found on the machine, for example returning a successful state of the execution and continuing with the installation of the application or setting up a variable in the MSI as such:

```

On error Resume Next
Dim strComputer
Dim objWMIService
Dim colProcessList
Dim objProcess
strComputer = "."

Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\" &
strComputer & "\root\cimv2")
Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process Where Name =
'notepad.exe'")
If colProcessList.Count > 0 Then
    Session.Property("ISPROCESSRUNNING") = "Yes"
End If
Set objWMIService = Nothing
Set colProcessList = Nothing

```

This VBScript checks if the process "notepad.exe" is running on the local computer and will set the ISPROCESSRUNNING MSI variable to Yes. Here's a breakdown of what each line does:

- On Error Resume Next: This statement allows the script to continue execution even if an



error occurs.

- Dim strComputer: Declares a variable named strComputer to store the name of the computer. In this case, it is set to "." which represents the local computer.
- Dim objWMIService: Declares a variable named objWMIService to hold a reference to the WMI service.
- Dim colProcessList: Declares a variable named colProcessList to hold a collection of processes.
- Dim objProcess: Declares a variable named objProcess to represent a single process object.
- strComputer = ".": Sets the value of the strComputer variable to "." to represent the local computer.
- Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\" & strComputer & "\root\cimv2"): Retrieves a reference to the WMI service using the GetObject method and the appropriate WMI namespace.
- Set colProcessList = objWMIService.ExecQuery("Select \* from Win32\_Process Where Name = 'notepad.exe'"): Executes a query against the WMI service to retrieve a collection of processes with the name "notepad.exe".
- If colProcessList.Count > 0 Then: Checks if the count of processes in the collection is greater than zero, indicating that the "notepad.exe" process is running.
- Session.Property("ISPROCESSRUNNING") = "Yes": If the "notepad.exe" process is running, it sets a property named "ISPROCESSRUNNING" to the value "Yes". This property can be accessed by an installer session to perform conditional actions based on the process status.
- Set objWMIService = Nothing and Set colProcessList = Nothing: Releases the references to the WMI service and the process collection to free up system resources.

Make sure that the ISPROCESSRUNNING property is available in the MSI before executing the script.

## Detect Process with PowerShell

With PowerShell we can achieve the same results when it comes to process detection. The following code can be used:

```
$notepad = Get-Process notepad -ErrorAction SilentlyContinue  
if ($notepad) {
```





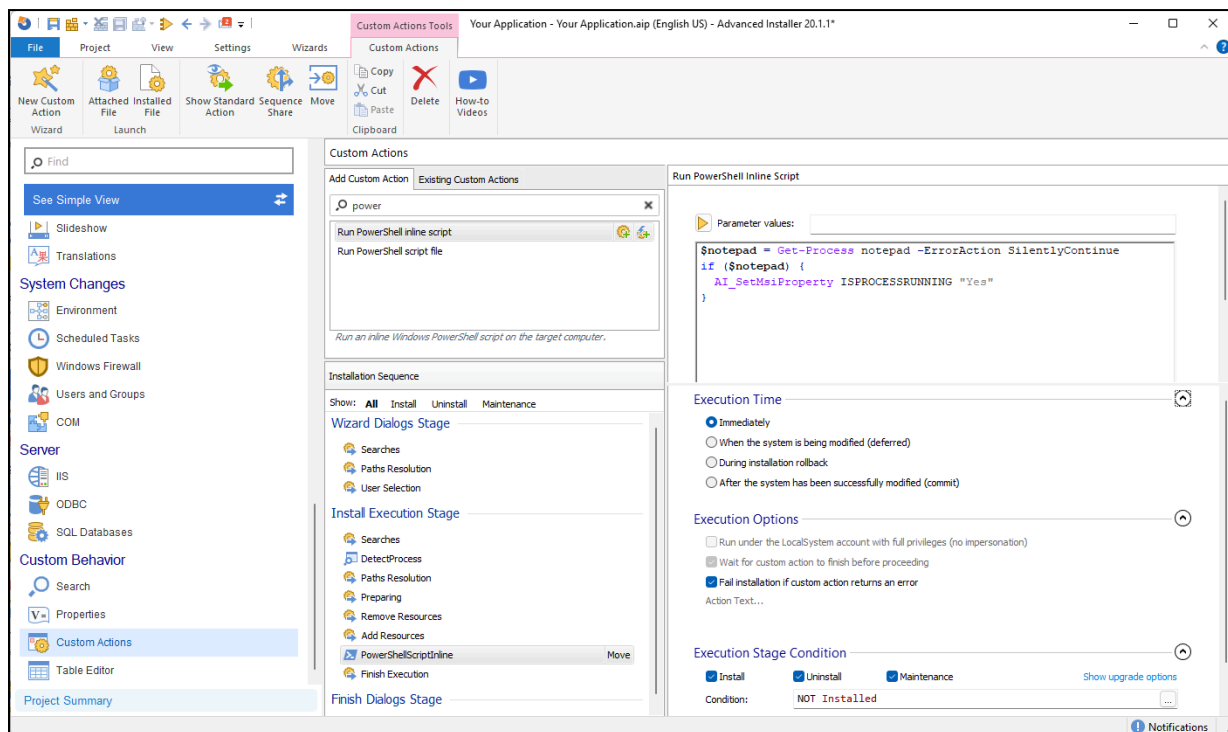
```
Write-host "notepad is running"
}
```

Of course you can also set an MSI Property with PowerShell in case a process is running to achieve the same result as Advanced Installer does. To do this, we can use the following code:

```
$notepad = Get-Process notepad -ErrorAction SilentlyContinue
if ($notepad) {
    AI_SetMsiProperty ISPROCESSRUNNING "Yes"
}
```

You can also write the PowerShell code directly into Advanced Installer by doing the following:

1. Navigate to the Custom Actions Page
2. Search for "Run PowerShell inline script" custom action and add it in sequence
3. Write the above script



#### 4. Build and run the installation



## Wait for Process with VBScript

As a last example we can think of is the case where you need to wait for a specific process to close before continuing with the installation process. To achieve this with VBScript, the following code can be used:

```
On error Resume Next
Dim strComputer
Dim objWMIService
Dim colProcessList
Dim objProcess
strComputer = "."

Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" &
strComputer & "\root\cimv2")
Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process Where Name =
'notepad.exe'")
Do While colProcessList.Count > 0
    Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process Where
Name = 'notepad.exe'")
    Wscript.Sleep(1000) 'Sleep 1 second
Loop
Set objWMIService = Nothing
Set colProcessList = Nothing
```

While notepad.exe is running, the script continues to run, thus blocking the installation to continue. Of course this can be later modified to show a certain message to the user until the process is closed. Here's an explanation of what each line does:

- On Error Resume Next: This statement allows the script to continue execution even if an error occurs.
- Dim strComputer: Declares a variable named strComputer to store the name of the computer. In this case, it is set to "." which represents the local computer.
- Dim objWMIService: Declares a variable named objWMIService to hold a reference to the WMI service.
- Dim colProcessList: Declares a variable named colProcessList to hold a collection of processes.
- Dim objProcess: Declares a variable named objProcess to represent a single process object.



- `strComputer = "."`: Sets the value of the `strComputer` variable to `"."` to represent the local computer.
- `Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")`: Retrieves a reference to the WMI service using the `GetObject` method and the appropriate WMI namespace.
- `Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process Where Name = 'notepad.exe'")`: Executes a query against the WMI service to retrieve a collection of processes with the name `"notepad.exe"`.
- `Do While colProcessList.Count > 0`: Starts a loop that continues as long as there are processes in the collection.
- `Set colProcessList = objWMIService.ExecQuery("Select * from Win32_Process Where Name = 'notepad.exe'")`: Re-executes the query to refresh the collection of processes with the name `"notepad.exe"`.
- `Wscript.Sleep(1000)`: Pauses the script for 1 second using the `Sleep` method to avoid continuous CPU usage during the loop.
- `Loop`: Returns to the start of the loop and checks the process collection count again. If there are still processes, the loop continues.
- `Set objWMIService = Nothing` and `Set colProcessList = Nothing`: Releases the references to the WMI service and the process collection to free up system resources.

## Wait for Process with PowerShell

With PowerShell it's even easier to wait for a process because PowerShell offers the [Wait-Process cmdlet](#) which can be easily used as such:

```
Wait-Process -name notepad
```

Of course you can easily add it in Advanced Installer as previously shown with the process detection.

## Firewall

To make the environment more secure it's important to properly define and configure the firewall of your machines. However, there might be times when a specific executable must be added as an exception to the Inbound or Outbound rules of the firewall in order to have access.



In this article let's have a look at how you can configure firewall rules via MSI with Advanced Installer, VBScript and Powershell.

## Firewall rules with VBScript

Although you can use the **HNetCfg.FwAuthorizedApplication** object with VBScript to define firewall rules, the easiest method is to call the [netsh.exe](#) utility that it's included in Windows. This command-line utility allows you to modify the network configuration of a certain machine that is currently running. One of the commands available for netsh is **advfirewall** which allows you to change to the netsh advfirewall context. Jumping further into the context, you can type

```
netsh advfirewall firewall
```

Into a cmd window and this will give you the following options:

```
?      - Displays a list of commands.
add      - Adds a new inbound or outbound firewall rule.
delete   - Deletes all matching firewall rules.
dump     - Displays a configuration script.
help     - Displays a list of commands.
set      - Sets new values for properties of a existing rule.
show     - Displays a specified firewall rule.
```

So basically if we want to add a firewall rule we can use:

```
netsh.exe advfirewall firewall add rule name=FRIENDLYNAME dir=IN/OUT
action=ALLOW/DENY program=PATHTOEXE enable=YES/NO profile=domain
```

If we want to remove a firewall rule we can use:

```
netsh.exe advfirewall firewall delete rule name=FRIENDLYNAME
```



Now that we are aware of how netsh is working with firewall rules, let's assume we have a HelloWorld.exe that we want to add to the inbound firewall and we want to allow everything. With VBScript we can produce the following:

```
Dim WshShell
Dim programPath2, programfiless, programfiles
Set WshShell = CreateObject("Wscript.Shell")
programfiless=WshShell.ExpandEnvironmentStrings("%ProgramFiles(x86)%")
programfiles=WshShell.ExpandEnvironmentStrings("%ProgramW6432%")

ProgramPath2 = programfiless & "\Program Files (x86)\Caphyon\Firewall
App\HelloWorld.exe"

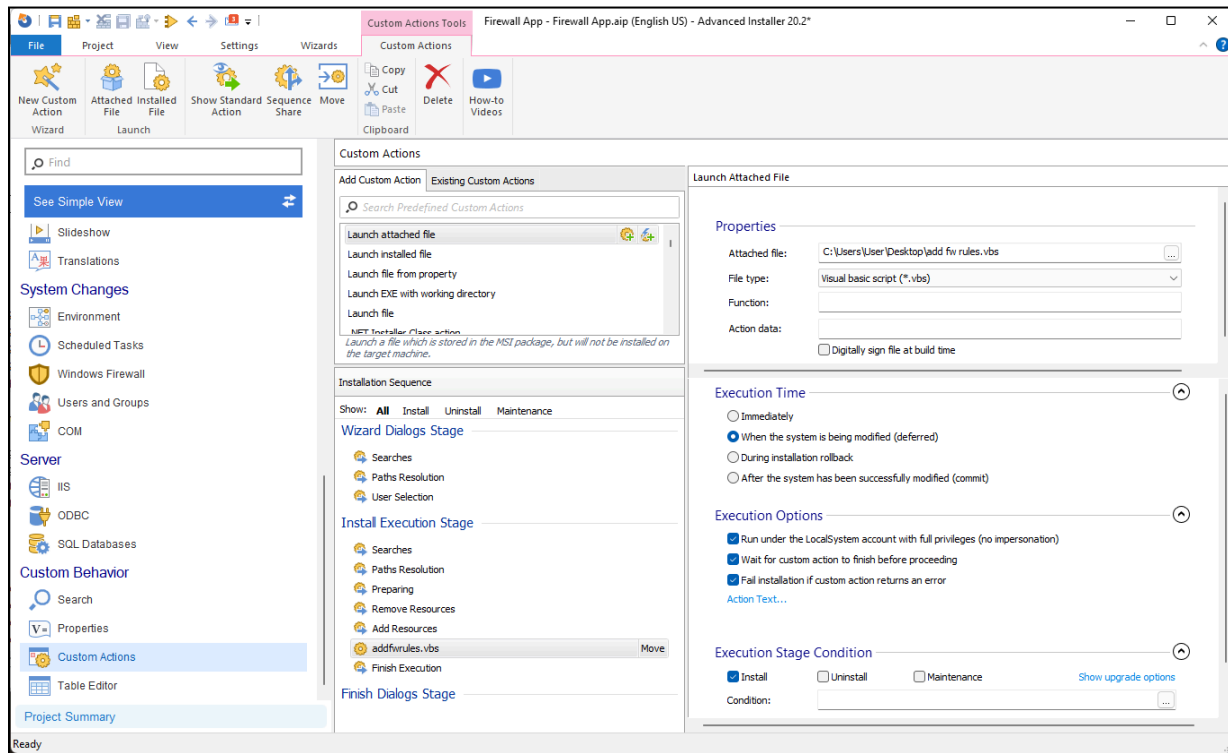
WshShell.Run "netsh.exe advfirewall firewall add rule name=HelloWorld dir=in action=allow
program=" & chr(34) & ProgramPath2 & chr(34) & " enable=yes profile=domain ", 0, False
```

This VBScript performs the following actions:

- Dim WshShell: Declares a variable named WshShell to hold a reference to the Windows Script Host Shell object.
- Dim programPath2, programfiless, programfiles: Declares variables to store the paths of program files.
- Set WshShell = CreateObject("Wscript.Shell"): Creates an instance of the Windows Script Host Shell object.
- programfiless = WshShell.ExpandEnvironmentStrings("%ProgramFiles(x86)%"): Retrieves the path of the "Program Files (x86)" folder using the %ProgramFiles(x86)% environment variable.
- programfiles = WshShell.ExpandEnvironmentStrings("%ProgramW6432%"): Retrieves the path of the "Program Files" folder using the %ProgramW6432% environment variable.
- ProgramPath2 = programfiless & "\Program Files (x86)\Caphyon\Firewall App\HelloWorld.exe": Concatenates the program file path with the specific file name to create the full path of the executable file "HelloWorld.exe".
- WshShell.Run "netsh.exe advfirewall firewall add rule name=HelloWorld dir=in action=allow program=" & chr(34) & ProgramPath2 & chr(34) & " enable=yes profile=domain ", 0, False: Runs the netsh.exe command to add a firewall rule named "HelloWorld" with the specified properties. The command allows incoming traffic (dir=in), allows the specified program (program=) with the path of "HelloWorld.exe", enables the rule (enable=yes), and applies the rule to the domain profile.



Next, open Advanced Installer and navigate to the Custom Actions Page. In here, search for the **Launch attached file** and select the location of the VBScript. Next, configure the custom action to execute as shown below:



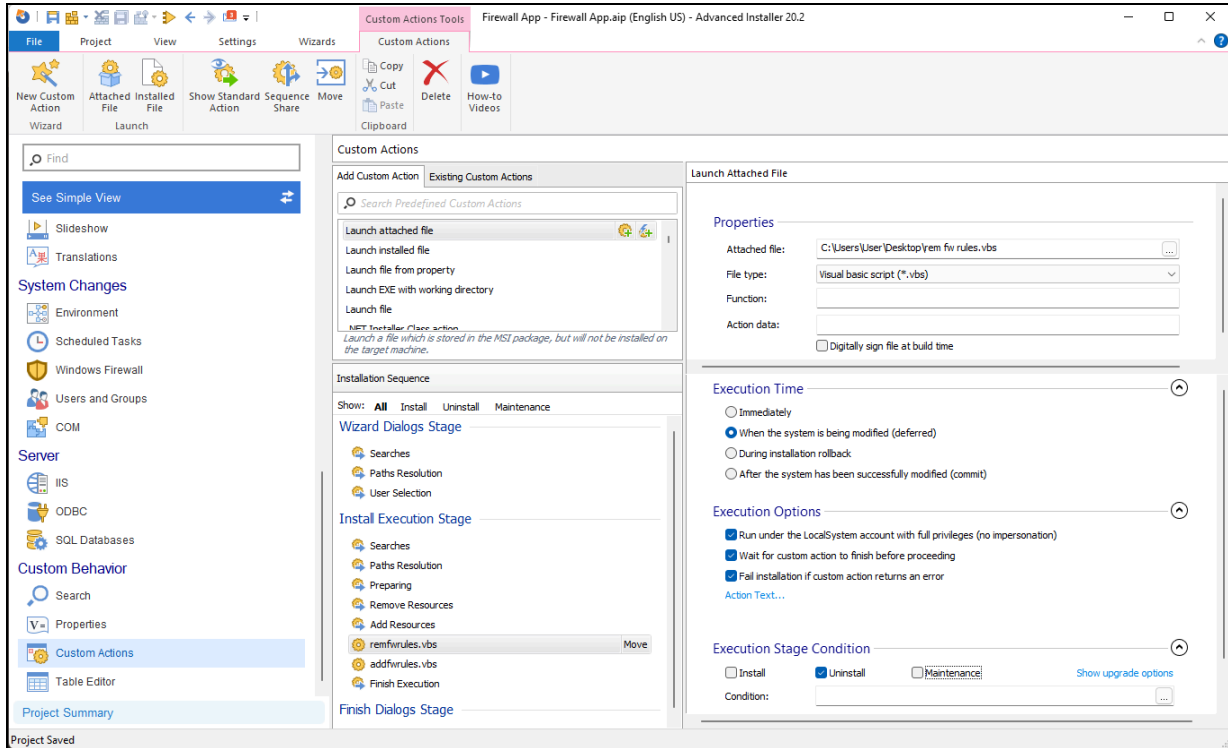
As a best practice it's also important to remove the firewall rule during the uninstallation. For that, it means we need another Custom Action and a different VBScript to remove our rule. The VBScript code is:

```
Dim WshShell
Set WshShell = CreateObject("Wscript.Shell")

WshShell.Run "netsh.exe advfirewall firewall delete rule name=HelloWorld"
```

After that, follow the same exact steps as above and configure the custom action as following:





## Firewall rules with PowerShell

While **netsh** is still available and widely used by the community, starting with Windows 8.1 you can use the built-in **NetSecurity** PowerShell module to manage firewall operations.

In general, there are 85 commands available in this module that you can use in Windows 10/11, but we are only interested in two of them. To add a firewall rule you can simply do:

```
$HelloWorldLocation = ${env:ProgramFiles(x86)} + "\Caphyon\Firewall App\HelloWorld.exe"
```

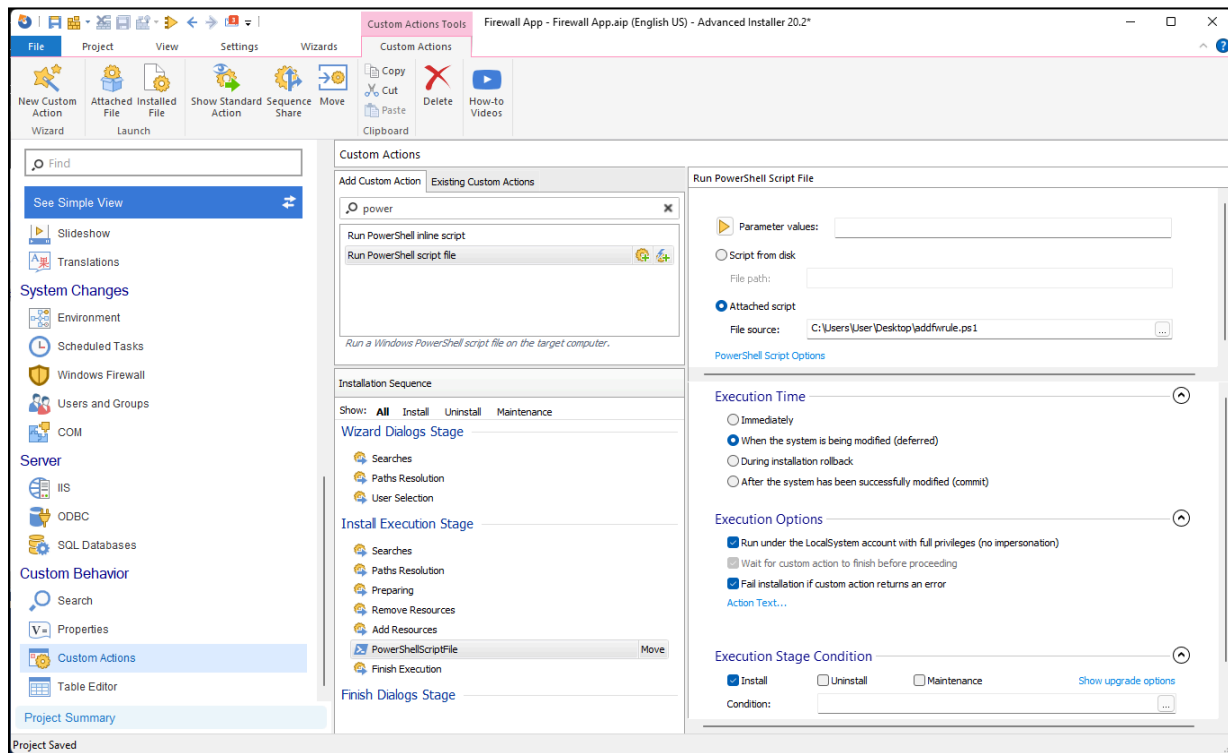
```
New-NetFirewallRule -Program $HelloWorldLocation -Action Allow -Profile Domain  
-DisplayName "HelloWorld" -Description "Block Firefox browser" -Direction Inbound
```

To remove a firewall rule is even simpler as we only use the [Remove-NetFirewallRule](#) PowerShell cmdlet:

```
Remove-NetFirewallRule -DisplayName "HelloWorld"
```



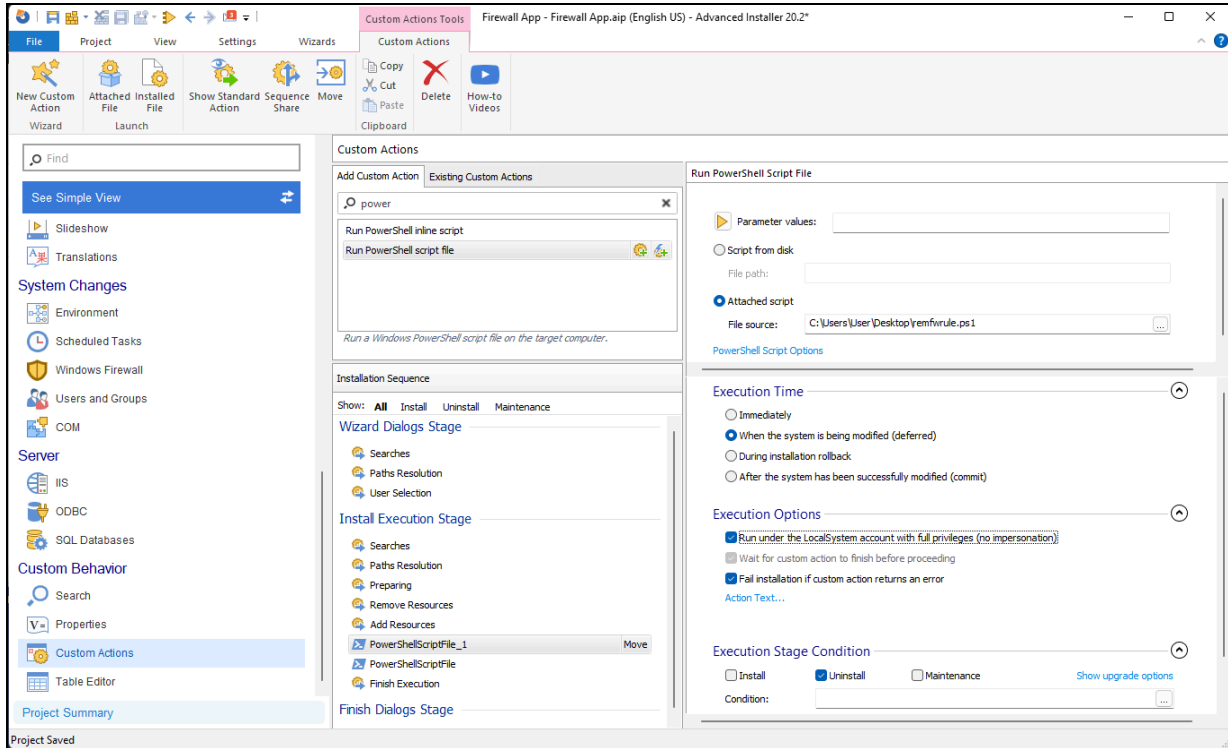
Next, open Advanced Installer and navigate to the Custom Actions Page. In here, search for the **Run PowerShell script file** and select the location of the PowerShell script. Next, configure the custom action to execute as shown below:



To also add the remove firewall PowerShell script, follow the same steps as above and do the following configurations:





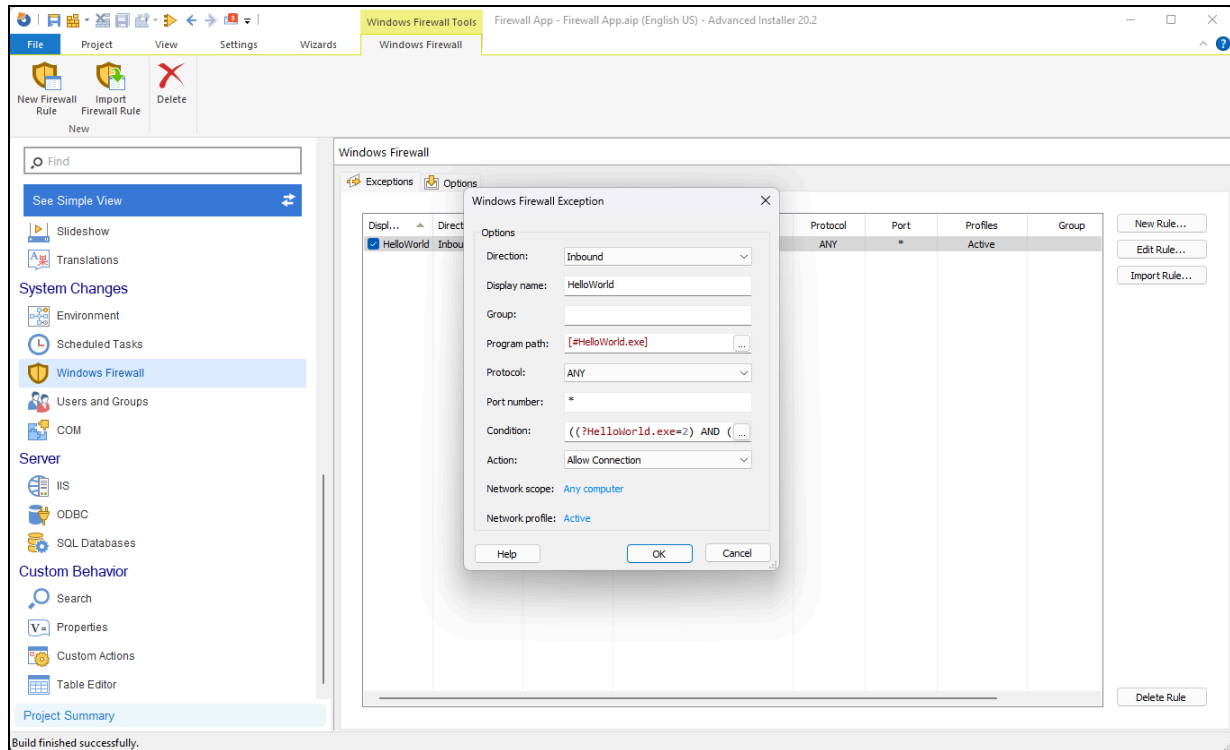


## Firewall rules with Advanced Installer

If you don't like to code, Advanced Installer made it much simpler to add firewall rules. First, navigate to the [Windows Firewall page](#).

Next, click on **New Rule**. This will open a new window in which you can define the necessary details for your exception:

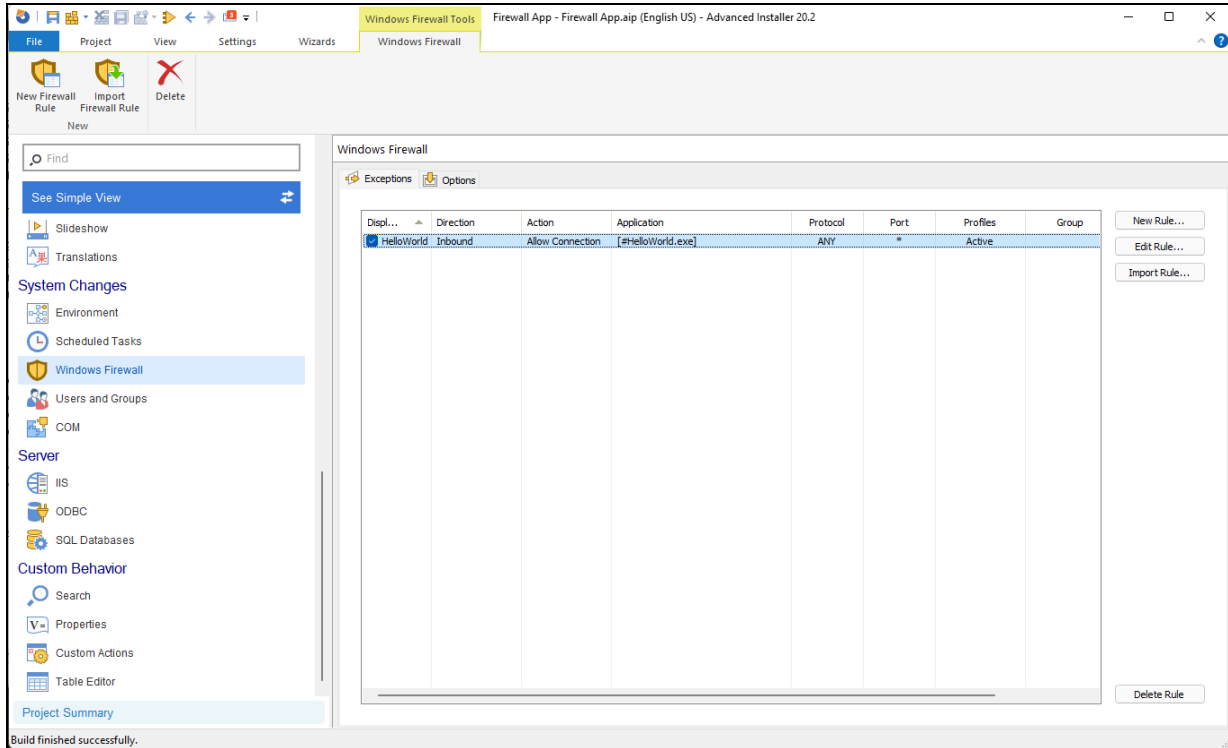




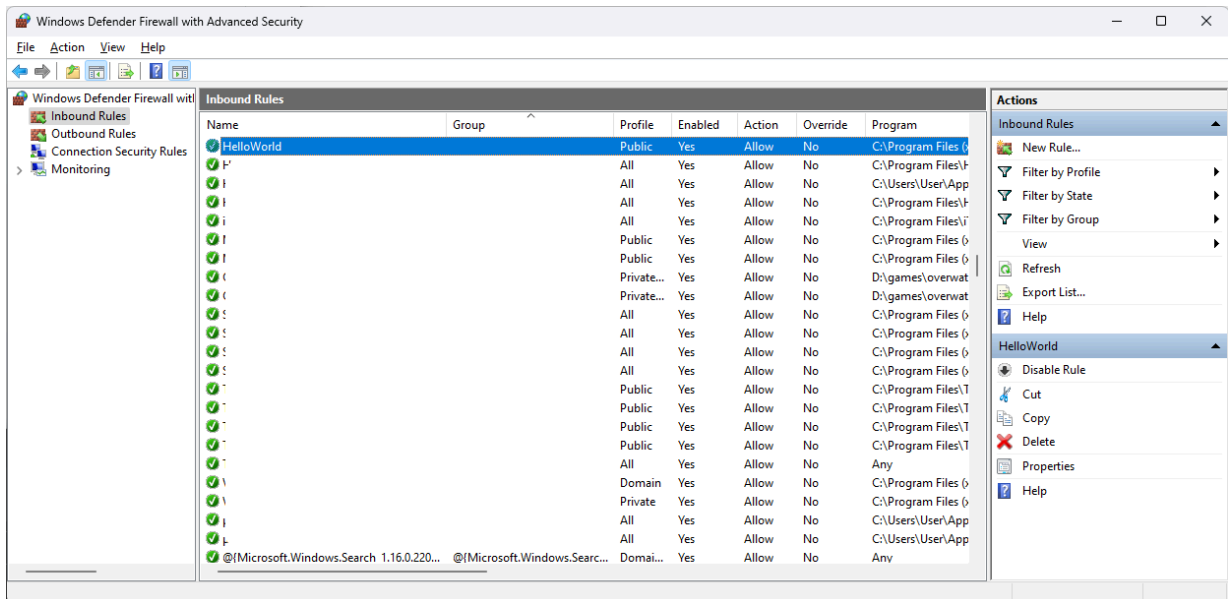
As you can see, you can easily choose the direction, display name, program path, protocol and other settings directly from the GUI. In our case we wanted to mimic the above usages of netsh and PowerShell and left everything as before in the GUI.

And that is it, Advanced Installer will automatically create the exception during the installation and during the uninstallation it will remove the exception from the firewall, not needing to create two separate actions for it.





All you have to do is build and install the MSI package. After the installation, if we check the Inbound rules, our rule is there:



## Install/uninstall driver

Working with drivers is not something to be worried about. Although the MSI technology does not offer a native way via the database tables that you can use, the OS is offering multiple choices for you to achieve this goal.

Let's have a look at how you can install drivers with MSI by taking different approaches.

### DPIInst

Driver Package Installer (DPIInst) is a component of [Driver Install Frameworks \(DIFx\)](#) version 2.1. DIFx simplifies and customizes the installation of driver packages for devices that you wish to install on the computer. This type of installation is commonly known as a software-first installation. DPIInst also automatically updates the drivers for any installed devices that are supported by the newly installed driver packages.

DPIInst searches for INF files for driver packages in the DPIInst working directory which by default is the DPIInst root directory, which is the directory that contains the DPIInst executable (DPIInst.exe).

You can also use the /path command-line switch to specify a custom DPIInst working directory.

The log files for the DPIInst utility can be found in the %SystemRoot%\DPIInst.log. However, DPIInst does not come natively with the OS and must be added into the MSI package, preferably near the driver .inf files.

To install a driver with DPIInst, the following command can be used:

```
DPIInst_x64.exe /F /LM /S
```

For the full list of commands which DPIInst supports, you can use the following:

```
DPIInst_x64.exe /?
```

### PnPUtil



Unlike DPInst, [PnPUtil](#) tool comes natively into the OS and lets an administrator perform actions on driver packages. lets an administrator perform actions on driver packages. With it you can add a driver package to the store, install a driver package on the computer and delete a driver package.

To install a driver with PnPUtil, the following command can be used:

```
PNPUtil.exe /add-driver PATH\DRIVERNAME.inf /install
```

For the full list of commands which PnPUtil supports, you can use the following:

```
PNPUtil.exe /?
```

Now that we know two methods which we can use to install the drivers, let's see how we can use them in VBScript or PowerShell.

### Installing drivers with VBScript

Let's take a look at both utility tools and create the necessary scripts to install a driver. Let's assume that the driver **.inf** name is HP.inf. Also, let's assume that we are going to place the DPInst.exe utility directly into the C:\Windows\DPInst folder and there we are going to place the .inf file as well.

The script to install the driver is:

```
Option Explicit
On Error Resume Next

Dim strCmd,WshShell,strInstalldir

Set WshShell = CreateObject("WScript.Shell")
strInstalldir = WshShell.ExpandEnvironmentStrings( "%SYSTEMROOT%" )

strCmd = chr(34) & strInstalldir & "\DPInst\DPInst_x64.exe" & chr(34) & " /F /LM /S"
```



```
WshShell.Run strCmd  
  
Set WshShell = Nothing
```

This VBScript performs the following actions:

- Option Explicit: Enables explicit variable declaration, ensuring that all variables are declared before use.
- On Error Resume Next: Instructs the script to continue execution even if an error occurs.
- Dim strCmd, WshShell, strInstallDir: Declares variables to hold the command, Windows Script Host Shell object, and the installation directory.
- Set WshShell = CreateObject("WScript.Shell"): Creates an instance of the Windows Script Host Shell object.
- strInstallDir = WshShell.ExpandEnvironmentStrings("%SYSTEMROOT%"): Retrieves the path of the Windows installation directory using the %SYSTEMROOT% environment variable.
- strCmd = chr(34) & strInstallDir & "\DPInst\DPInst\_x64.exe" & chr(34) & " /F /LM /S": Constructs the command to be executed. It combines the installation directory path with the relative path to the "DPInst\_x64.exe" executable. The /F, /LM, and /S are command-line switches or parameters for the executable.
- WshShell.Run strCmd: Runs the command stored in strCmd using the Run method of the Windows Script Host Shell object. This executes the DPInst\_x64.exe installer with the specified command-line switches.
- Set WshShell = Nothing: Releases the reference to the Windows Script Host Shell object.

In summary, the script runs the DPInst\_x64.exe installer, located in the DPInst subfolder under the Windows installation directory, with the command-line switches /F, /LM, and /S. The purpose and functionality of the DPInst\_x64.exe installer may depend on the specific software or device driver being installed.

The script to uninstall the driver is:

```
Option Explicit  
On Error Resume Next  
  
Dim strCmd, WshShell, strInstallDir, strCmd1, strCmd2  
Set WshShell = CreateObject("WScript.Shell")
```



```
strInstalldir = WshShell.ExpandEnvironmentStrings( "%SYSTEROOT%" )

strcmd= chr(34) & strInstalldir & "\DPInst\DPInst_x64.exe" & chr(34) & " /S /U " & chr(34) &
strInstalldir & "\DPInst\HP.inf" & chr(34) & " /D"

WshShell.Run strCmd

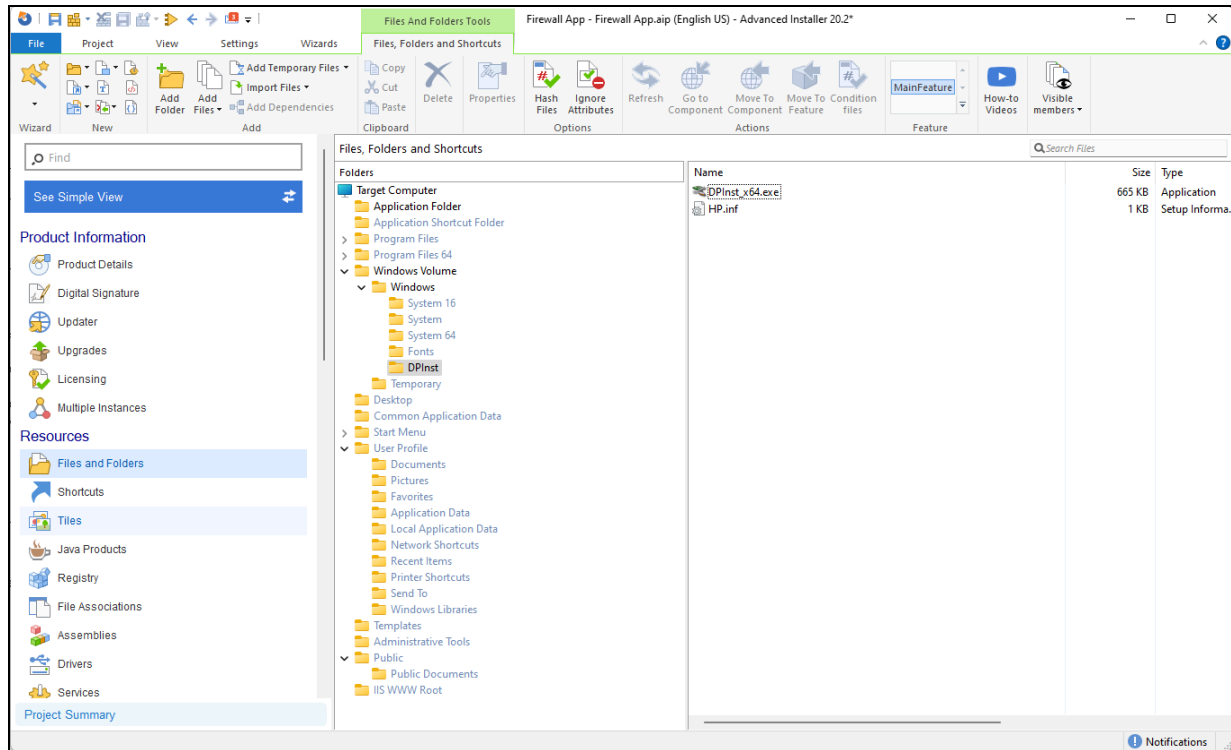
Set WshShell = Nothing
```

This VBScript performs the following actions:

- Option Explicit: Enables explicit variable declaration, ensuring that all variables are declared before use.
- On Error Resume Next: Instructs the script to continue execution even if an error occurs.
- Dim strCmd, WshShell, strInstalldir, strCmd1, strCmd2: Declares variables to hold the commands, Windows Script Host Shell object, and the installation directory.
- Set WshShell = CreateObject("WScript.Shell"): Creates an instance of the Windows Script Host Shell object.
- strInstalldir = WshShell.ExpandEnvironmentStrings("%SYSTEROOT%"): Retrieves the path of the Windows installation directory using the %SYSTEROOT% environment variable.
- strCmd = chr(34) & strInstalldir & "\DPInst\DPInst\_x64.exe" & chr(34) & " /S /U " & chr(34) & strInstalldir & "\DPInst\HP.inf" & chr(34) & " /D": Constructs the command to be executed. It combines the installation directory path with the relative paths to the "DPInst\_x64.exe" executable and "HP.inf" file. The /S, /U, and /D are command-line switches or parameters for the executable.
- WshShell.Run strCmd: Runs the command stored in strCmd using the Run method of the Windows Script Host Shell object. This executes the DPInst\_x64.exe installer with the specified command-line switches and the "HP.inf" file for driver uninstallation.
- Set WshShell = Nothing: Releases the reference to the Windows Script Host Shell object.

Once we have the scripts done and the DPInst utility downloaded, open Advanced Installer and first navigate to the Files and Folders Page. In here, create a new directory under **Windows Volume\Windows** called **DPInst** and add the DPInst utility with the HP.inf file near it.

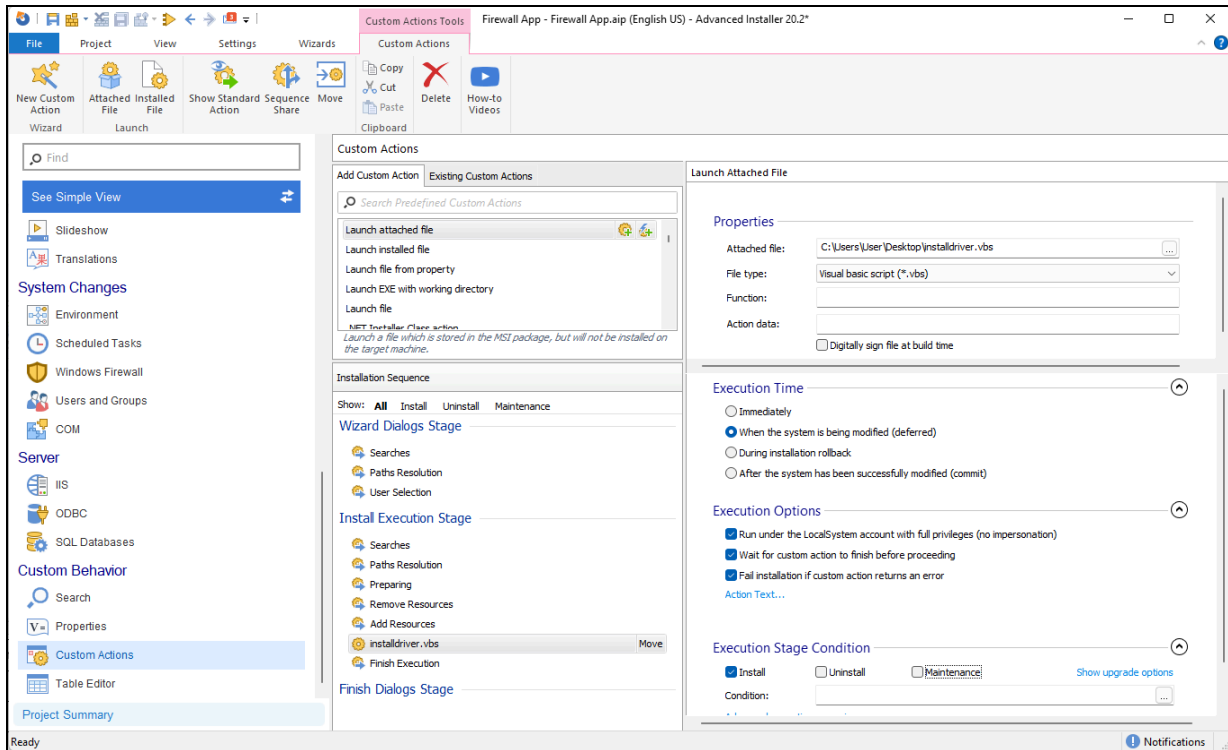




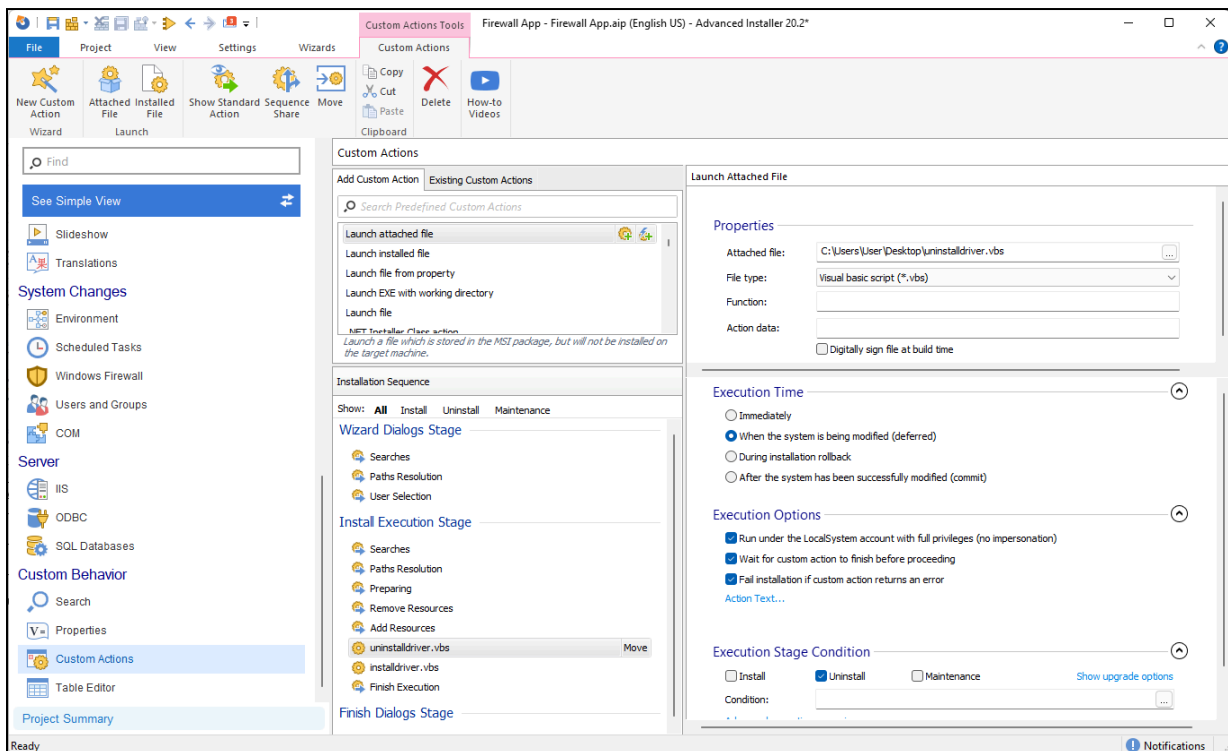
Next, navigate to the Custom Actions Page and add the **Launch attached file** predefined custom action into the sequence, select the installation vbscript file that was previously created and configure the Custom Action as such:







Repeat the same steps for the uninstall script and configure the custom action as follows:



And that is it, build your package and install it and the driver will appear as installed.

If we don't want to use the DPInst method and want to go with the PnPutil one, the VBScript for installation should look like this:

```
Option Explicit
On Error Resume Next

Dim strCmd,WshShell,strInstalldir,strCmd1, strCmd2
Set WshShell = CreateObject("WScript.Shell")
strInstalldir = WshShell.ExpandEnvironmentStrings( "%SYSTEMROOT%" )

strCmd= "pnputil.exe /add-driver " & chr(34) & strInstalldir & "\DPInst\HP.inf" & chr(34)

WshShell.Run strCmd

Set WshShell = Nothing
```

This VBScript performs the following actions:

- Option Explicit: Enables explicit variable declaration, ensuring that all variables are declared before use.
- On Error Resume Next: Instructs the script to continue execution even if an error occurs.
- Dim strCmd, WshShell, strInstalldir, strCmd1, strCmd2: Declares variables to hold the commands, Windows Script Host Shell object, and the installation directory.
- Set WshShell = CreateObject("WScript.Shell"): Creates an instance of the Windows Script Host Shell object.
- strInstalldir = WshShell.ExpandEnvironmentStrings("%SYSTEMROOT%"): Retrieves the path of the Windows installation directory using the %SYSTEMROOT% environment variable.
- strCmd = "pnputil.exe /add-driver " & chr(34) & strInstalldir & "\DPInst\HP.inf" & chr(34): Constructs the command to be executed. It combines the pnputil.exe utility command /add-driver with the path to the "HP.inf" file for driver installation. The chr(34) is used to enclose the path in double quotes.
- WshShell.Run strCmd: Runs the command stored in strCmd using the Run method of the Windows Script Host Shell object. This executes the pnputil.exe utility with the /add-driver command and the specified driver inf file for installation.
- Set WshShell = Nothing: Releases the reference to the Windows Script Host Shell object.



The script to remove the driver with PnPUtl is:

```
Option Explicit
On Error Resume Next

Dim strCmd,WshShell,strInstalldir,strcmd1, strcmd2
Set WshShell = CreateObject("WScript.Shell")
strInstalldir = WshShell.ExpandEnvironmentStrings( "%SYSTEMROOT%" )

strcmd= "pnputil.exe /delete-driver " & chr(34) & strInstalldir & "\DPInst\HP.inf" & chr(34)

WshShell.Run strCmd

Set WshShell = Nothing
```

This VBScript performs the following actions:

- Option Explicit: Enables explicit variable declaration, ensuring that all variables are declared before use.
- On Error Resume Next: Instructs the script to continue execution even if an error occurs.
- Dim strCmd, WshShell, strInstalldir, strCmd1, strCmd2: Declares variables to hold the commands, Windows Script Host Shell object, and the installation directory.
- Set WshShell = CreateObject("WScript.Shell"): Creates an instance of the Windows Script Host Shell object.
- strInstalldir = WshShell.ExpandEnvironmentStrings("%SYSTEMROOT%"): Retrieves the path of the Windows installation directory using the %SYSTEMROOT% environment variable.
- strCmd = "pnputil.exe /delete-driver " & chr(34) & strInstalldir & "\DPInst\HP.inf" & chr(34): Constructs the command to be executed. It combines the pnputil.exe utility command /delete-driver with the path to the "HP.inf" file for driver deletion. The chr(34) is used to enclose the path in double quotes.
- WshShell.Run strCmd: Runs the command stored in strCmd using the Run method of the Windows Script Host Shell object. This executes the pnputil.exe utility with the /delete-driver command and the specified driver inf file for deletion.
- Set WshShell = Nothing: Releases the reference to the Windows Script Host Shell object.

Do the same steps as we did before to add the VBScript files in the custom actions, build the installer and the driver should get installed.



## Installing drivers with PowerShell

Let's assume that the driver **.inf** name is HP.inf. Also, let's assume that we are going to place the DPInst.exe utility directly into the C:\Windows\DPInst folder and there we are going to place the .inf file as well.

The script to install the file as we did with VBScript is:

```
$DPInstLoc = $env:SystemRoot + "\DPInst\DPInst_x64.exe"

$cmd = "$DPInstLoc /F /LM /S"
Invoke-Expression $cmd
```

The script performs the following actions:

- `$DPInstLoc = $env:SystemRoot + "\DPInst\DPInst_x64.exe"`: Sets the variable `$DPInstLoc` to the path of the `DPInst_x64.exe` file located in the `%SystemRoot%\DPInst` directory. The `$env:SystemRoot` environment variable represents the path to the Windows installation directory.
- `$cmd = "$DPInstLoc /F /LM /S"`: Constructs a command string that includes the value of `$DPInstLoc` and additional command-line arguments. In this case, the command is `DPInst_x64.exe /F /LM /S`. The `/F` switch specifies that existing driver packages should be deleted, `/LM` specifies that the driver should be installed for all users on the local machine, and `/S` enables silent installation without displaying any user interface.
- `Invoke-Expression $cmd`: Executes the command stored in the `$cmd` variable using the `Invoke-Expression` cmdlet. This cmdlet interprets and runs the command as if it were typed directly into the PowerShell console.

The script to uninstall the driver is:

```
$DPInstLoc = $env:SystemRoot + "\DPInst\DPInst_x64.exe"
$INFLocation = $env:SystemRoot + "\DPInst\HP.inf"
$cmd = "$DPInstLoc /S /U $INFLocation"
Invoke-Expression $cmd
```

The script performs the following actions:

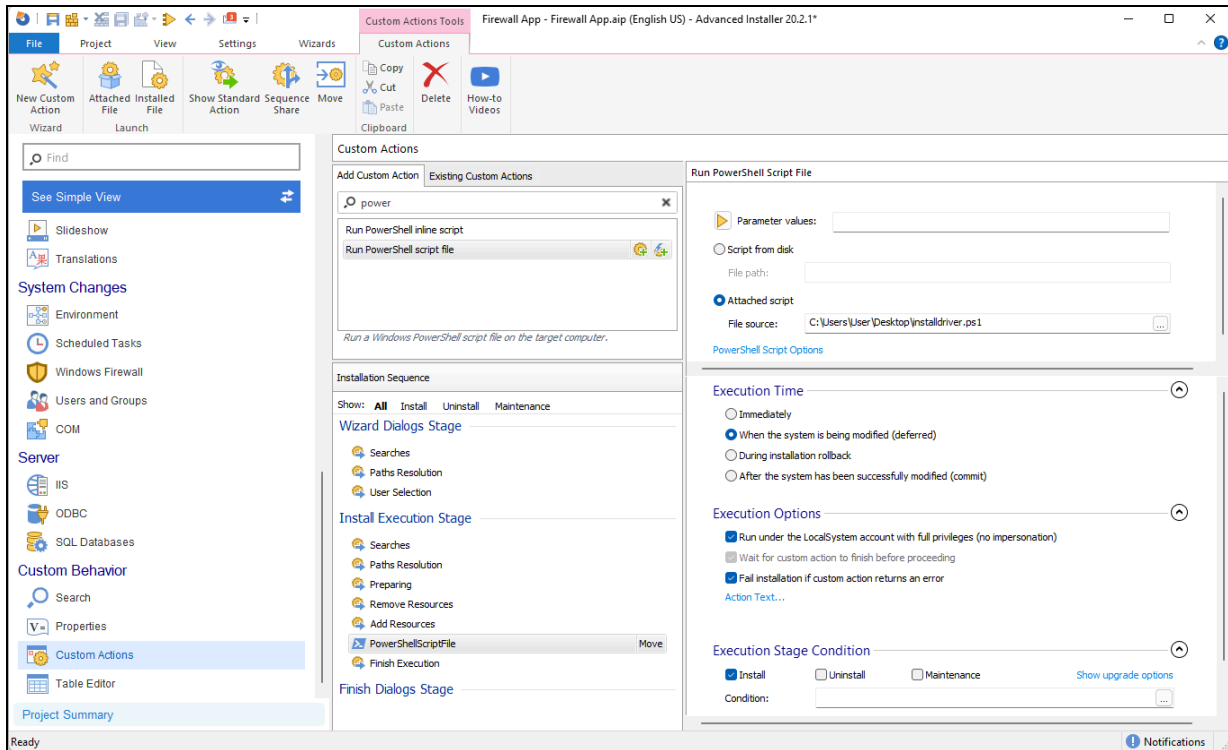


- `$DPInstLoc = $env:SystemRoot + "\DPInst\DPInst_x64.exe"`: Sets the variable `$DPInstLoc` to the path of the `DPInst_x64.exe` file located in the `%SystemRoot%\DPInst` directory. The `$env:SystemRoot` environment variable represents the path to the Windows installation directory.
- `$INFLocation = $env:SystemRoot + "\DPInst\HP.inf"`: Sets the variable `$INFLocation` to the path of the `HP.inf` file located in the `%SystemRoot%\DPInst` directory.
- `$cmd = "$DPInstLoc /S /U $INFLocation"`: Constructs a command string that includes the values of `$DPInstLoc` and `$INFLocation` as well as additional command-line arguments. In this case, the command is `DPInst_x64.exe /S /U HP.inf`. The `/S` switch enables silent installation without displaying any user interface, and the `/U` switch specifies the INF file to be used for uninstallation.
- Invoke-Expression `$cmd`: Executes the command stored in the `$cmd` variable using the Invoke-Expression cmdlet. This cmdlet interprets and runs the command as if it were typed directly into the PowerShell console.

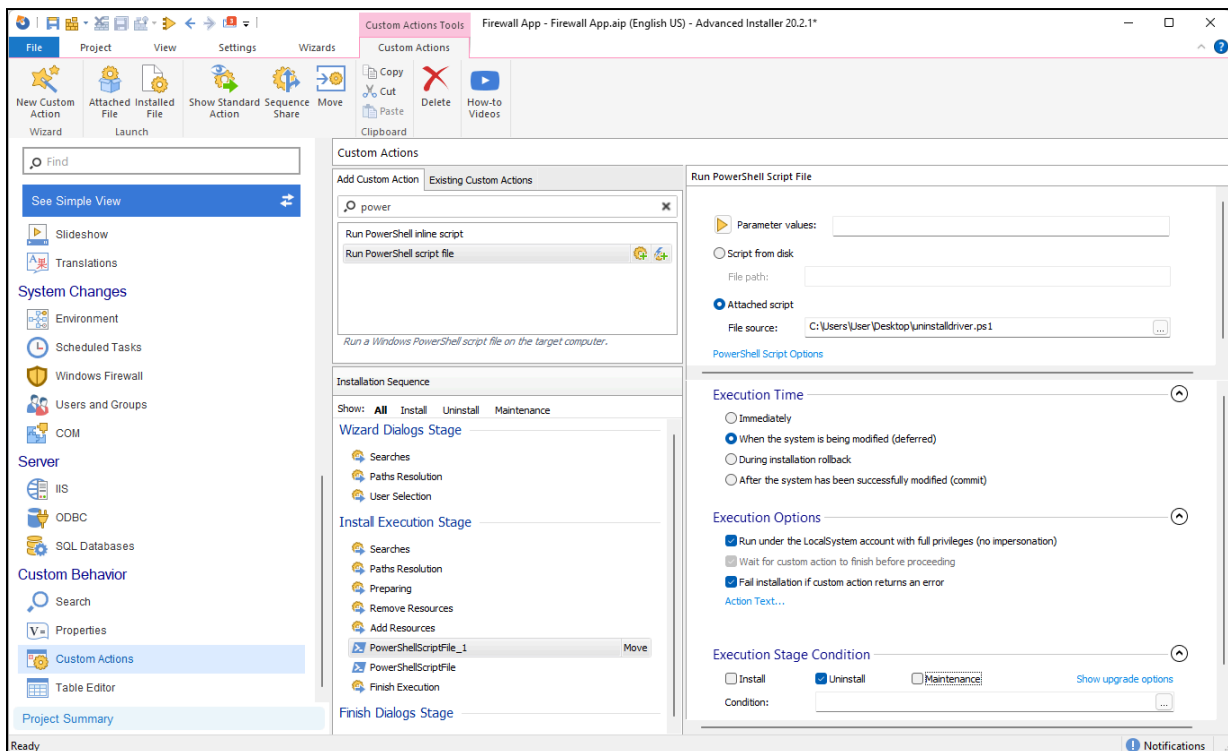
Once we have the scripts done and the `DPInst` utility downloaded, open Advanced Installer and first navigate to the Files and Folders Page. In here, create a new directory under **Windows Volume\Windows** called **DPInst** and add the `DPInst` utility with the `HP.inf` file near it.

Next, navigate to the Custom Actions Page and add the **Run PowerShell script file** predefined custom action into the sequence, select **Attached Script** and select the file that was previously created and configure the Custom Action as such:





Repeat the same process for the uninstall script and configure the Custom Action as follows:



Next, build the package and during installation/uninstallation the PowerShell scripts will run and install/uninstall the driver.

If we are going with the PnPUTil route, the script to install is quite simple:

```
$DriverPath = $env:windir + "\DPIInst"

Get-ChildItem $DriverPath -Recurse -Filter "*inf" | ForEach-Object { PNPUtil.exe /add-driver
$_ .FullName /install }
```

The script performs the following actions:

- `$DriverPath = $env:windir + "\DPIInst"`: Sets the variable `$DriverPath` to the path of the `DPIInst` directory located in the Windows installation directory (`%windir%`). The `$env:windir` environment variable represents the path to the Windows directory.
- `Get-ChildItem $DriverPath -Recurse -Filter "*inf"`: Retrieves all the files with the `".inf"` extension located in the `$DriverPath` directory and its subdirectories using the `Get-ChildItem` cmdlet. The `-Recurse` parameter ensures that files are searched recursively.
- `ForEach-Object { PNPUtil.exe /add-driver $_.FullName /install }`: For each `".inf"` file found in the previous step, it executes the `PNPUtil.exe` utility to add and install the driver specified by the `$_` variable (represents the current file object). The `/add-driver` switch is used to add the driver package, and the `/install` switch is used to install the driver.

The script to uninstall the driver with PnPUTil is:

```
$DriverPath = $env:windir + "\DPIInst\HP.inf"
$Arguments = "pnputil /delete-driver $DriverPath"
Start-Process -FilePath PowerShell.exe -ArgumentList $Arguments -Wait
```

The script performs the following actions:

- `$DriverPath = $env:windir + "\DPIInst\HP.inf"`: Sets the variable `$DriverPath` to the path of the `"HP.inf"` file located in the `DPIInst` directory within the Windows installation directory (`%windir%`). The `$env:windir` environment variable represents the path to the Windows directory.



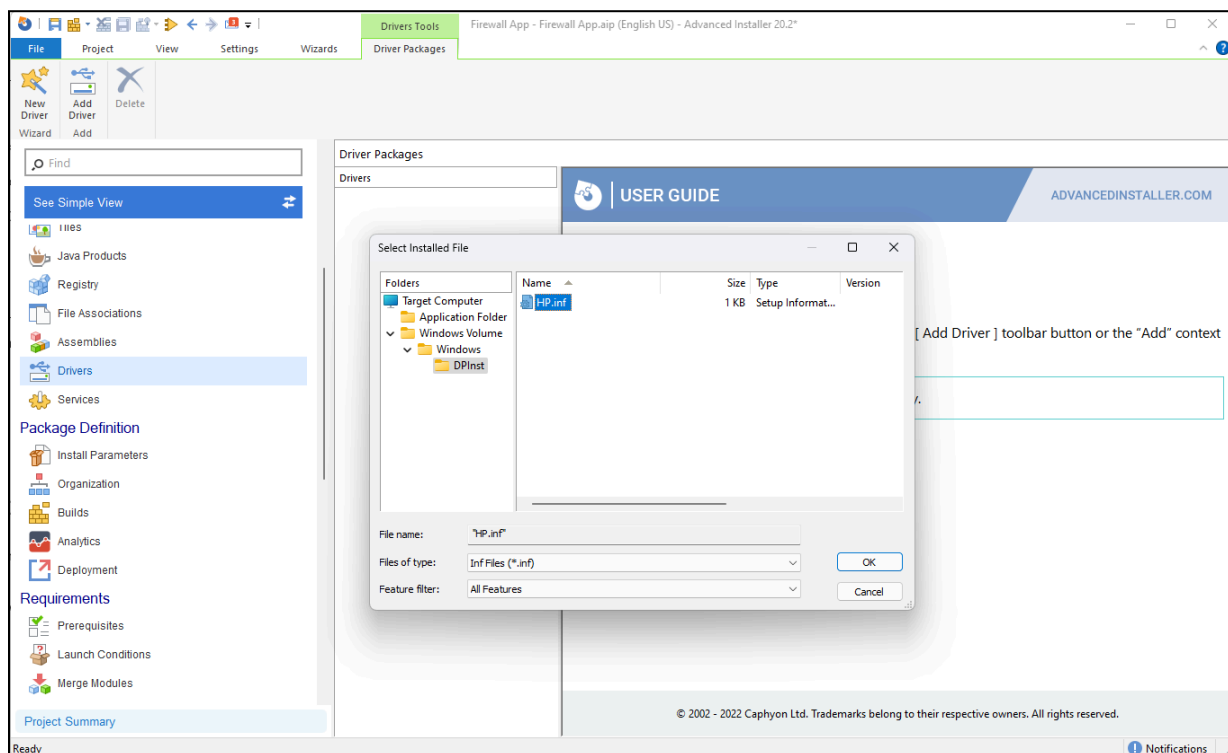
- `$Arguments = "pnputil /delete-driver $DriverPath"`: Sets the variable `$Arguments` to the command-line arguments that will be passed to the PowerShell process. In this case, it constructs a command to delete the driver specified by the `$DriverPath` variable using the `pnputil` utility.
- `Start-Process -FilePath PowerShell.exe -ArgumentList $Arguments -Wait`: Starts a new instance of the PowerShell process and passes the `$Arguments` as command-line arguments. The `-Wait` parameter ensures that the script waits for the PowerShell process to complete before continuing.

Once we have the scripts we need to do the same steps as we did with the `DPIInst` method and then build and install the package.

## Installing drivers with Advanced Installer

Advanced Installer makes it much easier to handle driver operations by providing a simple and intuitive GUI for these actions.

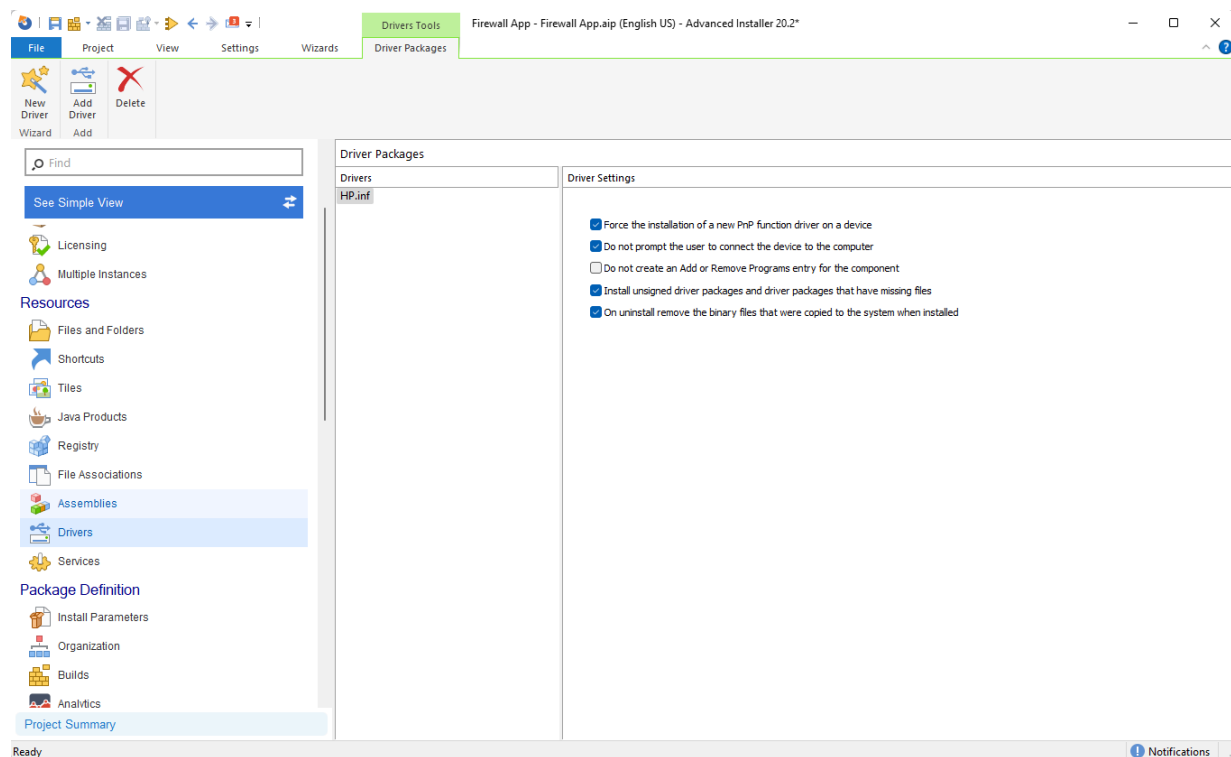
Navigate to the Drivers page and click on **New Driver**. A window will open for you to select the `.inf` file which must be present in the package.



Advanced Installer parses the `.INF` file and detects what is needed and you have multiple settings to choose from:





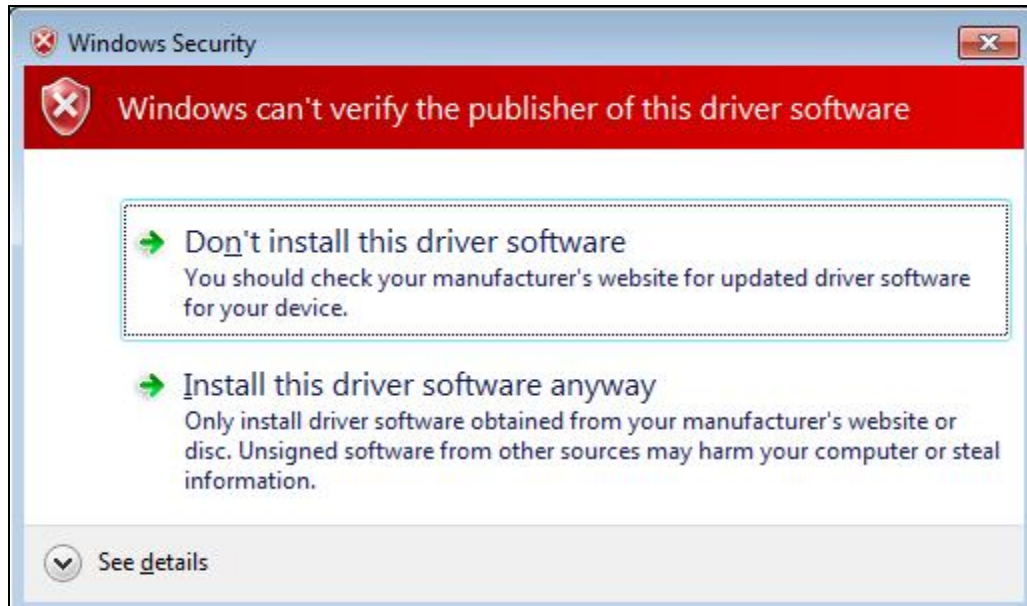


And that is it, all you have to do is build the MSI and install the package. Simple right?

## Install Unsigned Drivers

There are cases when we try to install drivers on Windows and the following windows appears:





And we must click : "Install this driver software anyway".

This is happening because that particular driver is unsigned. In the IT Pros world we need to make sure that these types of drivers are installed silently and no interaction to the user is necessary. We can install the driver without any prompt in following way :

Tools that you need: (most are from the Windows Driver Kit – the latest version of [Windows Driver Kit W11 22H2](#)):

[Inf2Cat.exe](#) (To generate the unsigned catalog file from our INF)

[Makecert.exe](#) (Used to create our certificate)

[Signtool.Exe](#) (Sign our catalog file with an Authenticode digital signature)

[Certmgr.exe](#) (Used to add and delete our certificate to the system root)

Let's have a look at each step that you must take to get your unsigned certificates installed silently.

Create a digital certificate by using the MakeCert tool



Open an **x86/x64 Free Build Environment** command prompt with administrator permissions, by right-clicking **x86 Free Build Environment** on the **Start** menu, and then selecting **Run as administrator**.

At the **x86/x64 Free Build Environment** command prompt, type the following command on a single line (it appears here on multiple lines for clarity and to fit space limitations):

```
makecert -r -n "CN=Name"  
        -ss CertStore  
        -sr LocalMachine
```

Ex: *makecert -r -n CN="TestCert" -ss Root -sr LocalMachine*

The meaning of each parameter is as follows:

- **-r**

Specifies that the certificate is to be "self-signed," rather than signed by a CA. Also called a "root" certificate.

- **-n "CN= Name "**

Specifies the name associated with this new certificate. It is recommended that you use a certificate name that clearly identifies the certificate and its purpose.

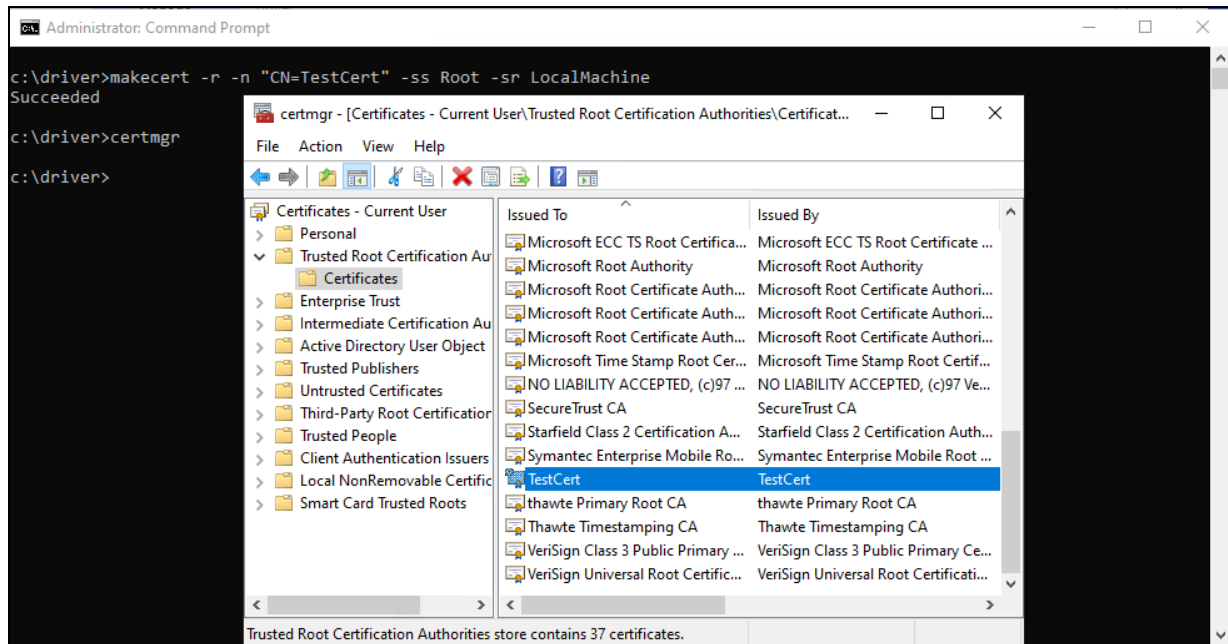
- **-ss CertStore**

Specifies the name of the certificate store in which the new certificate is placed.

- **-sr LocalMachine**

Specifies that the certificate store created by the -ss option is in the per computer store, instead of the default per user store.





The command returns the message "Succeeded" when the store and certificate are created.

### Create a .cat (catalog) file for the driver

We notice that some drivers don't contain a cat file, so we'll need to generate one.

Open the .INF file in a text editor. Ensure that under the [version] section that you have an entry specifying a .cat file. If it's not there, add the line

at the end of the section. For example:

```
[version]
Signature=xxxxxx
Provider=xxxxxx
CatalogFile=MyCatalogFile.cat
```



"MyCatalogFile.cat" is the name of the cat file that we want to generate. Not having a line specifying this will result in an "error 22.9.4 - Missing 32-bit catalog file entry" when we run Inf2Cat.exe.

Command line:

```
Inf2Cat.exe /driver:"<Path to folder containing driver files
```

Ex : Inf2cat.exe /driver:[PathToINFwithoutFile] /os:10\_x64,10\_x86

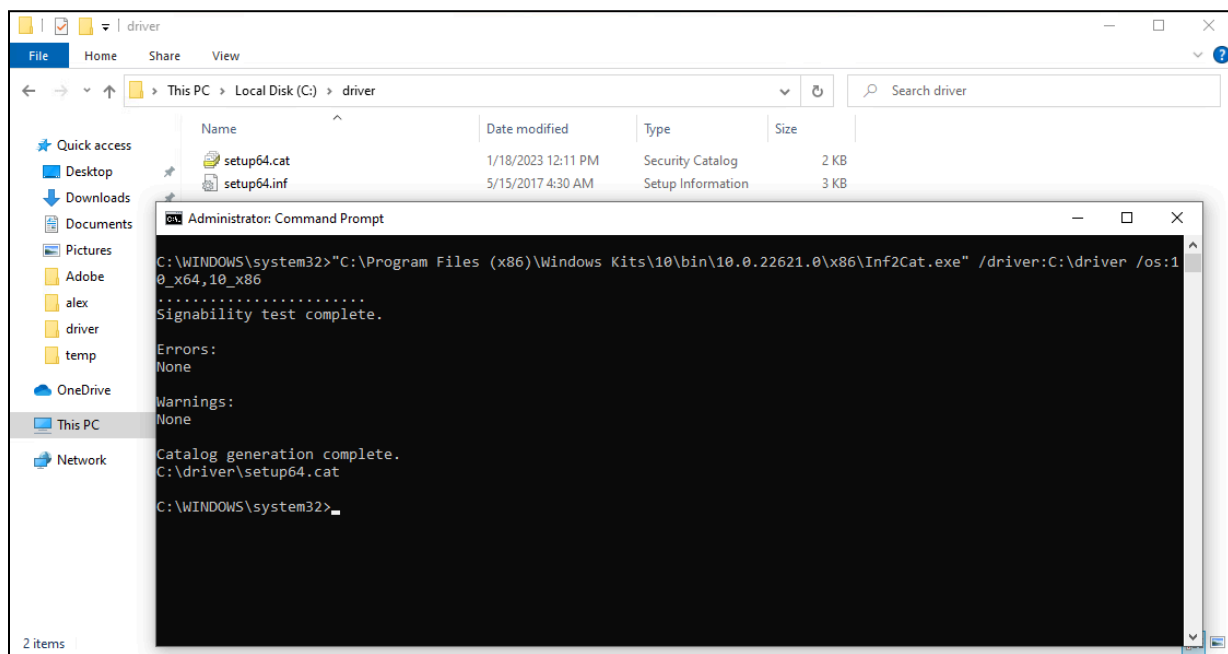
The meaning of each parameter is as follows:

- **/driver:** c:\toaster\device

Specifies the location of the .inf file for the driver package. You must specify the complete folder path. A '.' character does not work here to represent the current folder.

- **/os:** 10\_x86 or 10\_x64

Identifies the 32-bit version of Windows 10 as the operating system. Run the command **inf2cat /?** for a complete list of [supported operating systems and their codes](#).



## Sign the catalog file using SignTool

```
signtool sign /v /sm /s Root /n "TestCert" /t http://timestamp.digicert.com path\example.cat
```

The meaning of each parameter is as follows:

- **/sm**

Specifies that a machine store, instead of a user store, is used

- **/s CertStore**

Specifies the name of the certificate store in which SignTool searches for the certificate specified by the parameter /n. In our case we look in the Root

- **/n "Name "**

Specifies the name of the certificate to be used to sign the package. You must include enough of the name to allow SignTool to distinguish it from others in the store. If this name includes spaces, then you must surround the name with double quotes.

- **/t path to time stamping service**

Specifies the path to a time stamping service at an approved certification authority. If you purchase your certificate from a commercial vendor, they should provide you with the appropriate path to their service.

- **example.cat**

Specifies the path and file name of the catalog file to be signed.

Signtool indicates completion with the following message:



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19044.2364]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd c:\driver

c:\driver>signtool sign /v /sm /s Root /n "TestCert" /t http://timestamp.digicert.com C:\driver\setup64.cat
The following certificate was selected:
    Issued to: TestCert
    Issued by: TestCert
    Expires:   Sat Dec 31 15:59:59 2039
    SHA1 hash: 96295713ED6D5EBDFC7F0D0E1E20F76BC6953E82

Done Adding Additional Store
Successfully signed and timestamped: C:\driver\setup64.cat

Number of files successfully Signed: 1
Number of warnings: 0
Number of errors: 0

c:\driver>
```

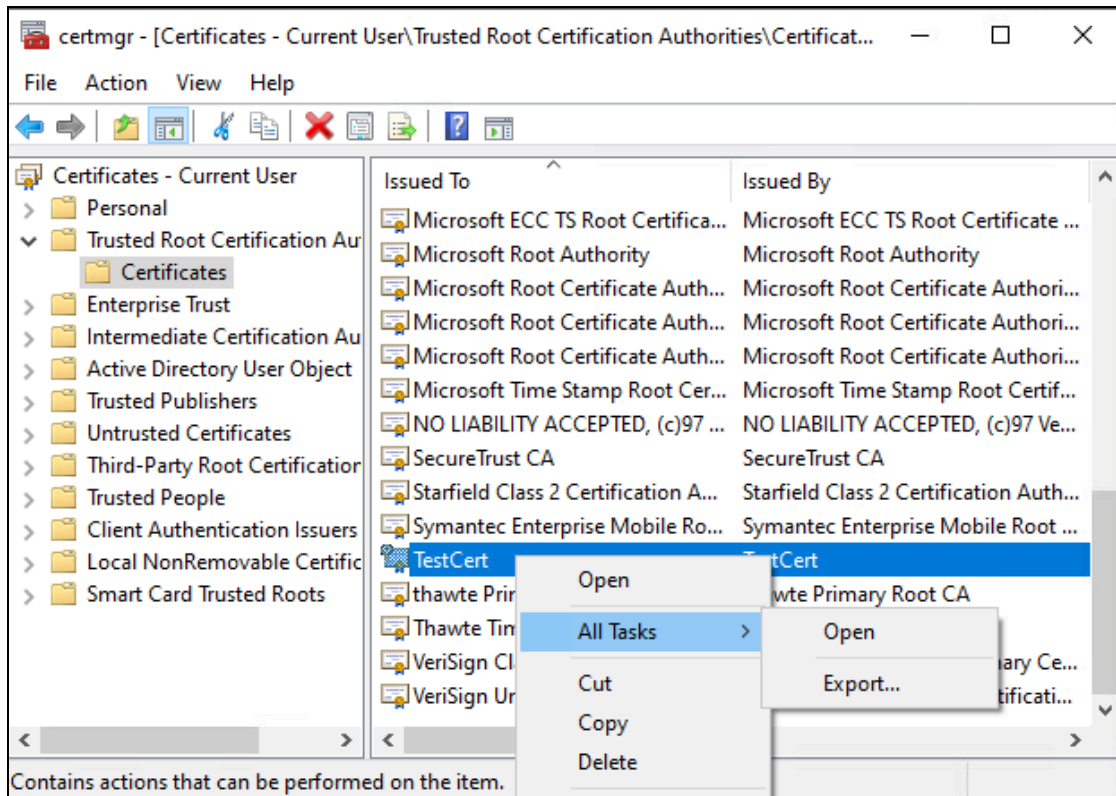
Successfully signed and timestamped: C:\toaster\device\example.cat

Export the certificate from certstore manually

Run an administrator command: certmgr.exe

Export the certificate manually:





Install the certificate to Root and TrustedPublisher

Command line:

```
certutil.exe -addstore "Root" [PathToCertificatewithFile]
```

```
certutil.exe -addstore "TrustedPublisher" [PathToCertificatewithFile]
```





```
Administrator: Command Prompt

c:\driver>certutil.exe -addstore "Root" "C:\Users\AlexMarin\Desktop\testcert.cer"
Root "Trusted Root Certification Authorities"
Signature matches Public Key
Related Certificates:

Exact match:
Element 4:
Serial Number: dbeb9d1d1cec9d8a4535256ff27ac180
Issuer: CN=TestCert
NotBefore: 1/18/2023 12:36 PM
NotAfter: 12/31/2039 3:59 PM
Subject: CN=TestCert
Signature matches Public Key
Root Certificate: Subject matches Issuer
Cert Hash(sha1): 96295713ed6d5ebdfc7f0d0e1e20f76bc6953e82

Certificate "TestCert" already in store.
CertUtil: -addstore command completed successfully.

c:\driver>certutil.exe -addstore "TrustedPublisher" "C:\Users\AlexMarin\Desktop\testcert.cer"
TrustedPublisher "Trusted Publishers"
Signature matches Public Key
Certificate "TestCert" added to store.
CertUtil: -addstore command completed successfully.

c:\driver>
```

for remove :

```
certutil.exe -delstore "Root" [PathToCertificatewithFile]

certutil.exe -delstore "TrustedPublisher" [PathToCertificatewithFile]
```

Now we can install the driver without the prompt.

## Build the MSI

Now that we have understood how you sign a driver we also need to learn how you add the actions in the MSI. Basically all you need are two steps:

1. Install the certificate
2. Install the driver

For the second step we already had a look [a chapter earlier](#) on how to achieve that, so basically all we have to do is create a script that performs the certutil commands previously mentioned. Let's assume that the driver .inf name is HP.inf and let's assume that we are going to place the certificate directly into the C:\Windows\DPInst.

For VBScript:



Option Explicit

On Error Resume Next

Dim strCmd,WshShell,strInstalldir

Set WshShell = CreateObject("WScript.Shell")

strInstalldir = WshShell.ExpandEnvironmentStrings( "%SYSTEMROOT%" )

strCmd = "certutil.exe -addstore " & chr(34) & "Root" & chr(34) & " " & chr(34) & strInstalldir & "  
" & "\DPInst\TestCer.cer" & chr(34)

WshShell.Run strCmd

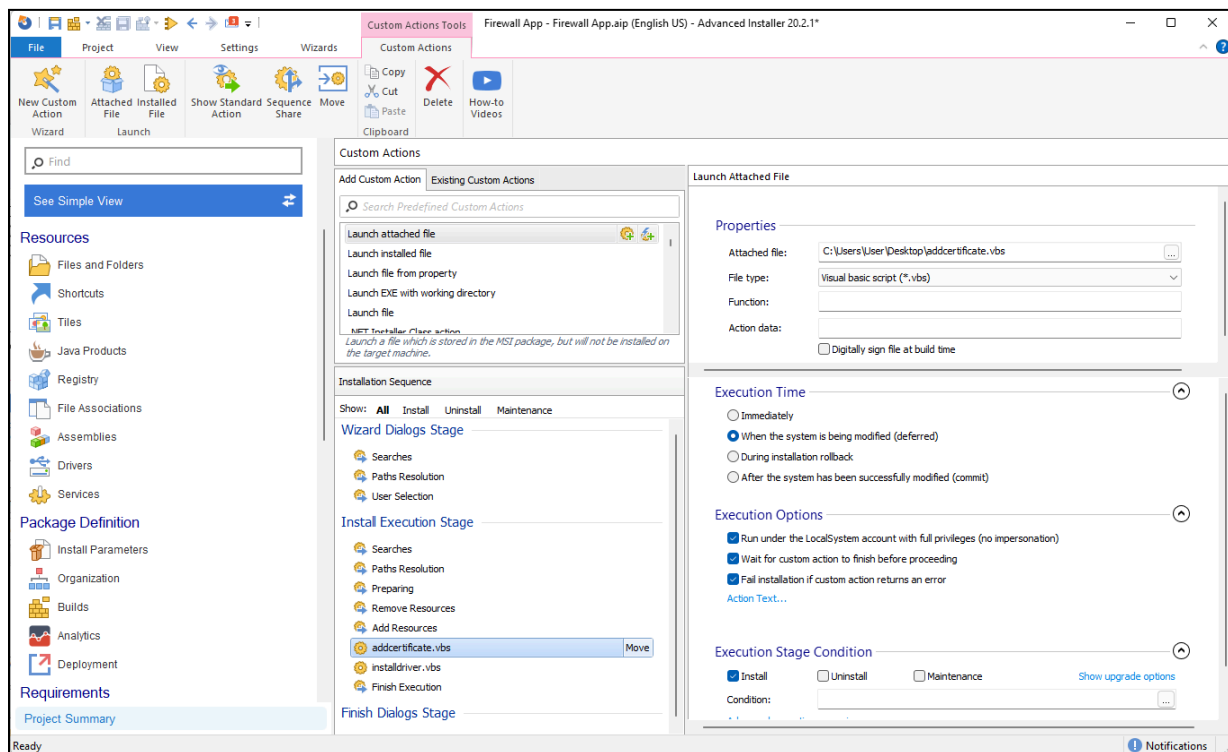
The script performs the following actions:

- Option Explicit: This statement enforces variable declaration in the script, ensuring that all variables are explicitly declared before they are used.
- On Error Resume Next: This statement allows the script to continue running even if an error occurs, bypassing the error and continuing with the next line of code.
- Dim strCmd, WshShell, strInstalldir: Declares three variables: strCmd to hold the command to be executed, WshShell to access the Windows Script Host Shell object, and strInstalldir to store the expanded value of the %SYSTEMROOT% environment variable.
- Set WshShell = CreateObject("WScript.Shell"): Creates an instance of the Windows Script Host Shell object, which allows the script to run shell commands and interact with the Windows environment.
- strInstalldir = WshShell.ExpandEnvironmentStrings("%SYSTEMROOT%"): Retrieves the value of the %SYSTEMROOT% environment variable using the ExpandEnvironmentStrings method of the WshShell object. The %SYSTEMROOT% variable represents the path to the Windows installation directory.
- strCmd = "certutil.exe -addstore " & chr(34) & "Root" & chr(34) & " " & chr(34) & strInstalldir & "\DPInst\TestCer.cer" & chr(34): Constructs the command to be executed. It uses the certutil.exe utility to add a certificate (TestCer.cer) to the "Root" certificate store. The path to the certificate file is obtained by combining strInstalldir (Windows installation directory) with the relative path \DPInst\TestCer.cer. The chr(34) is used to insert double quotes into the command string.
- WshShell.Run strCmd: Executes the command stored in the strCmd variable using the Run method of the WshShell object. This runs the command in a separate process.



In summary, the script uses VBScript to execute the certutil.exe command and add a certificate (TestCer.cer) to the "Root" certificate store. The script retrieves the Windows installation directory, constructs the command with the appropriate paths and options, and then runs it using the WshShell object.

Once the script is created, navigate to the Custom Actions Page and add the **Launch attached file** predefined custom action into the sequence, select the installation vbscript file that was previously created and configure the Custom Action as such:



Make sure that the script which installs the certificate is placed before the script which installs the driver under the "Install Execution Stage" section.

With PowerShell it's even easier because we can use the [Import-Certificate cmdlet](#):

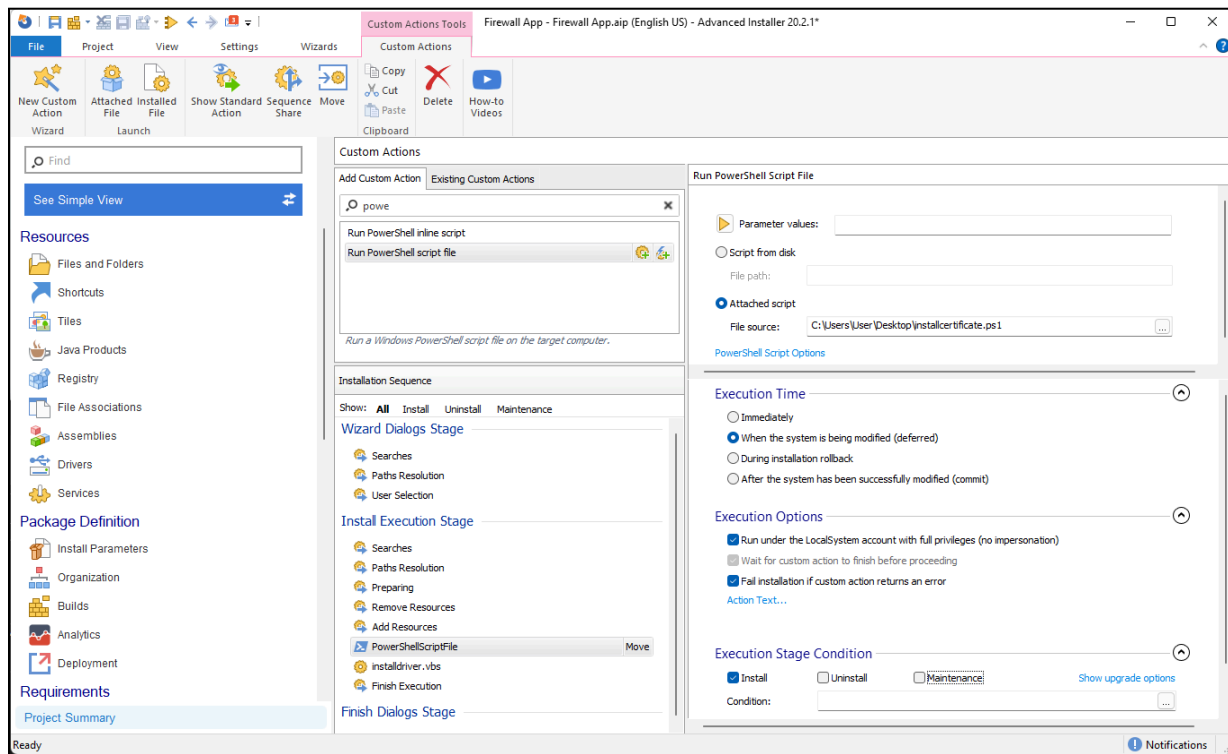
```
$CerLocation = $env:SystemRoot + "\DPIInst\TestCer.cer"
```

```
Import-Certificate -FilePath $CerLocation -CertStoreLocation Cert:\LocalMachine\Root
```



```
Import-Certificate -FilePath $CerLocation -CertStoreLocation  
Cert:\LocalMachine\TrustedPublisher
```

Once we have the script ready, navigate to the Custom Actions Page and add the **Run PowerShell script file** predefined custom action into the sequence, select **Attached Script** and select the file that was previously created and configure the Custom Action as such:



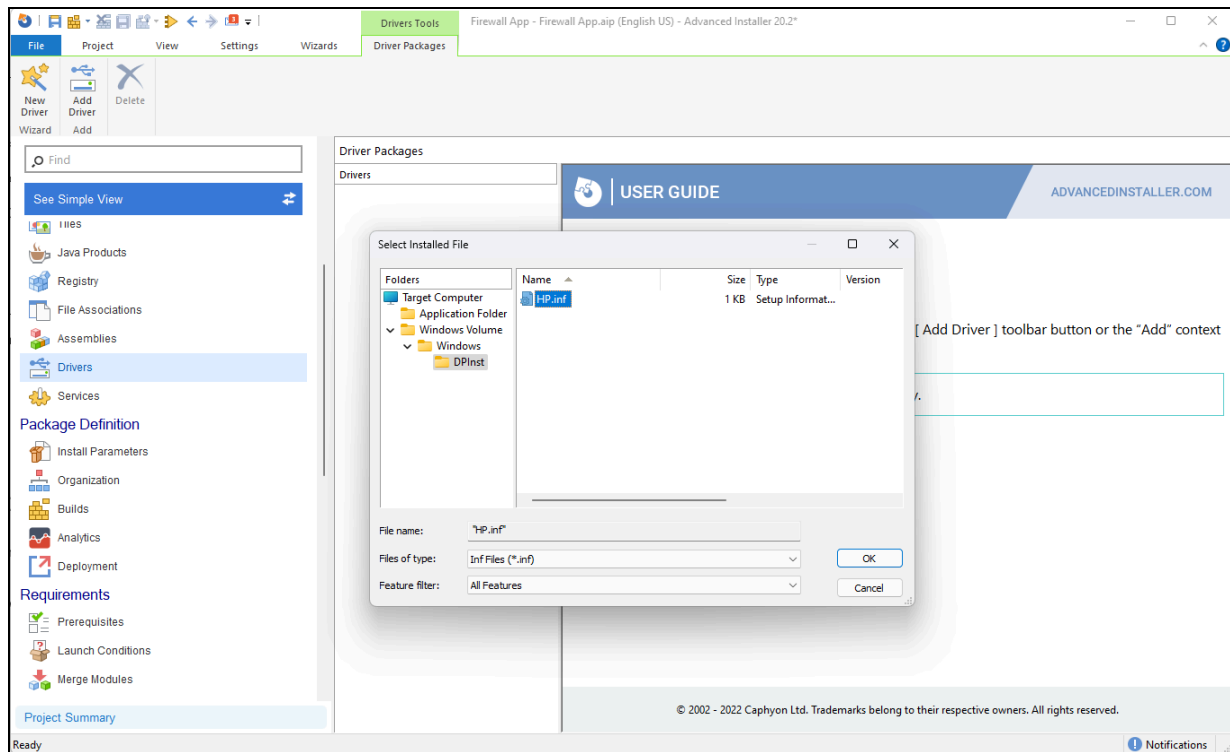
Next, build the package and during installation/uninstallation the PowerShell scripts will run and install/uninstall the driver without asking if we want to install an unsigned driver.

## Installing unsigned drivers with Advanced Installer

As you can see, handling unsigned certificates is not an easy task and it's definitely time consuming. Advanced Installer makes it easier to install unsigned drivers with just a click of a button..and yes that is not a figure of speech.

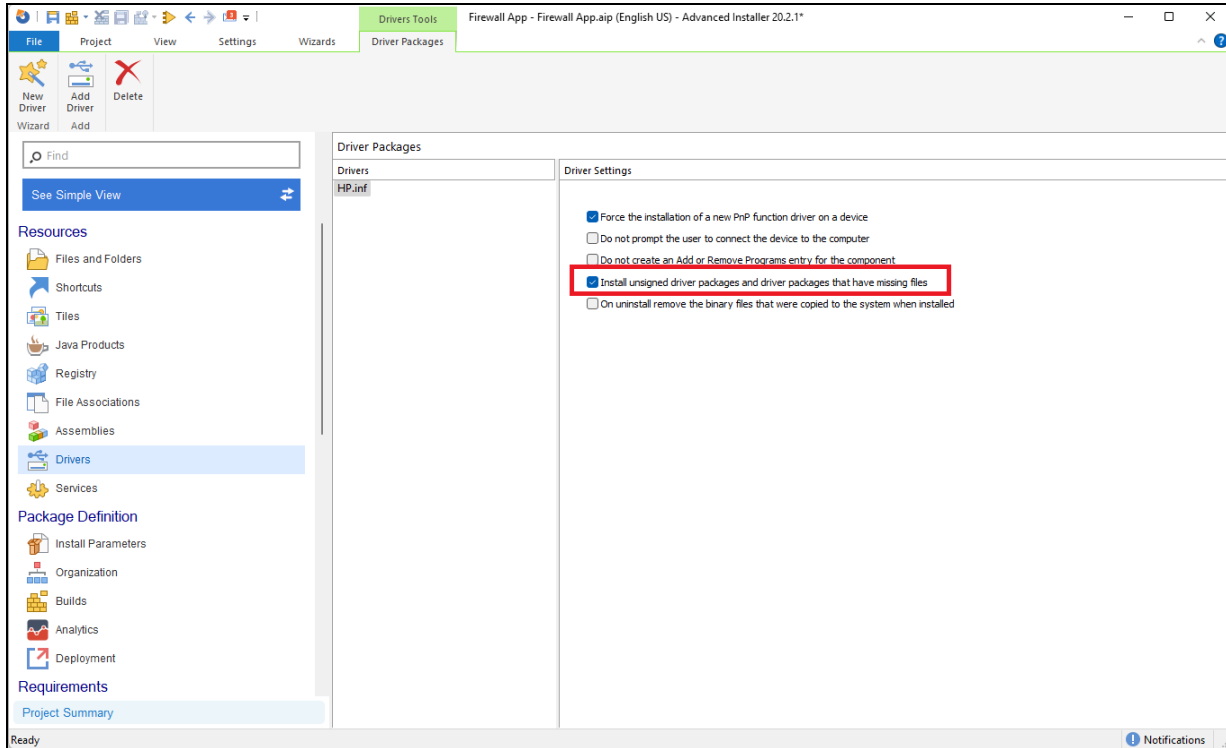
Navigate to the Drivers page and click on **New Driver**. A window will open for you to select the .inf file which must be present in the package.





Next, all you have to do is just select “Install unsigned driver packages and driver packages that have missing files” and Advanced Installer does everything for you!





And that is it! Next, just build and install the package and you should have a clean installation without any warning messages from the OS.

## DLL/OCX register/unregister

Object Linking and Embedding (OLE) is a proprietary Microsoft technology that allows developers to embed and link to other objects. With OLE Control Extension (OCX), you can develop and use custom user interface elements. You can also achieve OLE controls via Dynamic-Link Library(.dll) files.

However, to make sure the data transfer between applications work , these OLE controls must be registered on the system. How can we do that? That's what we will show you in this article.

### What is the Regsvr32 tool?

Regsvr32 is a command-line utility that allows registering and unregistering OLE controls such as DLLs and ActiveX controls in the Windows Registry. Regsvr32 can be found in:



- For 64-bit version: %systemroot%\System32\regsvr32.exe
- For 32-bit version: %systemroot%\SysWoW64\regsvr32.exe

The syntax of the Regsvr32 is actually quite simple:

Regsvr32 [/u] [/n] [/i[:cmdline]] <dllname>

/u - Unregister control

/i - Call DllInstall passing it an optional [cmdline]; when it is used with /u, it calls dll uninstall

/n - do not call DllRegisterServer; this option must be used with /i

/s – Silent

/e - Suppress only the GUI success message but shows the GUI error message

Regsvr32 must always be used from an elevated command prompt.

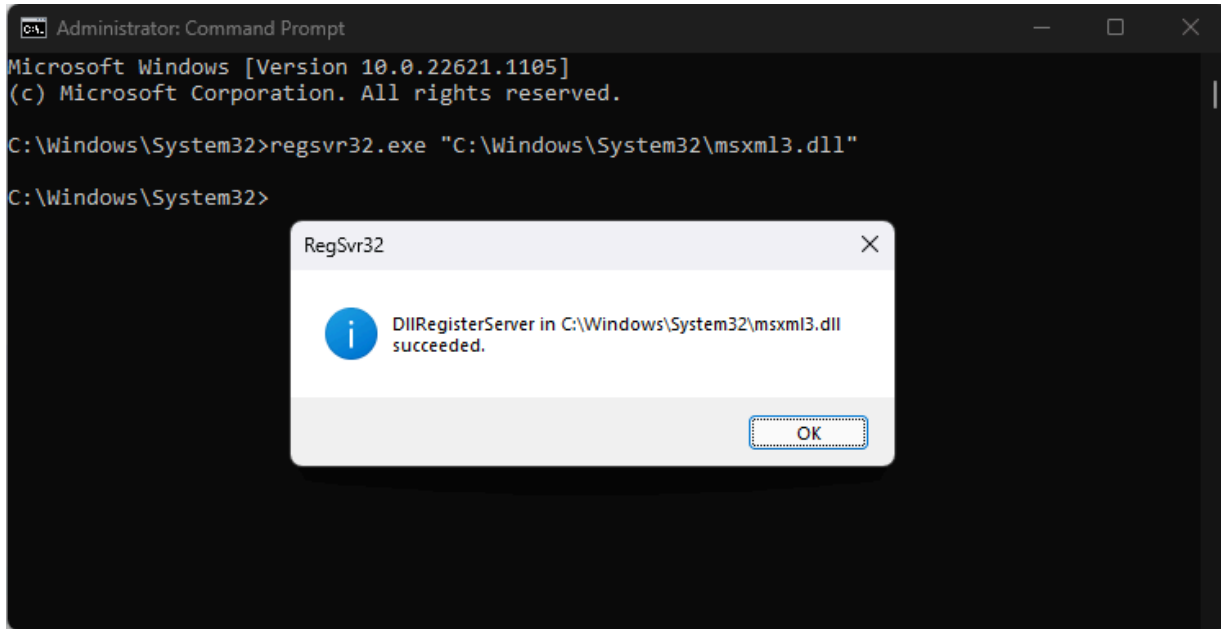
For example, to register a DLL/OCX, you can use:

```
Regsvr32.exe PATH TODLL\name.DLL
```

To unregister a DLL/OCX, you can use:

```
Regsvr32.exe /U PATH TODLL\name.DLL
```





## How to Register DLL/OCX with VBscript?

Now that we know about the Regsvr32 utility, we can build the following scenario. Let's assume that we have a DLL in our project which will be installed in %systemroot%\DLL. So, we can create the following registration script:

```
Option Explicit
On Error Resume Next

Dim strCmd,WshShell,strInstalldir

Set WshShell = CreateObject("WScript.Shell")
strSysRoot = WshShell.ExpandEnvironmentStrings( "%SYSTEMROOT%" )
strInstalldir = strSysRoot & "\DLL"

strCmd = "regsvr32.exe " & chr(34) & strInstalldir &
"\libifcoremd.dll" & chr(34) & " /s"

WshShell.Run strCmd
```

The script performs the following actions:

- Option Explicit: This statement enforces variable declaration in the script, ensuring that





all variables are explicitly declared before they are used.

- On Error Resume Next: This statement allows the script to continue running even if an error occurs, bypassing the error and continuing with the next line of code.
- Dim strCmd, WshShell, strInstallDir: Declares three variables: strCmd to hold the command to be executed, WshShell to access the Windows Script Host Shell object, and strInstallDir to store the path to the DLL file.
- Set WshShell = CreateObject("WScript.Shell"): Creates an instance of the Windows Script Host Shell object, which allows the script to run shell commands and interact with the Windows environment.
- strSysRoot = WshShell.ExpandEnvironmentStrings("%SYSTEMROOT%"): Retrieves the value of the %SYSTEMROOT% environment variable using the ExpandEnvironmentStrings method of the WshShell object. The %SYSTEMROOT% variable represents the path to the Windows installation directory.
- strInstallDir = strSysRoot & "\DLL": Constructs the path to the DLL file by appending the "\DLL" folder to the strSysRoot variable. This assumes that the DLL file is located in the "DLL" folder within the Windows installation directory.
- strCmd = "regsvr32.exe " & chr(34) & strInstallDir & "\libifcoremd.dll" & chr(34) & " /s": Constructs the command to be executed. It uses the regsvr32.exe utility to register a DLL file (libifcoremd.dll). The path to the DLL file is obtained by combining strInstallDir with the relative path \libifcoremd.dll. The chr(34) is used to insert double quotes into the command string, and /s is a parameter to silently register the DLL without displaying any user interface.
- WshShell.Run strCmd: Executes the command stored in the strCmd variable using the Run method of the WshShell object. This runs the command in a separate process.

To unregister the DLL, we can use the following script:

```
Option Explicit
On Error Resume Next

Dim strCmd, WshShell, strInstallDir

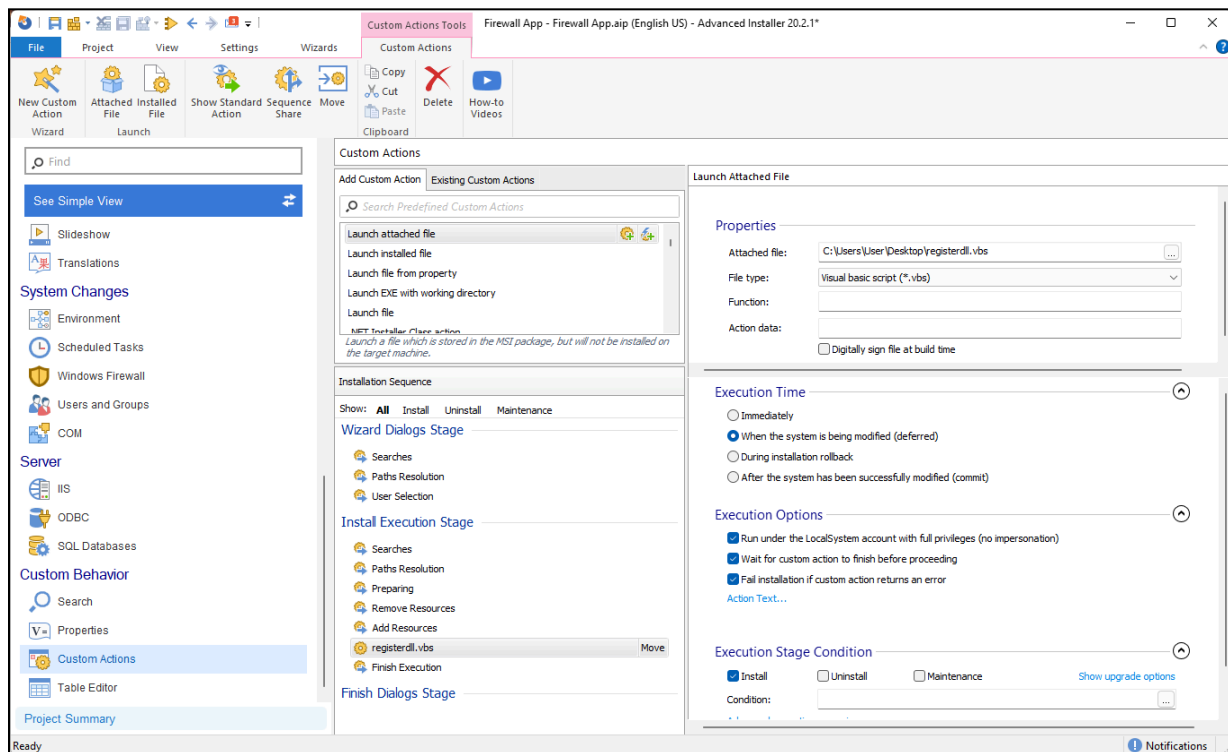
Set WshShell = CreateObject("WScript.Shell")
strSysRoot = WshShell.ExpandEnvironmentStrings( "%SYSTEMROOT%" )
strInstallDir = strSysRoot & "\DLL"

strCmd = "regsvr32.exe /U " & chr(34) & strInstallDir &
"\libifcoremd.dll" & chr(34) & " /s"

WshShell.Run strCmd
```

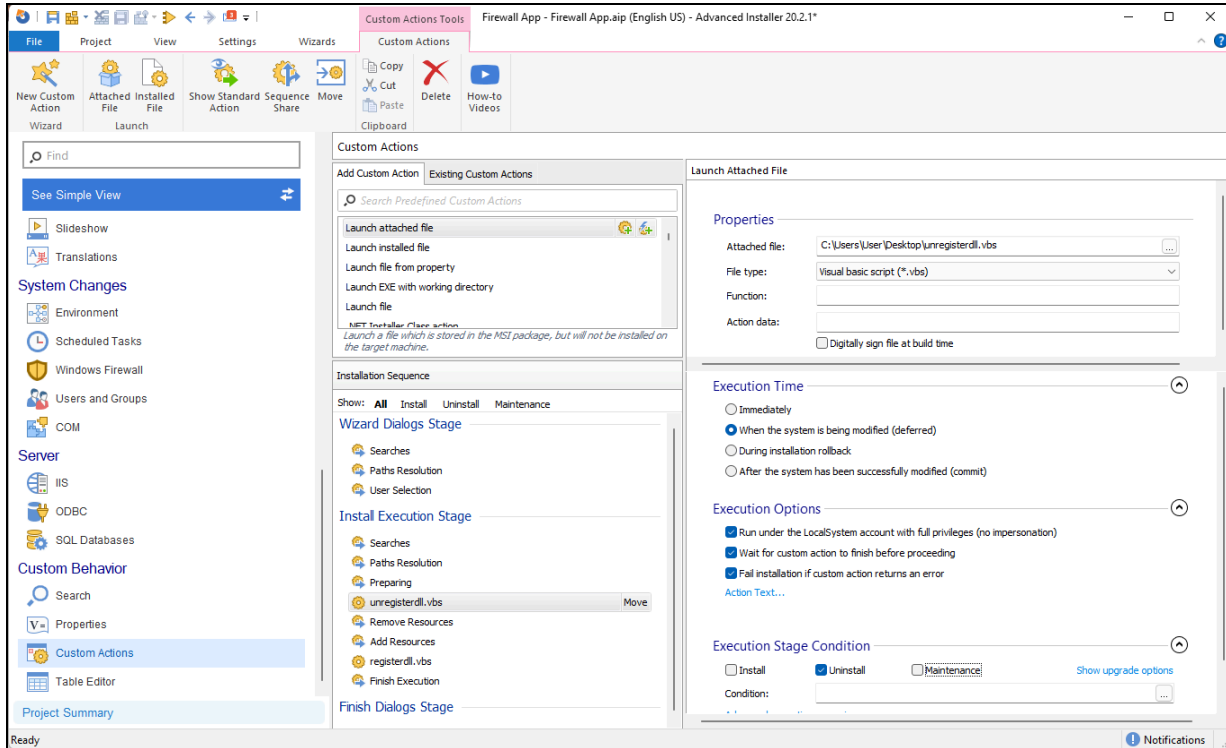


- Now that the scripts are created, we need to open Advanced Installer and navigate to the Custom Actions page.
- There, search for the **Launch attached file** predefined custom script and add it into the sequence.
- Select your previously created registration script and configure the custom action as follows:



We also want to unregister the DLL during uninstallation, so let's create another custom action the same way as we did previously. Except this time, select the unregister script and configure the Custom Action as seen below:





And that is it, all you need to do is **Build the MSI and install it**. The DLL will be registered.

## How to Register DLL/OCX with PowerShell?

Using the same logic as above, we can construct the following script to register a DLL with PowerShell:

```
$Arguments = "C:\Windows\DLL\libifcoremd.dll", "/s"

Start-Process -FilePath 'regsvr32.exe' -Args $Arguments -Wait
-NoNewWindow -PassThru
```

The script performs the following actions:

- **\$Arguments**: Declares a variable to hold the arguments for the regsvr32.exe command and assigns the arguments to be passed to the regsvr32.exe command. The first argument is the path to the DLL file (C:\Windows\DLL\libifcoremd.dll), and the second argument (/s) is a parameter to silently register the DLL without displaying any user interface.
- **Start-Process -FilePath 'regsvr32.exe' -Args \$Arguments -Wait -NoNewWindow**



-PassThru: Executes the regsvr32.exe command using the Start-Process cmdlet. The -FilePath parameter specifies the path to the executable (regsvr32.exe). The -Args parameter specifies the arguments to be passed to the executable, which are stored in the \$Arguments variable. The -Wait parameter ensures that the script waits for the command to complete before continuing. The -NoNewWindow parameter prevents the command from opening a new window. The -PassThru parameter returns an object representing the newly created process, allowing for further interaction if needed.

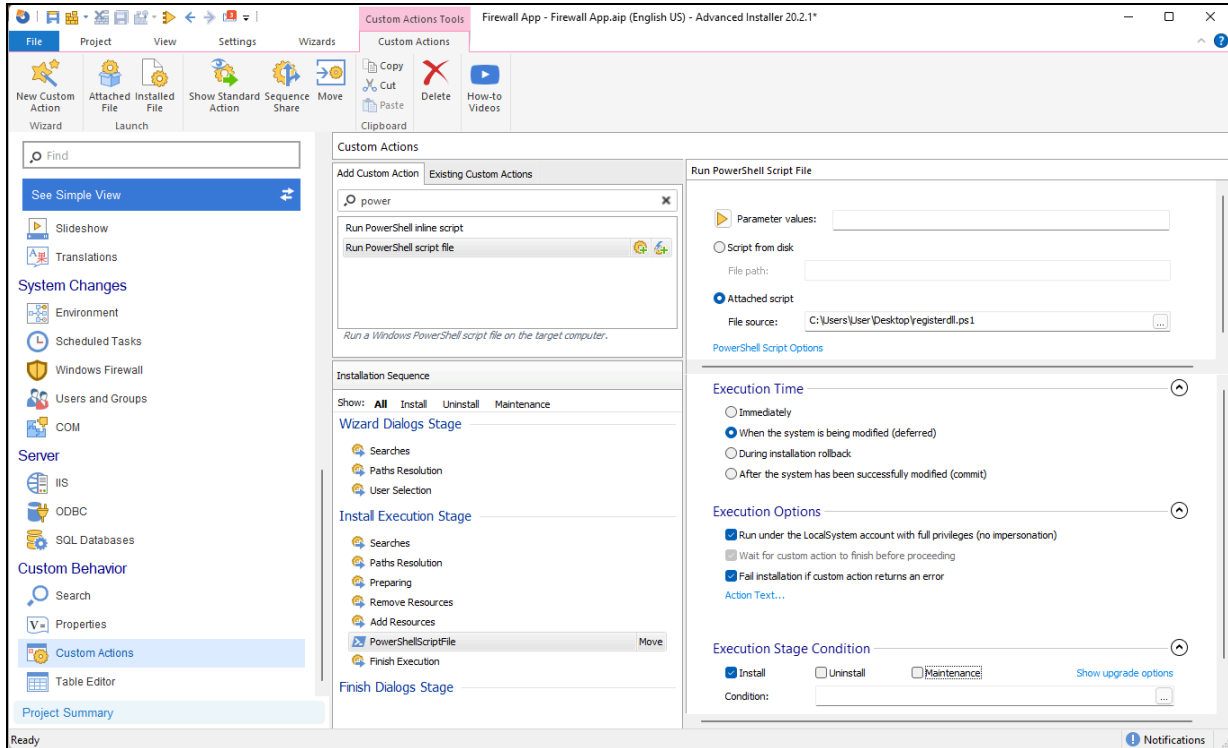
To unregister the DLL, we can use the following script:

```
$Arguments = "/u","C:\Windows\DLL\libifcoremd.dll", "/s"

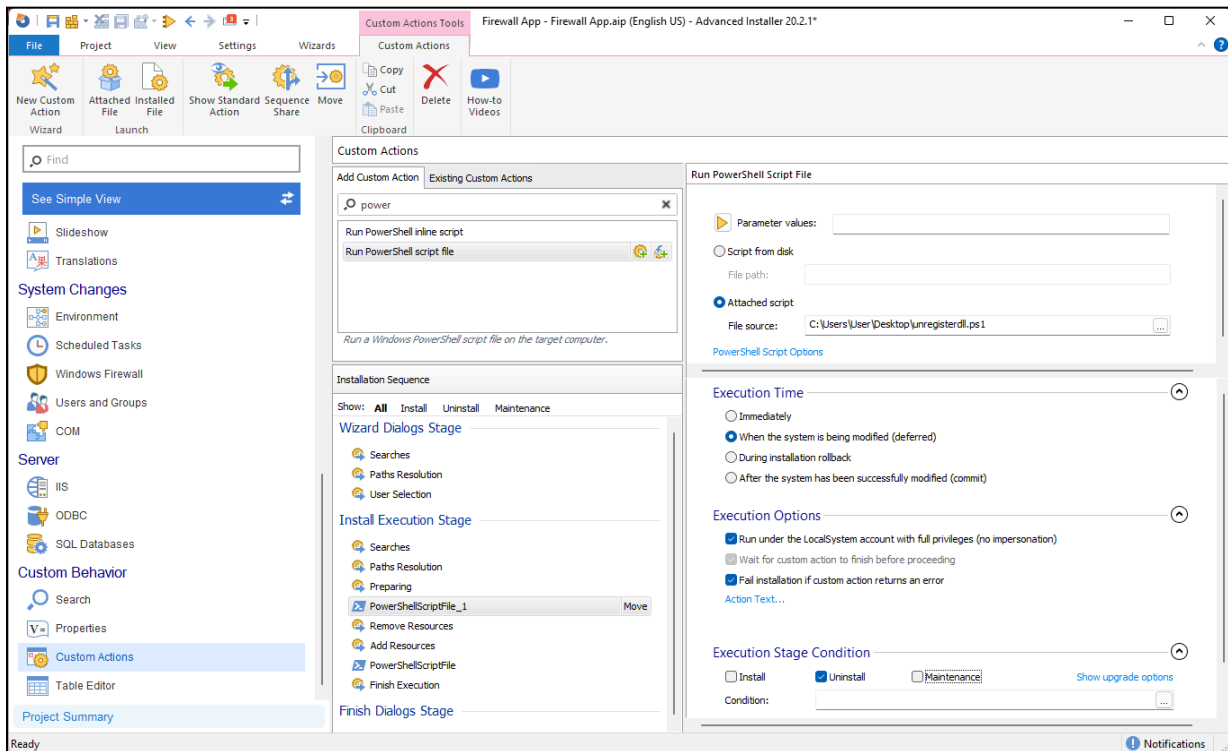
Start-Process -FilePath 'regsvr32.exe' -Args $Arguments -Wait
-NoNewWindow -PassThru
```

- When the scripts are created, we navigate to the Custom Actions page.
- Search for the **Run PowerShell script file** predefined Custom Action and add it in the sequence.
- Once we select the registration PowerShell script, we can proceed to configure the Custom Actions as follows:





For the unregister action, we follow the same steps as above and configure the Custom Actions:



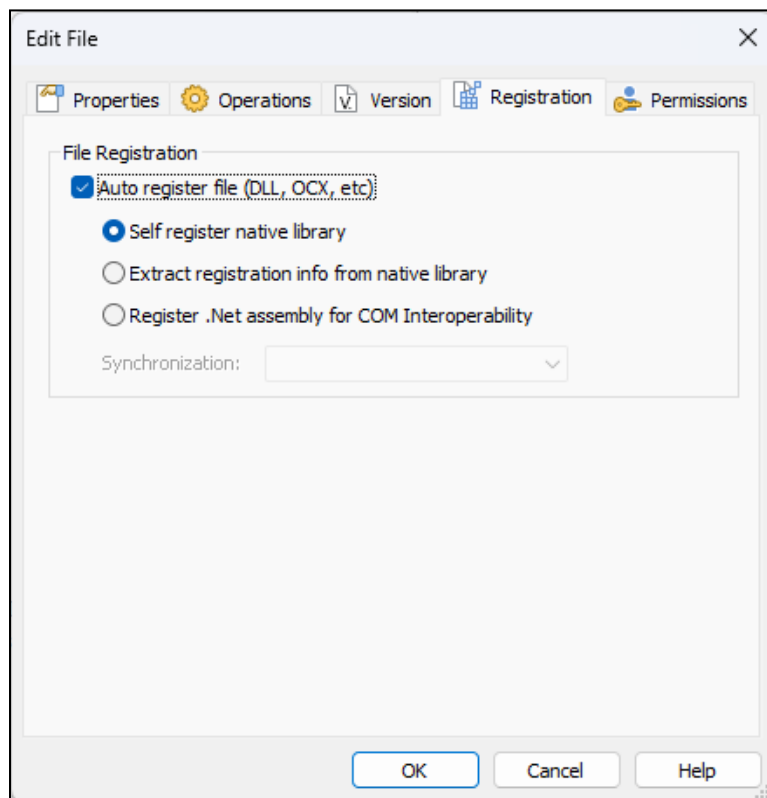
And you're done, now you can Build the MSI and install it. The DLL will be registered.

## How to Register DLL/OCX with Advanced Installer?

Advanced Installer makes it much easier to handle OLE control registration by providing a [Registration Tab GUI](#).

Accessing the GUI is quite easy with these steps:

1. First navigate to the [Files and Folders](#) page
2. Add the DLL/OCX files.
3. Once the files are added, right-click on the DLL/OCX and select **Properties**.
4. Then, navigate to the Registration Tab.



As you may notice, there are three methods for registering files, two for native libraries and one for .NET assemblies.

- Self-register native library

This file is marked for self-registration, however, the self-registration method for registering



components has many drawbacks (like not being able to roll back the changes if something fails later in the install) and it is against Microsoft guidelines.

- Extract registration info from the native library

All the necessary registry entries and keys are installed separately. You can see them in the [Registry](#) and [COM](#) pages. **This is the preferred way to register a file.**

- Register .NET assembly for COM interoperability

It creates the required registry entries so that your assembly is operable through COM. You can see those registry entries in the Registry page.

## Write line in hosts file

On the Windows operating system, the hosts file is a plain text file that maps hostnames to IP addresses. It functions as a local DNS (Domain Name System) resolver, allowing you to specify the IP address associated with a specific hostname.

The hosts file is located at %SystemRoot%\System32\drivers\etc\hosts, where %SystemRoot% represents the Windows installation directory (e.g., C:\Windows). The file has no file extension by default and is just titled "hosts."

Each line in the hosts file begins with an IP address and ends with one or more hostnames, separated by spaces or tabs. As an example:

127.0.0.1    localhost
------------------------

The IP address 127.0.0.1 is associated with the hostname localhost in this example. In a web browser, typing localhost will resolve to the specified IP address.

To add custom mappings between hostnames and IP addresses, manually edit the hosts file. This can be useful for a variety of purposes, including redirecting a domain name to a different IP address for testing or blocking access to specific websites by redirecting them to a non-existent or local IP address.

Please keep in mind that editing the hosts file usually necessitates administrative privileges. As a result, you may need to use an administrator account to run a text editor or any program that modifies the hosts file.



There might be cases where the modification of the hosts file is required directly from the MSI package, and the only way to do this is only via Custom Actions.

## Write in hosts file with VBScript

If you want to use VBScript to write in the hosts file, this is quite easy to accomplish.

```
Const ForReading = 1
Const ForWriting = 2
Const HostsFilePath = "C:\Windows\System32\drivers\etc\hosts"

Dim objFSO, objFile
Dim strIPAddress, strHostname, strNewEntry

' Prompt user for the IP address and hostname
strIPAddress = "127.0.0.1"
strHostname = "example.com"

' Create the new hosts file entry
strNewEntry = strIPAddress & vbTab & strHostname

' Open the hosts file for appending
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile(HostsFilePath, ForAppending,
True)

' Check if the entry already exists in the hosts file
Do Until objFile.AtEndOfStream
    If LCase(objFile.ReadLine) = LCase(strNewEntry) Then
        MsgBox "The entry already exists in the hosts file."
        objFile.Close
        WScript.Quit
    End If
Loop

' Close the file and reopen it for writing
objFile.Close
Set objFile = objFSO.OpenTextFile(HostsFilePath, ForAppending,
```





```
True)

' Write the new entry to the hosts file
objFile.WriteLine(strNewEntry)
objFile.Close
```

The above VBScript performs the following actions:

- It defines constants for file reading (ForReading) and file writing (ForWriting) modes, and specifies the path to the hosts file (HostsFilePath), which is typically located at "C:\Windows\System32\drivers\etc\hosts".
- It declares variables to store the IP address, hostname, and the new entry that will be added to the hosts file.
- The IP address is set to "127.0.0.1" (loopback address) and the hostname is set to "example.com".
- It creates a string (strNewEntry) by combining the IP address and hostname, separated by a tab character.
- It uses the FileSystemObject to access the hosts file and open it in appending mode (ForAppending). If the hosts file doesn't exist, it will be created.
- It reads each line of the hosts file to check if the new entry already exists. If a matching entry is found, a message box is displayed, and the script terminates.
- If the entry is not found, the file is closed and reopened in appending mode to write the new entry using the WriteLine method.
- Once the new entry is written, the file is closed again.

## Write in hosts file with PowerShell

If you want to use PowerShell to write in the hosts file, this is also quite easy to accomplish.

```
$hostsFilePath = "$env:SystemRoot\System32\drivers\etc\hosts"

# Specify the hostname and IP address
$hostname = "example.com"
$ipAddress = "127.0.0.1"

# Check if the hosts file exists
```



```

if (Test-Path $hostsFilePath) {
    # Check if the entry already exists in the hosts file
    $existingEntry = Get-Content $hostsFilePath | Where-Object { $_
-likes "$ipAddress *$hostname*" }

    if ($existingEntry) {
        Write-Host "Entry already exists in the hosts file."
    }
    else {
        # Append the new entry to the hosts file
        $newEntry = "$ipAddress $hostname"
        Add-Content -Path $hostsFilePath -Value $newEntry
        Write-Host "Entry added to the hosts file."
    }
}
else {
    Write-Host "Hosts file not found."
}

```

The script performs the following actions:

- **\$hostsFilePath**: Specifies the path to the hosts file (C:\Windows\System32\drivers\etc\hosts).
- **\$hostname**: Specifies the hostname to be added to the hosts file.
- **\$ipAddress**: Specifies the corresponding IP address for the hostname.
- **if (Test-Path \$hostsFilePath) { ... }**: Checks if the hosts file exists at the specified path. If it does, the script proceeds to modify the file. Otherwise, it outputs a message indicating that the hosts file was not found.
- **\$existingEntry = Get-Content \$hostsFilePath | Where-Object { \$\_ -likes "\$ipAddress \*\$hostname\*" }**: Reads the content of the hosts file and searches for an existing entry that matches the specified IP address and hostname. If a matching entry is found, it is stored in the **\$existingEntry** variable.
- **if (\$existingEntry) { ... }**: Checks if an existing entry was found in the hosts file. If so, it outputs a message indicating that the entry already exists.
- **else { ... }**: If no existing entry was found, the script continues to add the new entry to the hosts file.
- **\$newEntry = "\$ipAddress \$hostname"**: Constructs the new entry by combining the IP address and hostname.
- **Add-Content -Path \$hostsFilePath -Value \$newEntry**: Appends the new entry to the hosts file using the Add-Content cmdlet.
- **Write-Host "Entry added to the hosts file."**: Outputs a message indicating that the new entry has been successfully added to the hosts file.



The script checks if the hosts file exists and if the entry already exists in the file. If the entry is found, it displays a message indicating that the entry already exists. If the entry doesn't exist, it appends the new entry to the hosts file using the Add-Content cmdlet.

I have compiled a list of the most used scripts in Software Packaging that you can download from [here](#).

## Working with Conditional Statements

Conditional statements give you greater control and flexibility when creating MSI packages. This allows you to build robust setups that can be tailored to suit various system configurations, making it easier for users to install as required.

Conditional statements in MSI packaging give you the control to decide when an action should be taken. This means that certain files or applications will only get installed if a certain set of conditions are met. As an example, you can specify that a file should only be installed if another file is already present on the system.

To use conditional statements in MSI packaging, you'll need to specify the condition in the appropriate section of the MSI package. For example, to conditionally install a file based on the presence of a specific registry key, you would add the following code to the component where the file is located:

```
<Component Id="MyComponent" Directory="INSTALLDIR"> <File Id="MyFile"
Source="MyFile.txt">
<Condition><![CDATA[REGISTRY_VALUE_EXISTS("HKEY_LOCAL_MACHINE\Software\MyApp",
"MyKey")]]></Condition> </File> </Component>
```

In this example, the file "MyFile.txt" will only be installed if the registry key "HKEY\_LOCAL\_MACHINE\Software\MyApp\MyKey" exists on the system.

[Custom actions](#) are yet another way to use conditional statements. Custom actions are scripts or programs that are executed during the installation process and can be used to perform tasks that the standard MSI package does not support.



For example, you can use a custom action to check for the presence of a specific file on the system, and use that information to conditionally install a file. To do this, you would create a custom action that checks for the presence of the file, and set a property that can be used as a condition in the appropriate section of the MSI package.

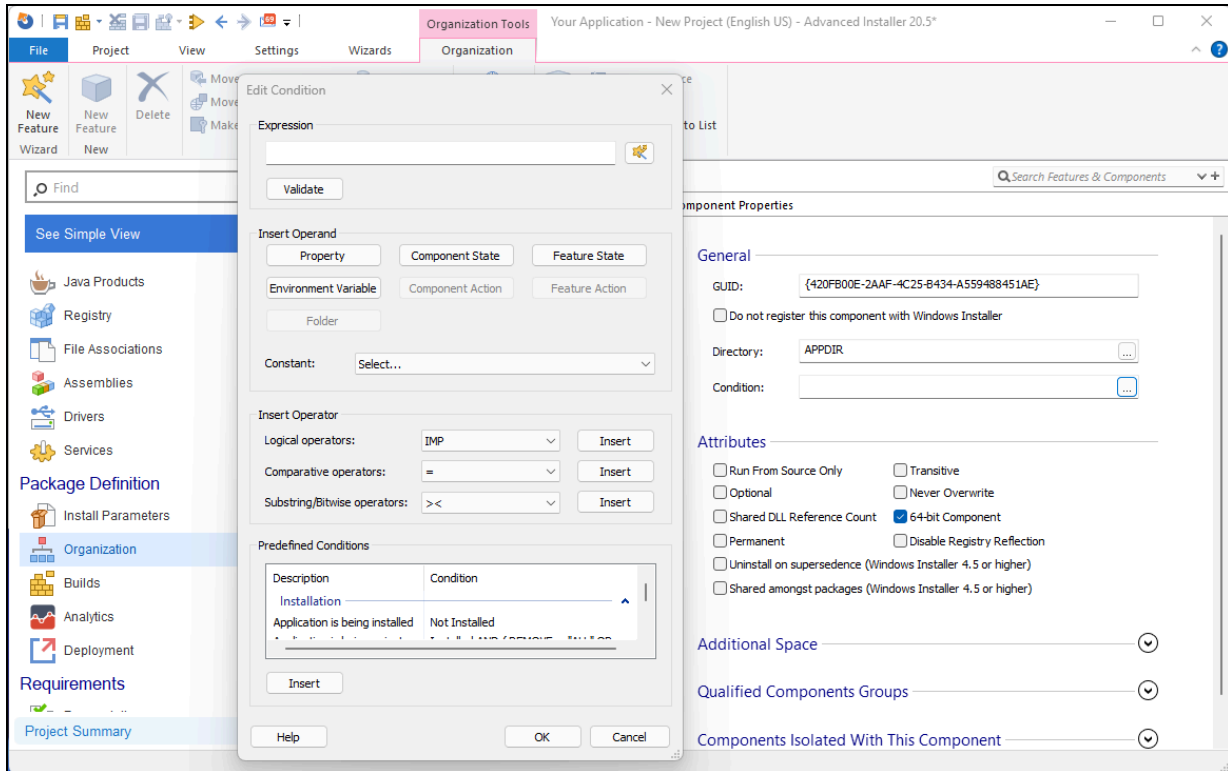
The last type of conditional statements that you could use with MSI are called [launch conditions](#). Prior to initiating the installation process, launch conditions must be satisfied. This can include verifying that certain system requirements are met, for example, determining if specific hardware or software components are present, or the presence of certain registry keys. In other words, launch conditions are a type of conditional statement that are used to control the behavior of the installation process based on the system configuration. By using launch conditions, you can ensure that the installation package is installed only on systems that meet the specified requirements, reducing the potential for errors and improving the overall reliability of the installation process.

## Component Conditions

Advanced Installer makes it much easier to work with conditions while building MSI components. You can easily set conditions on files, registry keys and other elements to control their setup process. This streamlined interface really simplifies the entire MSI component creation process based on system configuration.

To add a condition to an MSI component in Advanced Installer, simply select the component and add the appropriate condition under **Component Properties**. There are numerous conditions to choose from, including system properties, registry keys, file and folder presence, and custom conditions.



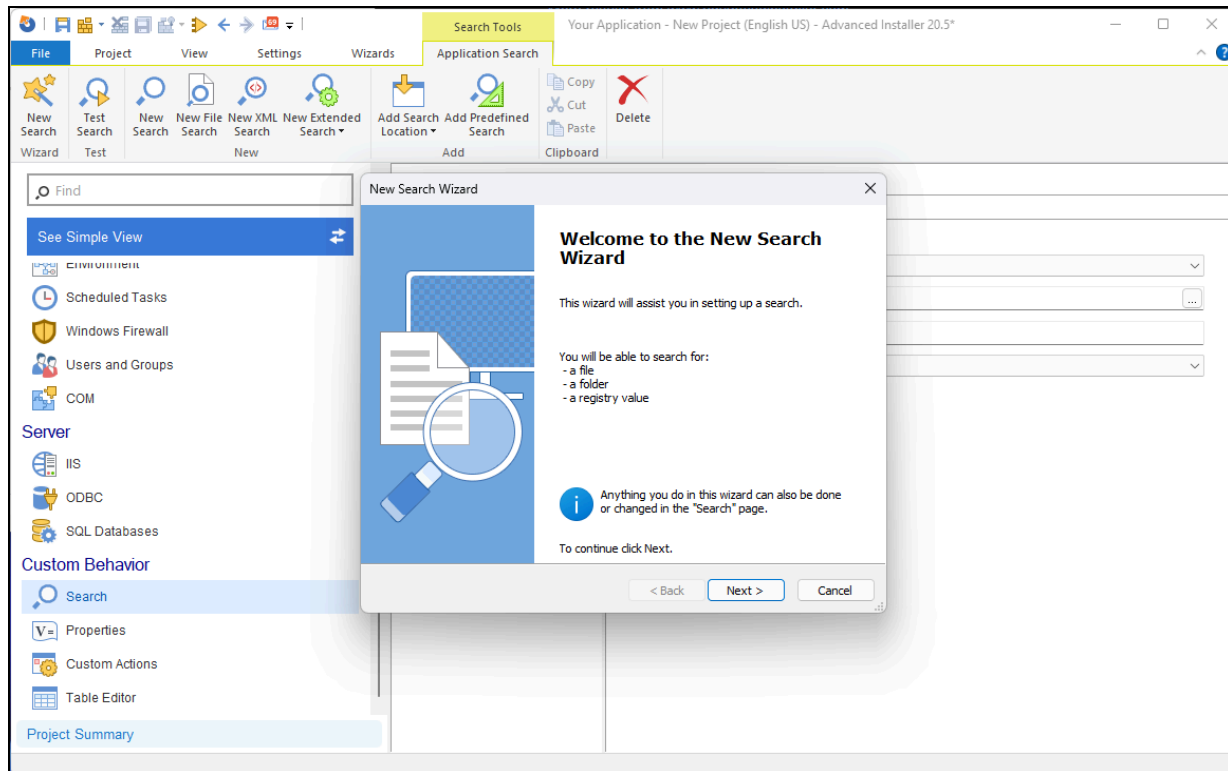


To perform the same actions as explained earlier where the file is only installed if a certain registry key exists, we must do some steps:

1. Search for the registry key and store this result into a property. We are going to use the default RESULT\_PROPERTY

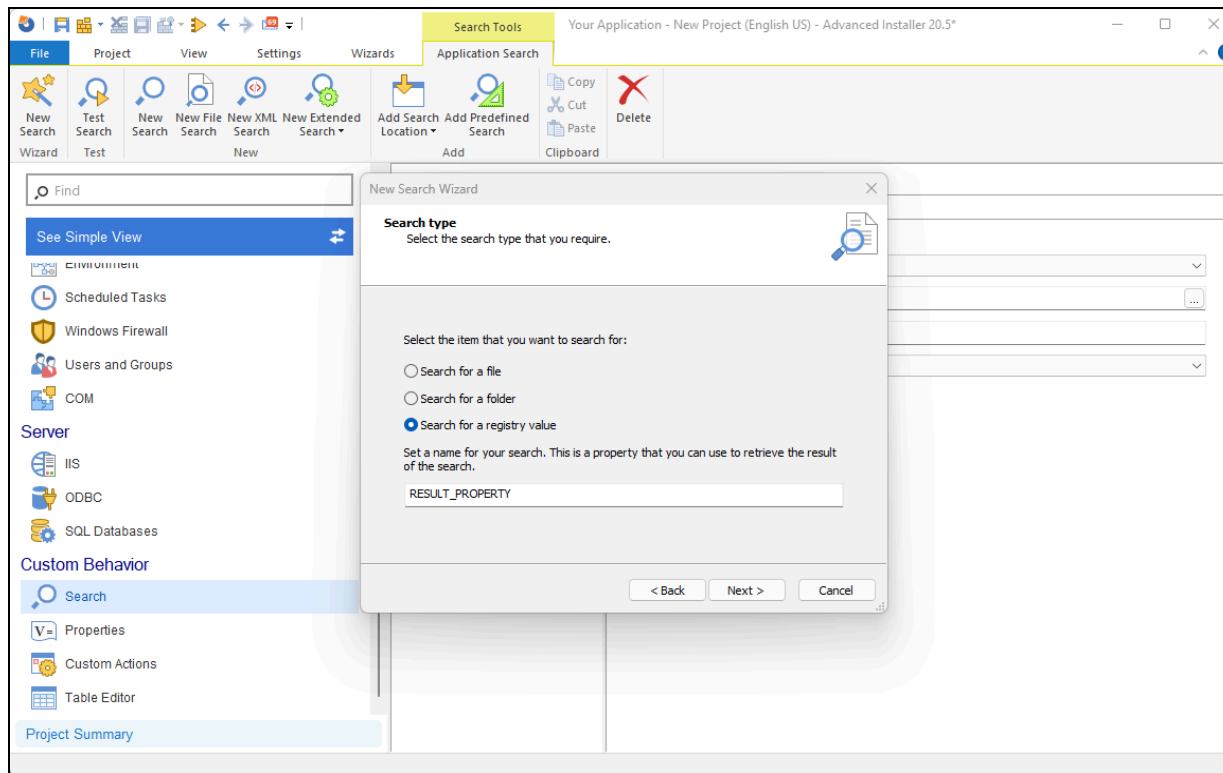
To do this, navigate to the Search page and click on New Search. A wizard will appear that will guide you through the process.





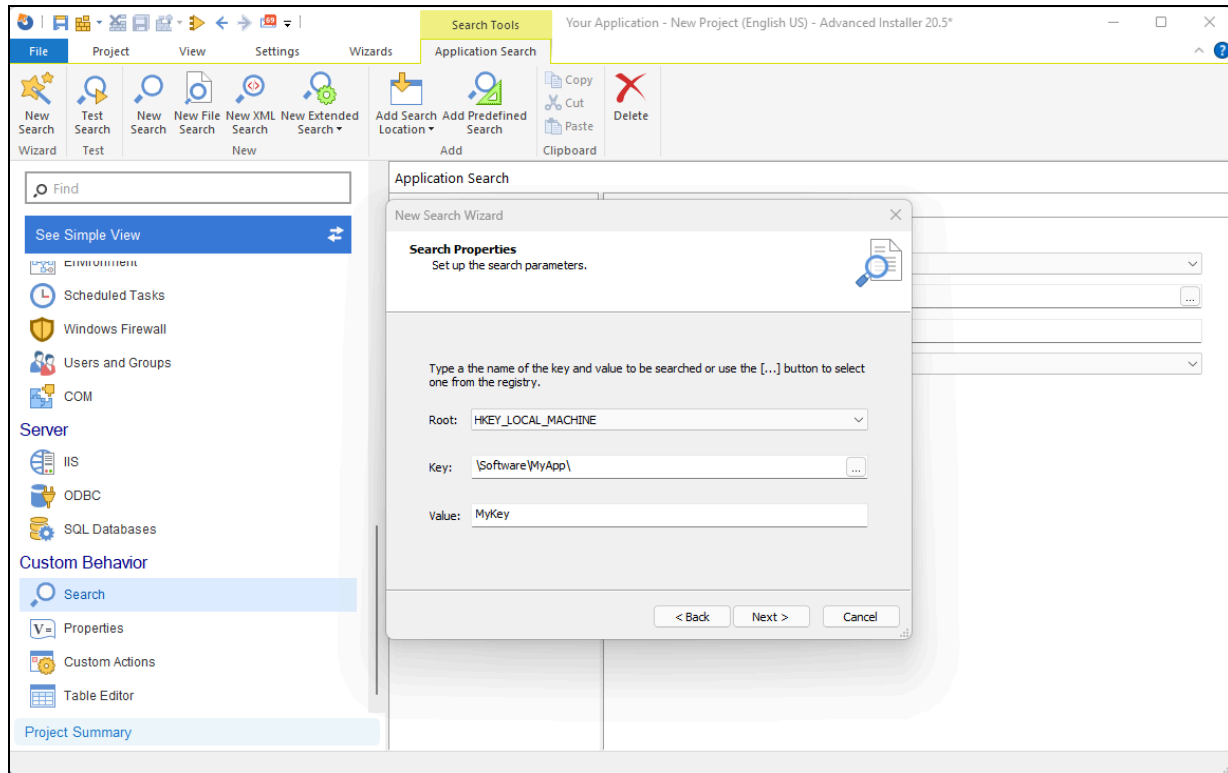
Next, select to Search for a registry value. As mentioned, we will leave the search results in the `RESULT_PROPERTY`.





Next, define the registry key value that we are searching for.

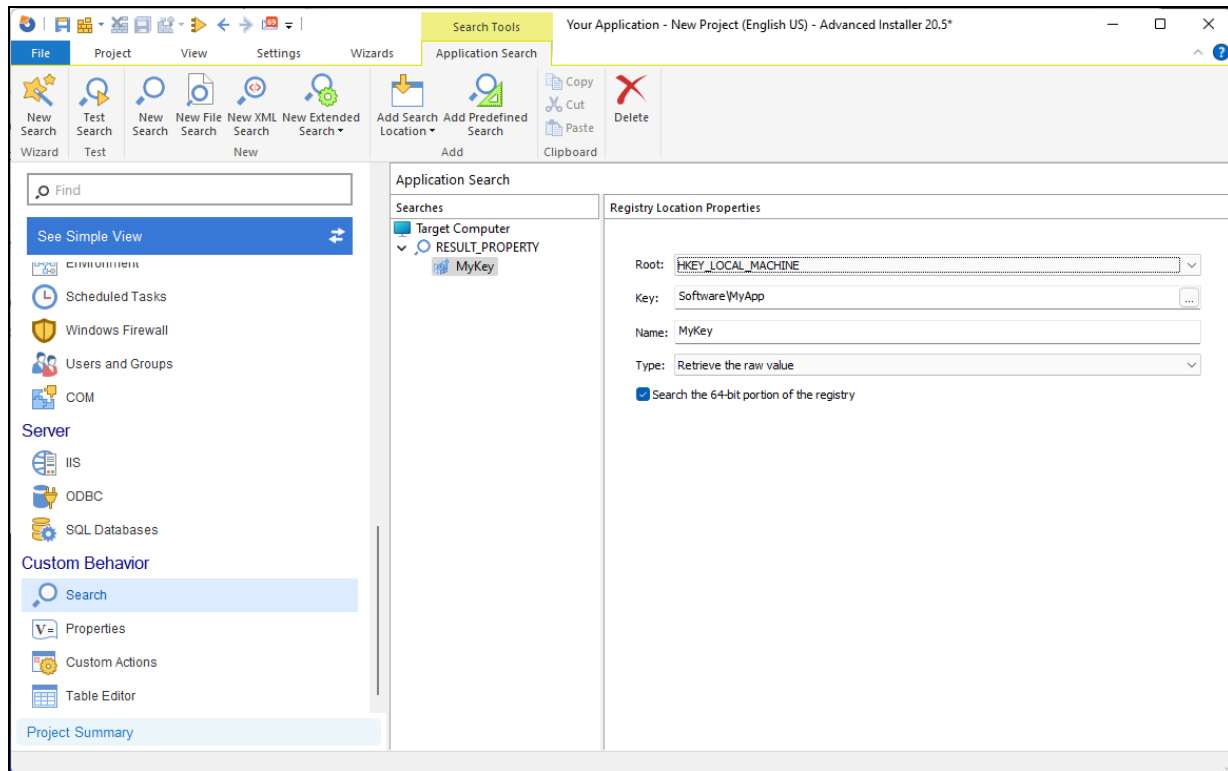




Select to retrieve the value and finish the wizard. After the wizard is completed check if your registry key is in the 64-bit portion of the registry and check the appropriate checkbox. You can also do a Test Search just to confirm that everything is in order.







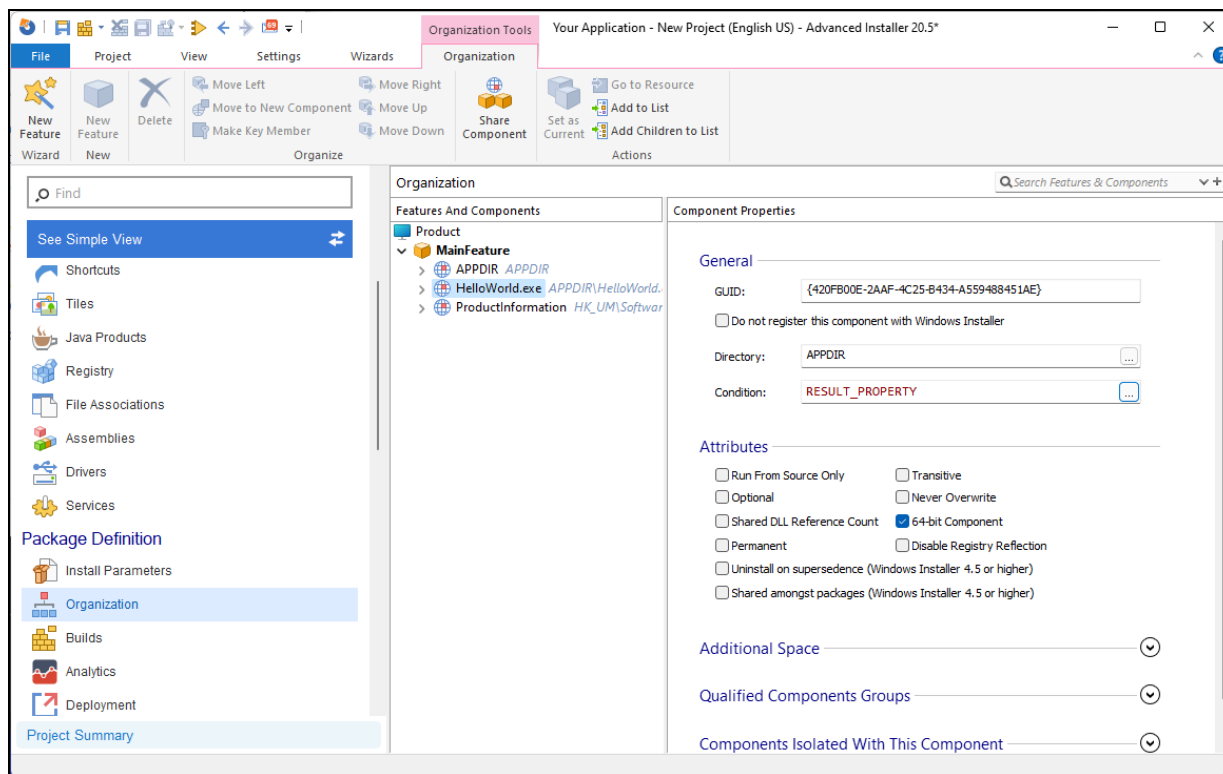
2. Use the above created Property to define the condition on the component where the file is located.

A property has two states:

- It has a value and therefore it exists
- It has no value and therefore it does not exist

Because we don't care about what value the property outputs and we are only interested if it exists or not, defining the condition on the component where the file is located is quite simple. All we have to do is navigate to the Organization page and we set the condition for our HelloWorld.exe component to RESULT\_PROPERTY.





During the install phase, the MSI will first perform the search function to see if the registry key exists and populate the property, afterwards it will parse through all the components which are marked to install. Once it reaches our component with the HelloWorld.exe file, it will first check if the condition is met, meaning if the property is present or not. If the property is present, it means the condition has been passed and the component is then installed on the machine.

## Launch Conditions

In Advanced Installer, configuring a launch condition is a straightforward process. You can create a launch condition to check for the presence of specific system requirements, such as the availability of a specific version of the .NET Framework, or the presence of specific registry keys or files on the system.

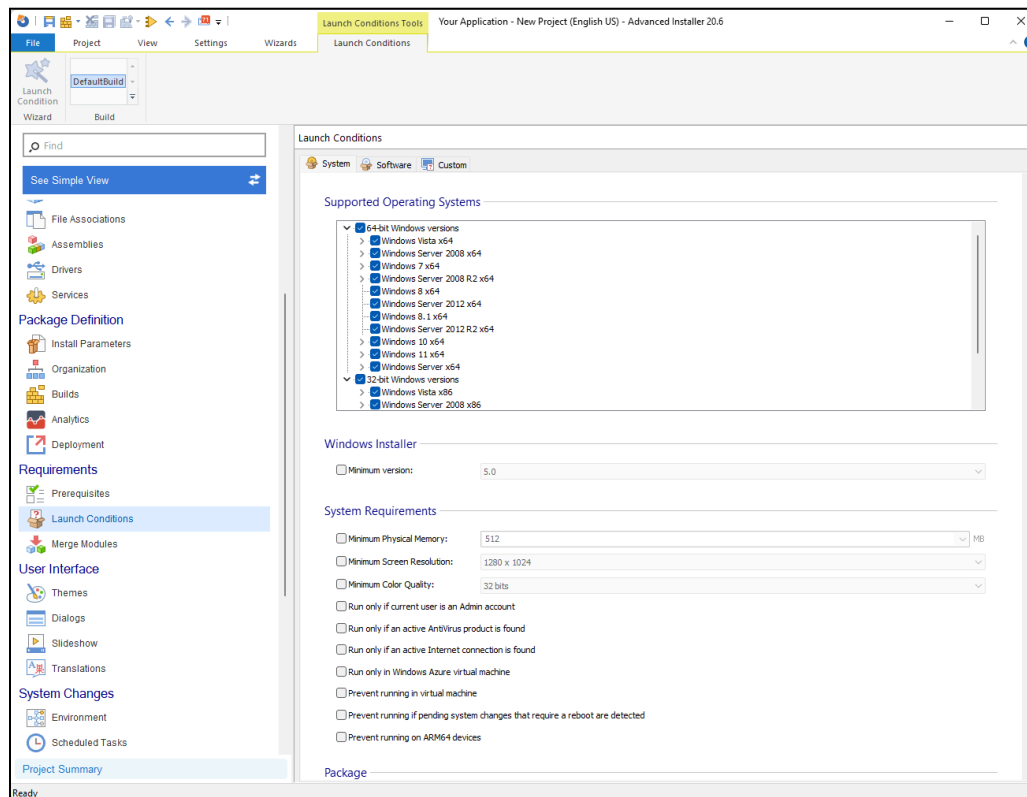
To configure a launch condition in Advanced Installer, you can simply navigate to the [Launch Conditions page](#) and create a new launch condition. Advanced Installer offers by default multiple pre-configured types of launch conditions and we can separate them into:

- System
- Software



## System Launch Conditions

When it comes to system launch conditions, you can easily define on which Operating System your package will be supported, what minimum version of Windows Installer is needed, but also more in-depth checks such as minimum physical memory that the user needs to run the software, minimum screen resolution, minimum color quality and so on. For more details check [out this page](#).



## Software Launch Conditions

Coming back to our scenario where you would need a certain version of .NET Framework installed on the machine, if we navigate to the [Software launch conditions tab](#) we can see that Advanced Installer offers many predefined checks for some of the most popular software products out there such as:

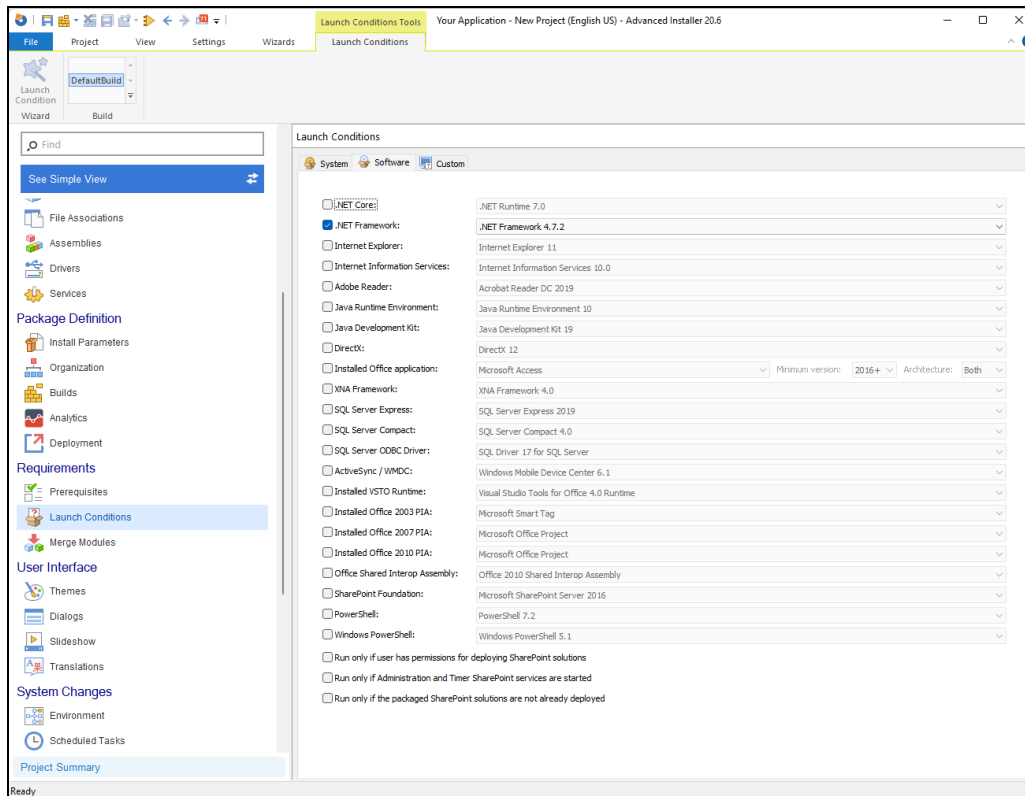
- .NET Core
- .NET Framework
- Internet Explorer
- Internet Information Services (IIS)



- Adobe Reader
- Java Runtime Environment (JRE)
- Java Development Kit (JDK)
- DirectX
- Office applications
- XNA Framework
- SQL Server Express
- SQL Server Compact
- SQL Server ODBC Driver
- ActiveSync / WMDC
- VSTO Runtimes
- Office 2003 PIA
- Office 2007 PIA
- Office 2010 PIA
- Office Shared Interop Assembly
- Sharepoint Foundation
- PowerShell
- Windows PowerShell

In our case, all we need to do is check .NET Framework and select the desired version, for the scope of this example we went with .NET Framework version 4.7.2.



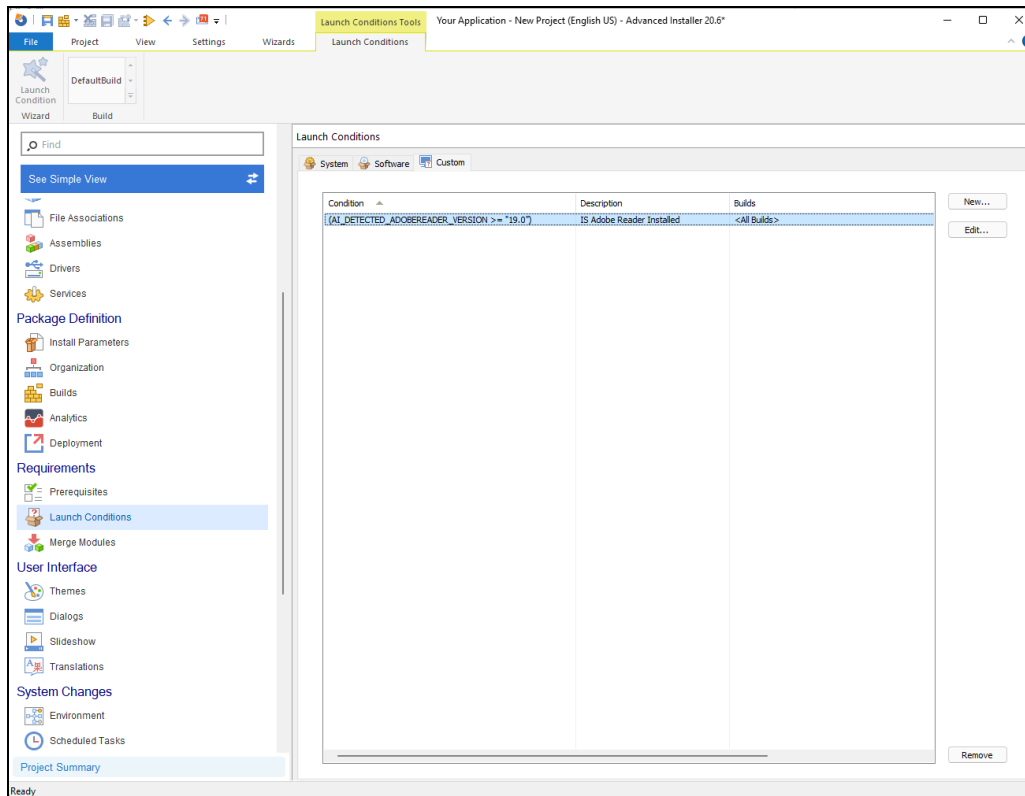


## Custom Launch Conditions

In Advanced Installer, you can define [custom launch conditions](#) by using installer properties or environment variables. These properties can be predefined or set by searches and custom actions.

To create a new launch condition, simply click on the "New" button, the "New" list context menu item, or press the Insert key while the list control is focused. The Edit Launch Condition Dialog will pop up, allowing you to specify a launch condition.





To modify a launch condition, use the "Edit" button, the "Edit" list context menu item, or press the Space key while an element from the list control is selected. The Edit Launch Condition Dialog will pop up, allowing you to edit the launch condition.

If you no longer need a launch condition, you can remove it using the "Remove" button, the "Remove" list context menu item, or by pressing the Delete key while an element from the list control is selected.

Defining custom launch conditions in Advanced Installer is a best practice that can help to ensure the reliability and efficiency of the installation process. By specifying the conditions that must be met before installation, you can reduce the potential for errors and ensure that the installation package is installed only on systems that meet the required specifications.

## Custom Actions as Conditional Statements

One of the key benefits of custom actions is their ability to act as conditional statements, enabling you to create complex installation packages that can adapt to different system configurations.



Custom actions can be used to perform a wide range of tasks during the installation process, including creating registry keys and values, copying files, and executing scripts or other programs. By using custom actions as conditional statements, you can control the behavior of the installation process based on the system configuration.

For example, you can use a custom action to check for the presence of a specific file or registry key on the system, and use that information to conditionally install or execute other custom actions. You can also use custom actions to perform tasks that are specific to a particular system configuration, such as installing a driver or configuring a network adapter.

This can be done using a wide range of syntax and scripting languages, including VBScript, JavaScript, C++ and [even C#](#).

Depending on the approach you want to take with Custom Actions you can either:

- Write the custom action to initialize a variable as we did above and then check during the [custom launch condition](#) if the variable is initialized or not
- Write the custom action to produce an error if the check is not passed. When the custom action is inserted in the Sequence with Advanced Installer make sure that the "Fail installation if custom action returns an error" is checked. The installer is awaiting a return code either with 0 (success) either with 3010 (reboot)

Let's take the second scenario and consider that we want to create a script that checks if a certain registry key exists. If that registry key is missing we will give back the return code 100 which will be interpreted as a failure.

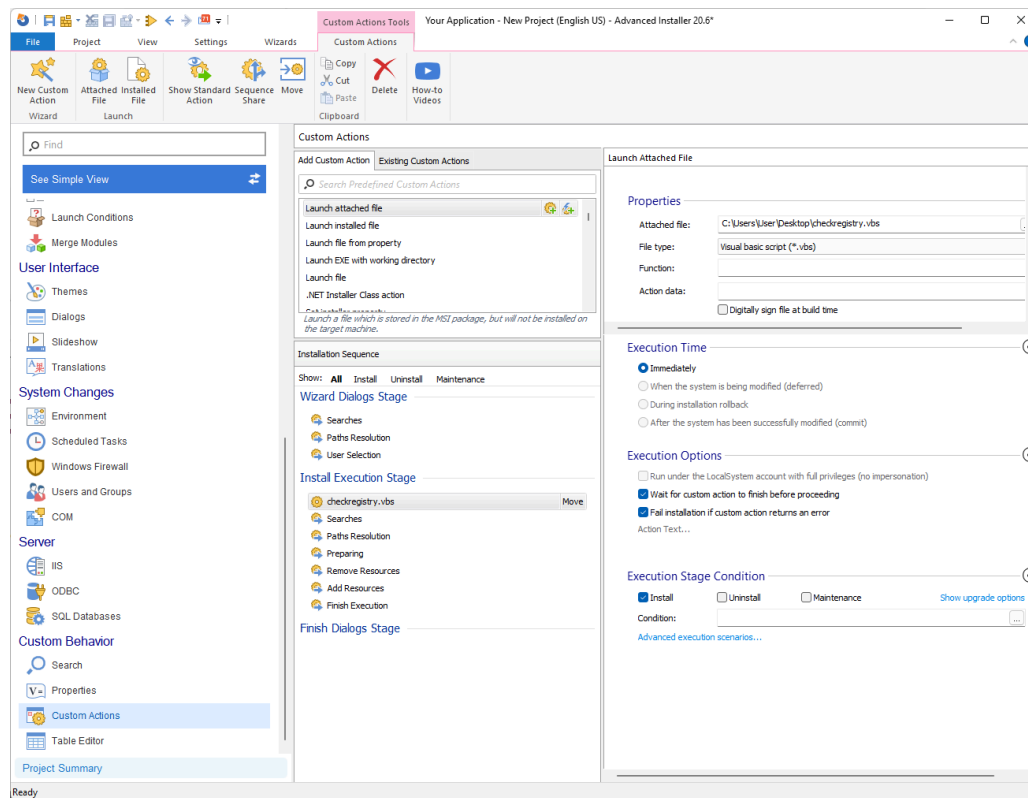
```
Const HKEY_LOCAL_MACHINE = &H80000002
Dim RegKey
Dim objWshShell
Set objWshShell = CreateObject("WScript.shell")
strComputer = "."
Set objRegistry = GetObject("winmgmts:\\." & strComputer &
"\root\default:StdRegProv")
RegKey= "ProgramFilesDir"
strKeyPath = "SOFTWARE\Microsoft\Windows\CurrentVersion"
objRegistry.GetStringValue
HKEY_LOCAL_MACHINE, strKeyPath, RegKey, strValue
'check if the value exists
If IsNull(strValue) Then
    wscript.quit (100)
```



```
End If
```

As you can see above, we are checking with VBScript if the ProgramFilesDir which is located in HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion has any value. If the value is null then we will return the error code 100.

Now that we have the script created, all we need to do is navigate to the Custom Action page and set it up like so:



If you remember our discussion in the [MSI Packaging Fundamentals ebook](#), immediate execution custom actions can be executed before the [InstallInitialize Action](#), and this is exactly what we need because we need to run this script as soon as possible to confirm that the system has the requested parameters.

As you can see in the Install Execution Stage, the custom action is set at the top to be the first CA that is executed, and when it comes to Execution Stage Conditions we are only setting it to run during the Install phase.





# Working with Dependencies

While conditional statements give you the possibility to clearly define the needed environment for your installer, it doesn't let you modify the system in order to make the proper changes for your application to be installed. As seen in the chapter above, conditional statements for software applications can be easily added into the MSI by using Advanced Installer, but what if you want to check if that particular dependency is installed and if not, install it yourself?

This is where the Prerequisites feature comes in handy and lets you quickly create a bundle which contains your application and the desired dependencies.

Prerequisites are software components that must be installed before the installation of an application to ensure its proper functioning. Advanced Installer provides a wide range of prerequisites that can be easily added to MSI packages, including .NET Framework, Visual C++ Redistributable, and SQL Server Express.

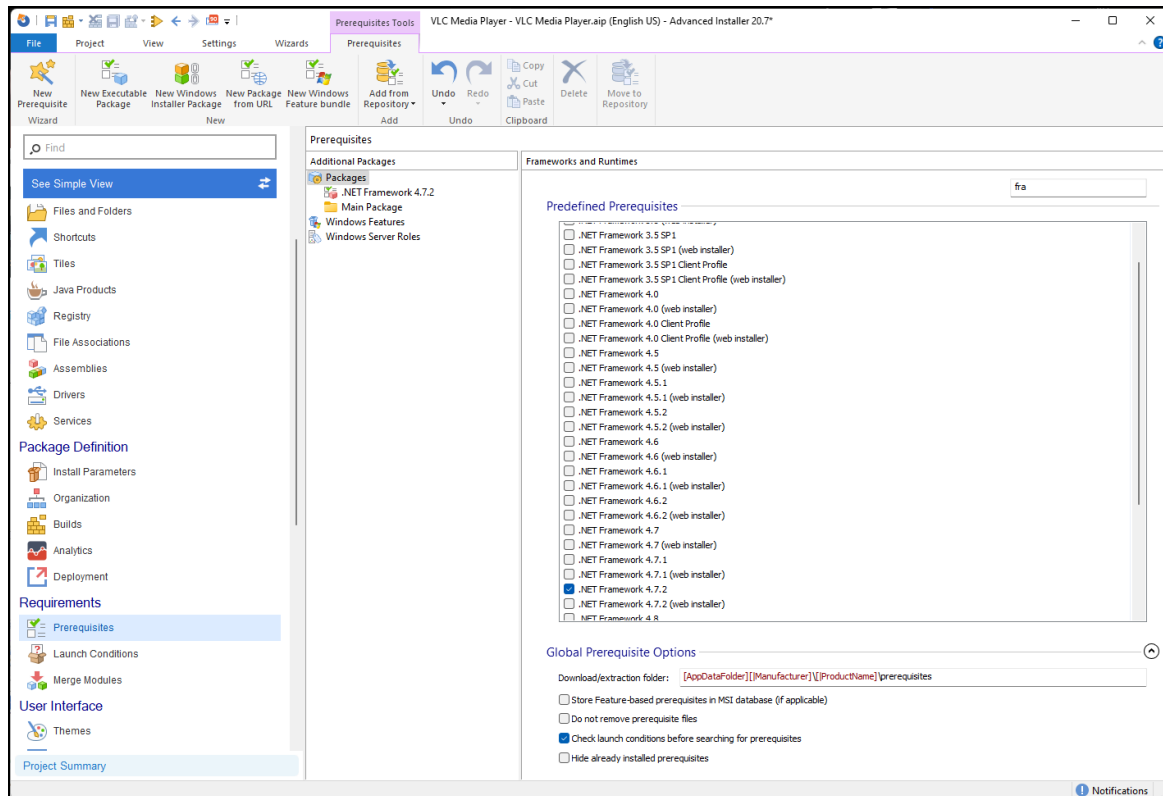
Adding prerequisites to MSI packages is important for several reasons. Firstly, it ensures that the application will run correctly on target systems, reducing the risk of compatibility issues and user frustration. Secondly, it simplifies the deployment process by automating the installation of required software components. Finally, it can improve the performance of the application by ensuring that it has access to the latest software components.

Adding prerequisites with Advanced Installer is a straightforward process that can be done in just a few steps. Here's how to do it:

Open your Advanced Installer project and go to the Prerequisites page. Advanced Installer provides a wide range of options for configuring prerequisites, including the ability to specify a specific version of the prerequisite, the installation path, and the location of prerequisite files.

This view enables you to incorporate existing installers in your package, enable Windows features, and configure specific Windows Server roles.





All prerequisite setup files can be bundled with your package or placed online and accessed via a URL. If the prerequisite is not found during installation, it will be installed automatically.

Advanced Installer can also download and install prerequisites from a remote location, such as a web server or network share. This ensures that the most recent versions of the prerequisite are always used, lowering the possibility of compatibility issues.

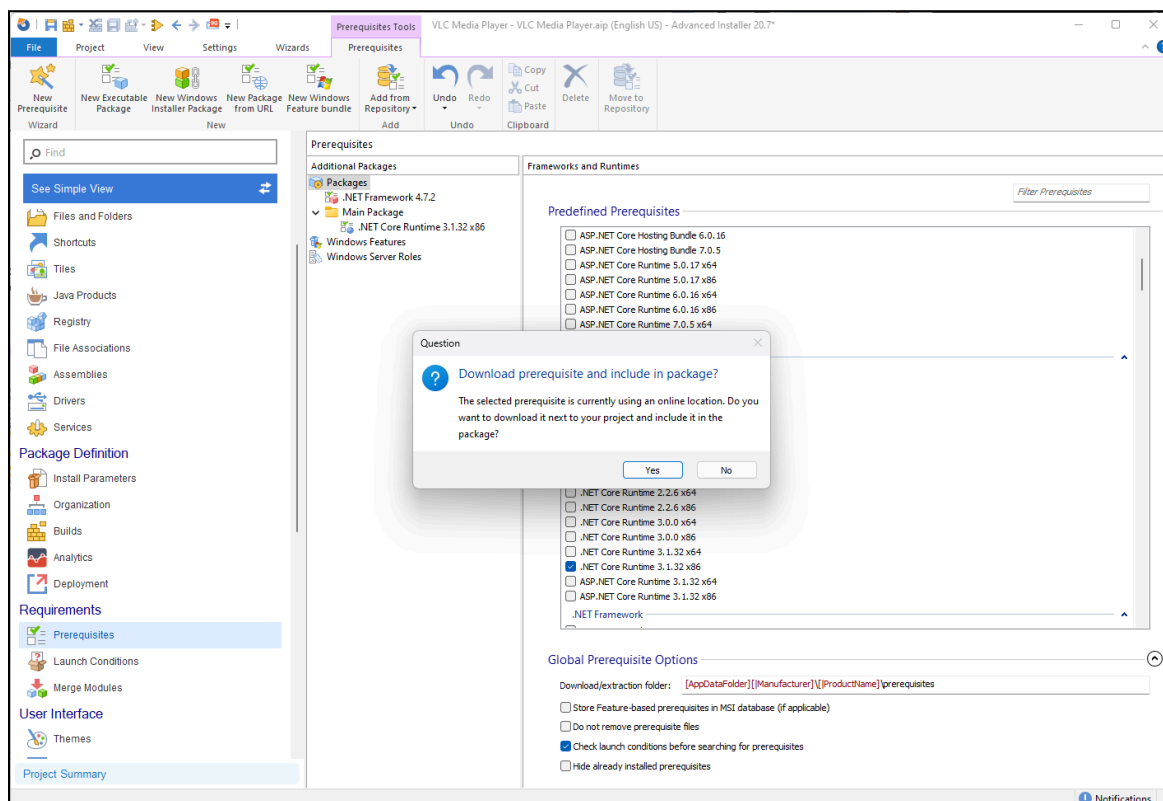
In the main Prerequisites page you can see on the right hand menu that you have a ton of predefined prerequisites that you can choose from. Some of these prerequisites are:

- .NET Framework
- .NET Core
- .NET Runtime
- SQL Server Compact
- SQL Server Express
- MySQL Server
- SQL Server OLE DB
- Adobe products
- JRE
- JDK
- Silverlight
- Python



- Internet Explorer
- DirectX
- XNA
- Access Runtimes
- Visual C++ Redistributables
- VSTO
- IIS
- Apache Tomcat
- MSML
- Windows Installer
- PowerShell
- And many more

When you select a predefined prerequisite, Advanced Installer asks you if you want to download the package next to your project and include it in the package, making the overall process much simpler.

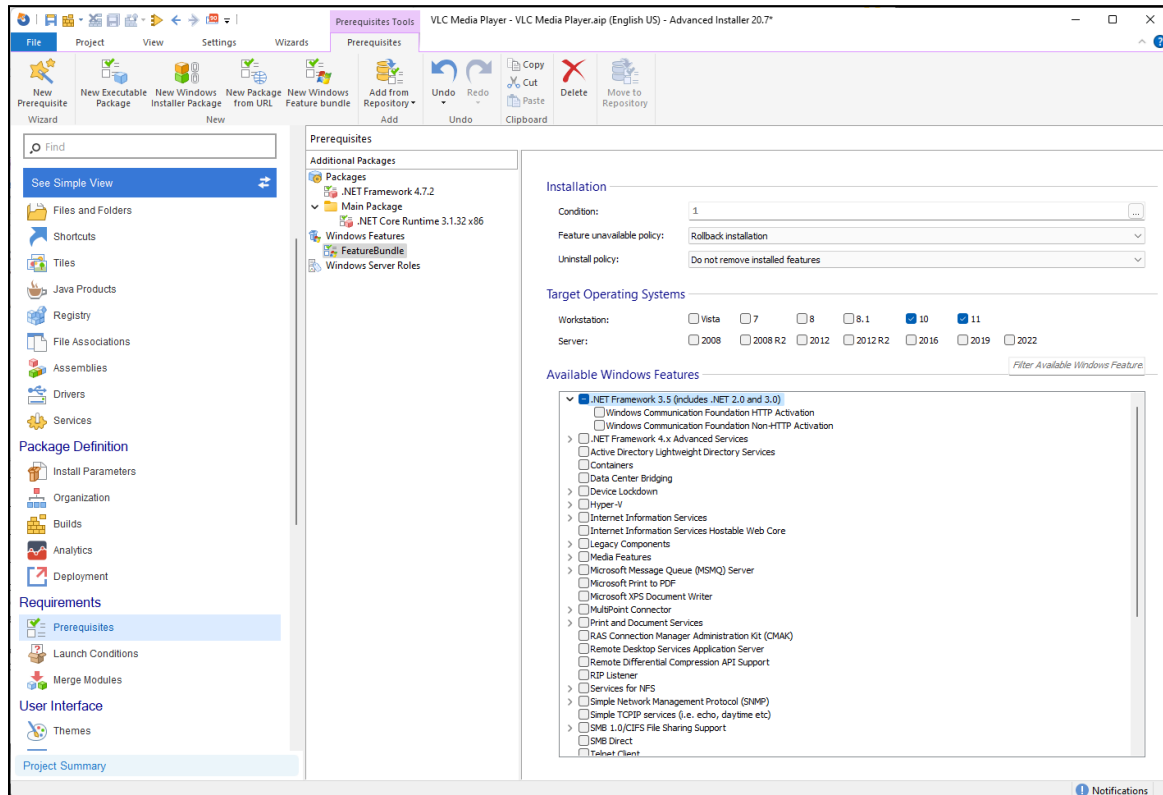


You can also select the extract folder for each selected prerequisite, keep the prerequisite files after installation, hide installed prerequisites and check the launch conditions before searching for the prerequisites.



Advanced Installer also allows you to enable Windows Features. Some Windows programs and features must be enabled before applications can use them. Other features are enabled by default, but you can disable them if your application does not require them.

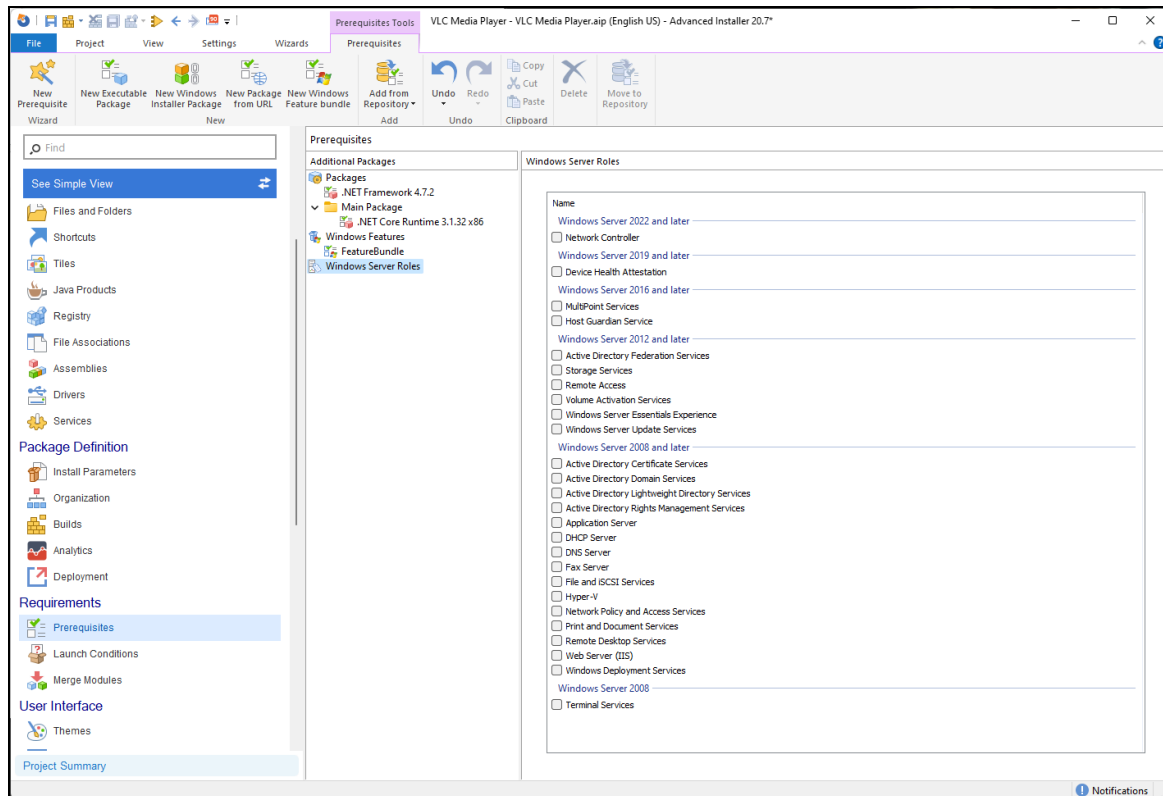
Use the [Windows Feature bundle] toolbar button or the "New Windows Feature bundle" context menu item to add a Windows Features bundle then select the Target Operating System. Once you have chosen the OS, the Available Windows Features will be populated with all the options available for those particular OSes.



Advanced Installer also offers multiple Windows Server roles from which you can choose to include in your package. Roles selected in this view only apply when running the package on a Windows Server. They will be ignored on any other OS type.

To include a server role in your package, select the Windows Server Roles tree item, then check the roles you want to include. The roles are organized by the earliest supported version of the target server's operating system.





Of course, you can add your own packages and create a suite installation, [a chapter we have already covered in the first MSI Packaging fundamentals ebook](#).

## Creating Transform Files

MSI (Microsoft Installer) transform files are an extremely useful tool for customizing and modifying MSI packages without having to directly edit the original package. Transform files, also known as MST files, enable you to modify an existing MSI file by specifying changes such as adding or removing features, changing installation paths, changing registry settings, and more.

Before we go any further, consider the following best practices for ensuring the effectiveness and dependability of your MSI transform files:

- **Understand the MSI File Structure:** Familiarize yourself with the structure and components of an MSI file. This will help you identify the areas you want to modify and ensure that your transform file integrates smoothly with the original package. For this



steps, we really recommend that you have a look over our first [MSI Packaging Essentials free Ebook](#).

- Use a Reliable MSI Editor: Use a reliable MSI editor tool, such as Advanced Installer, to create and edit transform files. Advanced Installer provides a user-friendly interface and powerful features for working with MST files.
- Test Thoroughly: Always test your transform files on different target machines and operating systems to ensure compatibility and proper installation. Testing helps identify any issues or conflicts that may arise due to the modifications made in the transform file.
- Document Changes: Keep detailed documentation of the changes made in the transform file. This will help you track modifications, troubleshoot issues, and maintain a record of the customization process.
- Follow Standard Naming Conventions: Use standard naming conventions for your transform files to ensure consistency and avoid confusion. Consider including version numbers, date stamps, or a meaningful description in the filename.
- Versioning and Upgrades: If you plan to create multiple versions or upgrades of your software, maintain consistency in the naming and structure of your transform files to facilitate smooth upgrades and ensure compatibility between different versions.

## Click-Once Apps

ClickOnce is a Microsoft deployment technology that facilitates deploying Windows applications.

In one of our previous articles, we discussed [How to Replace the ClickOnce app with MSIX](#). But, what if you want to ["repackage"](#) a ClickOnce application? That's what we'll be covering in this article.

If you want to learn more about repackaging, take a look at our [repackaging best practices](#).

### What are the challenges of repackaging a ClickOnce application?

ClickOnce shares some similarities with MSIX – the main one being that they are **per-user applications**.

**Per-user applications** are kind of difficult to repackage. In general, IT Pros are used to building MSI and EXE packages to be [installed per-machine](#). That means that most of the applications



that we find on the market today require administrative rights to be installed. So, most vendors prefer the per-machine method of deploying applications.

This comes as no surprise, since the [per-machine approach](#) makes it much easier to [manage your applications in the infrastructure](#).

One mistake that we see happening very frequently is for IT Pros to convert ClickOnce applications (per-user) to per-machine applications without considering all the issues that could occur during the execution of that particular application.

Never change the installation type of an application unless you fully understand how it works.

Usually, you want an application to save some settings or running information in files. For per-user applications, most vendors consider placing those setting files near the executable.

The problem comes if you try to convert the application to per-machine. Why? Because users in an infrastructure will probably not have the necessary rights to write in a per-machine location.

Some might say that you can always give [additional permissions](#) when you build your package, but that is a security point that needs a deeper discussion.

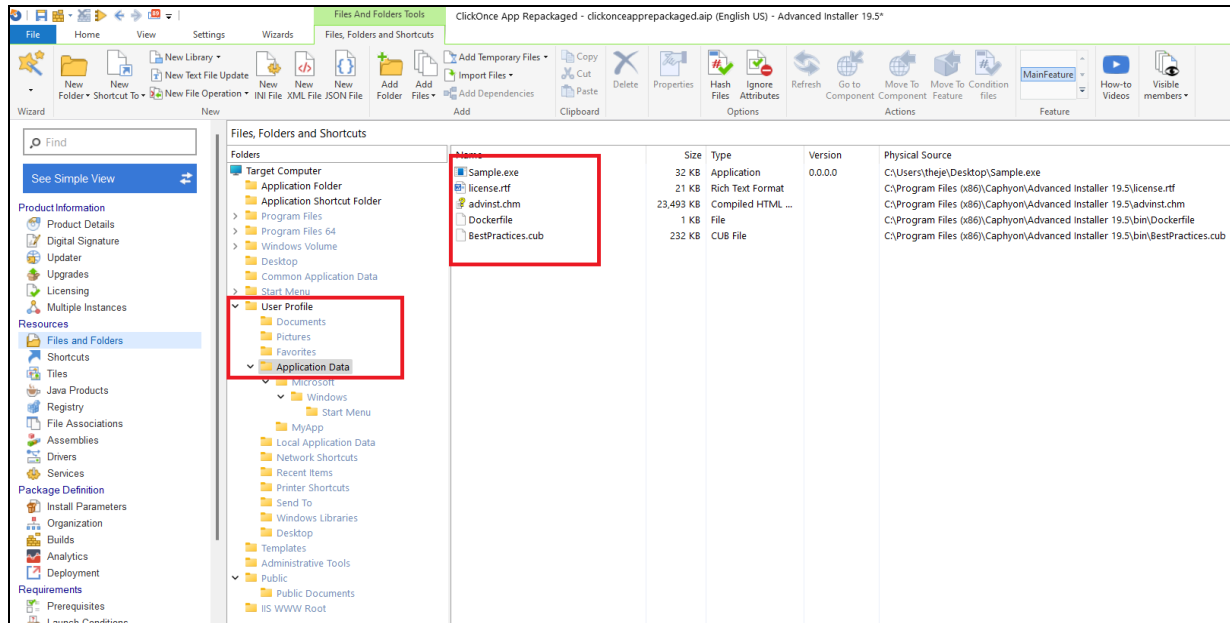
For now, let's consider that additional permissions are not allowed on per-machine installations. Check out the [MSI Permissions Guide: Three Ways to Add Rights With your Installer](#)

## How to Repackage ClickOnce Applications?

As you will see, repackaging a ClickOnce application is not very different from capturing other types of installers.

1. Open [Advanced Repackager](#) and go through the standard process of repackaging an application. A full tutorial can be found [here](#).
2. Once you repackaged your application, your AIP(Advanced Installer Project) should contain all the files in a per-user location:





## Best practices for Per-user applications

There are two main golden rules for per-user applications:

1. Make sure the **self-healing mechanism** is working properly at all times
2. Don't change an **application installation behavior**

Following [the second rule](#), while using **ClickOnce applications**, we must place files exactly where the original installer places them. In our case, all the files are copied directly into **%appdata%**.

But, how do you address this situation when you have a multiple-user setup on a single machine?

When you deploy per-user applications within any infrastructure, it takes time before it reaches the user. It also uses up additional bandwidth for the actual installer to be downloaded into the user cache again. So, what do we do in this case?

We need to place all the files in a per-machine folder (e.g. C:\Program Files\My App) and copy them for each user when he/she logs into the machine with the Active Setup mechanism.

The challenge here is making sure we also follow golden rule number one: making sure that the self-healing mechanism works.

If you want to read more about it, we already [touched on this subject in this article](#).





When it comes to MSI's, if the MSI is not present in the original install location and files need to be copied, then the self-healing process fails. That is one of the reasons we are not leaving the files as they were captured (in %appdata) and we are moving them in a per-machine location (C:\Program Files\My App) and then copying them using a [Custom Action](#).

One other aspect we need to consider is the shortcut. In our example, the shortcut is placed under each user Start Menu folder, located in:

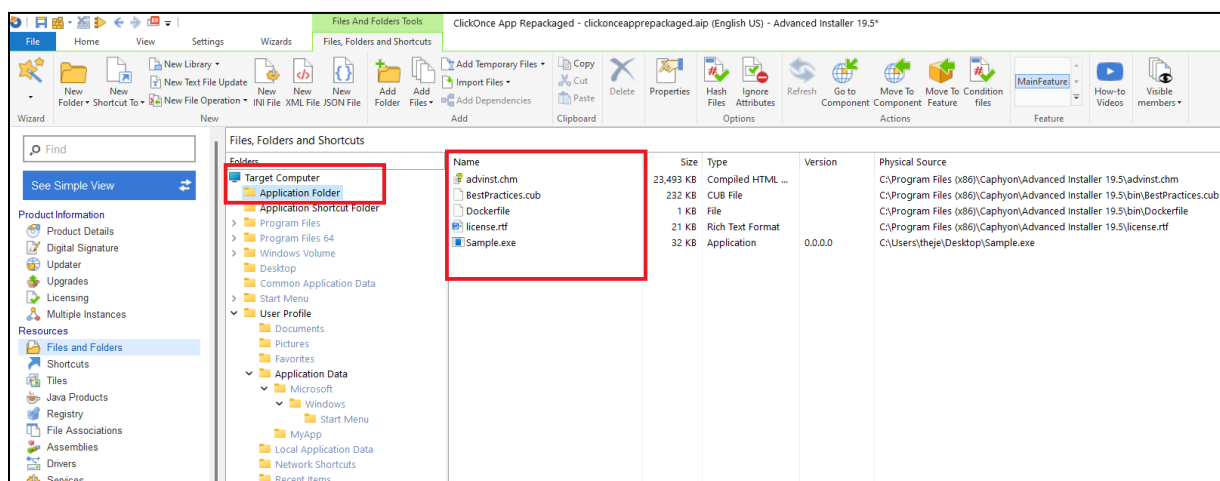
```
C:\Users\YOUR USER\AppData\Roaming\Microsoft\Windows\Start Menu
```

To make it easier, we will place the shortcut on a per-user machine, but the [target of the shortcut](#) will point to %appdata%, meaning it will open the executable from each user location.

## How to adjust the package?

Knowing what we know now, we can adjust the package.

1. Let's first move all the files into the Application folder:



2. Now that the files are placed on a per-machine location (C:\Program Files (x86)\Caphyon\ClickOnce App Repackaged), it's time to add the Custom Action that will copy the files during the [Active Setup](#). To do this, navigate to the [Custom Actions Page](#) and add a new "Launch attached file" action. For this, we are using this simple VBScript:

```
Option Explicit

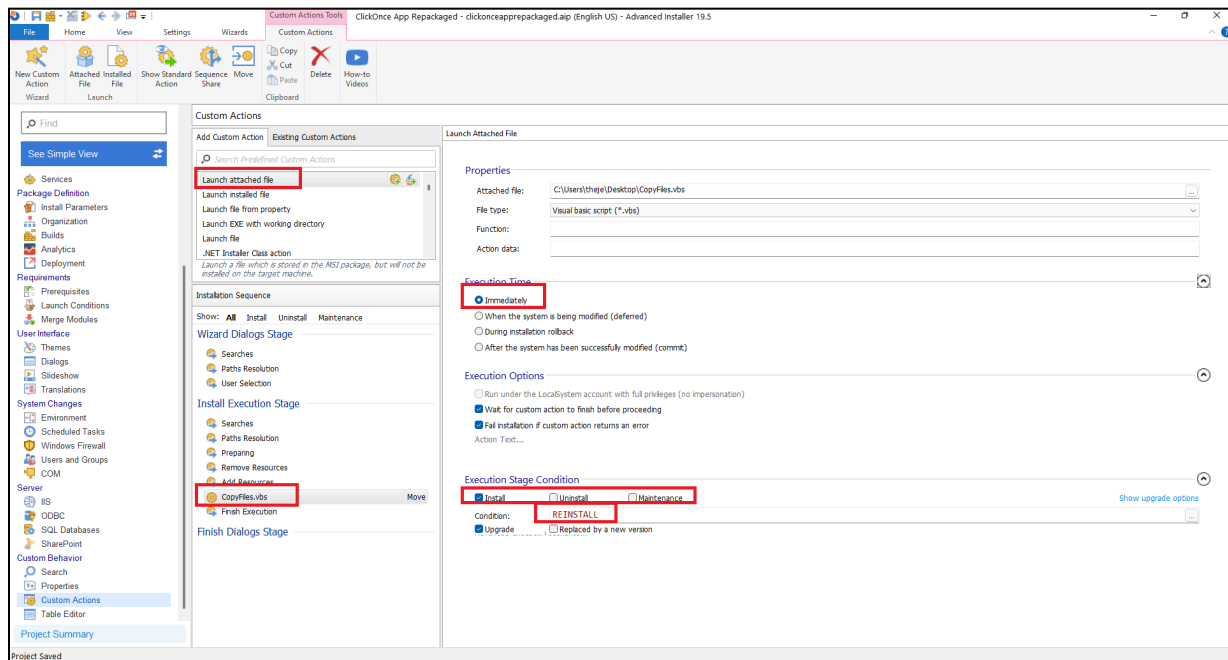
Dim fso, objWShell, appData
```



```
Set fso = CreateObject("Scripting.FileSystemObject")
Set objWShell = WScript.CreateObject("WScript.Shell")
appData = objWShell.expandEnvironmentStrings("%APPDATA%")

fso.CopyFile "C:\Program Files (x86)\Caphyon\ClickOnce App
Repackaged\*.\"", appData + "\"
```

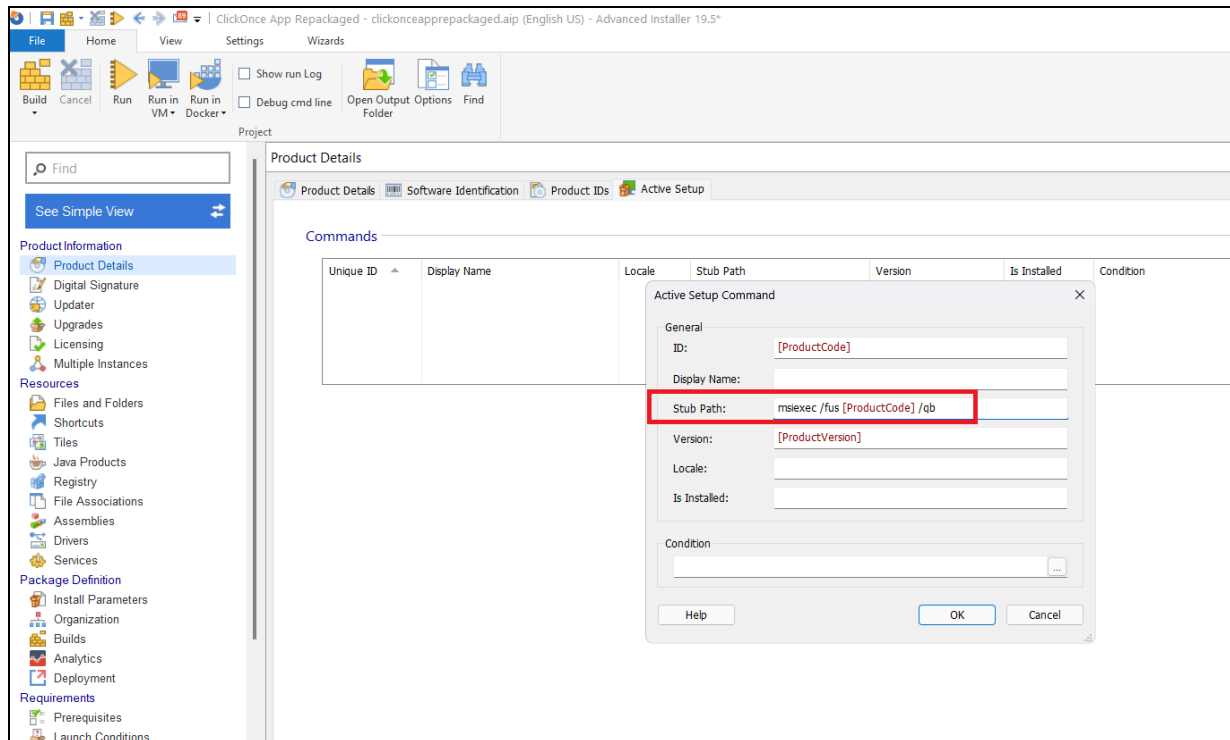
3. Once you have selected the VBScript, configure the rest of the settings, as shown below:



4. The next step is to implement a simple Active Setup into the MSI. In Advanced Installer, navigate to the [Product Details page](#) and click on the Active Setup tab.

5. There, click on **New** and only change the **Stub Path** from /fou to /fus and click **OK**.



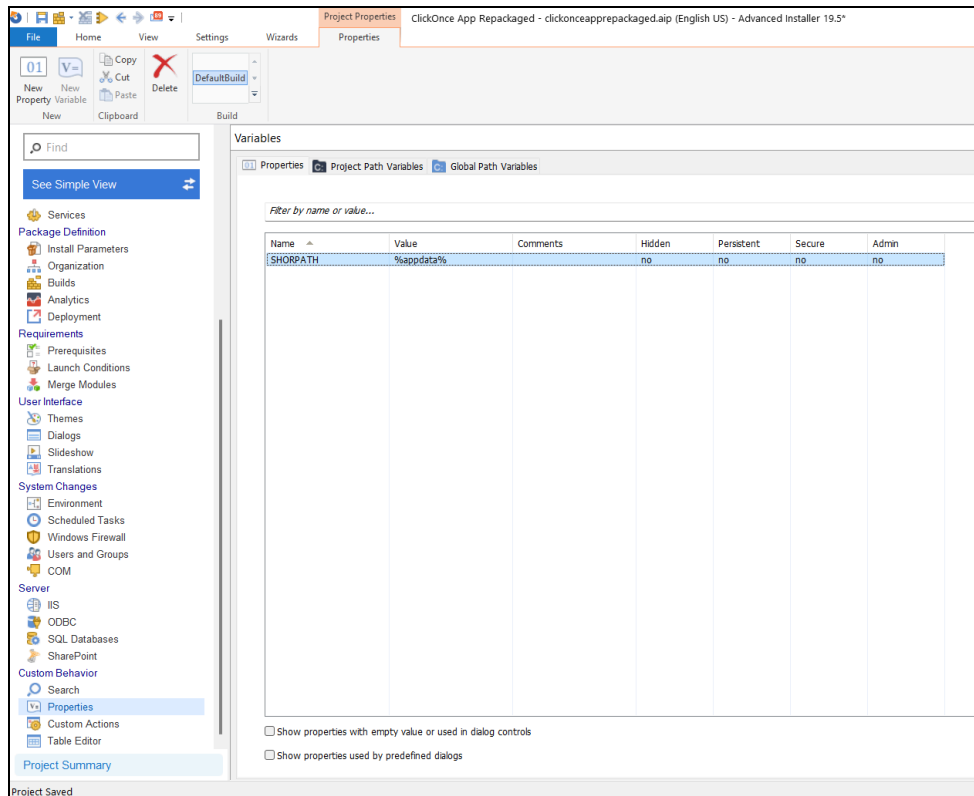


6. Now, the package is installing on a per-machine location and we have an Active Setup which will trigger the Custom Action that is introduced to copy the files to a per-user location once a user logs in.

7. The final step is to adjust the shortcut to point to the correct executable location. This is a two step process.

7.1 First, navigate to the [Properties page](#) and create a new property that looks something like this:





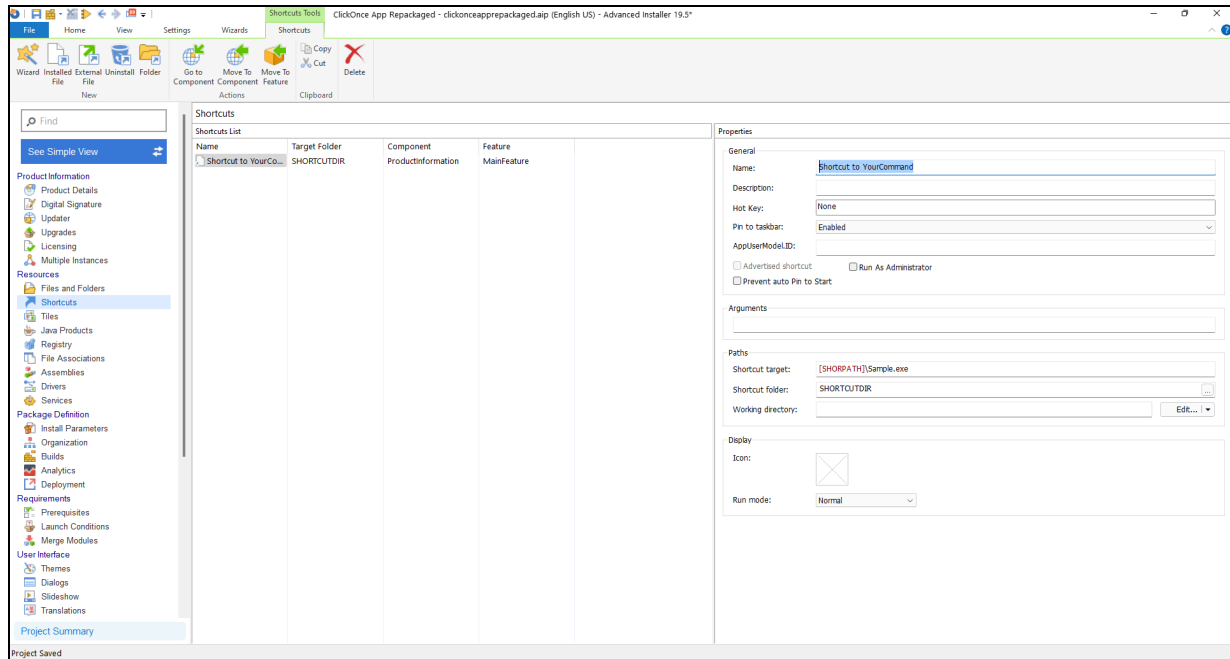
7.2 Once we have the property created, navigate to the [Shortcuts Page](#) and create a new “External File” shortcut. The most important part here is to define the **Shortcut Targetdir** to point to:

```
[SHORPATH]\Sample.exe
```

SHORPATH is the property we previously created, replace it with the property name you have created in your project.

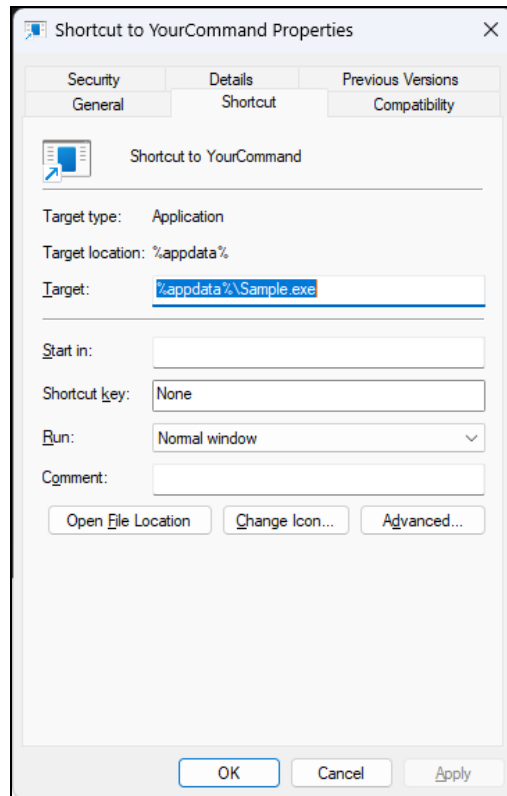
8. At the end, the shortcut should look something like this:





As you can see, the shortcut is placed on a per-machine location, but the target of the shortcut will point at the end to %appdata%\Sample.exe:





# Working with Services

## Introduction to Services

Services play a critical role in the installation and management of software applications. They are independent background processes that provide specific functionality or perform system tasks. Services are components that can be installed, configured, and managed as part of an application installation in the context of Windows Installer (MSI).

## Understanding Services in MSI

Services are represented as components in MSI, which include files, registry entries, and configuration settings required for the service to function properly. The MSI package defines these components, which can be installed, started, stopped, and managed using the Windows Service Control Manager (SCM).



## Benefits of Using Services in MSI

Including services in your application installation provides several advantages:

- **Scalability:** Services can be designed to handle heavy workloads and manage system resources efficiently. They can operate in the background, ensuring continuous operation even when the user is not logged in.
- **Flexibility:** Services enable applications to perform tasks autonomously by providing a flexible architecture. They can be set to start automatically with the operating system or to be triggered by specific events, allowing for a more seamless user experience.
- **Centralized Management:** The SCM can centrally manage services, allowing users or administrators to start, stop, pause, and resume them as needed. This centralized management makes administration and troubleshooting easier.
- **Integration with System Tools:** Services can be integrated with system tools and utilities to interact with other services and respond to system events. This integration improves your application's overall functionality and interoperability.
- **Security:** Services can be configured with specific user accounts and security permissions to ensure that they run with the appropriate level of access and follow best security practices. This aids in the protection of sensitive data and system resources.

## Creating and Configuring Services in MSI

You must define the necessary components, files, registry entries, and configuration settings to create and configure services within an MSI package. The following are the key steps:

- **Component Definition:** Create a component that contains the service files, dependencies, and any additional resources that the service requires.
- **Service Installation:** Specify the details of the service installation, such as the display name, description, startup type (automatic, manual, or disabled), and dependencies on other services, if any.
- **Service Control:** Define the installation actions, such as starting or stopping the service, as well as the error control settings and recovery options.



- **Service Properties:** Set the service's additional properties, such as the service account, password, and other security-related settings.
- **Service Customization:** Customize the service's behavior by providing parameters, command-line arguments, or any other configuration needed for the service to function properly.

The MSI package installs and configures the services defined in the package using the Service Control Manager (SCM). The installer communicates with the SCM to create service entries, configure dependencies, and set the appropriate startup type.

The installer ensures that the services are properly managed during maintenance operations such as repair, modification, or uninstallation. This includes starting, stopping, and updating the service configuration as needed to maintain the installation's integrity.

Advanced Installer tools provide additional functionality to improve service management in MSI installations. These are some examples:

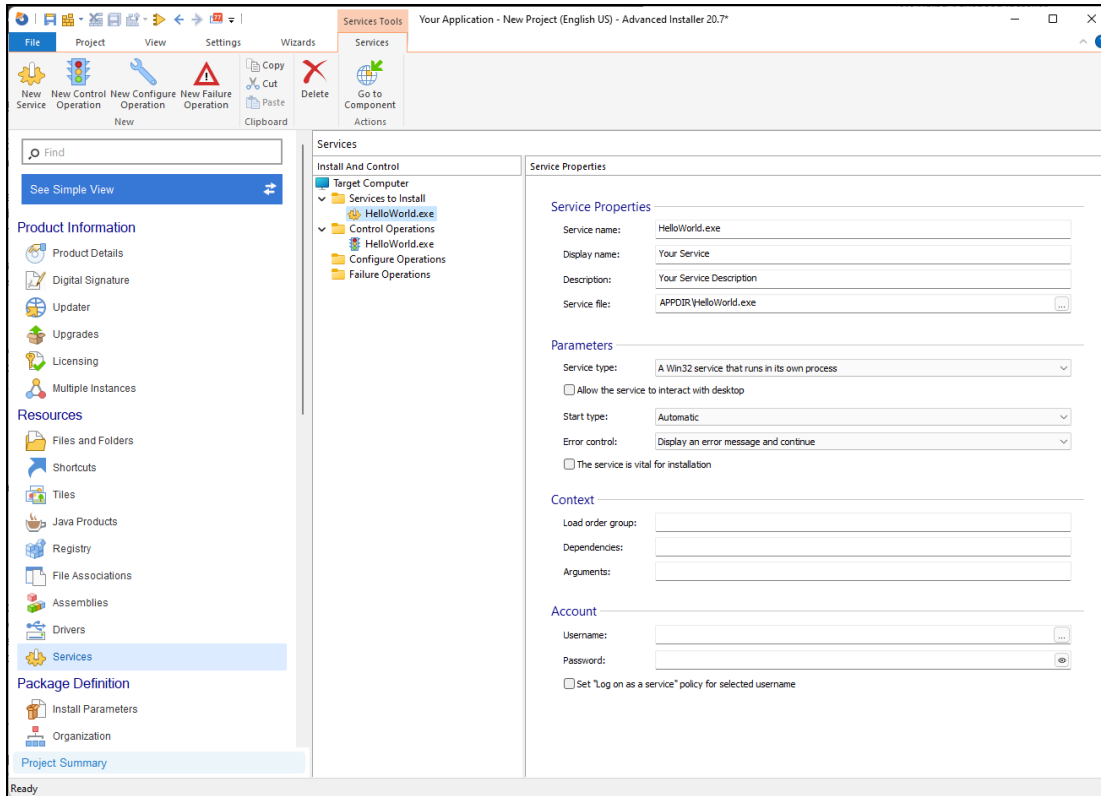
- **Service Start Conditions:** Specify when the service should start, such as after the system boots, when a specific event occurs, or when a specific file is modified.
- **Service Failure Actions:** Define what to do if the service fails to start or stops unexpectedly. Restarting the service, running a specific program, or sending alert notifications are examples of these actions.
- **Service Monitoring:** In the event of service failures or issues, monitor the status of installed services and provide feedback or notifications to users or administrators.

## Creating and configuring services with Advanced Installer

In Advanced Installer, you have full control over the installation, configuration, and management of Windows native services using Advanced Installer's intuitive interface.







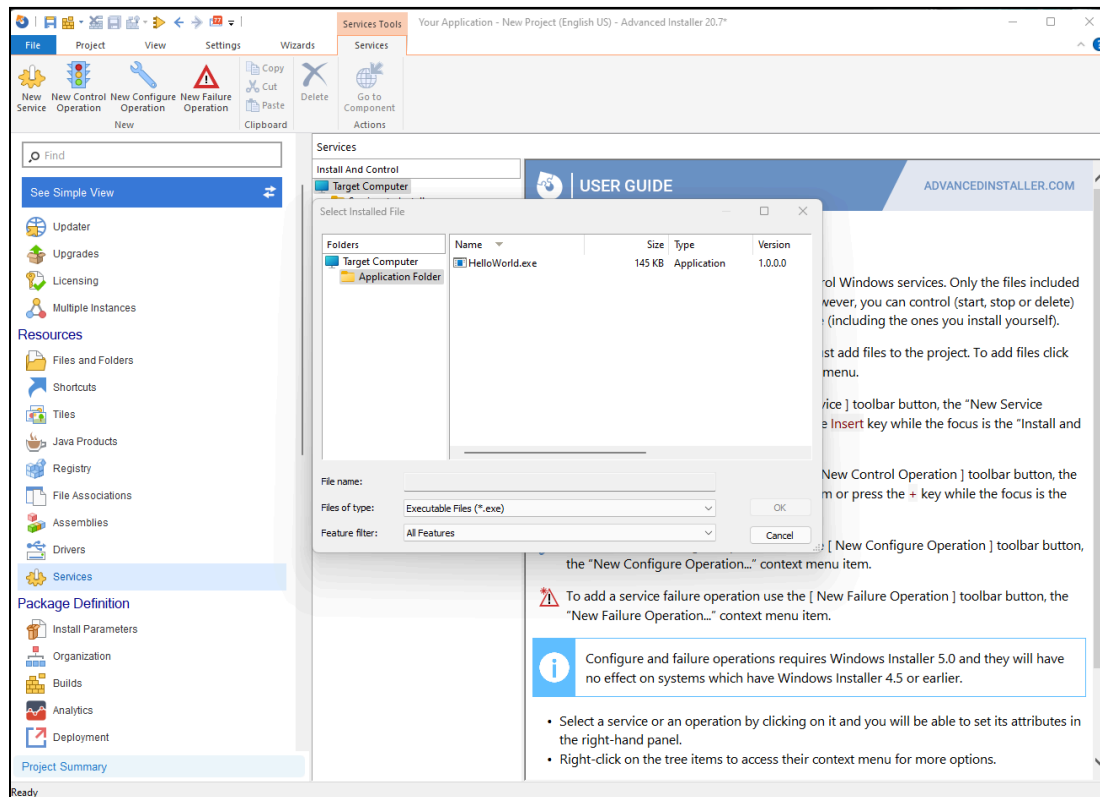
Advanced Installer's service management interface includes a comprehensive set of features for managing services. The interface is split into four sections:

- **Services to Install:** This branch contains a list of all the services that your package will install. You can specify the services that will be installed and their properties.
- **Control Operations:** Control actions for services during installation or uninstallation can be defined here. This includes launching, stopping, and deleting services as needed.
- **Configure Operations:** During installation or uninstallation, you can configure service-specific operations in this branch. You can specify custom actions or configurations that must be carried out for specific services..
- **Failure Operations:** This branch allows you to configure service failure actions. When a service fails or encounters errors, you can specify what actions should be taken.

## Service Installation



To add a new service for installation, you can use the "New Service" option. This can be accessed through the toolbar button, context menu item, or by pressing the Insert key while the "Install and Control" panel is focused. A file picker dialog will appear, allowing you to select the file that contains the service.



When configuring a service in Advanced Installer, you can fine-tune its behavior and characteristics using a range of properties. Here are the key properties you can set:

- **Service Name:** This property specifies the service's unique identifier within the Windows API functions.
- **Display Name:** The display name is the friendly name that users will see. It can be translated into multiple languages.
- **Description:** You can provide a detailed description of the service in the description field. It, like the display name, can be localized.
- **Service File:** This property specifies the service's source file. You can navigate through your project files and choose the appropriate file.
- **Service Type:** You can run a Win32 service in its own process or a Win32 service that shares a process.



- Allow the service to interact with the desktop: This option specifies whether the service may interact with the user by displaying a user interface. It should be noted that this option is not recommended for services installed on Windows Vista or later.
- Start Type: You can configure the service to start automatically when the operating system boots, start on demand when the user initiates it, or disable it entirely.
- Error Control: This property defines the system's behavior when the service fails to start.
- The service is vital for installation: If this option is selected, the package installation will be aborted if the service fails to install.
- Load Order Group: If the service belongs to a specific group, you can specify the group's name. Otherwise, leave this field empty.
- Dependencies: In this field, you can specify any active applications or services that need to be running before this service starts. Use [~] as a separator for multiple dependencies.
- Arguments: This property allows you to pass command line arguments to the service when it starts.
- Account: You can specify the user account under which the service will run. If left empty, the service will run under the LocalSystem account. Use the specified format (<Domain\_Name><User\_name>) for user accounts, and use a dot (.) as the domain name for local user accounts.

Services which interact with the desktop can use only the **LocalSystem** account.

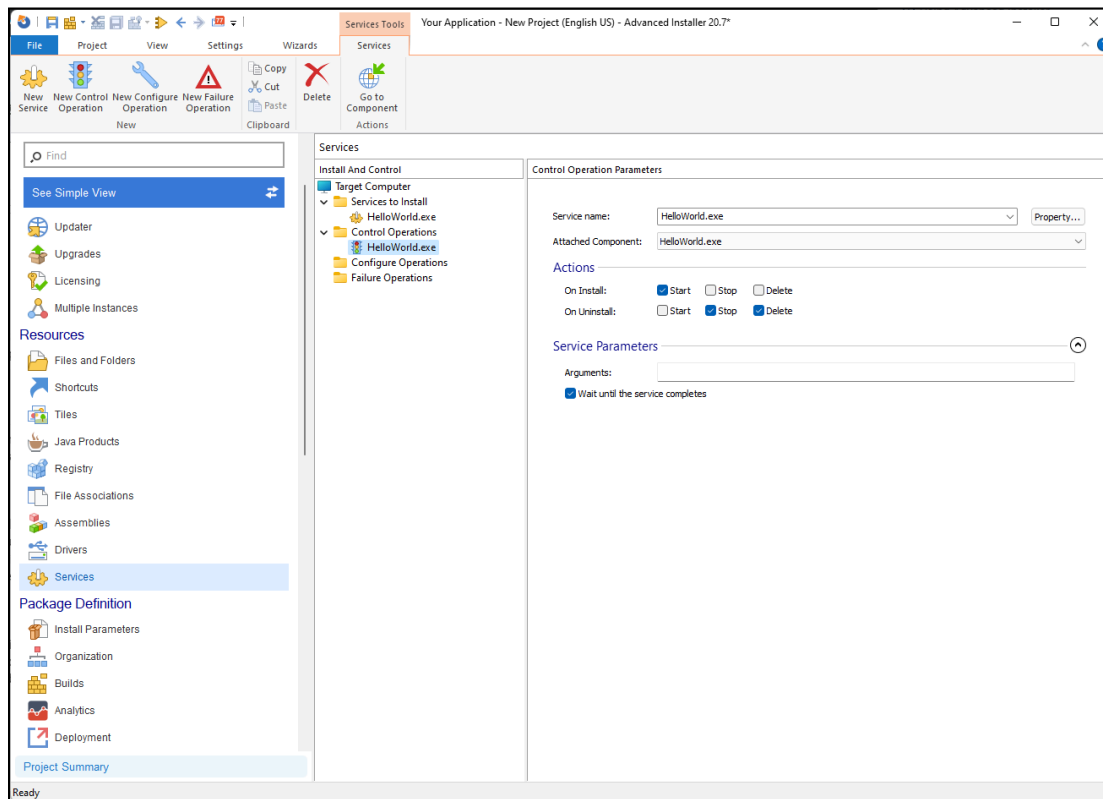
- Password: If applicable, you can provide the password for the service user account. Note that the LocalSystem account does not require a password.
- Set "Log on as a service" policy: Enabling this option sets the "Log on as a service" policy for the specified user account.

If you want to install a service for a specific user, you must take specific steps. These steps are detailed in the [How-To Install a Service for a Custom User](#).

## Control and Configure Operations

Service control operations allow you to have full control over the behavior of services after they are installed using Advanced Installer. With these operations, you can start, stop, or delete services based on your installation package's requirements.





Advanced Installer's service control operation section includes the following properties:

- **Service Name:** This field contains the name of the service you wish to manage. You can either select a service from the combo box or type the name of a service that is already installed on the target machine. The service name and installer properties can both be localized.
- **Attached Component:** You can specify the component that will house the control operation here. This enables you to link the operation to a specific component in your package. Simply select the desired component from the drop-down list box that contains all of the components in your package.

After the service is installed, you have three available options:

- **Start:** The service will be started automatically.
- **Stop:** The service will be stopped.
- **Delete:** The service will be removed.



If all three options are selected, the existing service will be stopped and removed, and the service in the package will be installed and started.

During the uninstallation of your package, you can specify the behavior for the service:

- Start: The service will be started.
- Stop: The service will be stopped.
- Delete: The service will be removed.

For uninstallation, you can only start a service that will still be present on the target machine after the uninstallation is complete.

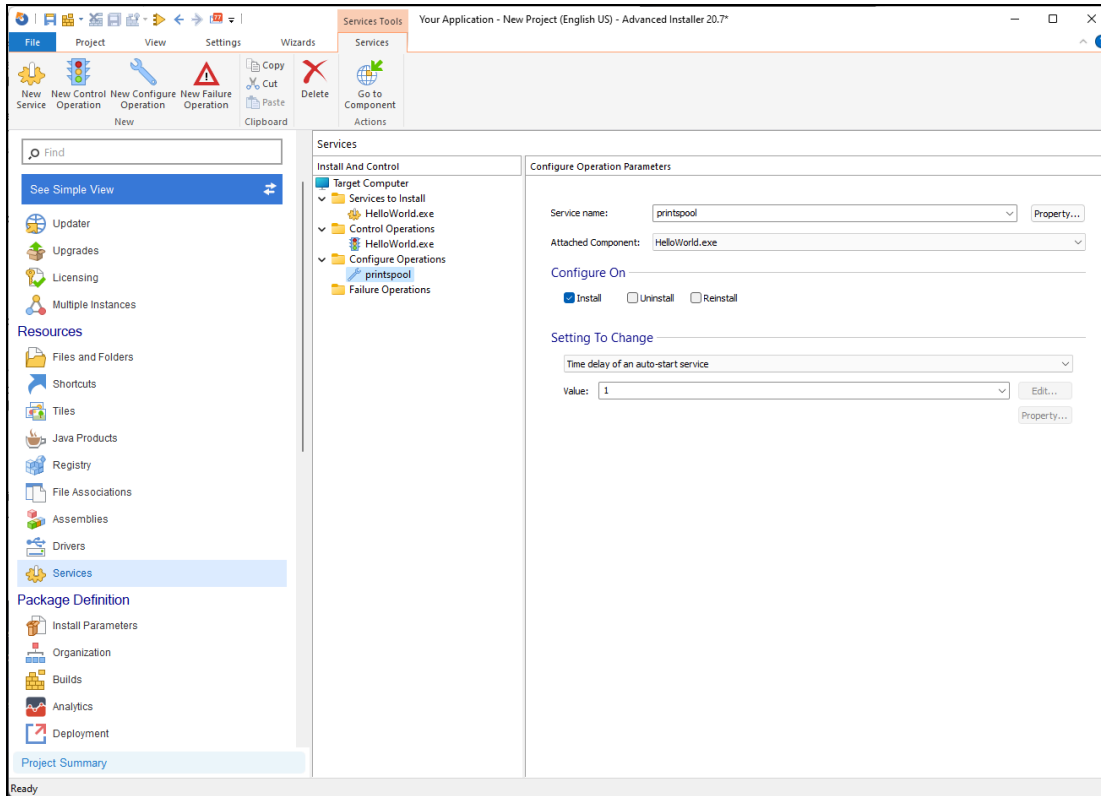
In the service parameters section, you can provide a list of arguments separated by [~] that will be executed when the service starts. These arguments can be localized, and the field accepts formatted types, allowing you to perform advanced editing with installer properties and smart edit control.

Enabling the “Wait until the Service Completes” option instructs the installer to wait for a maximum of 30 seconds for the service to complete before proceeding. This is useful when critical events must be completed before proceeding with the installation. If this option is disabled, the installer will not proceed until the Service Control Manager (SCM) reports that the service is in a pending state.

Several attributes for Merge Module projects can be made configurable at merge time. These are the name, the events, the argument, and the wait. This adaptability enables you to tailor the behavior of service control operations during the merge process.

Also, service configure operations in Advanced Installer allow you to modify the settings of services that are already installed or being installed by your current package. With these operations, you can make changes to service configurations to ensure they meet your application's specific requirements.





The service configure operation section in Advanced Installer provides the following properties:

- **Service Name:** This field contains the service name that you want to configure. You can either select a service from the combo box or type the name of a service that is already installed on the target machine. The service name and installer properties can both be localized.
- **Attached Component:** You can specify which component will contain the configure operation here. You have fine-grained control over when and how the configuration changes are applied by associating the operation with a specific component within your package. Simply select the desired component from the drop-down list box that contains all of the components in your package.

The "Configure On" property determines when the service configuration changes should be applied. You can select one or more options from the following:

- **Install:** Configure the service during the installation of the component.
- **Uninstall:** Configure the service during the uninstallation of the component.
- **Reinstall:** Configure the service during the reinstallation of the component.

By combining these options, you can precisely control when the service configuration changes are applied.



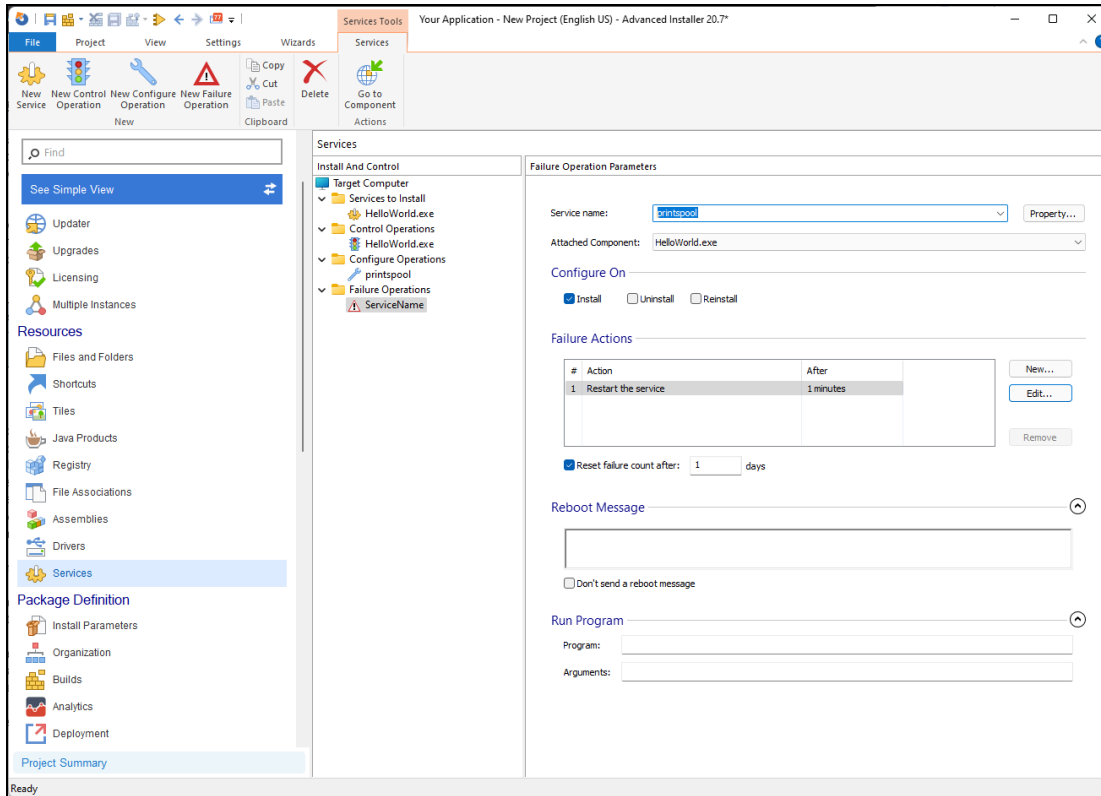
In the "Setting To Change" section, you can specify the changes to be made to the service configuration. You can dynamically configure the service by setting specific values or using installer properties. You can easily select a property to assign the desired value by clicking the "Property..." button..

**"Time delay of an auto-start service"** is applied only on auto-start services or services installed by the package configured with Automatic Start Type from the "Parameters" field in the Service [Properties](#) page.

## Failure Operations

Failure actions are an essential aspect of managing services, ensuring that they respond appropriately in case of failures. Advanced Installer provides a comprehensive set of tools to configure failure actions for services, whether they are already installed or being installed by your current package. The "New Service Failure Operation" option allows you to define failure actions for services. This feature enables you to specify actions to be taken when a service fails to start or encounters errors. You can define custom actions, such as restarting the service or sending error notifications, to ensure smooth operation and prompt error handling.





Advanced Installer's service failure operation section includes the following properties:

- **Service Name:** This field contains the name of the service for which the failure actions are to be configured. You can either choose a service from the combo box or manually enter the name of a service that is already installed on the target machine. The service name and installer properties can both be localized.
- **Attached Component:** You can specify which component will contain the failure operation here. You have fine-grained control over when the failure actions are applied by associating the operation with a specific component within your package. Simply select the desired component from the drop-down list box that contains all of the components in your package.

The "Configure On" property determines when the service failure actions should be configured. You can select one or more options from the following:

- **Install:** Configure the failure actions during the installation of the component.
- **Uninstall:** Configure the failure actions during the uninstallation of the component.
- **Reinstall:** Configure the failure actions during the reinstallation of the component.

By combining these options, you can precisely control when the failure actions are applied.





In the "Failure Actions" section, you can specify the actions to be taken if the service fails. The following properties are available:

- **Reset failure count after:** The Service Control Manager (SCM) keeps track of how many times each service has failed since the system began. When the specified number of times the service fails, the system executes the action with the corresponding index in the "Failure Actions" list. You can specify a time (in minutes) when the failure count should be reset.

You can specify a message to be sent to network users before restarting the computer in response to a "Restart the computer" action specified in the "Failure Actions" list in the "Reboot Message" section. You have the option of sending a reboot message or deleting the current message and sending no message.

You can specify a program to run in response to a "Run a program" action specified in the "Failure Actions" list in the "Run Program" section. When the service fails, you can use a formatted string to delete the current command and run no program. You can leave this field blank to continue using the current run program.

Any changes made to the failure actions will take effect the next time the system is started.

## Service Example

Apache Tomcat is a popular open-source web server and servlet container for running Java-based web applications. Apache Tomcat is typically distributed as a set of executable files that must be manually installed and configured. However, you can repackage Apache Tomcat into an MSI installer using advanced packaging tools such as Advanced Installer, which simplifies the installation process and adds new features.

You can create an MSI package that includes the necessary files, configurations, and scripts to install and configure Apache Tomcat as a service on a Windows system by repackaging Apache Tomcat with Advanced Installer. This enables users to easily install and manage Apache Tomcat without requiring manual setup or configuration.

Once the repackaged MSI installer has been built, users can run it to begin the installation process. The MSI package will extract the required files and place them in the appropriate locations on the target system during installation. It will also create the necessary registry entries and configurations for Apache Tomcat to run as a service.

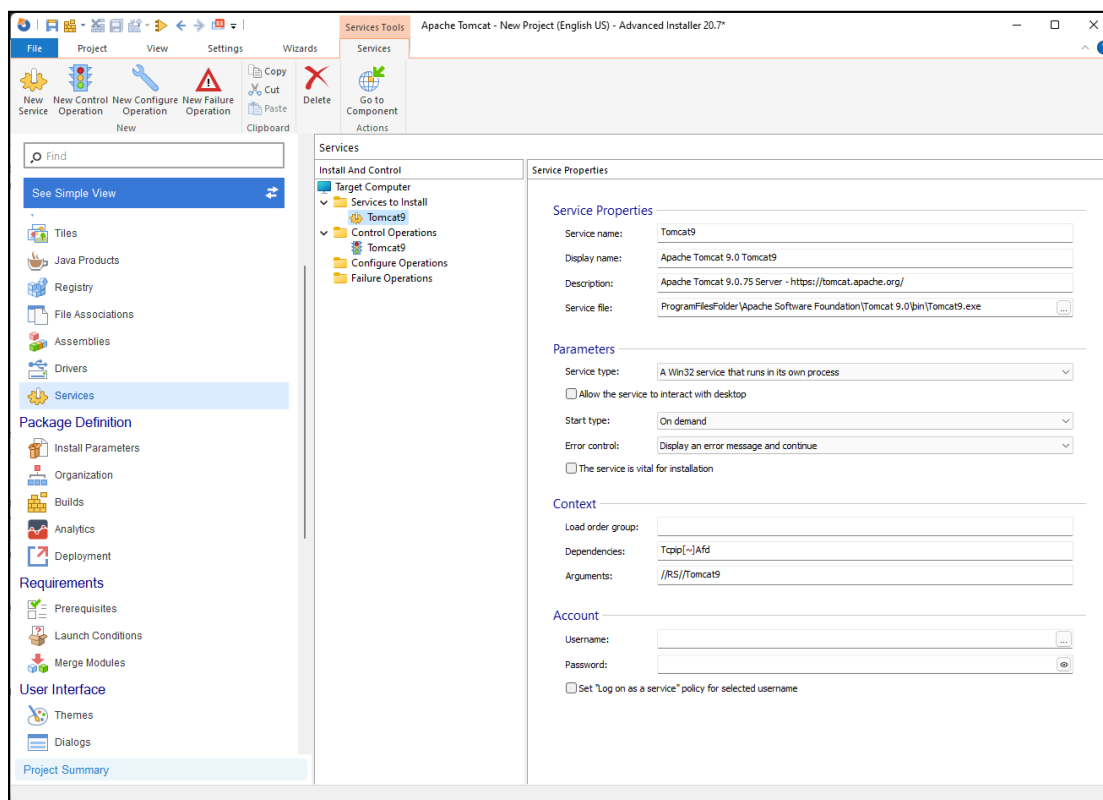


Users can view the installed Apache Tomcat service in Windows' Services management console after the installation is complete. The Services management console provides an interface for starting, stopping, and managing the system's installed services. In this case, the Apache Tomcat service will be listed among the installed services, allowing you to control its behavior and have it start automatically with the system.

But jumping to the Services Page, we can see that after we repackaged Apache Tomcat, we see 2 operations:

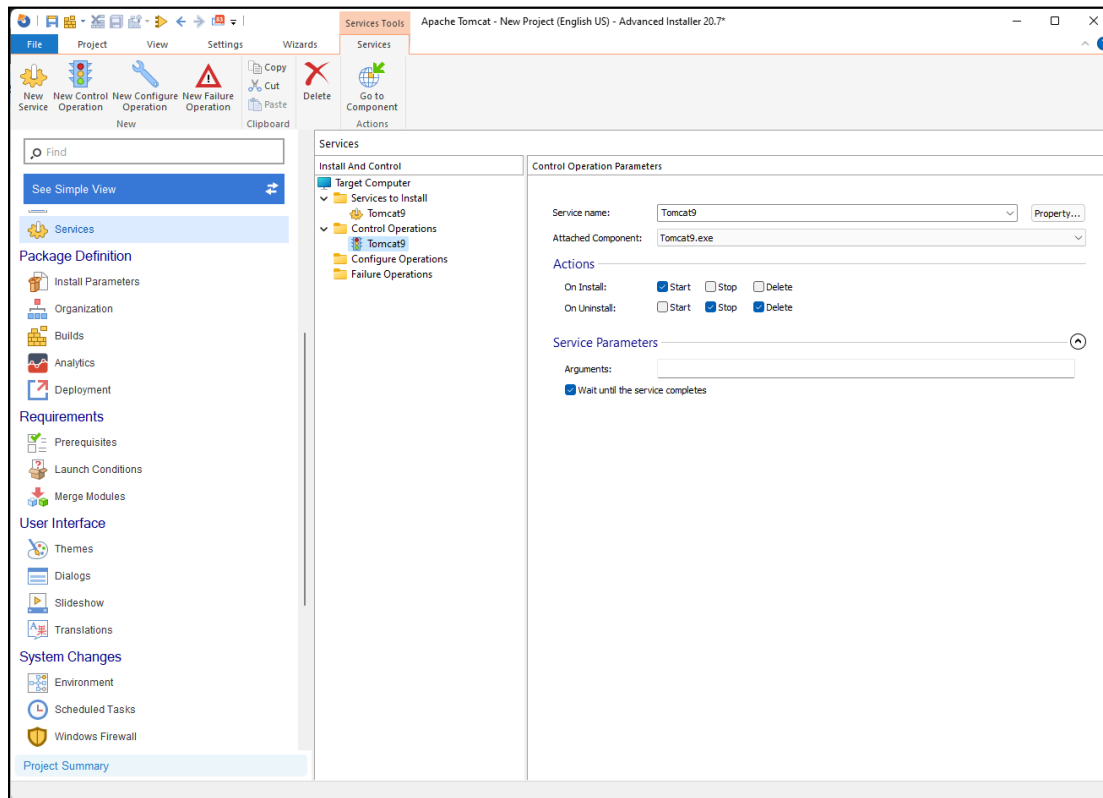
- Services to Install
- Control operations

This is because, as previously stated, after installing a service, you must also configure the operations that must occur once it is present on the machine.



Looking at how the service installation is configured, we can see that Advanced Installer has already implemented some best practices and detected the necessary configurations. This is a Win32 service that runs its own process; the start type is on demand; and we discovered some dependencies and arguments that were passed during the executable's installation.





When it comes to controlling what happens with the service after it is installed or removed, the Control Operations tab provides a graphical interface to define what you require. In this case, after installing the service, we must start it, and when uninstalling, it is best practice to first stop the service and then delete it. If the service is not stopped before deleting it, the operation may fail.



# Introduction to MSI Upgrades

## What is an MSI Upgrade?

An MSI upgrade refers to the process of installing a new version of a software product that replaces an existing installation. It involves updating the installed product with a higher version while preserving user data and settings. MSI upgrades provide a comprehensive update mechanism that allows users to transition seamlessly from one version to another.

When an MSI upgrade process is started, the following actions are considered:

- **Versioning:** A version number is assigned to each version of a software product. A major version, a minor version, a build number, and a revision number are all part of the version number. Upgrades usually necessitate increasing the major or minor version number.
- **Upgrade Table:** The MSI database's Upgrade table contains information that defines how the upgrade process should be carried out. It contains upgrade codes, product codes, and version ranges that are used to determine compatibility between old and new versions.
- **Detection:** The installer checks for the presence of a previously installed version during the upgrade process by comparing product codes and version numbers. If a compatible version is found, the installer will begin the upgrade process.
- **Upgrade Options:** Depending on the upgrade scenario, the installer provides the user with options such as "Upgrade," "Repair," or "Uninstall." The user can choose to upgrade the current installation or take other actions based on their needs.
- **Files and Settings:** The installer replaces old files with new ones, updates registry entries, and modifies configuration settings to reflect the changes introduced in the new version during the upgrade. To ensure a smooth transition, user-specific data and settings are typically preserved.

When doing MSI upgrades, it is important that you follow some of the best practices which were developed during the years:

- **Versioning:** To avoid confusion and ensure proper compatibility checks during the upgrade process, use a consistent versioning scheme. When significant changes are introduced, the major or minor version number is incremented.



- **Testing:** Test the upgrade process thoroughly to ensure compatibility, data integrity, and a smooth transition from the old to the new version. To identify and address any issues, perform functional testing, compatibility testing, and user acceptance testing (UAT).
- **Compatibility Checks:** To ensure a smooth upgrade process, conduct thorough compatibility checks between different versions of the software. Check to see if the new version is compatible with your existing data, settings, and dependencies.
- **Documentation:** To guide users through the upgrade process, provide clear and comprehensive documentation, including release notes and upgrade instructions. To manage user expectations, communicate any changes, new features, or known issues.
- **Rollback Plan:** Prepare a rollback strategy in case the upgrade process fails. This entails making a backup of the existing installation or having a mechanism in place to revert to a previous version if necessary.
- **User Communication:** Communicate effectively with users to inform them of the availability of upgrades, benefits, and any actions they must take. To keep users informed about the upgrade process, use notifications, emails, or in-app messages.
- **Customization:** Provide options for customizing the upgrade process to meet the needs of different users. During the upgrade process, allow users to select installation options, configure settings, or provide feedback.
- **Compatibility with Previous Versions:** When designing upgrades, keep backward compatibility in mind. Check that the new version can interact with previous versions of the software and that data compatibility issues are handled.
- **Automation and Deployment:** To speed up the upgrade process, use automation tools or software deployment systems, especially in enterprise environments. This helps to ensure consistency, reduces manual effort, and allows for centralized upgrade management.

## Patch vs upgrade

Windows Installer (MSI) provides two primary mechanisms for managing software updates and new releases: patches and upgrades. Both are used to modify an existing installation, but they have different characteristics and are used in different scenarios.

An MSI patch (MSP) is primarily used to fix specific issues or bugs in an existing installation without affecting the installed product significantly. It typically targets a specific version or range of versions and addresses specific software problems. Patches are intended to be small and focused, addressing specific issues without introducing new features or functionality.

Patching is a differential process that updates only the modified files or components. A patch package usually includes the binary differences between the old and new versions of the files,



allowing for quick updates. Only the affected files are replaced during patch installation, reducing installation time and impact on system resources.

MSI patches are version-specific and typically target a specific software base version. They are intended to update a specific set of versions, addressing only the issues or changes identified between those versions. Patches frequently have strict version dependencies and may necessitate additional prerequisites to ensure compatibility.

They are also typically installed silently, with no user interaction required. They operate in the background, frequently during system maintenance or automated update processes. Patches are intended to be non-intrusive and seamless, providing fixes while not interfering with the user's workflow.

MSI upgrades, on the other hand, entail installing a new version of the software that includes enhancements, new features, or significant changes to the product. It is a full update that replaces the current installation with a newer version. Upgrades frequently necessitate careful planning and testing to ensure a smooth transition from the old to the new version.

Upgrades entail completely replacing the installed product. The previous version is uninstalled, and the new version is installed separately. Uninstalling previous versions, copying new files, modifying registry entries, and updating the installation database are all possible steps in this process. Upgrades necessitate a more involved installation process than patches.

MSI upgrades affect the entire installed base, allowing users to transition from one major version to the next. They typically support upgrades from multiple previous versions to the most recent release. To ensure a smooth transition, upgrades frequently include mechanisms to handle data migration, configuration updates, and compatibility checks.

Upgrades also give you more control and flexibility over your user experience. They can be installed interactively, allowing users to select installation options, configure settings, or provide feedback while the upgrade is taking place. User notifications, such as release notes or upgrade prompts, may also be included in upgrades to inform users about new features or changes.

Patches, on the other hand, are typically easier to test and deploy than upgrades because they address specific issues within a limited scope. Testing efforts can be concentrated on the affected areas, reducing overall testing effort. Patch deployment can be done automatically with software distribution tools or manually with patch management systems.

MSI upgrades do necessitate extensive testing to ensure compatibility, data integrity, and a smooth transition from the old to the new version. They entail a more thorough testing procedure, which includes functional testing, compatibility testing, and user acceptance testing



(UAT). Additional considerations for upgrades may include data migration, backward compatibility, or upgrade rollback options.

Be aware of the various restrictions that come with patches when working with them. Not taking them into consideration may cause your main application to stop working or the patch installation to fail.

Now we'll go over the most common restrictions to be aware of when working with patches. By understanding these constraints, you can ensure that your patching process runs smoothly and without incident.

The following are the most common patch restrictions:

- Between the original and new MSI file versions, do not change the primary keys in the File table.
- Files should not be moved from one folder to another.
- Files should not be moved from one cabinet to another.
- Do not rearrange the files in a cabinet.
- Change the Component GUID for any Component at all.
- Change the name of the Component's key file only if you want to change the Component GUID.
- Do not change the existing feature and component hierarchy. A new Feature can be added, but removing a parent Feature requires the removal of all its child Features.
- Components should not be removed from a Feature.
- The names of the Target and Upgraded MSI packages must be the same.
- The Product Code for the Target and Upgraded MSI packages must be the same.

You can find the full list of restrictions for patches on the [Microsoft website](#).

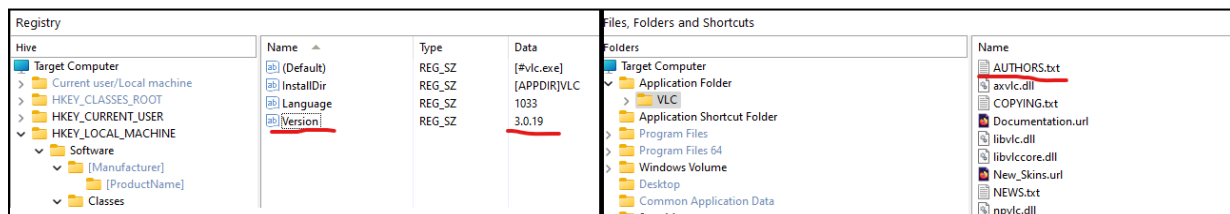
## Example patch for VLC



If you're new to patching, you might notice that creating an MSP is a difficult task. Unlike MSIs, which can be created with the click of a button, MSPs require additional steps and can appear more complicated. But don't be concerned! Here are the straightforward steps for creating an MSP in Advanced Installer:

1. Create the MSI for v1.0 of your app.
2. Create the MSI for v1.1 following the patch rules.
3. Create a patch project to compare the two MSI packages.

We already repackaged VLC Media player earlier, but let's assume that a new version came along and only changed 1 file and 1 registry.



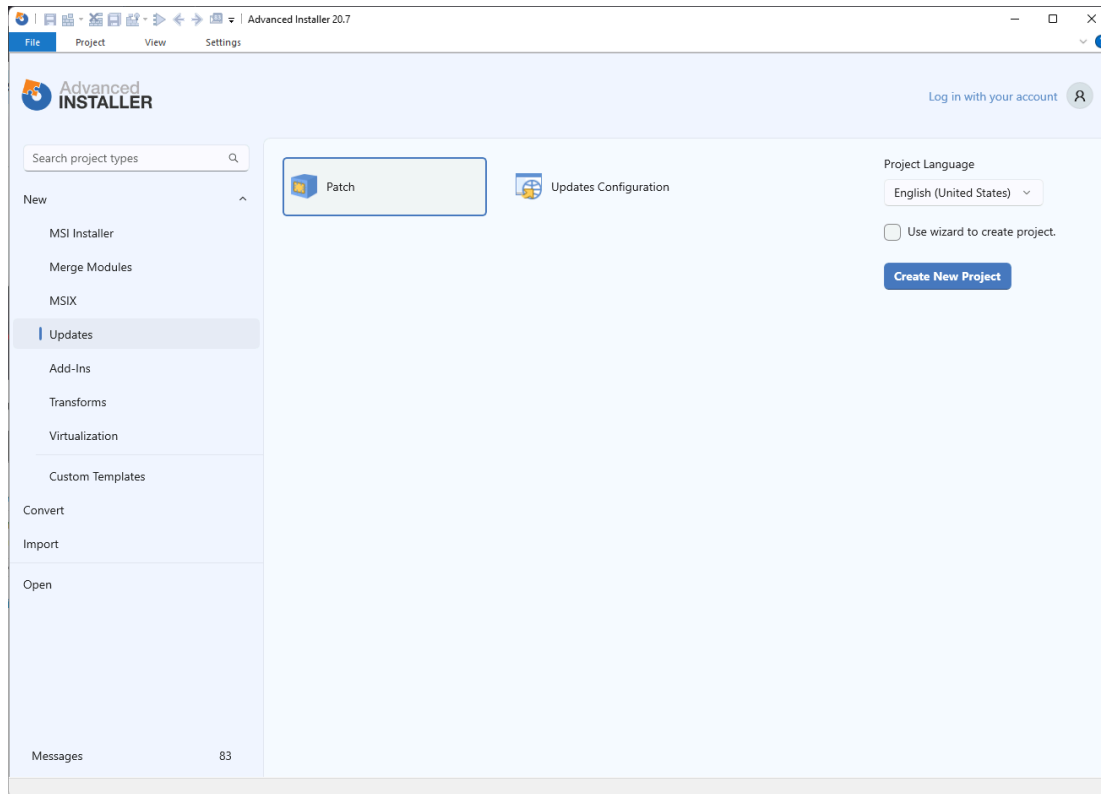
We will simply copy the resulted VLC Media Player MSI and change any desired file and registry and we will assume that this is the second MSI for VLC Media Player.

Once we have our 2 MSI files, let's get started with creating the patch file using Advanced Installer:

- Launch Advanced Installer and navigate to the Updates tab.



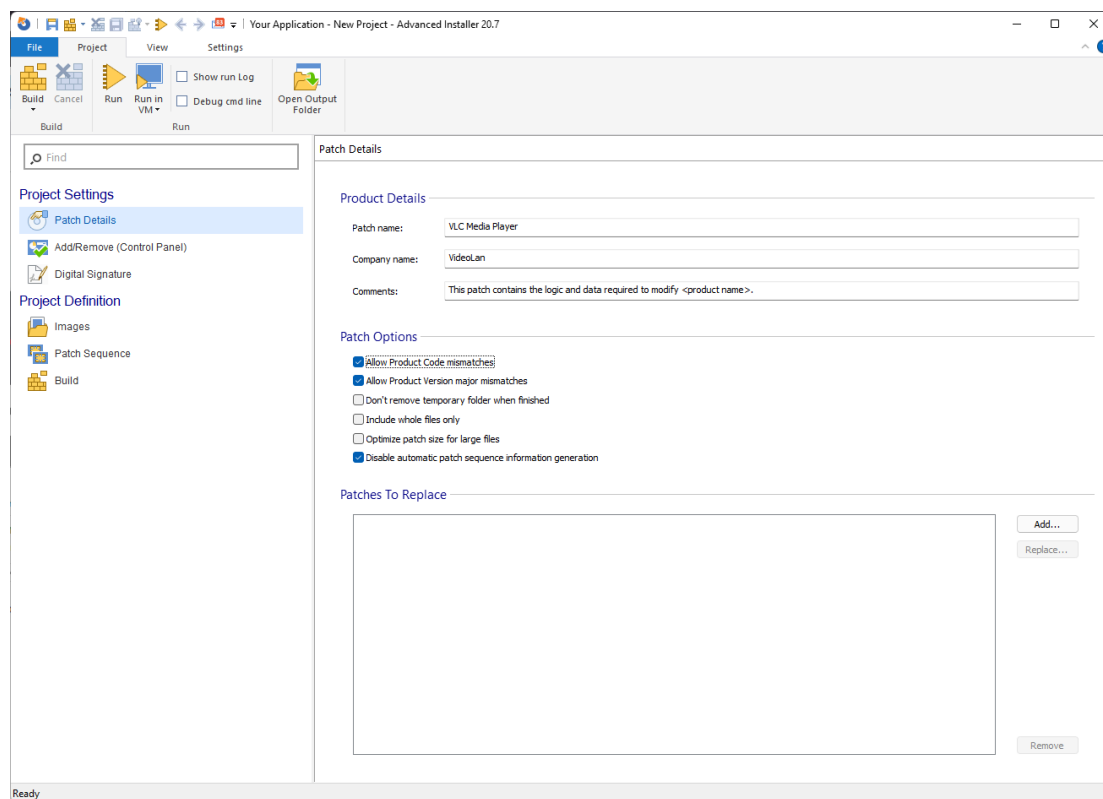




If you haven't installed Advanced Installer yet, you can download a 30-day full-featured trial from our website.

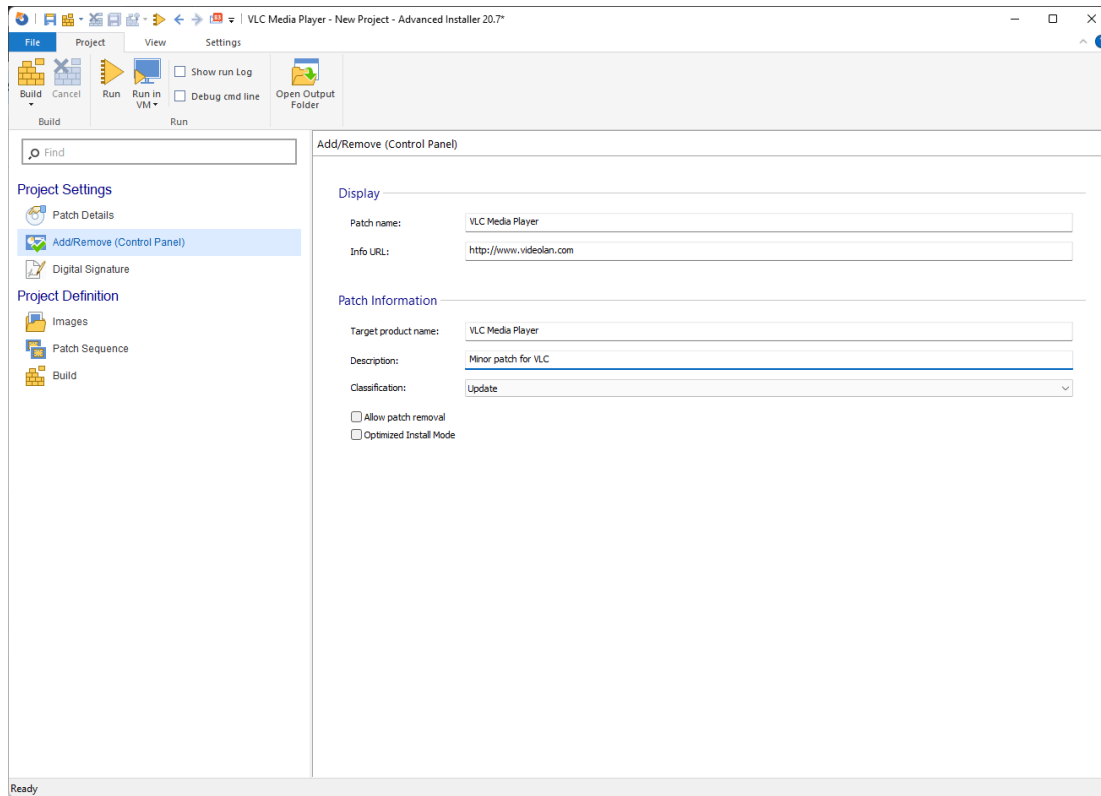
- Create a new Installer Patch project and fill in all the necessary information related to your patch identification in the Patch Details page and then in the Product Details.





- Configure the Add/Remove (Control Panel) information. This sets up the display name in the Control Panel. There, you can also set the option to Allow patch removal.

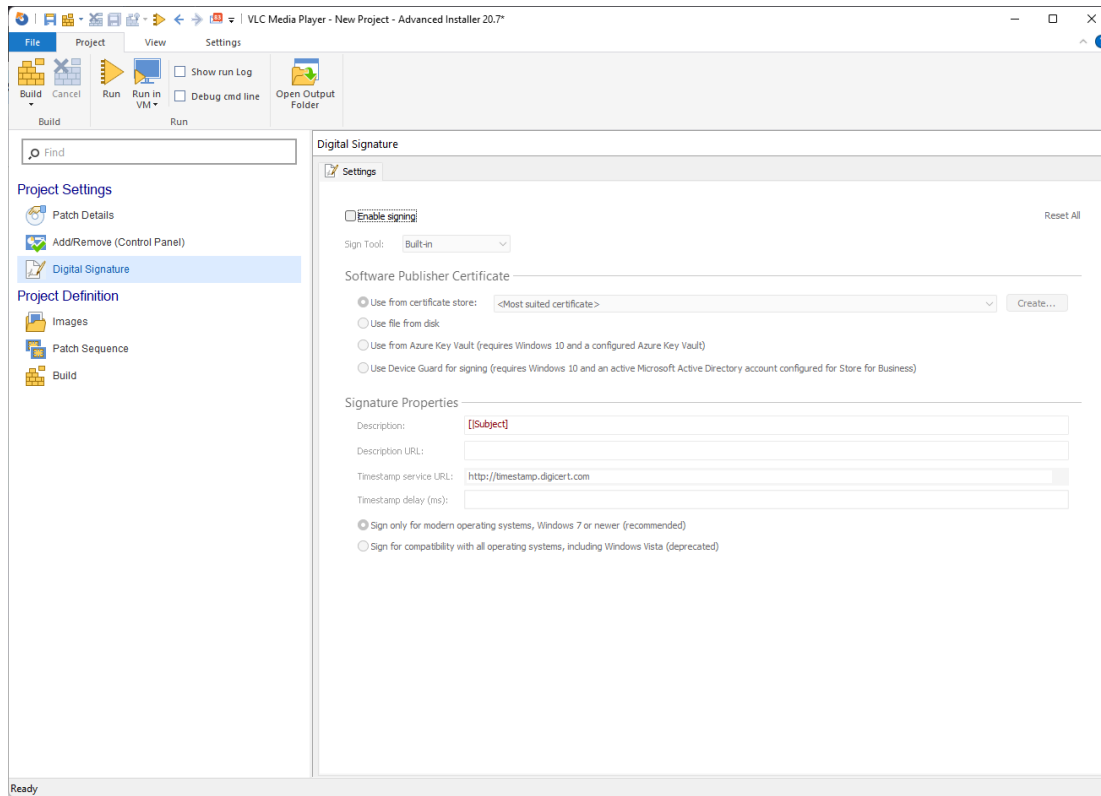




- Optional: Digitally sign the patch. This step helps ensure the integrity and authenticity of the patch.

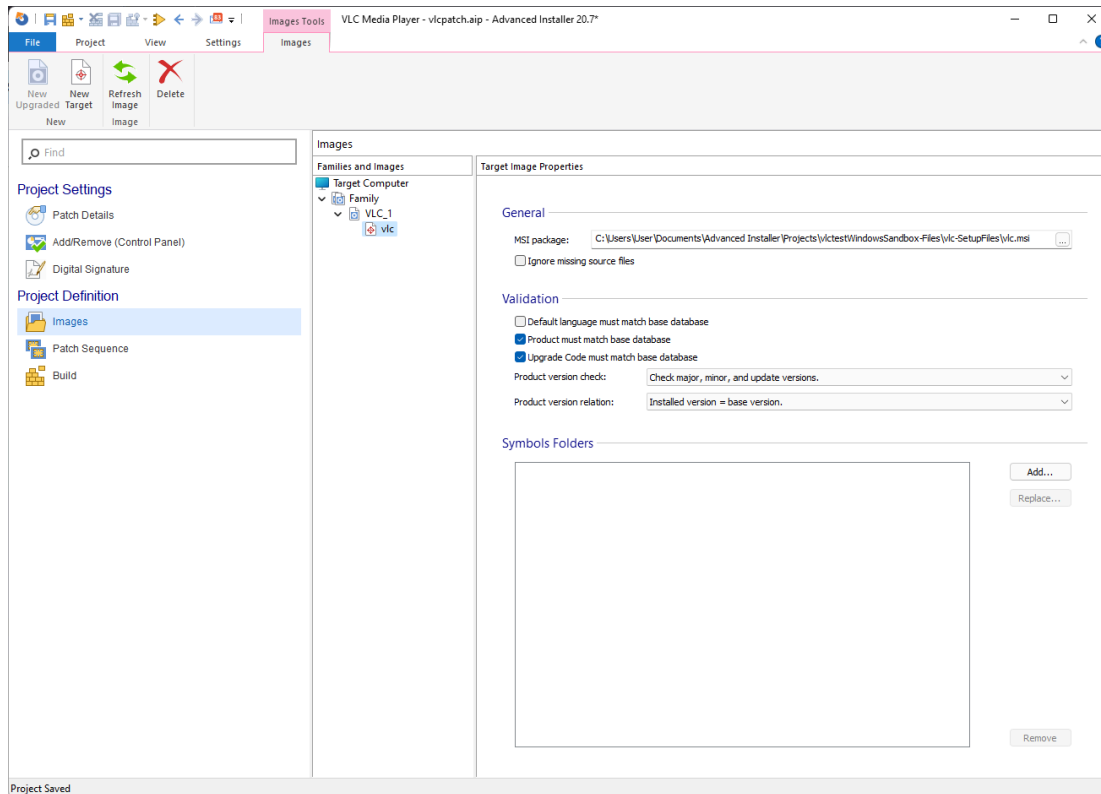
Note that signing the patch requires a valid digital certificate.





- Proceed to the Images page and right-click on the Target Computer. Here you need to first point out the Upgraded MSI and then the Targeted one. The upgraded MSI is the version you want to update to, while the targeted MSI is the base version you want to patch.



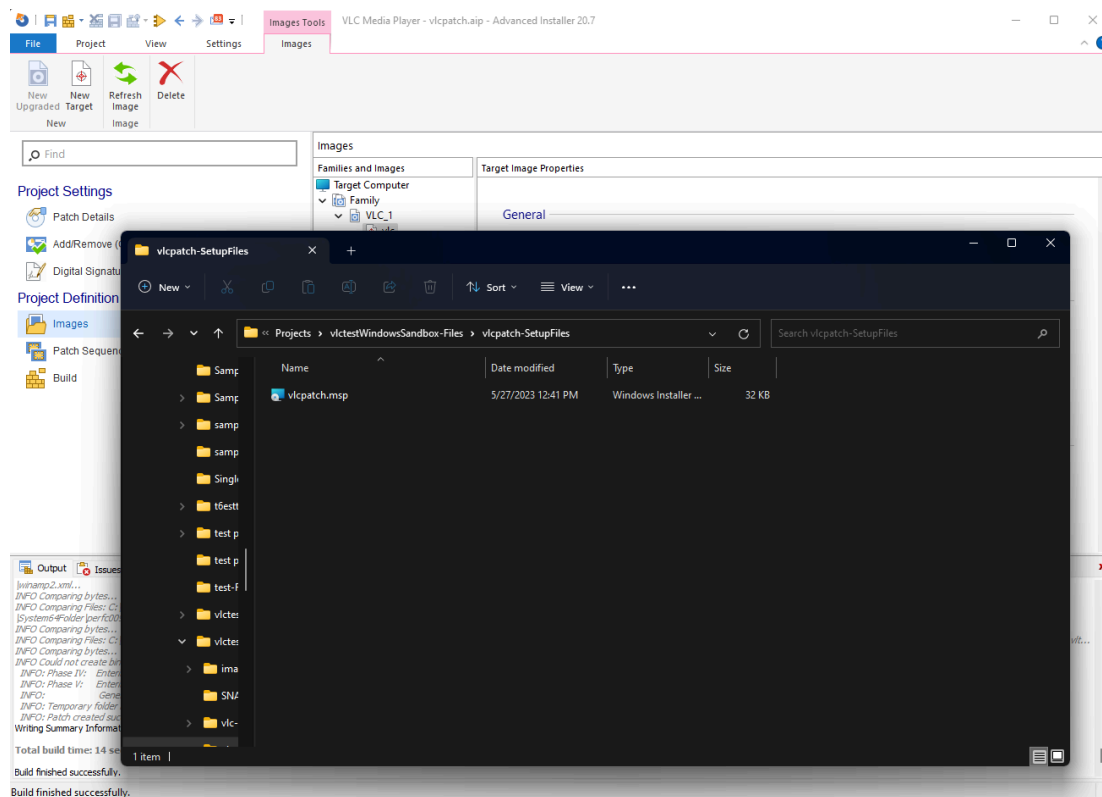


By pointing out these two MSI files, Advanced Installer will generate the necessary delta (difference) between them to create the patch.

- Save and build the patch project.

Once you have completed all the necessary configurations, save the patch project. Then, build the project to generate the patch file (MSP).





You can use the Windows Installer Patch command-line tool (Msiexec.exe) with the /p parameter to install an MSP (Microsoft Patch) file. The following is the general syntax for installing an MSP file:

```
msiexec /p PathToMSPFile /qb
```



# Package Deployment

## Command lines

### MSI Command Lines

MSI command lines are used to customize the installation and configuration of MSI (Microsoft Installer) packaged applications. The following are some of the most commonly used MSI command lines:

**INSTALL:** The INSTALL command starts the installation of an MSI package. It usually takes the form `msiexec /i <path to MSI file>`.

```
msiexec /i VLC Media Player.msi
```

**UNINSTALL:** The UNINSTALL command is used to uninstall an MSI package that has been installed. It has the syntax `msiexec /x <ProductCode>`, where the ProductCode represents the installed package's unique identifier.

```
msiexec /x {35A71788-FB15-4046-BFBA-58913DDE5D9C}
```

**REINSTALL:** The REINSTALL command is used to repair or reinstall an MSI package that has already been installed. It can be used to repair problems or to update the installation. `msiexec /f <ProductCode>` is the syntax.

```
msiexec /fus {35A71788-FB15-4046-BFBA-58913DDE5D9C}
```

**MODIFY:** The MODIFY command is used to modify an MSI package's installed features or settings. Users can add or remove specific components. `msiexec /i <path to MSI file> MODIFY=<feature name>` is the syntax.



```
Msiexec /i VLC Media Player.msi MODIFY=MainFeature
```

TRANSFORM: To apply an MST (transform) file to an MSI package, use the TRANSFORM command. MST files are modifications and customizations to the original MSI. `msiexec /i <path to MSI file> TRANSFORMS=<path to MST file>` is the syntax.

```
Msiexec /i VLC Media Player.msi TRANSFORMS=VLC.MST
```

PROPERTY: The PROPERTY command is used to set or change the value of an MSI package's property. Properties are used to manage different aspects of the installation. `msiexec /i <path to MSI file> <property name>=<value>` is the syntax.

```
Msiexec /i VLC Media Player.msi ALLUSERS=1
```

/qn: The /qn switch is used for silent installation, which means that no user interface is displayed during the installation process. To perform silent installations, it can be added to any installation command line.

```
Msiexec /i VLC Media Player.msi /qn
```

/LV: During the installation process, the /LV switch is used to generate a verbose log file. It collects detailed installation information that can be used for troubleshooting. `msiexec /i <path to MSI file> /L*V <log file path>` is the syntax.

```
Msiexec /i VLC Media Player.msi /L*V C:\temp\vlc.log
```

These are only a few examples of commonly used MSI command lines. Depending on the MSI package and its configuration, the command line options and parameters may differ. It is best to consult the documentation or vendor's instructions for the specific command lines needed for a given application.

For more information about MSI command lines check out our [in-depth user guide](#).

When deploying applications with SCCM or Intune, you don't need to specify the full path to the installation because when the user installs it from Software Center/Company Portal, the





installation runs directly from the downloaded directory which contains the files.

## PowerShell Command Lines

PowerShell scripts are executed using the PowerShell command-line environment, which is a powerful scripting and automation framework provided by Microsoft.

The general command line is:

```
Powershell.exe -file Install.ps1
```

PowerShell, on the other hand, introduced the concept of execution policies. PowerShell's execution policy determines the level of security for running scripts. It is a security feature that aids in the prevention of malicious or unauthorized script execution. The execution policy can be configured to control whether and where scripts can be run.

PowerShell has different execution policy levels available:

- **Restricted:** This is the default execution policy. It does not allow the execution of any scripts. It only allows individual commands to be run from the command line.
- **AllSigned:** With this execution policy, scripts can only be run if they are signed by a trusted publisher. PowerShell will prompt for confirmation before running scripts that are not digitally signed.
- **RemoteSigned:** Scripts downloaded from the internet or other remote sources must be signed by a trusted publisher. Locally created scripts do not require a digital signature.
- **Unrestricted:** This execution policy allows the execution of any script, regardless of its source. PowerShell will still display a warning before running scripts downloaded from the internet.
- **Bypass:** This execution policy disables the execution policy. No restrictions are applied, and PowerShell will not prompt for confirmation.
- **Undefined:** This execution policy is used when no execution policy is set. It inherits the execution policy from the parent or user's machine.

In most enterprises, the setting for the execution policy is usually set to either AllSigned or RemoteSigned. In this case it is recommended to sign your PowerShell scripts before executing.

But in case you are unable to sign them, there is an alternative way to run the script:



```
PowerShell.exe -executionpolicy bypass -file Install.ps1
```

By doing this, you are bypassing the set execution policy Sand no restrictions are applied, thus allowing you to run your PowerShell script.

Lastly, one other major aspect we need to consider is in regards of parameters. In PowerShell, script parameters are used to pass values or arguments to a script during runtime. They allow scripts to be flexible and configurable by accepting input from the user or other sources. Script parameters are defined within the script's code and can be accessed and used within the script.

```
param(  
    [Parameter(Mandatory=$true)]  
    [string]$ApplicationName,  
  
    [Parameter()]  
    [int]$Execution  
)
```

Looking at the above code, the parameter \$ApplicationName is mandatory and must be provided in order for the script execution to continue, while the \$Execution parameter is optional.

If we take a look at PSADT and how it's executed when you want to run the uninstallation part, you can use the DeploymentType parameter which is not mandatory.

```
powershell.exe -executionpolicy bypass -file deployapplication.ps1  
-DeploymentType Uninstall
```

The parameters are different with each and every script, but PSADT uses the DeploymentType parameter to know which part of the script you wish to execute. If you want to use the install part you can either remove the parameter or use the following:

```
powershell.exe -executionpolicy bypass -file deployapplication.ps1  
-DeploymentType Install
```

## VBScript Command Lines



When it comes to running VBScripts, you have two different script hosts available in the OS:

- Wscript
- Cscript

Wscript is a built-in Windows scripting host that allows you to run VBScript (Visual Basic Scripting Edition) and JScript (JavaScript) scripts on Windows. It interprets and executes these scripts, provides access to various system objects, and facilitates interactions with the Windows operating system.

Cscript is a Windows command-line scripting host that is used to run VBScript (Visual Basic Scripting Edition) and JScript (JavaScript) scripts. It is an alternative to the graphical scripting host, Wscript, and is typically used when scripts must be run from the command line or when a graphical user interface is not available or desired.

The primary distinction between Wscript and Cscript is how VBScript or JScript code is executed. Comparing the two we have the following:

Wscript:

- On Windows operating systems, Wscript (Windows Script Host) is the default scripting host for VBScript and JScript.
- Wscript is used by default when a script file is double-clicked or executed from the command prompt without explicitly specifying the scripting host.
- Wscript includes a graphical user interface (GUI) environment for displaying script dialogs and message boxes to the user.
- It is commonly used for interactive scripting as well as user-oriented script execution.

Cscript:

- On Windows operating systems, Cscript (Console Script Host) is a command-line scripting host for VBScript and JScript.
- Because it operates within the command prompt or batch files, it is intended for non-interactive scripting and batch processing.
- Cscript lacks a graphical user interface and displays all output in the command prompt window.
- It is commonly used for background script execution, automation tasks, and script execution in command-line environments.

To summarize, Wscript is used when running scripts with a graphical user interface, displaying message boxes, or interacting with users, whereas Cscript is used for command-line execution and non-interactive script processing. The choice between Wscript and Cscript is determined by the script's specific requirements and the desired execution environment.



In general, most VBScripts should be Wscript and Cscript compatible and can be run on either scripting host. However, in some cases, a VBScript may fail to run as expected when using Cscript. Among the most common reasons are:

- Graphical User Interface (GUI) interactions: Because Cscript operates in a command-line environment without a GUI, VBScripts that rely on graphical elements such as message boxes, dialog boxes, or user input prompts may not work properly. These scripts are more appropriate for Wscript.
- Wscript-specific methods: VBScripts that use Wscript-specific methods or properties, such as Wscript.Echo or Wscript.Sleep, may not work properly with Cscript. These methods are intended to be used in conjunction with the Wscript scripting host.
- External dependencies: When a VBScript is executed with Cscript, it may encounter errors or unexpected behavior if it relies on external dependencies that are not available or accessible in the command-line environment.
- Script-specific requirements: Certain VBScripts may have requirements or dependencies that are incompatible with Cscript. Scripts that use COM components or ActiveX controls, for example, may need additional configuration or permissions to run with Cscript.

In general the preferred execution method with VBScript is to use the wscript. However, let's have a look at how to run scripts in both ways.

For wscript:

```
Wscript.exe Install.vbs
```

For cscript:

```
Cscript.exe Install.vbs
```

VBScript, like PowerShell, supports the use of parameters. Parameters are used in VBScript to pass values to a subroutine or function when it is called. Parameters enable you to modify the behavior of a subroutine or function based on the values supplied at runtime.

You can use the WScript.Arguments collection to access the values passed from the command line when adding parameters to a VBScript. Here's how you can change a VBScript to accept and use parameters:

```
' Get the number of parameters passed  
numArgs = WScript.Arguments.Count
```



```
' Check if any parameters were passed
If numArgs > 0 Then
    ' Access the parameters using the WScript.Arguments collection
    ' Parameters are zero-based, so the first parameter is at index
0
    param1 = WScript.Arguments(0)

    ' Check if additional parameters were passed
    If numArgs > 1 Then
        param2 = WScript.Arguments(1)
    End If
Else
    ' Display a message if no parameters were passed
    WScript.Echo "No parameters were passed."
End If
```

To run a VBScript with parameters from the command line, you can run this command:

```
Wscript.exe MyScript.vbs param1 param2
```

## Deploy with SCCM

SCCM is an abbreviation for System Center Configuration Manager. It is a comprehensive systems management solution offered by Microsoft for managing large-scale software, operating system, and device deployments. SCCM enables IT administrators to automate software installations, manage updates and patches, deploy operating systems to new machines, and perform a variety of other administrative tasks across a network.

SCCM provides a centralized console through which administrators can efficiently manage and monitor software and hardware resource deployment and configuration. It includes inventory management, software distribution, patch management, remote control, and reporting capabilities. SCCM integrates with Active Directory to enable targeted deployments based on user and device information.

Administrators can use SCCM to create packages and programs that can be used to deploy software applications, perform system updates, and manage configurations across multiple



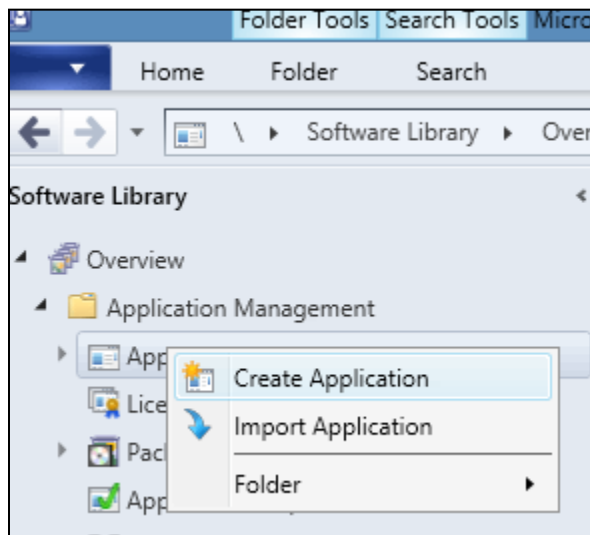
devices such as desktops, laptops, servers, and mobile devices. It supports both on-premises and cloud-based deployments, giving organizations of all sizes flexibility.

SCCM is critical for streamlining IT operations, reducing manual efforts, ensuring compliance, and ensuring a standardized and secure computing environment. It enables organizations to manage their software and hardware assets more efficiently, improve security and compliance, and simplify the process of deploying and managing systems across the enterprise.

When it comes to SCCM, there are two methods for deploying packages. If the package is an MSI, the steps are much simpler, as shown below. Of course, if you want to use wrappers or EXE bundles, you can, but there are some extra steps to take.

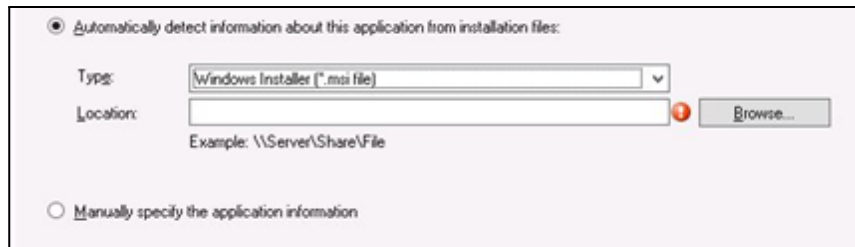
## Deploy MSI via SCCM

To deploy an MSI via SCCM, go to Software Library -> Application Management -> Applications -> right click *Create Application*

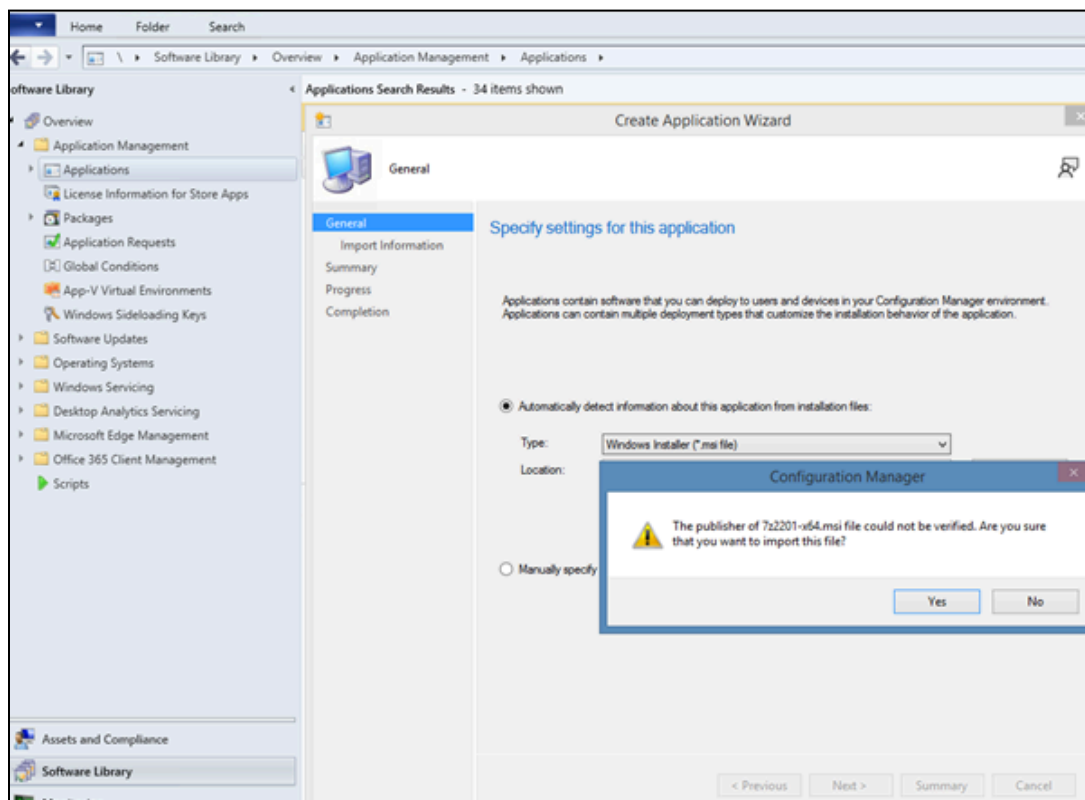


Because the source is an MSI, choose Windows Installer and browse for source content



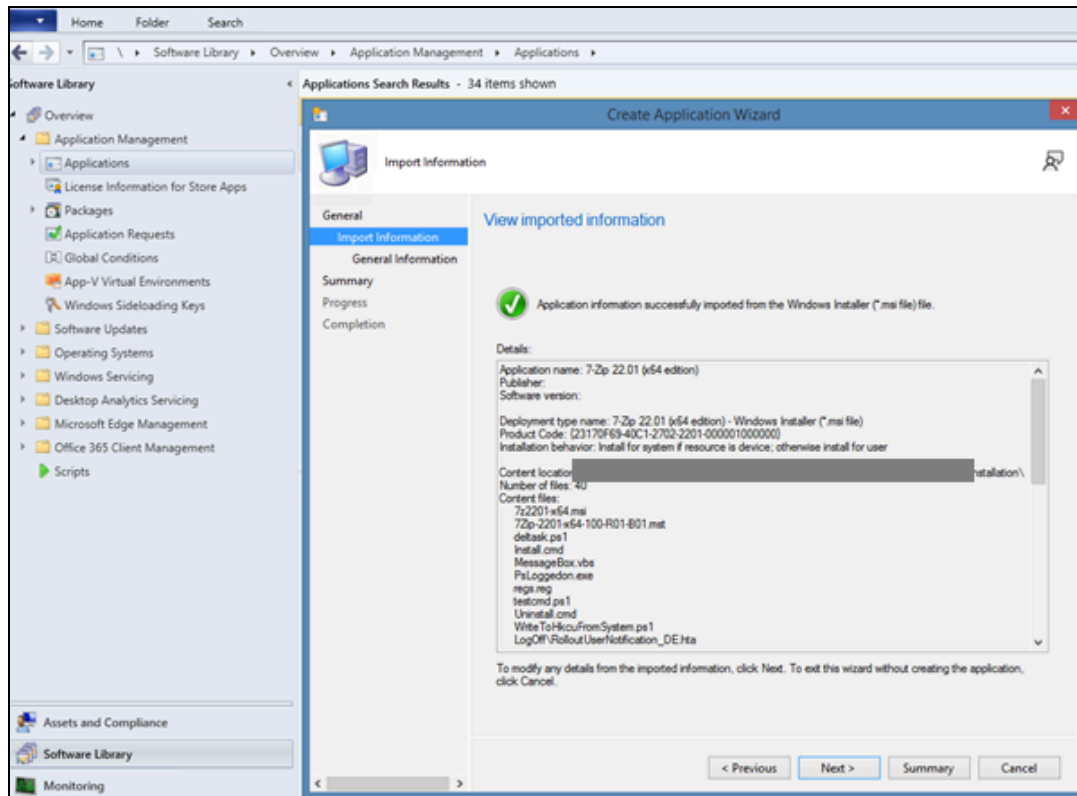


This warning is present with most MSI packages, so we can click YES:

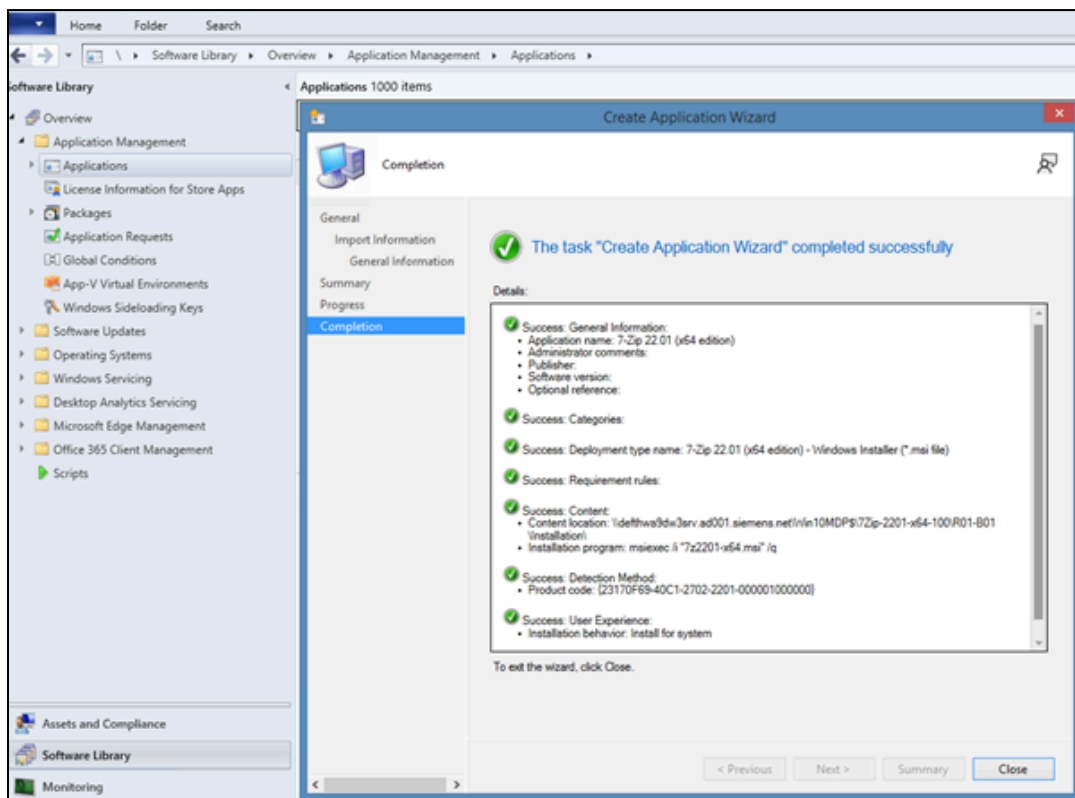
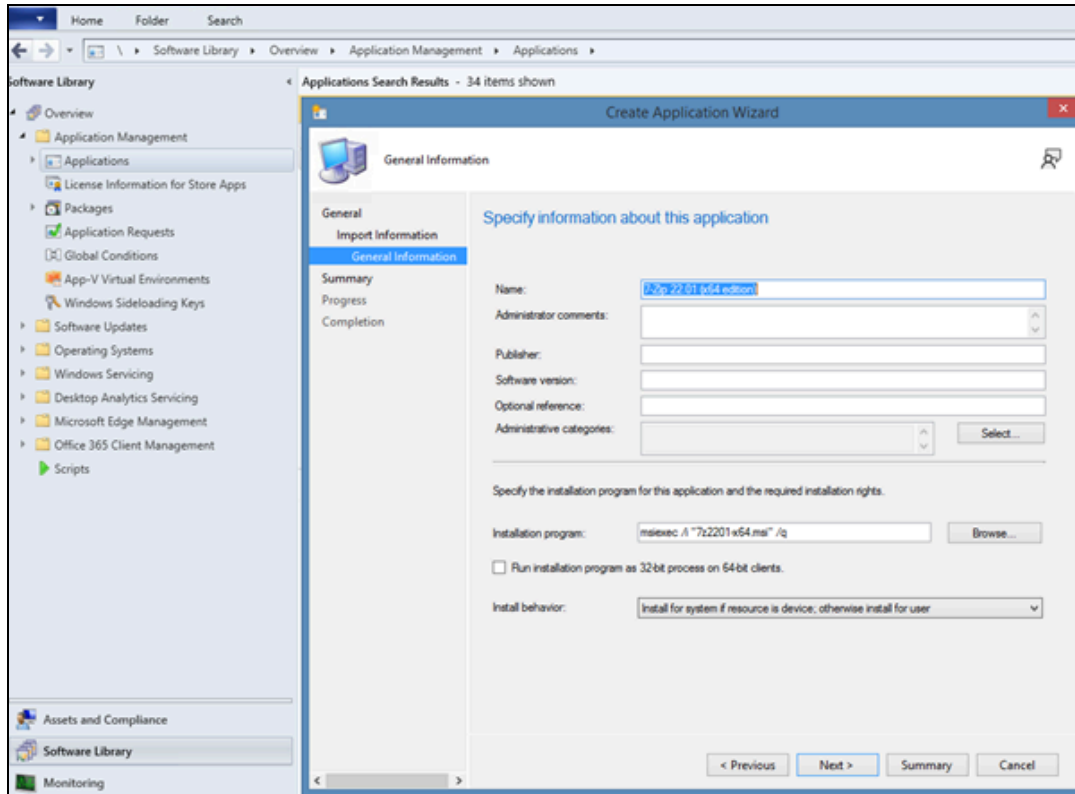


The next steps require only to acknowledge what is going to be created so click Next until the wizard is finished.





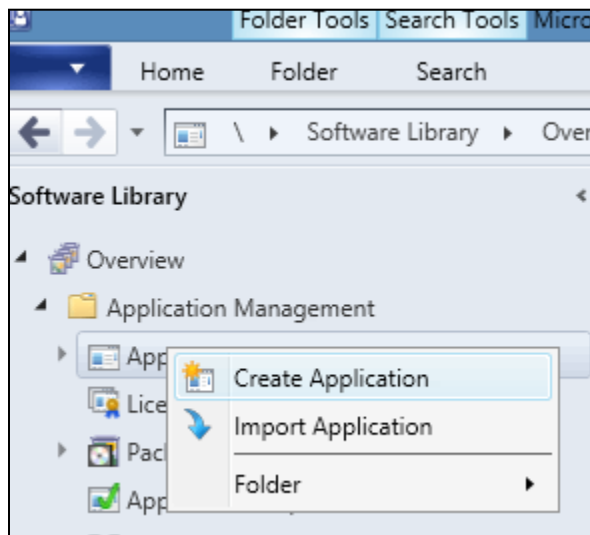




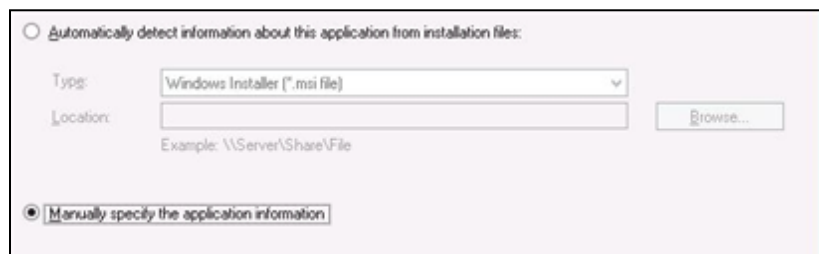
And that is it, all that is left to be done is to distribute the package content on all available Distribution Points and deploy the packaged to the desired list of devices/users in the infrastructure.

## Deploy EXE/VBscript/PowerShell via SCCM

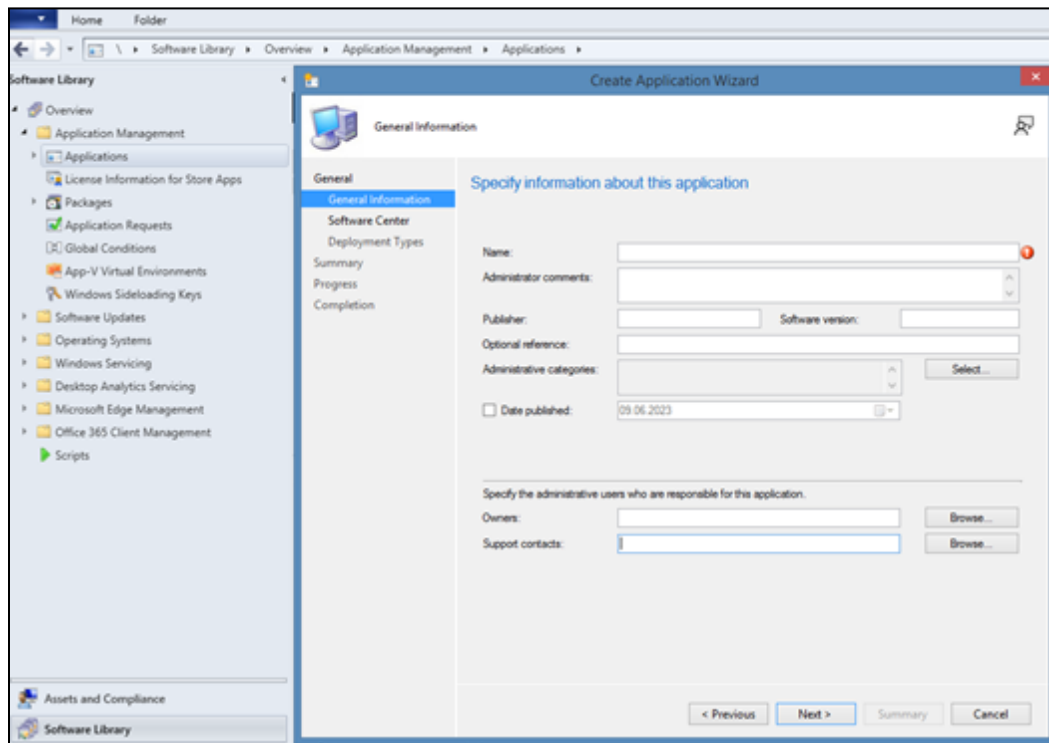
When it comes to other forms of installers, there are a few additional steps which need to be taken. First, go to Software Library -> Application Management -> Applications -> right click *Create Application*



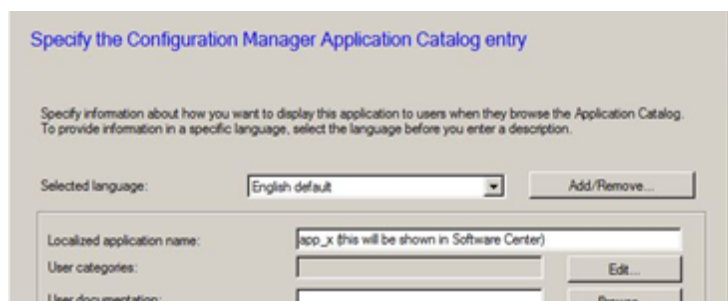
We will have to choose *Manually specify the application information*, which will give you the option to choose *Script Installer* later on this wizard.



Next, fill in all the information for the General Information tab. Keep in mind that “Name” is only an internal information in SCCM and not what the user sees in Software Center.



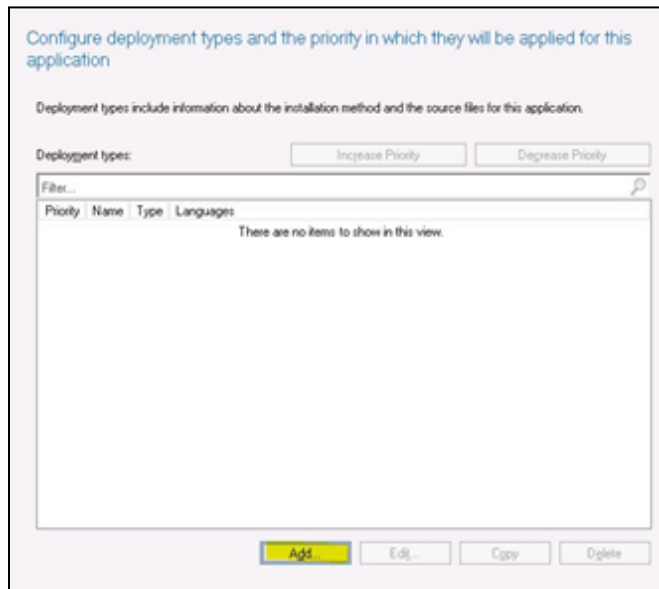
Next, select *English default* and *remove all other languages* if not specifically mentioned otherwise. The *Localized application name* is the name shown in Software Center:



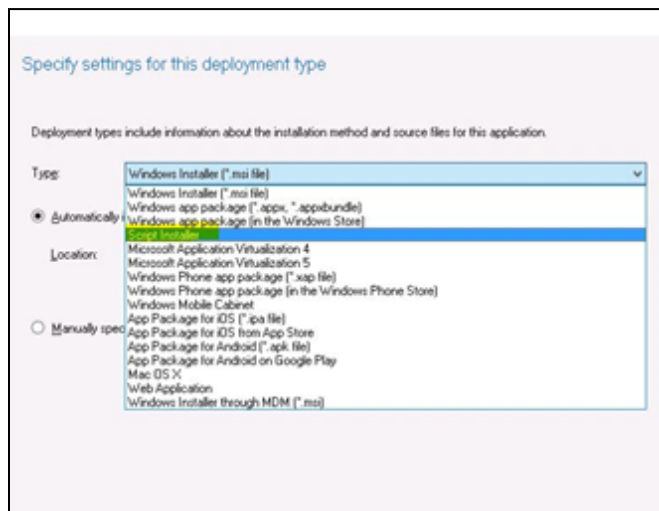
We will need to create a deployment type. A deployment type includes all of the information and instructions required for SCCM to successfully install and manage the application. Details such as the installation command, installation behavior, requirements, detection methods, and user



experience settings are all included. SCCM supports a variety of deployment types to accommodate a variety of application scenarios and deployment needs.

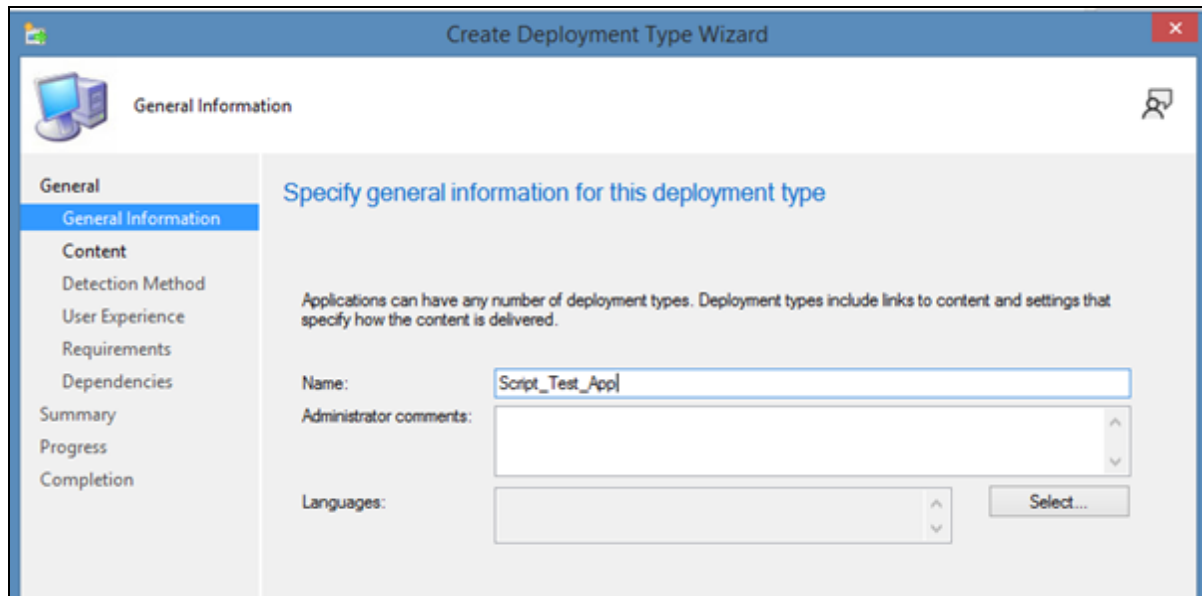


When we reach the specify settings dialog, we choose again *Script Installer*:

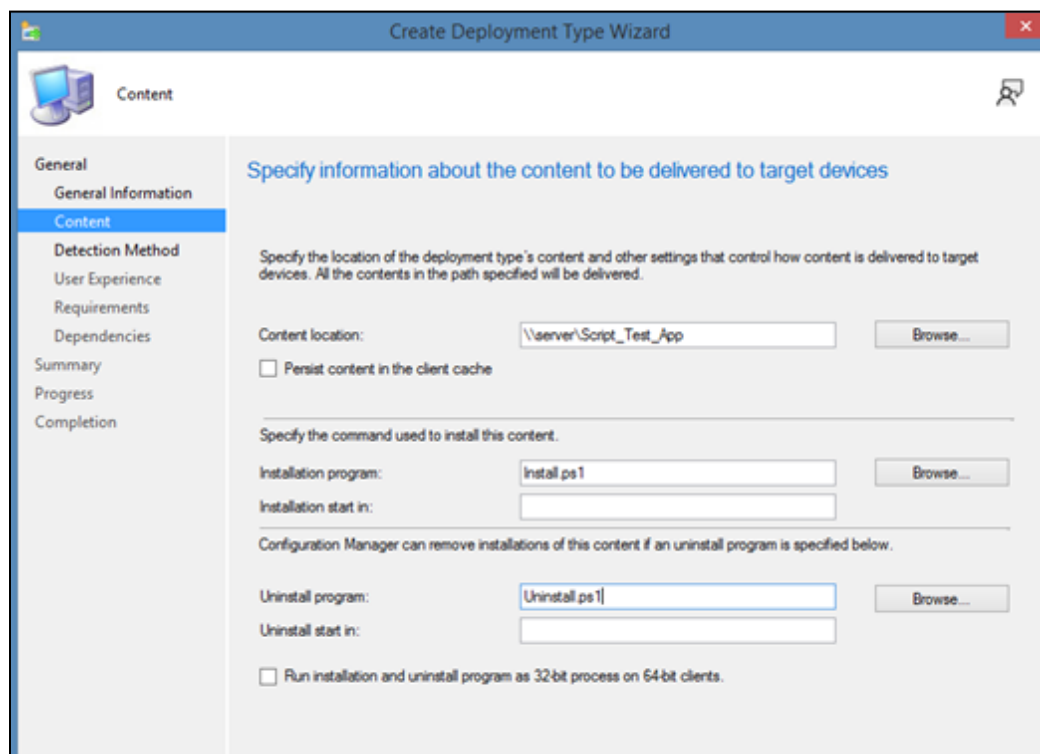


Next, we fill in the application deployment type name:





Now is the part where we add the Content location from the master share and the install and uninstall lines.



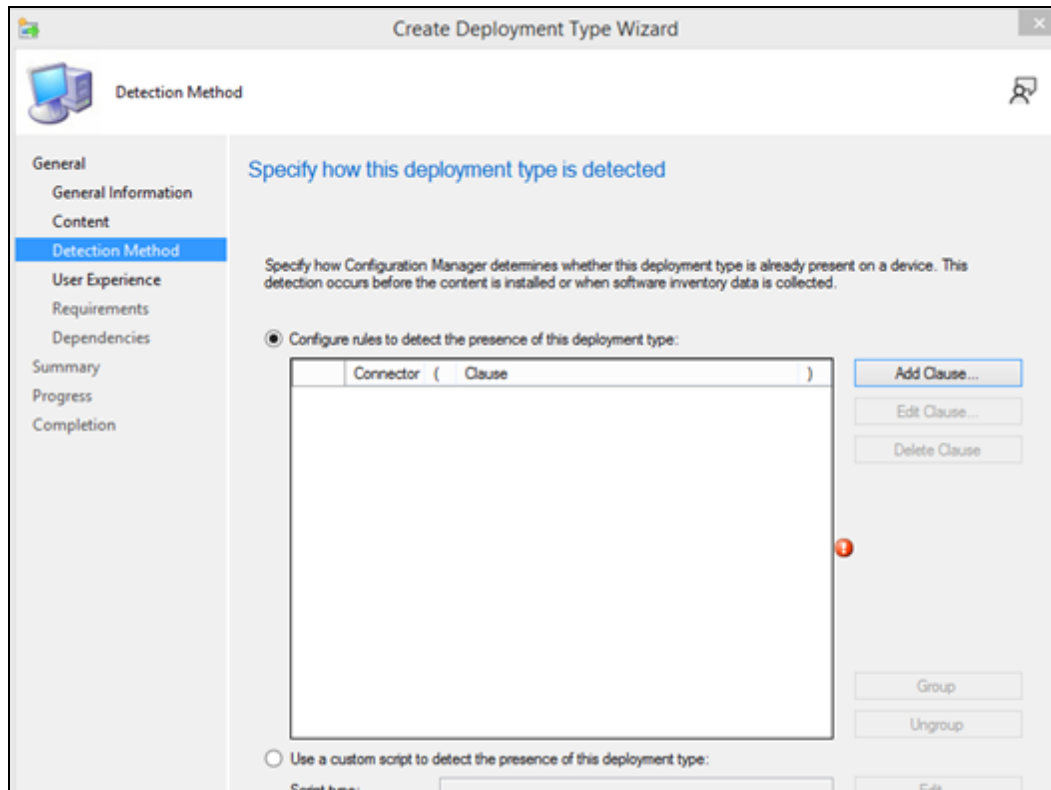
On the next tab, add the detection method. The detection method checks for the existence of an application using specific criteria or rules. When you deploy an application in SCCM, you define a detection method to ensure that the application is only installed on devices that require it or to determine whether an upgrade or update is needed.

The detection method can be configured to use various criteria, such as:

- **File or Folder Detection:** SCCM can check the device for the presence of a specific file or folder. It can look for the presence of an executable file, a configuration file, or a specific folder structure associated with the application, for example.
- **Registry Key Detection:** SCCM can check the device for the presence or value of a specific registry key. This is frequently used to identify applications that generate specific registry entries during installation.
- **Windows Installer Detection:** SCCM can use the Windows Installer database (MSI) to see if a particular product code, package code, or product version is already installed on the device.
- **Custom Script Detection:** Custom scripts (VBScript, PowerShell, etc.) can be used to define detection logic in SCCM. Administrators can create scripts to perform complex checks based on their needs.

With the MSI deployment method, this is easy because SCCM uses the Windows Installer detection which checks for the presence of the product code, and these steps which we are doing here are skipped. In case your package contains an MSI it is recommended to use the Windows Installer Detection.





Next, we reach the user experience settings. The configuration options that determine how an application installation or update is presented and handled on end-user devices are referred to as user experience settings with applications. Administrators can use these settings to customize the behavior and appearance of application deployments in order to provide a consistent user experience. The following are some of the most common user experience settings available in SCCM:

- **Install for User or System:** This setting determines whether the application is installed per user or per system. Per-user installation installs the application for the currently logged-on user, whereas per-system installation installs it for all users on the device.
- **Install Whether or Not a User is Logged On:** Determines whether the application installation should continue even if no user is currently logged in.
- **Allow Users to Interact with the Installation:** Allows or disables user interaction with the installation process, such as displaying or suppressing installation prompts or dialogs.

The most used configuration options for these step are:

- **Installation behavior:** Install for system (runs under NT System/Administrator account)
- **Logon requirement:** Whether or not a user is logged in
- **Installation program visibility:** Normal





- Allow users to view and interact with the program installation: Checked

**Create Deployment Type Wizard**

User Experience

General Information  
Content  
Detection Method  
**User Experience**  
Requirements  
Dependencies  
Summary  
Progress  
Completion

Specify user experience settings for the application

Installation behavior: Install for system

Logon requirement: Whether or not a user is logged on

Installation program visibility: Normal

☒ Allow users to view and interact with the program installation

Specify the maximum run time and estimated installation time of the deployment program for this application. The estimated installation time displays to the user when the application installs.

Maximum allowed run time (minutes): 120

Estimated installation time (minutes): 0

Click next and add installation requirements if needed and dependency application

**Add Dependency**

Define a group of applications that will satisfy a specific software dependency.

Specify one or more applications. If any one of the specified applications is present on a device, this dependency group is considered satisfied for that device. To automatically install one of these applications if none are detected, check Auto Install for the desired applications. Installation attempts will occur in the order listed.

Dependency group name: dependency

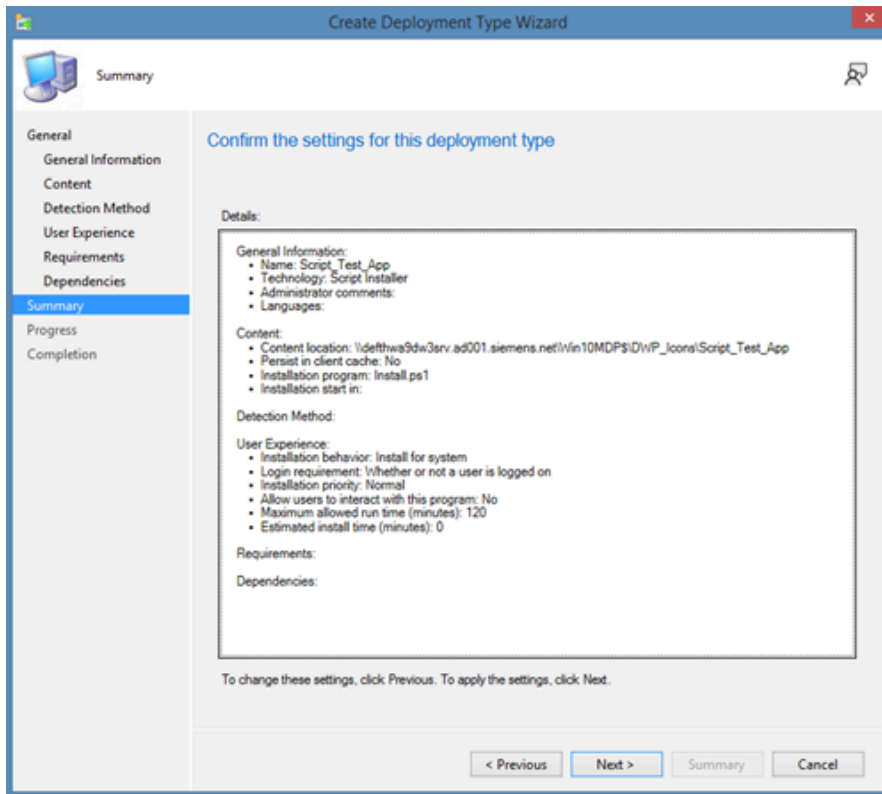
Priority	Application	Supported Deployment Types	Auto Install
----------	-------------	----------------------------	--------------

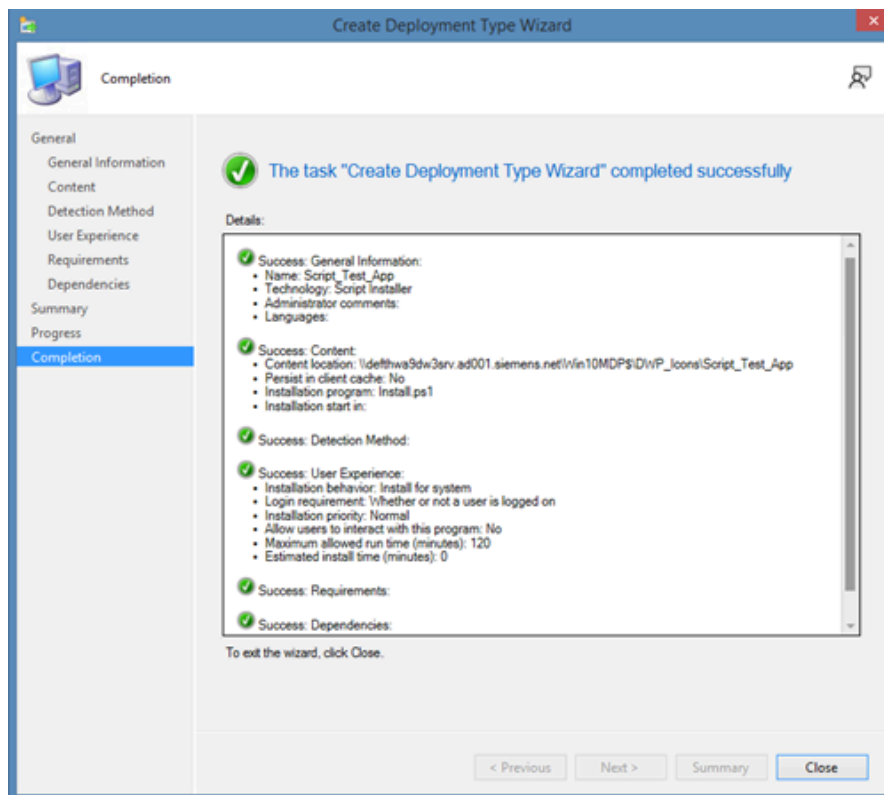
Increase Priority Decrease Priority Add... Delete

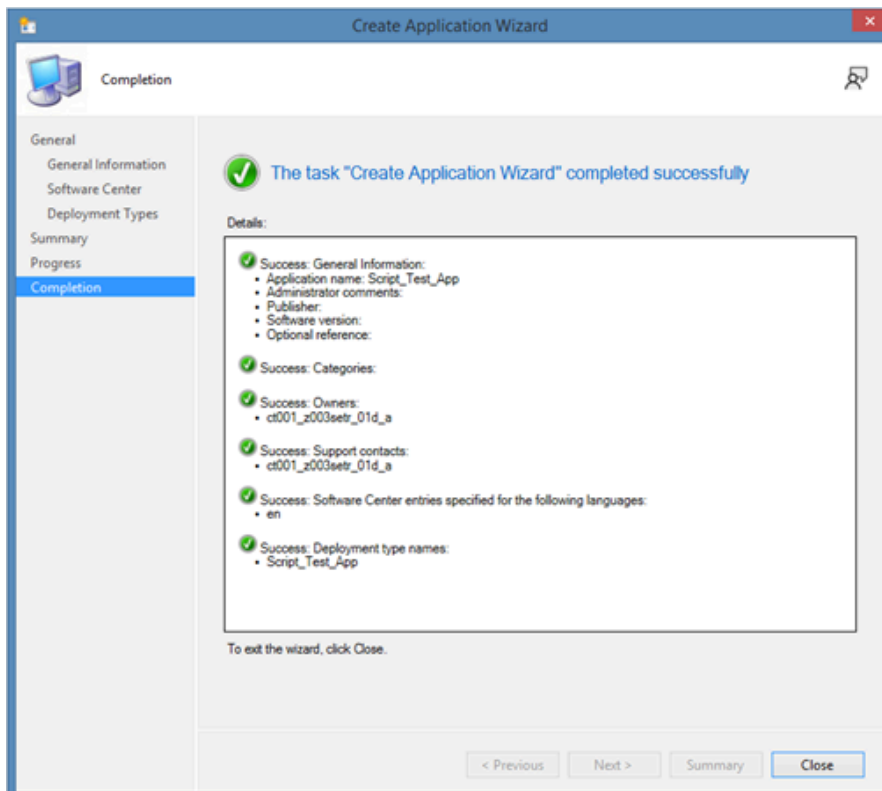
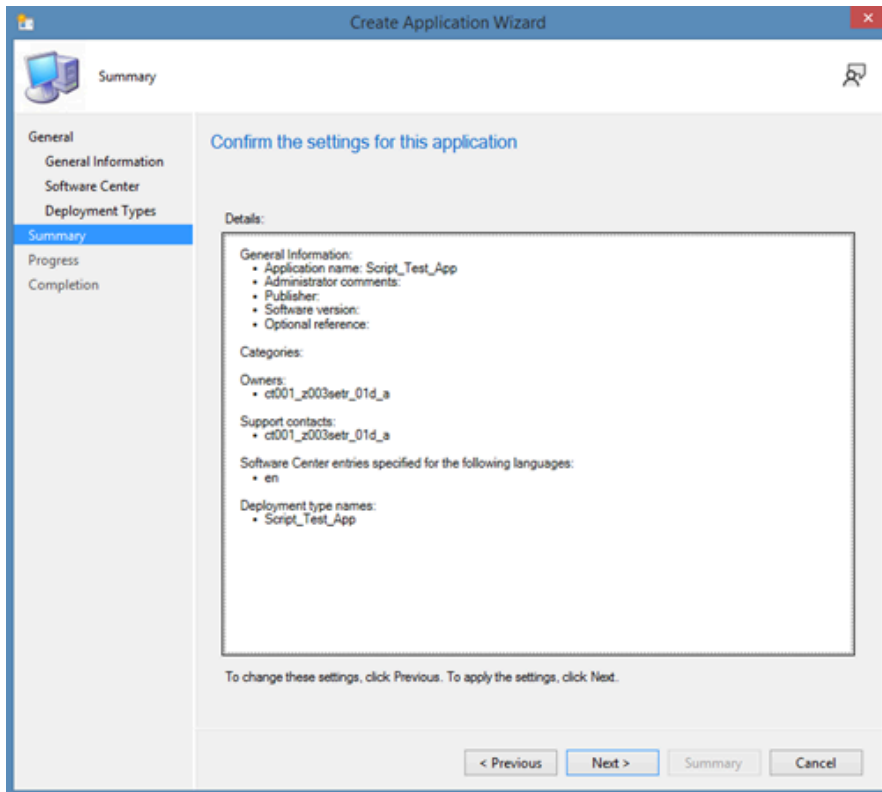
OK Cancel

In the final part, click Next until everything is done.









And that is it. As with MSI deployments, all that remains is to distribute the package content to all available Distribution Points and deploy the package to the desired list of infrastructure devices/users.

## Deploy with Intune

Intune is a cloud-based service provided by Microsoft that focuses on mobile device management (MDM) and mobile application management. It is also known as Microsoft Intune or Microsoft Endpoint Manager (MAM). It is part of the Microsoft Endpoint Manager tool suite and is intended to assist organizations in managing and securing their devices, applications, and data across multiple platforms such as Windows, macOS, iOS, and Android.

Intune includes a number of features and capabilities that help IT administrators manage and protect devices, deploy and manage applications, enforce security policies, and ensure compliance within their organization. Intune's key features include:

- Device Management
- Application Management:
- Data Protection:
- Compliance and Security:

Intune provides organizations with greater flexibility, scalability, and ease of use by providing a unified and cloud-native approach to managing and securing devices and applications. It works well with other Microsoft solutions and services, such as Azure Active Directory and Microsoft 365, to provide an all-encompassing endpoint management and security solution.

With Intune, organizations can use a centralized and cloud-based management console to implement modern management practices, empower their workforce to be productive on any device, and ensure the security and compliance of their digital assets.

Intune supports a variety of application deployment methods, but for the purposes of this book, we will focus on LOB (line of business) and Win32. Let's look at how these two deployment methods compare in Intune.

Looking at LOB (Line of Business) Applications, we can see the following conditions::

- LOB applications are typically custom-built or specialized applications that an organization develops in-house to meet their specific business needs.



- Deploying LOB applications entails directly uploading the application package (e.g., .appx, .msi) to Intune. After that, the package is distributed and installed on managed devices.
- LOB applications are primarily designed for modern platforms such as Windows 10 and later, but they may also support other platforms such as iOS and Android.
- LOB applications are ideal for deploying custom or business-specific applications within a company. They are frequently organization-specific and may not be available in public app stores.
- Intune offers LOB application management capabilities such as app installation, updates, and removal, as well as the ability to enforce policies and configurations specific to these applications.
- For LOB applications to be deployed, the application package must be available for upload, as well as proper signing certificates and relevant metadata.

Win32 Applications, on the other hand, are intended for a variety of purposes.:

- Win32 applications are traditional desktop applications that were not created with modern management platforms in mind.
- The Intune Win32 App Packaging Tool is used to create an application package for Win32 application deployment. The application installer (e.g., .exe), installation script, and other dependencies are typically included in the package. The package is distributed and installed on managed devices via Intune.
- Win32 applications work with a variety of Windows versions, including legacy Windows 7 and Windows 8.1, 10, as well as modern Windows 11.
- Win32 applications are appropriate for deploying traditional desktop applications, such as legacy or complex applications that may necessitate customization or special installation procedures.
- Intune provides Win32 application management capabilities such as app installation, updates, removal, and policy enforcement. Furthermore, Win32 applications can use detection rules to determine whether or not the application is already installed on the device.
- Converting the application installer into a format compatible with Intune, as well as relevant configuration files and detection rules, is required when creating a Win32 application package.

In summary, LOB applications are organization-specific applications, whereas Win32 applications are traditional desktop applications. LOB applications are typically designed for modern platforms, whereas Win32 applications are more universal. Both deployment methods offer management capabilities, but the packaging and deployment processes vary depending on the application. Depending on the application requirements and compatibility with the target devices and platforms, organizations can select the best deployment method.



To make it easier to understand the difference between the two, we can look at LOB applications somewhat as MSI deployments with SCCM and Win32 deployments are similar to Script installations in SCCM. Of course you need to take in consideration multiple factors when creating such applications and for now, the business standard usually leans to Win32 deployments. For now, let's take a look at how to deploy you application with both methods.

## Deploy MSI via LOBA

Deploying an MSI with LOB (Line of Business) applications in Intune involves a few steps. Here is a step-by-step guide to help you deploy an MSI using the LOB method in Intune:

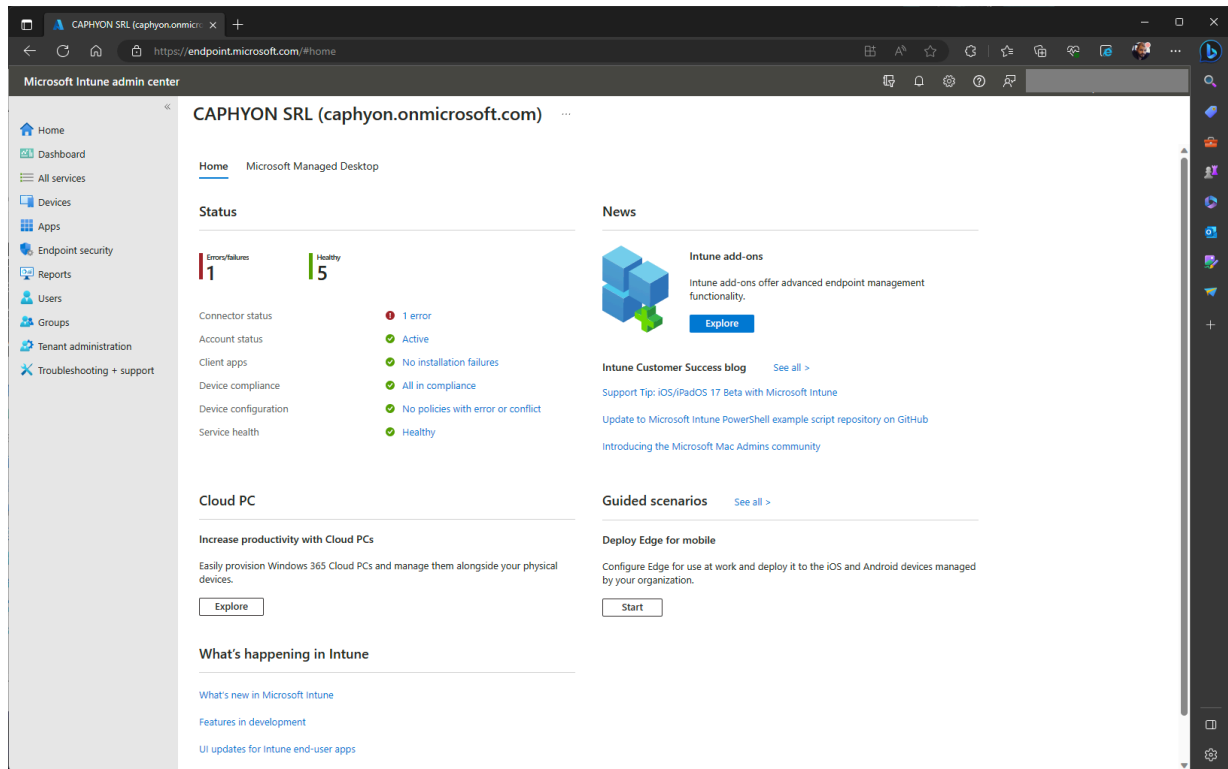
### Step 1: Prepare the MSI Package

Of course the first step is to create and prepare the MSI package with all the necessary configurations and adjustments before deploying it. In our case we will use the repackaged VLC Media Player MSI.

### Step 2: Upload the MSI to Intune

- Sign in to the [Microsoft Endpoint Manager admin center](#) with your Intune administrator credentials.

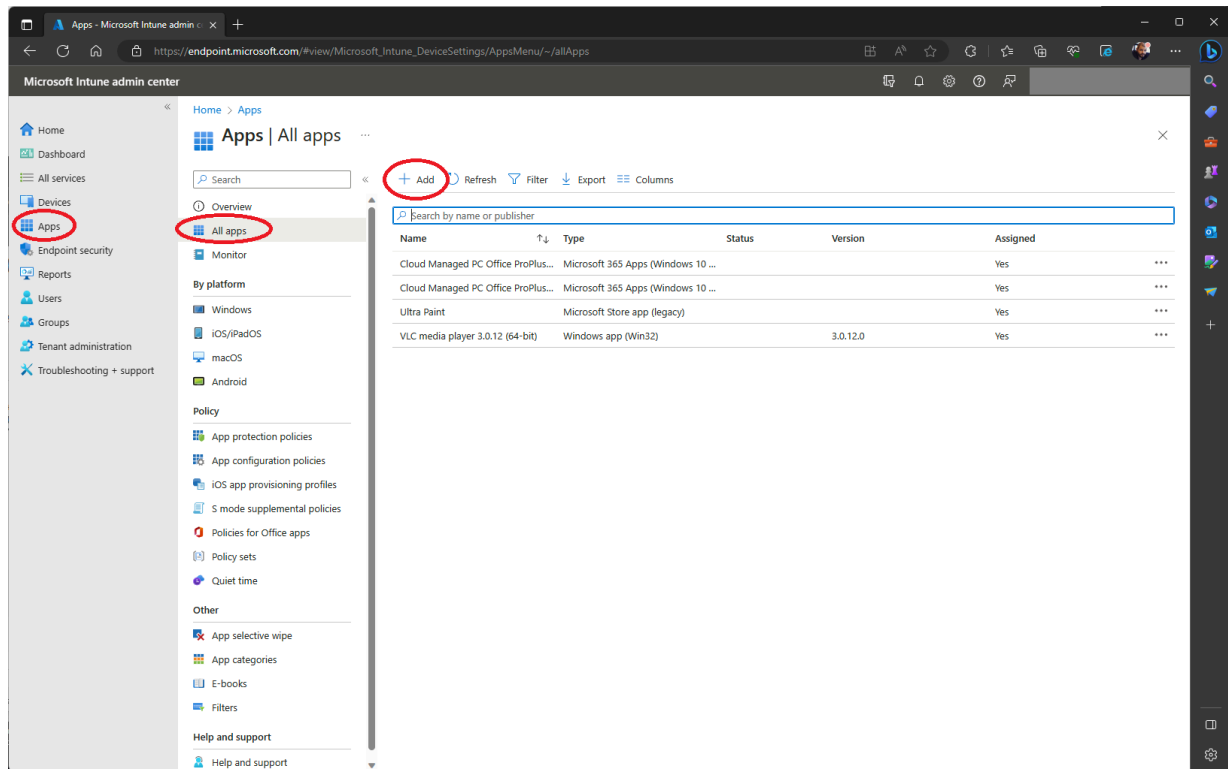




- Navigate to "Apps" > "All apps." Click on "Add" to add a new app.

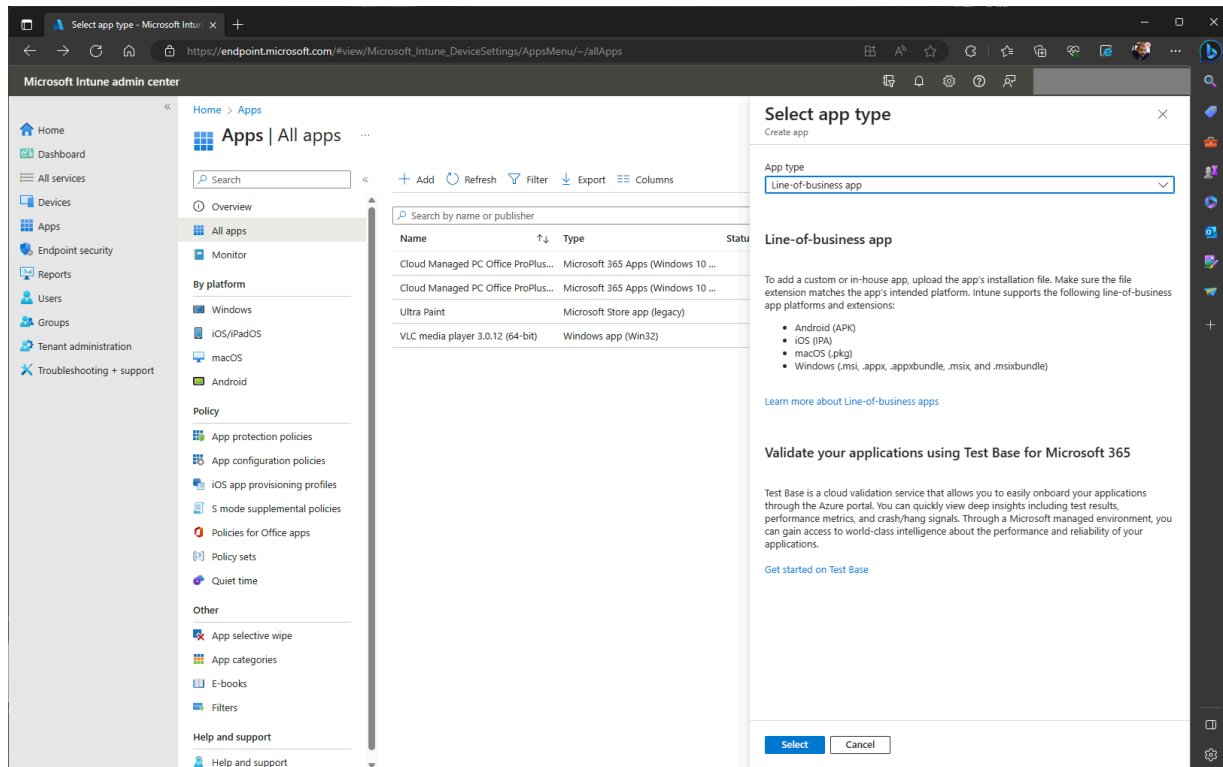






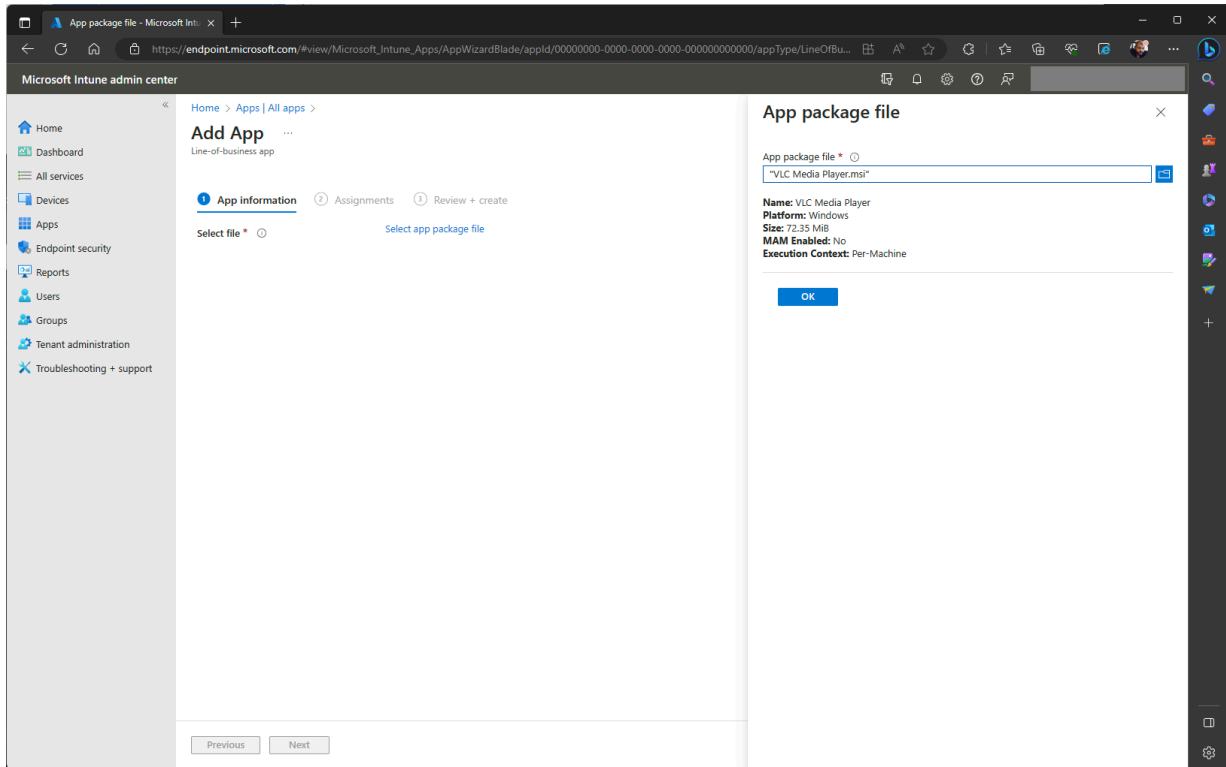
- Select "Line-of-business app" as the app type.





- Select the App Package File





### Step 3: Configure the App Details

- Provide the necessary details, such as the app name, description, and publisher information.



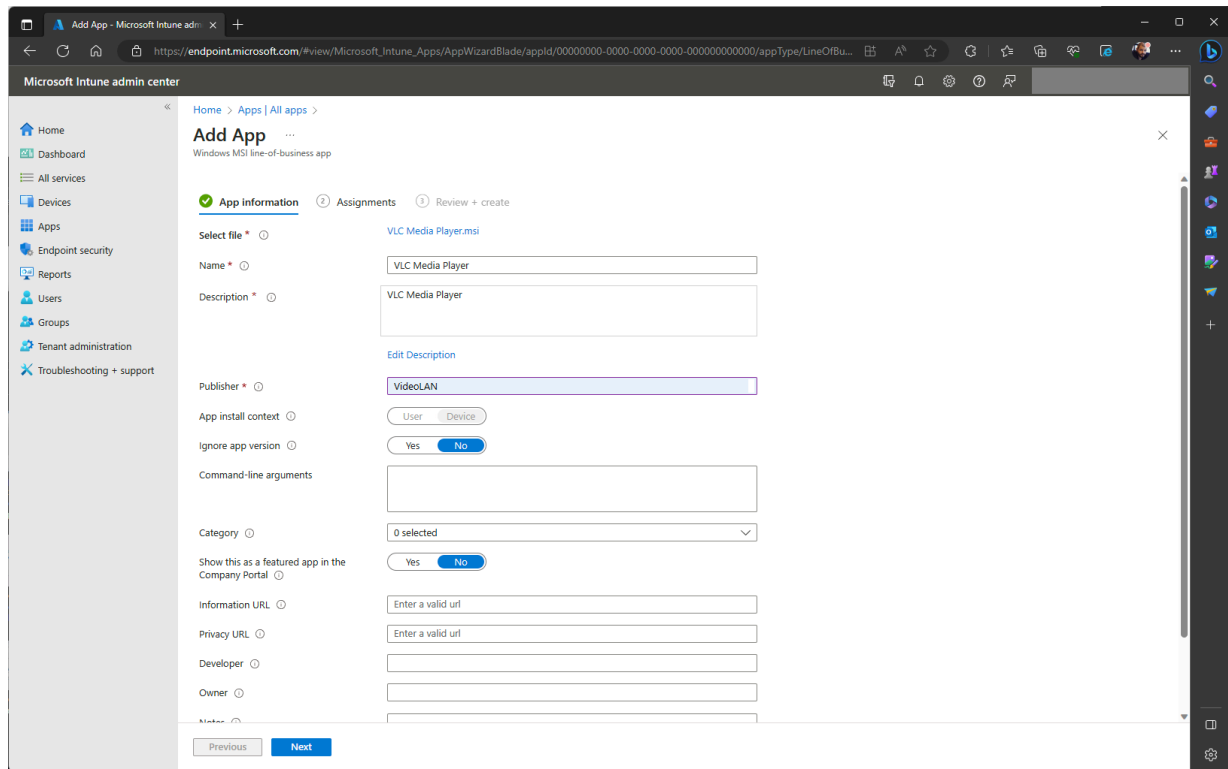


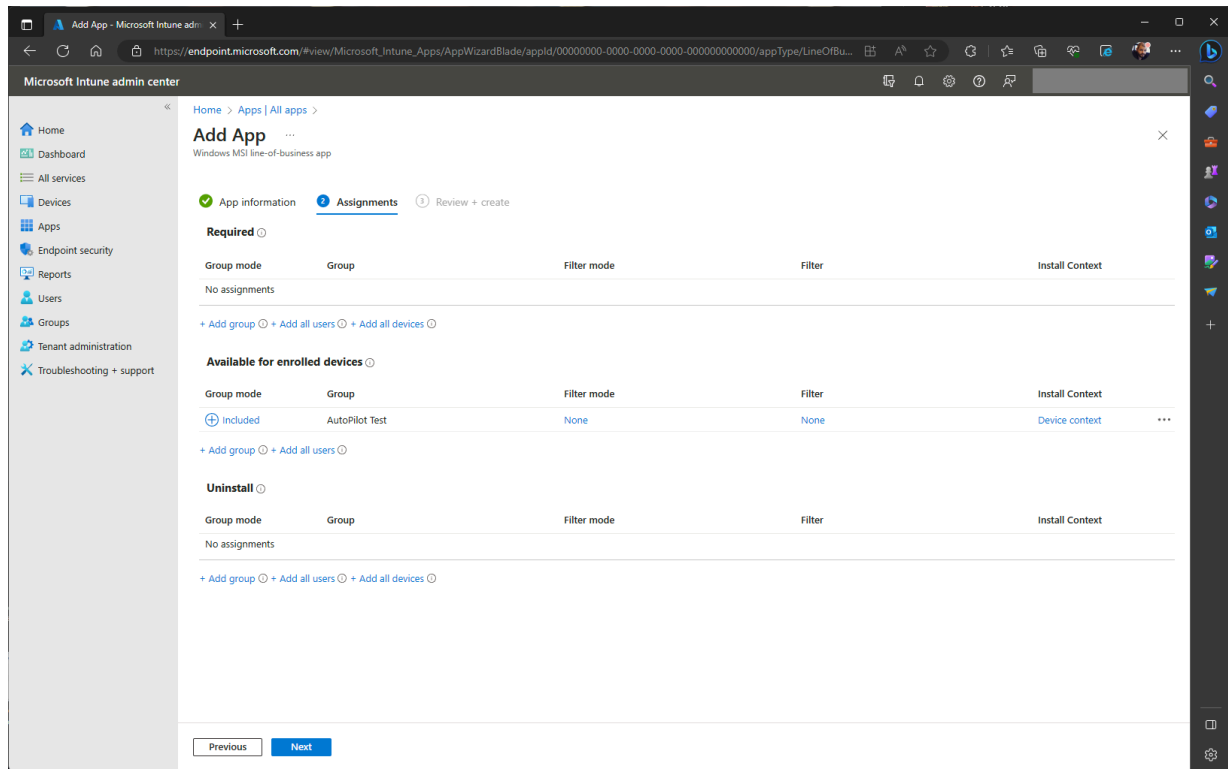
image-id	intune-mem-admin-center-loba-app-details.png
image-title	
image-alt-text	

As you can see, just as the case with deploying MSI packages with SCCM, no detection method is requested because Intune automatically picks up the Product Code of the MSI package and uses it as a detection to check if the application has been successfully installed on the target devices/users.

#### Step 4: Assign the App to Groups

- Click on "Assignments" to assign the app to specific user groups or device groups and choose the appropriate groups based on your deployment requirements.

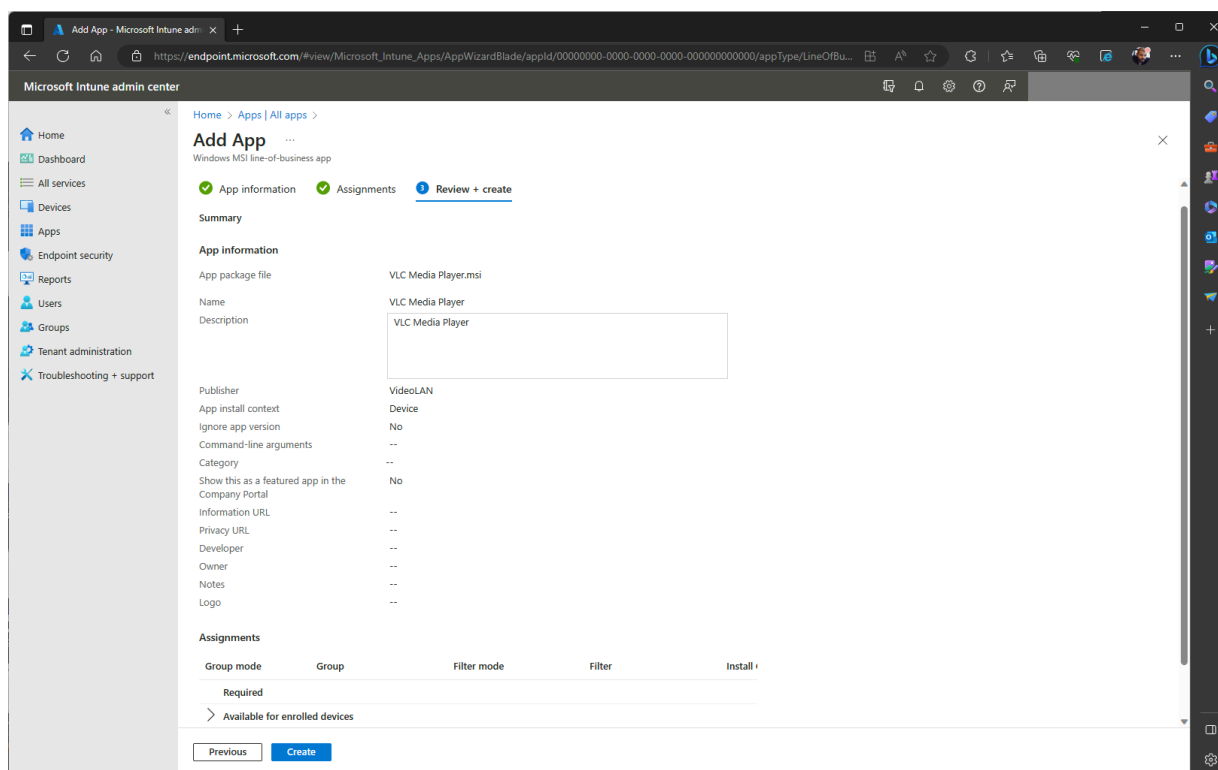




## Step 5: Review and Deploy the App

- Review the app settings and ensure everything is configured correctly. If everything is correct click on Create.





## Deploy EXE/VBScript/PowerShell via Win32

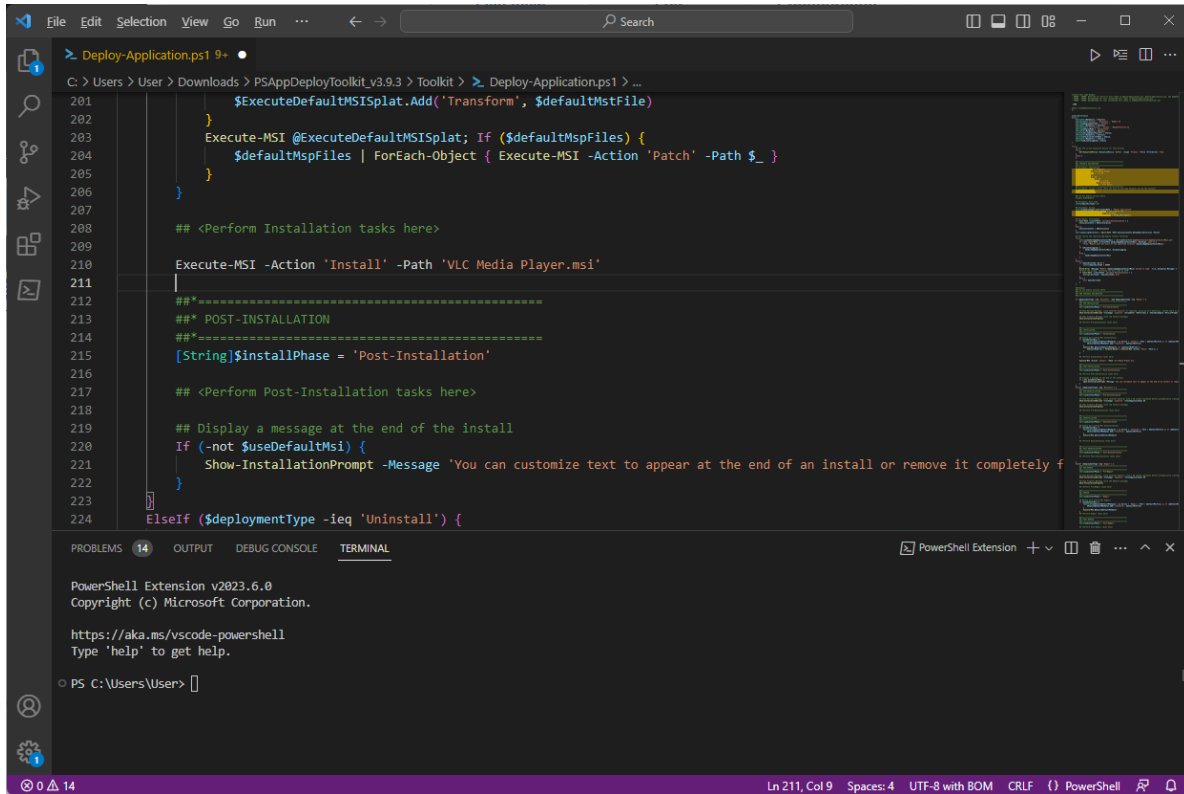
Deploying EXE installers, VBScript or PowerShell wrappers via the Win32 method requires more steps with Intune as it requires more steps with SCCM, so let's take a look at what is necessary to create and deploy a Win32 Application.

Let us assume that we have created a PowerShell wrapper for our VLC Media Player repackaged application with [PSADT](#).

### Step 1: Prepare the PowerShell Script

Create or obtain the PowerShell script that you want to deploy. Ensure that the script performs the desired actions and is compatible with the target devices and platforms. We won't go through all the steps on creating and modifying the PSADT template in this example. For more information [check out our first MSI Packaging Ebook](#).





```
281     $ExecuteDefaultMSISplat.Add('Transform', $defaultMstFile)
282   }
283   Execute-MSI @ExecuteDefaultMSISplat; If ($defaultMspFiles) {
284     $defaultMspFiles | ForEach-Object { Execute-MSI -Action 'Patch' -Path $_ }
285   }
286 }
287
288 ## <Perform Installation tasks here>
289
290 Execute-MSI -Action 'Install' -Path 'VLC Media Player.msi'
291
292 ##*=====
293 ##* POST-INSTALLATION
294 ##*=====
295 [String]$installPhase = 'Post-Installation'
296
297 ## <Perform Post-Installation tasks here>
298
299 ## Display a message at the end of the install
300 If (-not $useDefaultMsi) {
301   Show-InstallationPrompt -Message 'You can customize text to appear at the end of an install or remove it completely f
302 }
303
304 ElseIf ($deploymentType -ieq 'Uninstall') {
```

PowerShell Extension v2023.6.0  
Copyright (c) Microsoft Corporation.  
  
https://aka.ms/vscode-powershell  
Type 'help' to get help.

PS C:\Users\User> |

## Step 2: Package the PowerShell Script

When it comes to Win32 applications in Intune, you can't just upload the source media as it is and this must be converted to an .intunewin format using the [Microsoft Win32 Content Prep Tool](#). The Microsoft Win32 Content Prep Tool is a command-line utility provided by Microsoft that assists in the preparation of Win32 app packages for deployment via Microsoft Endpoint Manager (formerly known as Microsoft Intune). It is intended to streamline the packaging process and ensure that Win32 app packages meet the requirements for enterprise deployment.

First, download the tool from the official Github repository. Once the tool is downloaded, extract it from the zip file. Next, open up a command prompt and use the following command:

```
IntuneWinAppUtil -c <setup_folder> -s <source_setup_file> -o  
<output_folder> <-q>
```

The .intunewin file will be generated from the specified source folder and setup file. This tool will retrieve the necessary information for Intune from an MSI setup file. If the -a option is used, all catalog files in that folder are bundled into the .intunewin file.



It will be in quiet mode if -q is specified. The output file will be overwritten if it already exists. In addition, if the output folder does not already exist, it will be created.

```
Microsoft Windows [Version 10.0.22621.1778]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>C:\Users\User\Downloads\Microsoft-Win32-Content-Prep-Tool-master\IntuneWinAppUtil.exe -c C:\Users\
User\Downloads\PSAppDeployToolkit_v3.9.3\Toolkit -s "C:\Users\User\Downloads\PSAppDeployToolkit_v3.9.3\Toolkit\Dep
loy-Application.ps1" -o C:\Users\User\Downloads\PSAppDeployToolkit_v3.9.3\Toolkit\output -q
INFO Validating parameters
INFO Validated parameters within 11 milliseconds
INFO Compressing the source folder 'C:\Users\User\Downloads\PSAppDeployToolkit_v3.9.3\Toolkit' to 'C:\Users\User
\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Contents\IntunePackage.intunewin'
INFO Calculated size for folder 'C:\Users\User\Downloads\PSAppDeployToolkit_v3.9.3\Toolkit' is 77776188 within 8
milliseconds
INFO Compressed folder 'C:\Users\User\Downloads\PSAppDeployToolkit_v3.9.3\Toolkit' successfully within 1735 mill
iseconds
INFO Checking file type
INFO Checked file type within 8 milliseconds
INFO Encrypting file 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Con
tents\IntunePackage.intunewin'
INFO 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Contents\IntunePack
age.intunewin' has been encrypted successfully within 159 milliseconds
INFO Computing SHA256 hash for 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPa
ckage\Contents\3295de55-9096-4450-91ce-a823eac7b042' within 857 milliseconds
INFO Computed SHA256 hash for 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPa
ckage\Contents\3295de55-9096-4450-91ce-a823eac7b042' within 857 milliseconds
INFO Computing SHA256 hash for 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPa
ckage\Contents\IntunePackage.intunewin'
INFO Copying encrypted file from 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWi
nPackage\Contents\3295de55-9096-4450-91ce-a823eac7b042' to 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Contents\IntunePackage.intunewin'
INFO Copying encrypted file from 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWi
nPackage\Contents\IntunePackage.intunewin'
INFO Computing SHA256 hash for 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPa
ckage\Contents\IntunePackage.intunewin'
INFO Computed SHA256 hash for 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPa
ckage\Contents\IntunePackage.intunewin'
INFO Copying encrypted file from 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWi
nPackage\Contents\IntunePackage.intunewin' to 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Contents\IntunePackage.intunewin'
INFO File 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Contents\IntunePackage.intunewin' got updated successfully within 82
milliseconds
INFO Generating detection XML file 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Metadata\Detection.xml'
INFO Generated detection XML
INFO Compressing folder 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage' to 'C:\Users\User\Downl
oads\PSAppDeployToolkit_v3.9.3\Toolkit\output\Deploy-Application.intunewin'
INFO Calculated size for folder 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage' is 75396262 within 8 milliseconds
INFO Compressed folder 'C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage' successfully within 302 milliseconds
INFO Removing temporary files
INFO Removed temporary files within 54 milliseconds
INFO File 'C:\Users\User\Downloads\PSAppDeployToolkit_v3.9.3\Toolkit\output\Deploy-Application.intunewin' has been generated successfully

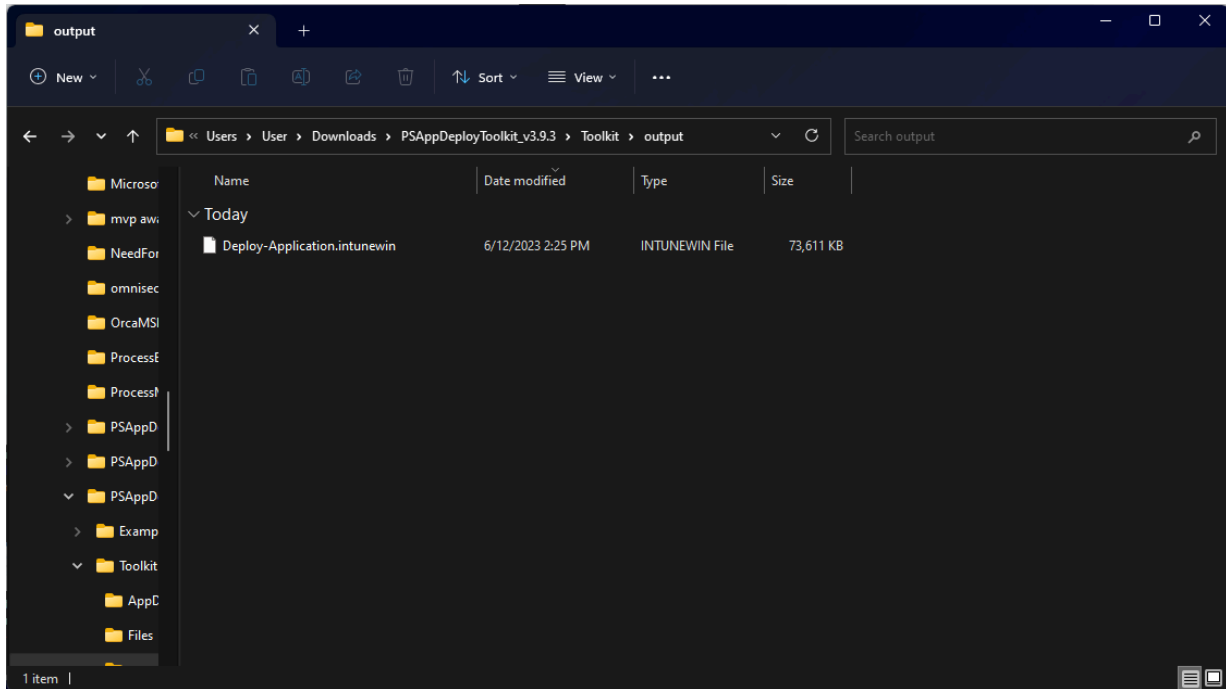
[=====] 100%
INFO Done!!!
C:\Users\User\AppData\Local\Temp\{e219a00b-7d1b-43df-bde9-e63194db0b36}\IntuneWinPackage\Contents\IntunePackage.intunewin'
```

The Microsoft Win32 Content Prep Tool does not have a GUI, but if you want one you can download the [IntuneWinAppUtil GUI utility for free](#).

Once the conversion is successful, the output folder should contain the .intunewin file necessary to upload to Intune.



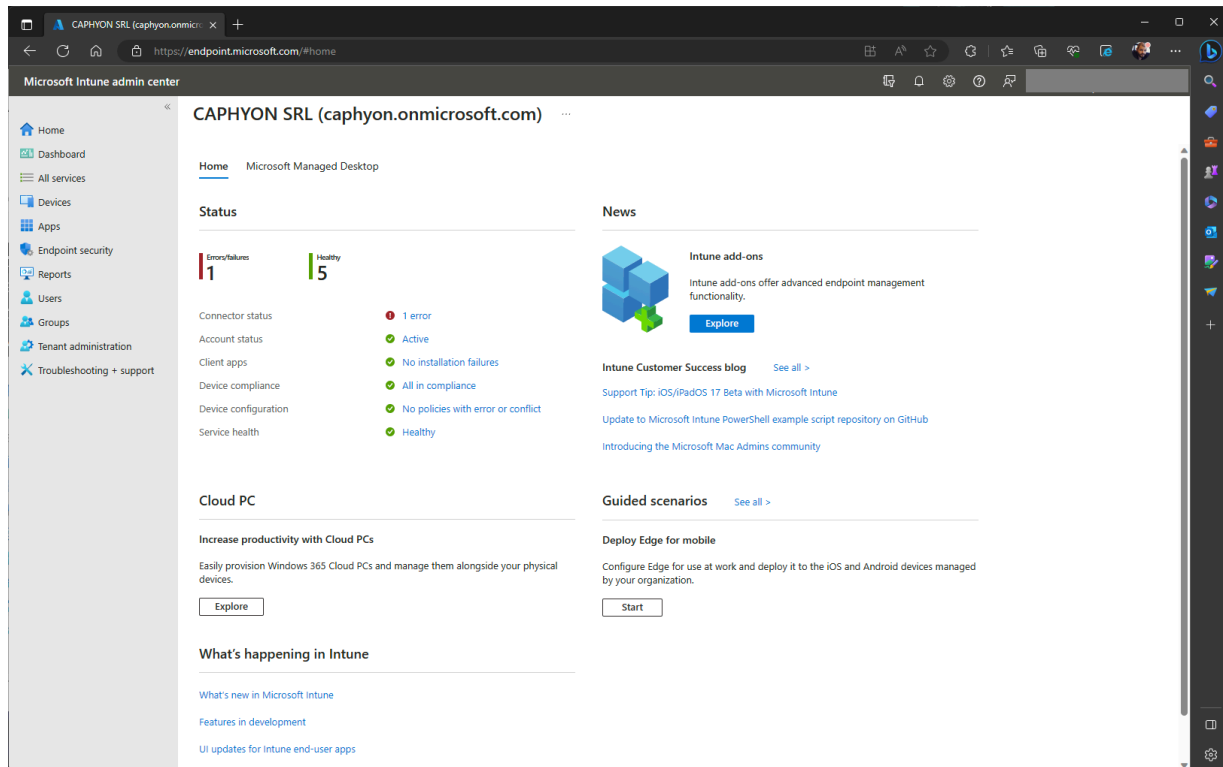




### Step 3: Create the Intune Win32 App Package

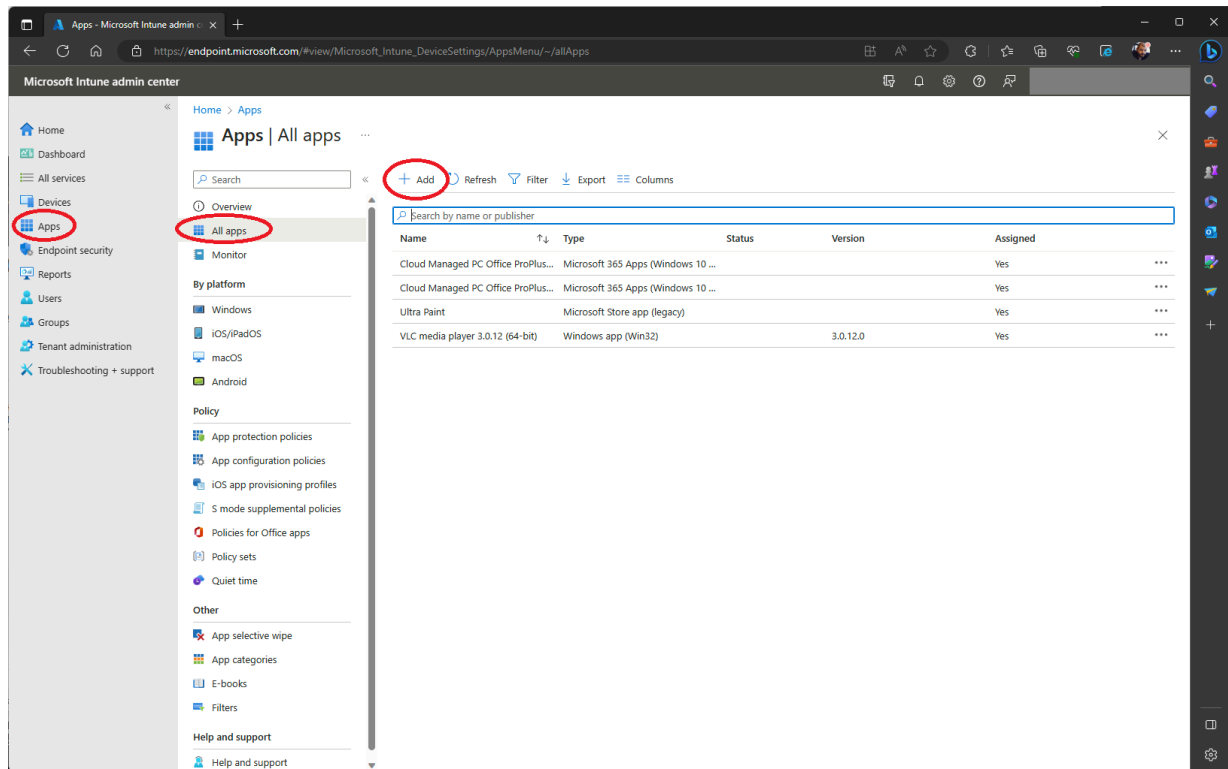
- Sign in to the [Microsoft Endpoint Manager admin center](#) with your Intune administrator credentials.





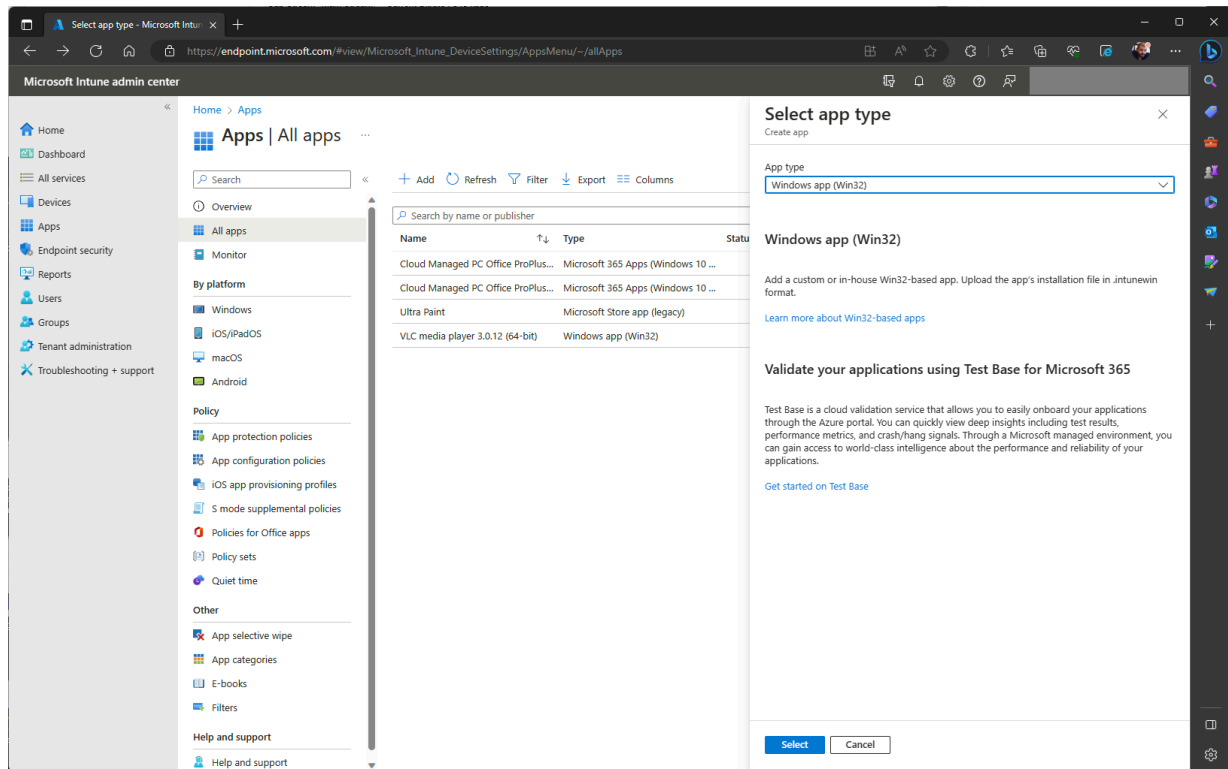
- Navigate to "Apps" > "All apps." Click on "Add" to add a new app.





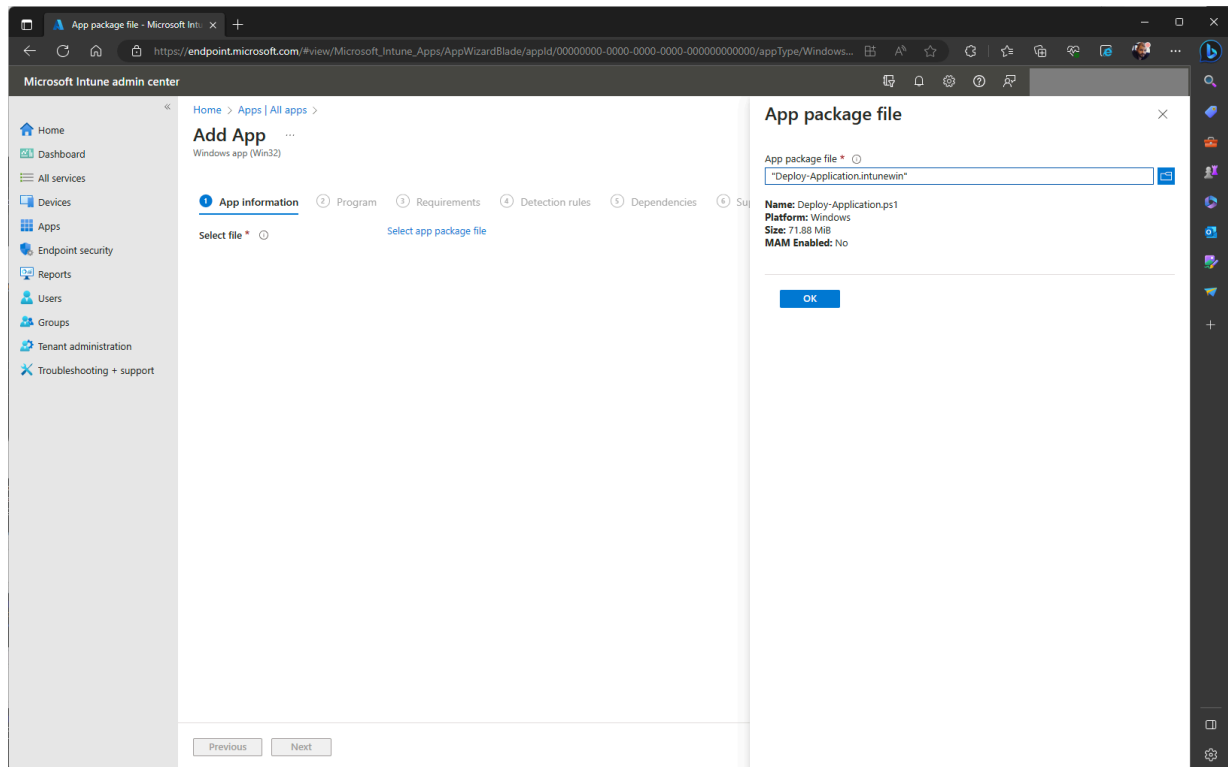
- Select "Windows app (Win32)" as the app type.





- Select the App Package File, in our case the .intunewin file created earlier





#### Step 4: Configure the App Details

- Provide the necessary details, such as the app name, description, and publisher information



Microsoft Intune admin center

Home > Apps | All apps >

## Add App

Windows app (Win32)

1 App information 2 Program 3 Requirements 4 Detection rules 5 Dependencies 6 Supersedence 7 Assignments 8 Review + create

Select file \* [Deploy-Application.intunewin](#)

Name \*

Description \*

[Edit Description](#)

Publisher \*

App Version

Category

Show this as a featured app in the Company Portal ☐ Yes ☒ No

Information URL

Privacy URL

Developer

Owner

Notes

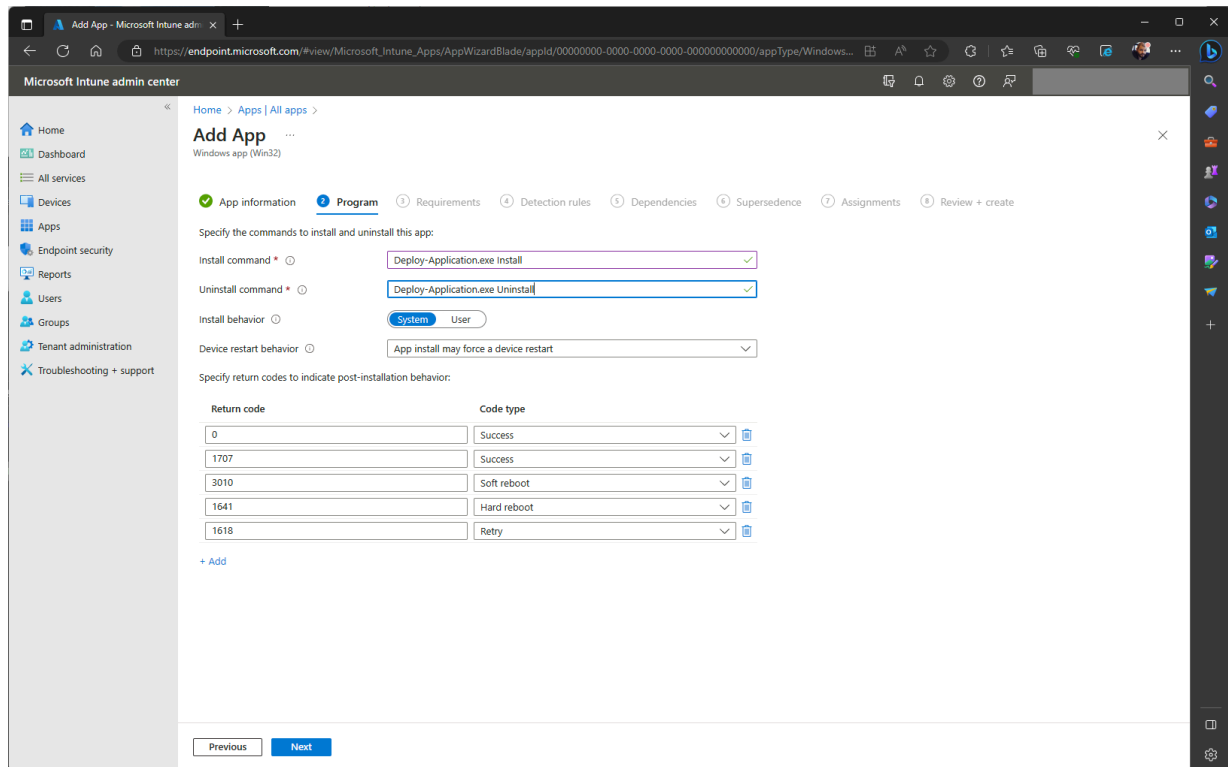
Logo  [Select image](#)

[Previous](#) [Next](#)

## Step 5: Define the Installation parameters and behaviors

- Fill in other relevant information, such as the installation command and uninstall command. Most of the behavior is similar to SCCM methods

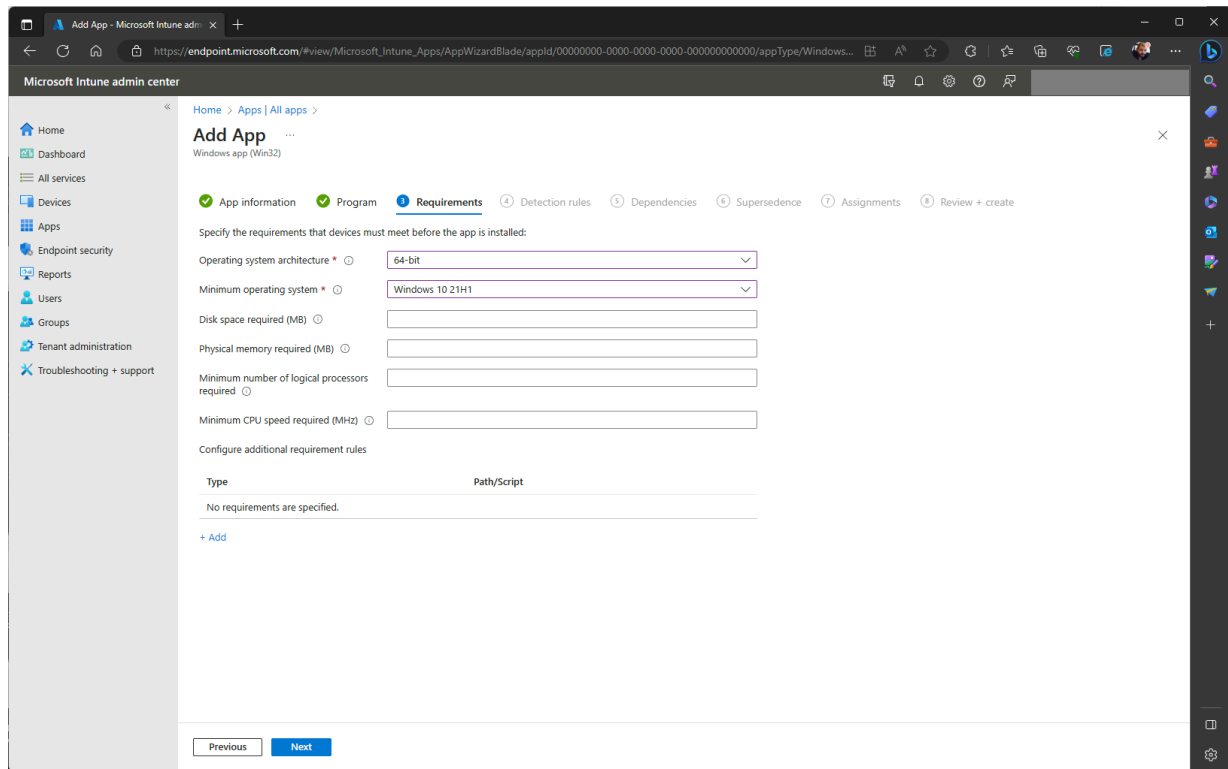




## Step 6: Define Application Requirements

- Specify any requirements or dependencies for the app, such as minimum operating system versions or device architectures.



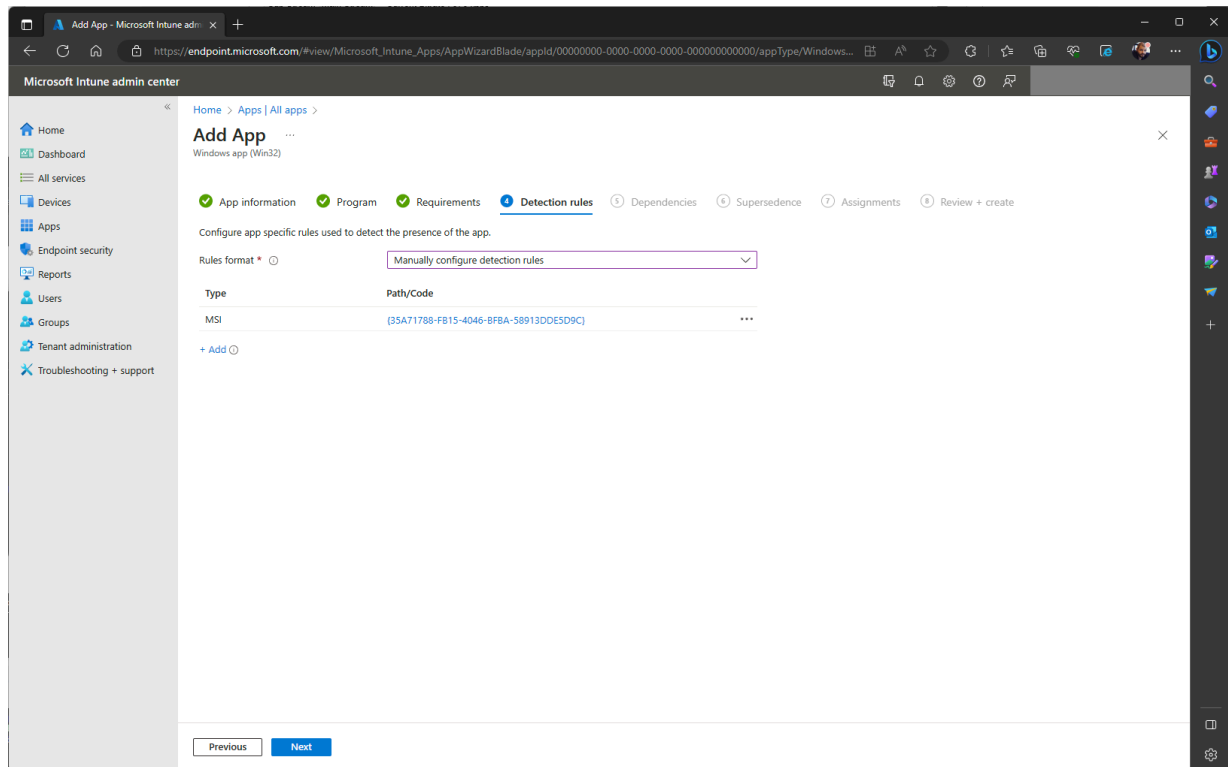


## Step 7: Define the Detection Method

- Specify the detection method for the app, which determines whether the app is already installed on the target device. Because we have an MSI we can still use the Product Code of our MSI as a detection method



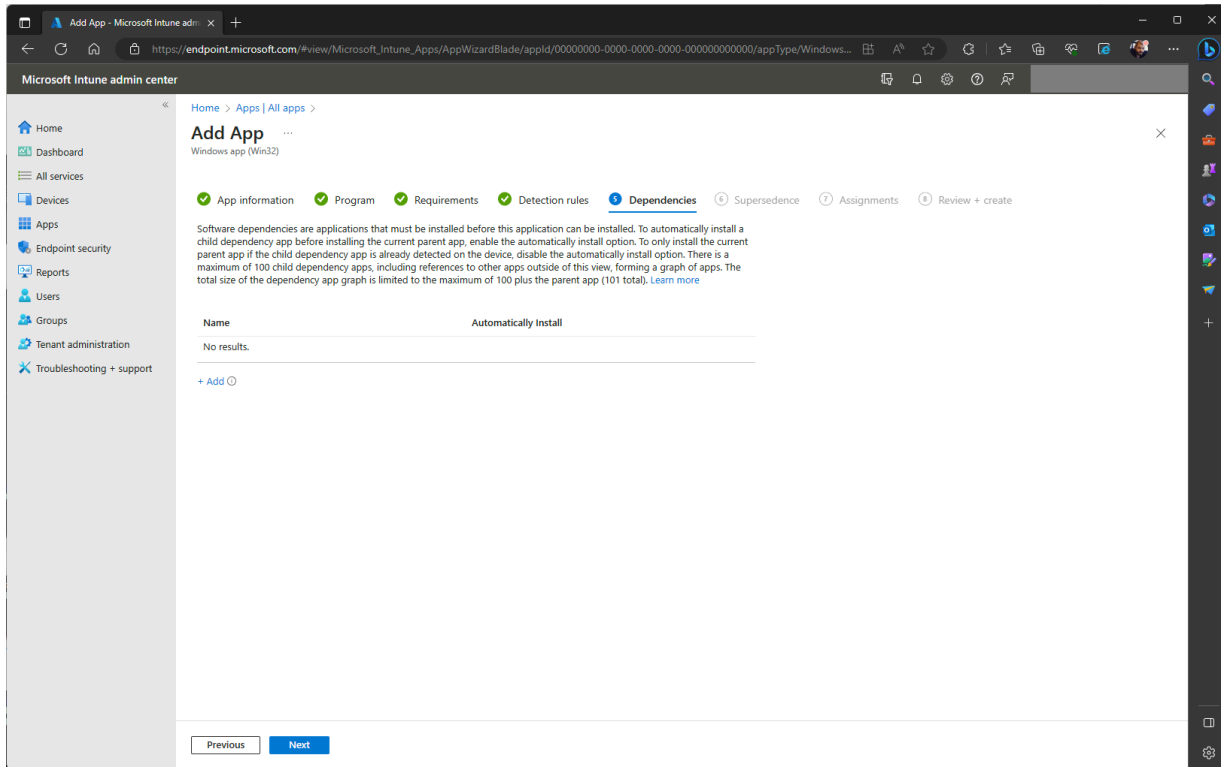




## Step 8: Configure Dependencies

- If required, add any dependencies to ensure the app is deployed correctly



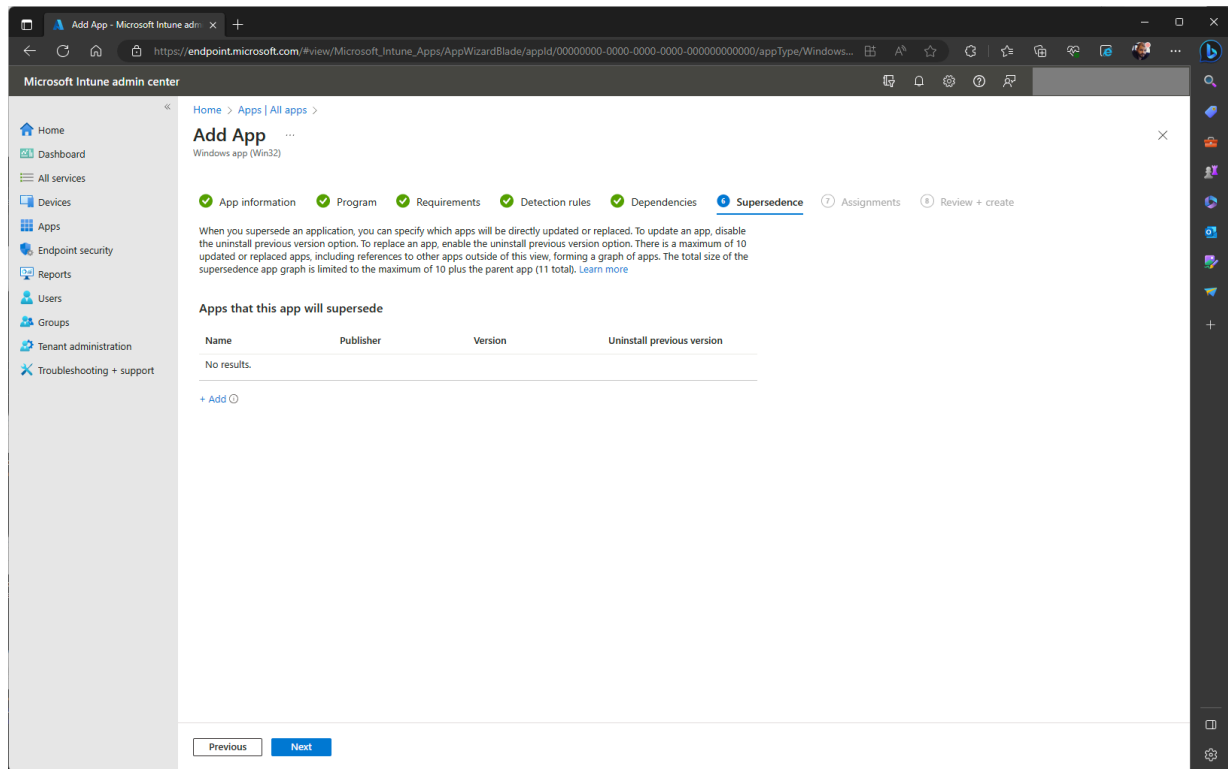


## Step 9: Configure Supersedence

Supersedence in Intune apps refers to the ability to replace or upgrade an existing deployed application with a newer version. It allows you to manage the lifecycle of applications by automatically detecting and handling updates or upgrades to applications in your environment.

When an application is superseded, it means that a new version of the application is available, and Intune will handle the process of replacing the older version with the new one on targeted devices. Supersedence helps ensure that devices stay up to date with the latest versions of applications, providing improved functionality, security updates, and bug fixes.

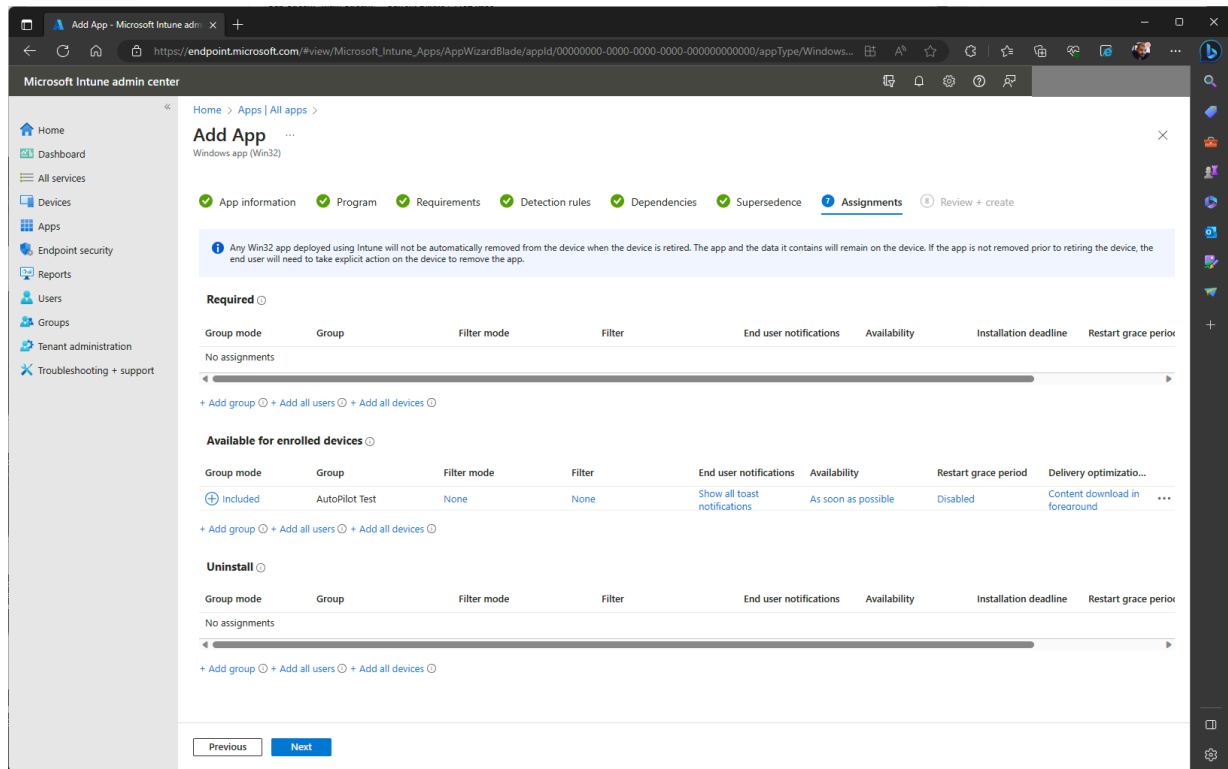




## Step 10: Assign the App to Groups

- Click on "Assignments" to assign the app to specific user groups or device groups and select the appropriate groups based on your deployment requirements.

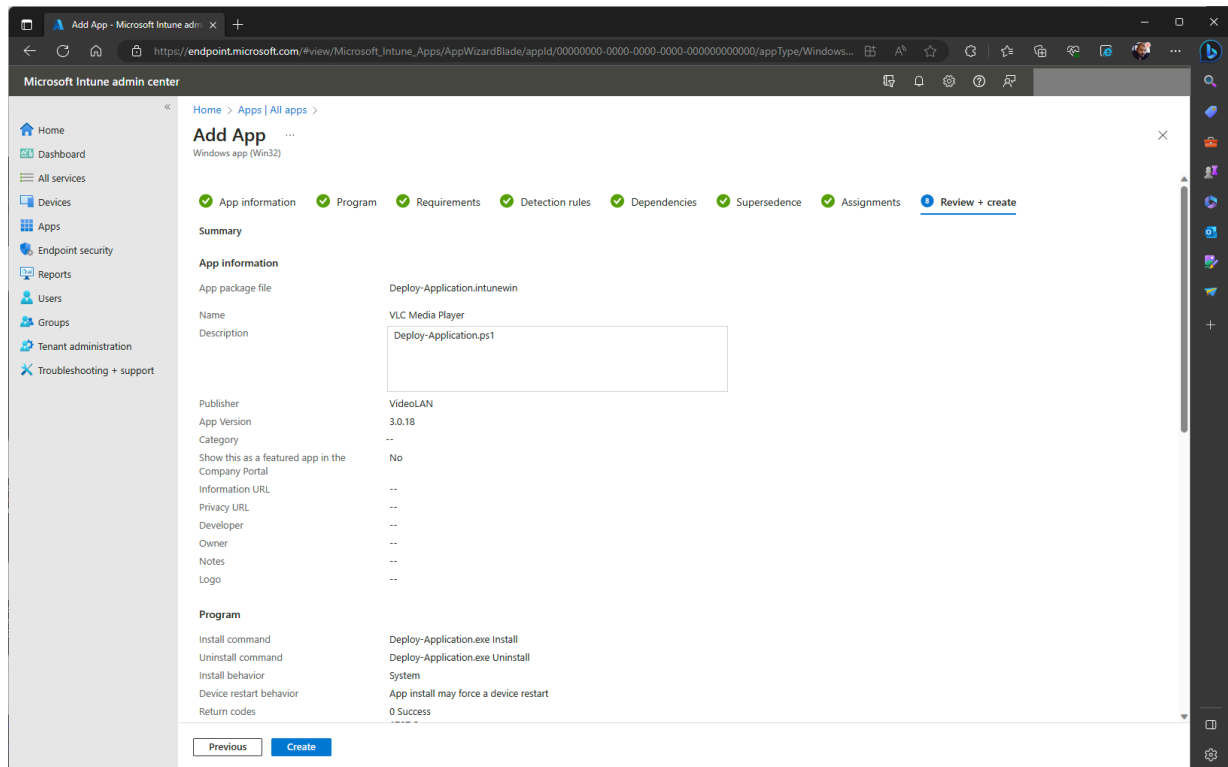




## Step 8: Review and Deploy the App

Review the app settings and ensure everything is configured correctly. If everything is correct click on Create.





As you can see, the Win32 deployments are more lengthy and require more steps, but if you look closely it gives you a more granular view and control of your application deployment, hence why this method has been widely adopted by the IT Professionals.



# Final Words

This book has provided a comprehensive exploration of various topics related to software installation and deployment. We have covered a wide range of concepts, tools, and techniques that are essential for any software developer, system administrator, or IT professional involved in the deployment process.

Throughout the book, we have discussed the importance of proper software packaging, the role of MSI technology, and the benefits of using advanced installation tools like Advanced Installer. We have explored best practices for creating MSI packages, customizing installations, handling upgrades and patches, and managing dependencies. We have also delved into advanced topics such as repackaging, Windows Installer customization, scripting, and automation. By understanding these advanced techniques, readers will be equipped with the knowledge and skills needed to streamline their software deployment processes, reduce errors, and improve overall efficiency.

Moreover, we have covered the use of SCCM for deploying MSI packages in enterprise environments, ensuring centralized management and control over software installations. The integration of SCCM with MSI technology provides a robust solution for large-scale software deployment and maintenance.

As we conclude this book, it is important to remember that the software deployment landscape is constantly evolving. New technologies, methodologies, and best practices emerge regularly, and it is crucial to stay updated with the latest advancements in the field. Continual learning and adaptation are key to ensuring smooth and successful software deployments in an ever-changing IT landscape.

We hope that this book has served as a valuable resource, providing practical insights and guidance to help you navigate the complex world of software installation and deployment.

Remember, successful software deployment is not just about the technical aspects; it is also about understanding the end-users, their needs, and delivering a seamless experience. With the right tools, knowledge, and mindset, you can make software deployment a smooth and rewarding process for both the developers and end-users.

Thank you for joining us on this journey, and we wish you all the best in your future software deployment endeavors!

Happy deploying!



# About the Author

Dive deep into the world of Microsoft Installer (MSI) with industry expert Alex Marin. This comprehensive guide unveils advanced techniques and best practices to streamline your application deployment processes. From intricate dependency management to automation strategies, “Advanced Techniques in MSI Packaging” empowers IT professionals to elevate their packaging skills to the next level.

Alex Marin brings over two decades of hands-on experience in IT engineering and management, specializing in application packaging and repackaging. His work has set industry standards for efficiency and reliability in software deployment, making him a trusted voice in the community.

Through this book, Alex shares his vast knowledge, practical insights, and innovative approaches to mastering MSI packaging.



## Alex Marin

IT Pro | Packaging Lead | Author

Follow Alex on

 YouTube ·  Advanced Installer Blog