Working with Windows Installer

— A Practical Guide to MSI —

By Nelson Ruest & Danielle Ruest

Edited By Bob Kelly

Les Entreprises Resolutions Ltd. Enterprises

AppDeploy.com

Abstract

Windows Installer is a comprehensive application installation system. Because it is so powerful, Windows Installer caw be complex to use. This guide demystifies Windows Installer and explains its concepts and principles in plain English. If you need to work with this services then read this guide. It will provide you with a complete overview of this powerful tool.

About the Authors

Danielle Ruest and Nelson Ruest are IT professionals specializing in systems administration, migration planning, software management and architecture design. They are authors of multiple books, and are currently working on the **Definitive Guide to Vista Migration** (www.realtime-nexus.com/dgvm.htm) for Realtime Publishers as well as the **Complete Reference to Windows Server Codenamed "Longhorn"** for McGraw-Hill Osborne. They have extensive experience in systems management and operating system migration projects.

Bob Kelly is the founder of **AppDeploy.com** where he produces the AppDeploy Library which contains an extensive document library and hours of video presentations on system and application deployment topics. Bob is also co-founder and president of iTripoli, Inc. who offer the **AdminScriptEditor** (www.adminscripteditor.com), a powerful suite of scripting tools for Windows Administrators. He has authored and edited several books and papers on the topics of scripting and desktop administration and is editor for a **Windows Vista community website** by Realtime Publishers (www.realtime-vista.com). For more on the AppDeploy Library, please visit www.appdeploy.com/library.

Les Entreprises Resolutions Ltd. Enterprises
www.Reso-Net.com

AppDeploy.com
AppDeploy.com is provided as a Service of RWK Systems, Inc.
www.appdeploy.com

Table of Contents

1. Working with Windows Installer

Use this powerful service to manage the package lifecycle in your network.

Most everyone who is involved in software management, whether it be software distribution or packaging, has heard of the Microsoft Windows Installer service (WIS). This powerful service has been created by Microsoft to help manage the software lifecycle on Windows systems. Today, Windows Installer is in its fourth edition. Version 4.0 has been specifically designed to run on Windows Vista and Windows Server Codenamed "Longhorn". Windows Installer version 3.1 runs on earlier operating systems that are based on the Windows code developed for Windows 2000 and beyond. That means it will work with Windows 2000 Service Pack 3 and above, Windows XP Service Pack 2 and Windows Server 2003. If you're running an earlier version of Windows, you'll need to use an earlier version of Windows Installer.

Windows Installer is available in a variety of formats; users of Windows Vista will find it embedded in the operating system while users of the other supported operating systems can use the redistributable download of Windows Installer which is available at the Microsoft Download Web site. Users of the .NET Framework, especially the latest edition of the .NET Framework, will also require the addition of the appropriate version of Windows Installer.

Note: For a full list of Windows Installer versions and its corresponding operating system, go to http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/released_versions_of_windows_installer.asp.

The Windows Installer redistributable download can be found at http://www.microsoft.com/downloads/details.aspx?FamilyID=889482fc-5f56-4a38-b838-de776fd4138c&DisplayLang=en.

But why use this service? If you're aware of the service, you're most likely aware of some of its features. First and foremost, Windows Installer provides a consistent, single point of interaction for commercial software or in-house application installations. This is a major change from the pre-Windows Installer days when system administrators and packagers had to deal with a multitude of installation tools, each with its own particular commands and its own particular idiosyncrasies. Using Windows Installer for installations is one way to reduce administrative overhead for software management because you only have to learn one single installation method. Of course, not all software or all in-house applications are integrated to the Windows Installer service and this despite the fact that it has been around for over four years. This is one reason why you use a packaging tool to prepare and customize your installations.

Second, Windows Installer provides a set of features that tie in very closely with the software lifecycle. As you know, software has its own lifecycle and it is your job to manage this lifecycle once a piece of software has entered into your network. This lifecycle and the relationship the Windows Installer service has with it are illustrated in Figure 1.

The third and most important aspect of the Windows Installer service is that it provides a series of features that were heretofore unavailable through conventional installation systems. Much of this functionality is due to the fact that Windows Installer actually stores an installation database on the target computer system each time it installs a piece of software. This database contains information about the installation, about the components that were installed by the installation and about the way those components were configured during installation.

This gives WIS the ability to provide a comprehensive set of features in support of this installation.

For example, because WIS includes the computer's pre-installation state in its database, it can support complete installation rollbacks in the case of a problem during installation, returning the target system to the same state it was in before the installation began. In addition, because it stores the software configuration in its database, it can automatically repair an installation should a problem occur with the program. This repair mode can be run through a maintenance mode, but it is also automatically run each time a user launches a program through its shortcuts or through the opening of a document generated by the program. And, for managed environments, it can automatically elevate a user's privileges during installation to insure that the installation will occur properly in locked-down environments. That's because Windows Installer is a service that runs in the background and is therefore always available. Finally, because of this database, it can completely remove an application from a system when it is time to retire or upgrade a software program from a network.

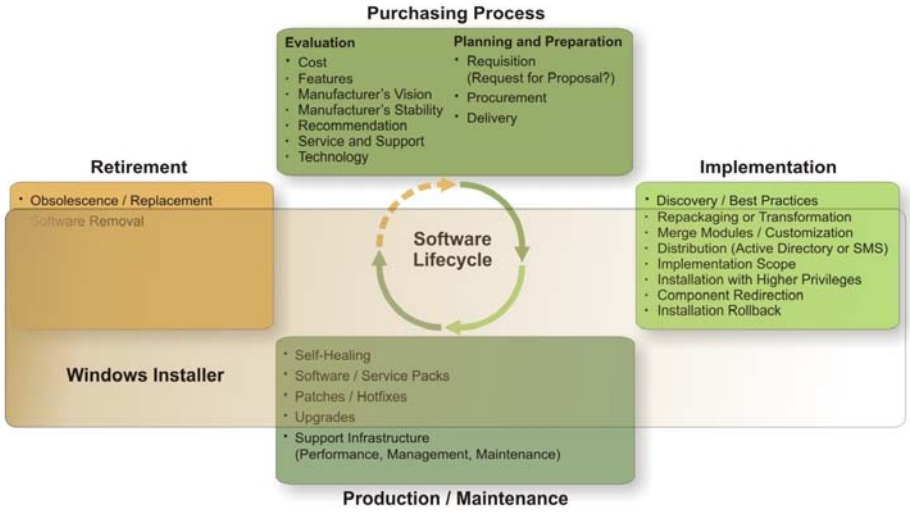


Figure 1: The interaction of MSI with the software lifecycle.

These are only some of the features that make Windows Installer a powerful installation system. These features are part of the reason why you should endeavor to integrate all of your installations to this potent service. Most, but not all, new software products on the market are now provided as MSI packages for these reasons.

1.1 Integrating Installations with the Windows Installer Service

This does not mean that you need to go out and buy a new version of all of the software products in your network. An average-sized network, say between 500 and 5,000 users, will most likely have between 100 to 300 software programs and applications in use within it. Some organizations of this size may have over 1,000 such programs in use. This is before they perform a software rationalization—a formal exercise that reviews and justifies the

existence of each software product or application within the network[1]. If you haven't done so yet, it is highly recommended that you perform such a rationalization in your network. This means getting rid of any programs that duplicate features or multiple versions of the same program. This will greatly reduce your software administration burden and potentially reduce licensing costs.

It is unrealistic to expect any organization to be able to simply go out and purchase new versions of each software product in their network in order to have versions that are integrated with the Windows Installer service. That's because of several reasons:

- The cost would be too prohibitive.
- New versions of your in-house applications aren't available on the market, you have to build them and doing so may also be cost-prohibitive.
- Some manufacturers simply don't offer new versions of their products.
- Though they are becoming fewer and fewer, some manufacturers still haven't integrated their software products to Windows Installer.

Given these reasons, you'll have to consider your options for moving to Windows Installer-integrated installations. The first thing you should do is categorize your software into the following three program types:

- **Native Windows Installer software:** This software includes any product that bears the Designed for Windows Server 2003, Designed for Windows XP or Designed for Windows 2000 logos or any software that does not include this logo, but has been set up to be installed through Windows Installer. Obviously, software that supports the logo may be more properly behaved in your network than software that does not include it. That's because the logo specifications include much more than Windows Installer integration[2].
- **MSI-integrated Corporate Applications**: New versions of your corporate applications should be integrated to the Windows Installer service in all cases.
- **Repackaged Legacy Software**: All products that are not upgraded and use an installation system other than Windows Installer should be repackaged to be integrated to this service. This also includes corporate applications that do not require recoding or cannot be recoded as well as legacy commercial software.

Next you should learn more about Windows Installer to see how is can help manage software in your network.

---

[1] For more information on the software rationalization process, see "Preparing for .NET Enterprise Technologies" by Ruest and Ruest, Addison-Wesley, ISBN: 0-201-73487-7.

[2] For more information on Microsoft Logo specifications, see http://www.microsoft.com/windowsserver2003/partners/isvs/cfw.mspx.

1.2 The Windows Installer Service

Like all system services, Windows Installer is a service listed in the Services section of the Computer Management console. This service is set to a manual startup and is activated only when you launch an installation that is integrated to it. This automatically starts the service and runs Windows Installer to perform the installation. You should not change the settings of this service because they are controlled by the operating system.

In addition, you need to know which version of the Windows Installer service you are running. Obviously, the newest version includes the most comprehensive feature set. To find out which version you are running, search for MSI.DLL in the Windows folder. Once you locate it, you can verify its properties to view which version you are running. Figure 2 shows the version number for this file.

Another and even easier way to find out which version you have installed is to simply type one of the following commands at the command prompt:

```
MSIEXEC /?
MSIEXEC /HELP
```

This will display a dialog box that lists all of the switches supported by the command as well as displaying the installed version of the service (see Figure 3).

As mentioned above, there are several different versions of this service. The latest is version 4.0 which only runs on Windows Vista or Windows Server “Longhorn”. Windows installer version 3.1 is the version that runs on Windows versions that are newer than Windows 2000 service pack 3 or above. If you have an OS that is older than this, you’ll have to work with WIS version 2.0. But, if you have such legacy operating systems in your network, you should be seriously asking yourself why. They are very costly to maintain and must be required for critical business reasons only because of this. If you’re putting in place an ESP as recommended in this book, you won’t be running such legacy systems in your network.

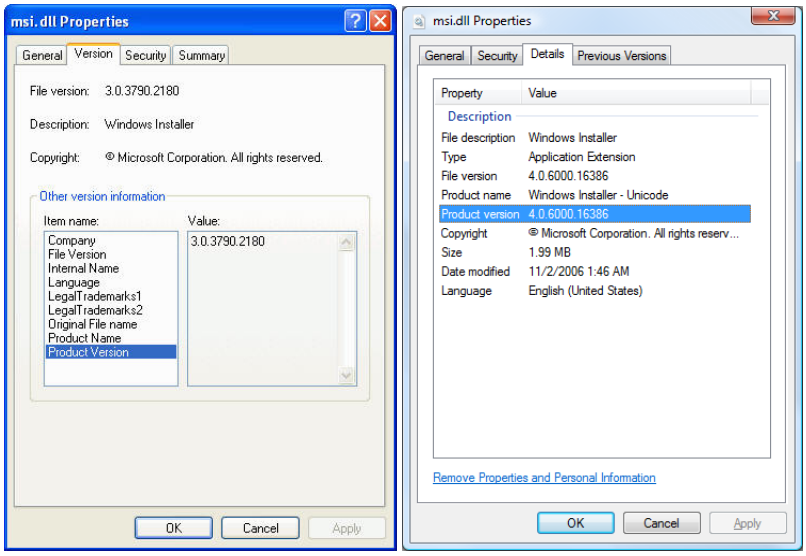


Figure 2: Identifying the Windows Installer service version in Windows XP and Vista.

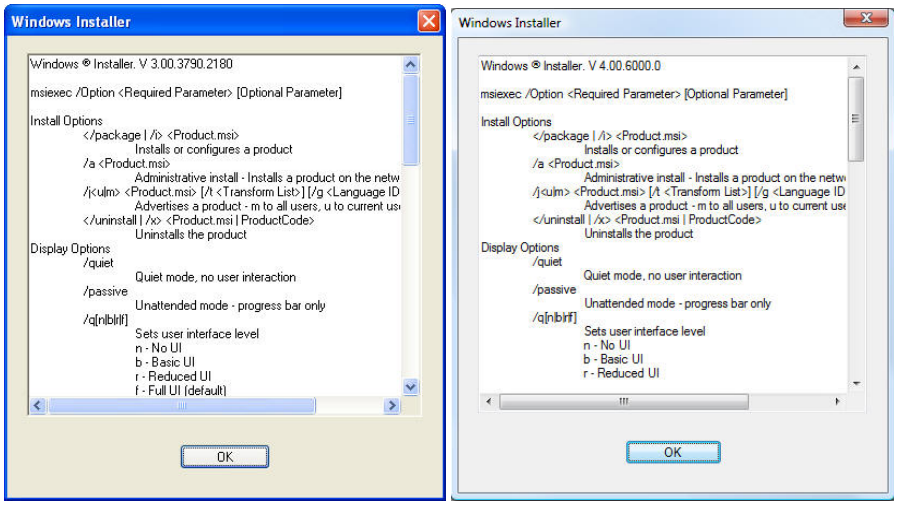


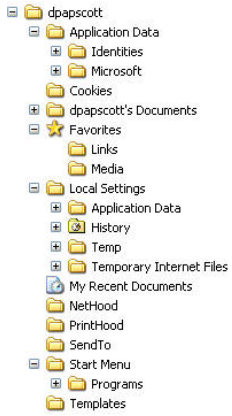
Figure 3: Getting Help on MSIEXEC.

Note: A complete list of the switches supported by the msiexec command on Windows can be found at http://support.microsoft.com/kb/314881.

1.3 Windows Security and Software Installations

Like Windows NT and Windows 2000, Windows XP, Vista and Windows Server 2003 use the NTFS file system. The advantage of this system over its predecessors is that every object stored in the system includes *attributes.* These attributes can contain security features — security features that are different for users, power users and administrators. The greatest limitations are applied to users. Since a user's main responsibility is to operate the system, they only need read and execute permissions for system components. By nature, NTFS protects system and application files by restricting access to these files.

The User Profile in XP



In Windows 2000/XP/2003, the user profile includes every element that is modifiable by the user.

But in Windows NT, users were given too much leeway. This is because software integration was not controlled effectively. Many software products would install into (and require constant read and write usage of) the system directories. Giving users these rights would open the system to potential damage and therefore higher support costs.

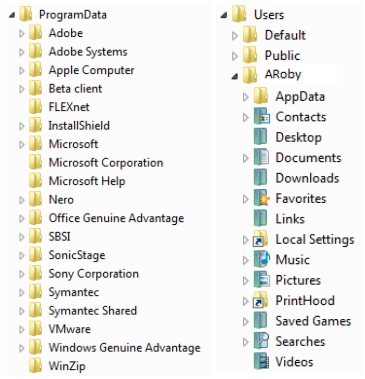
Realizing this, Microsoft released the "Zero Administration Kit" for Windows NT. This kit provided corporations with the tools to increase system "lock down" to further limit user access. But this system was complex to use and organizations often had to invest heavily into its management.

With Windows 2000, Microsoft changed the nature of the NTFS system lock down. They added further restrictions to users and changed the way applications work with the operating system. As a comparison, users in Windows NT have the same rights that power users do in Windows 2000. Today, actual users have significant restrictions within the operating system directories and within application directories.

In Windows XP/2003, Microsoft added more complete support for protected software operation within the operating system itself such as support for side by side dynamic link library (DLLs) in memory.

Software that follows the most recent guidelines for the Designed for Windows Logo program (see above) should not install any component in the system directories. That's because all software components now reside into the application's own directory in Program Files. In addition, every component that is modifiable by a user (configuration settings, user preferences, etc…) is stored within the directories containing the user profile. Here users rule and can read and write to their hearts' content. This is a good strategy because critical system and application files are protected for all users. If users damage something related to an application within their own profile, you can usually repair it by erasing the profile and recreating it. Of course, care must be taken during this operation because the profile doesn't only store application preferences, but also user preferences and sometimes user documents. It is a good idea to make sure you back up and restore documents and preferences once the profile is recreated.

The User Profile in Vista



In Windows Vista, the user profile is structured in a different manner. It also spans two different folders. Application data is in the **ProgramData** folder and

Windows System File Protection

Since Windows 2000, Windows also includes System File Protection (SFP). This feature stores a backup copy of many critical system files (within the %SystemRoot%\System32\DLLCache folder). A special agent is constantly watching the system directories. If, during the installation of a new application, a critical system DLL is replaced and the original system DLL is overwritten, this agent will automatically replace the new file with the original and proper file. Most files are contained within the cache folder and are restored quickly without notification. However, due to space considerations, Windows may attempt to pull an original file from the installation media (for which you may be prompted if it is not available at the time).

SFP in Vista

In Windows Vista, SFP has been upgraded to Windows Resource Protection (WRP).WRP provides protection for system registry keys as well as system files.

Files protected by SFP may only be updated by OS upgrades, service packs and hotfixes released by Microsoft. To further protect the operating system, SFP allows only operating system operations to update files within the DLLCache folder. It is important to note that Windows Installer cannot update protected files. If a Windows Installer package attempts to update such files it will return error 1933. This is why setups like Windows Media Player and Internet Explorer are not provided as MSI setups from Microsoft.

Managing Software in a Locked-down Environment

The new file structure for application location, application preference location and the Windows System File Protection make it even more difficult to update and install software on Windows systems, especially remotely. Of course, users who have local administration rights can install anything on a system. Power users have some more limited installation rights, but they can still modify some system components. Users, who are on the lowest end of the totem pole in terms of installation rights, cannot install anything on the system because they are only granted standard user access.

While this makes for more stable PCs, it does present a challenge for administrators: they need a proper vehicle to install software in locked-down environments. It's either that or grant all users administrative rights. Running a network where all users have administrative rights is a like running a Windows 95 network because you don't gain any of the advantages of a locked down environment.

While many desktop management solutions provide a client agent to address this issue, native support is also provided in Windows by the Windows Installer because it can provide elevated rights to install software packages within the security context of the user. This makes it possible to have a locked down environment and still allow installations in secure contexts. Of course, this does not solve all of the problems related to user installation rights, especially for those related to workstations or servers that are not connected to the network, but it goes a long way towards solving all problems related to network-based software installations in a locked-down environment.

By now you're starting to realize how many features and functionalities the Windows Installer service can offer you. Here's a more complete list of these features:

- Restore the target system to its pre-installation state or **Rollback**. This is one of the nicest aspects of Windows Installer since it tracks the state of a computer system before it begins a software installation. If for some reason, an installation fails, WIS returns the system to the previous state making sure that failed installations do not destabilize systems. This is done through the creation of temporary files during the installation. These files are only available during the installation itself. After the installation is complete, you must use the uninstall command to remove the application (/x).
- Provide **application resiliency**. This gives WIS the ability to check the health of an application and repair or reinstall damaged pieces of the application. This requires access to the original installation source (or a copy available at a specified location) of an application because the repair is performed from the original installation files.
- **Clean uninstallations**. Because WIS tracks all of the components making up an application during its installation, it can safely remove the application from a system, even if the application shares components with other applications on the same system. WIS also tracks which applications share the components and keeps them if there are still applications that require them on the system.
- **Control reboots** during installation. WIS gives you the ability to either call for or suppress reboots during the installation of your software. For example, this would let you install a required component such as the Microsoft SQL Server Desktop Engine (MSDE), reboot the computer if required and continue with the installation of your product.
- **Componentization** or separation of the components of an installation into discreet units that are treated as whole components by the Installer service.
- **Source list control**. Windows Installer lets you control the source locations for the installation. Each time a package is updated, WIS updates the folder from which it is updated and adds it to the source list if it isn't already there. The next time it needs to repair an installation, it will look to this folder for an installation source. For increased resiliency, several source locations may be specified for any one MSI package. Source list control is a very important part of WIS package management.
- **Merge module inclusion**. You can include mini-packages into your own software installation. For example, this is how you would include MSDE into your own package.
- **Command line options** for installations. Since WIS uses a single command line tool for installations—the MSIEXEC.EXE command—it allows you to use standard command line structures to install products. In addition, the command line supports

the modification of MSI packages through transforms and/or patches, letting you use one single interface for installation, patching and customization. The MSIEXEC command also includes several levels of logging which make it quite practical to use when you are having problems with an installation. Because WIS installations are performed through the MSIEXEC command, SETUP.EXE commands are no longer really needed, though they are often included in applications to make the install more transparent to users. A WIS installation that uses a SETUP.EXE command is really only calling MSIEXEC with the proper options and switches.

- Taking the extensive command line support provided by Windows Installer even further, you may also specify the value of any **public properties** right at the command line. Public properties act as variables that dictate the behavior of a Windows Installer Setup. There are several common public properties that control everything from how an application appears in the Add/Remove Programs applet, to how a required reboot should be handled. Further, authors may dictate their own public properties to allow custom command line option support for their Windows Installer setups.
- **Group Policy control**. Windows includes a series of Group Policy settings both at the user and at the system level for the control and operation of the Windows Installer service.
- **Installation on demand**. Because this function lets you choose whether or not a software component is installed during the software installation, it speeds up installations. With Installation on Demand, the feature or function that was not installed originally can be installed when it is first used by the user. This is another reason why original installation sources must be maintained on the network. Be careful with this feature because lots of users find it extremely annoying to see the Windows Installer service launch when they are in the middle of using a product simply because the feature wasn't installed originally.
- **Application advertisements**. This function is much like the Installation on Demand function, but performs even faster because all it does is place the shortcut to the application on the user's desktop. The application isn't actually installed until the user either clicks on the shortcut to use it the first time or when the user tries to open a document that requires the application.
- **Administrative installations**. This allows you to perform a single network installation which would then let users install the software without access to the original CD version. WIS provides one single standard format for these administrative installations.
- Encapsulated logic for **Visual Basic for Applications** (VBA) Installation. WIS includes the logic required to perform the installation of VBA and stores it as a reusable module. This means that all products that will include VBA will use one single standard and approved format for this installation.

You can see that as you learn more about WIS, that it has a compelling story for you to move to this standard in terms of software installations.

2.1 Windows Installer Architecture

The Windows Installer package is everything that is required to perform the installation of a software product. The first part of the package is the .MSI file. This file includes all of the instructions for the installation. Along with the instruction file, you'll also have .CAB files which are compressed files that contain the software parts to be installed. These software parts do not necessarily need to be compressed into .CAB files; they can simply be stored in a folder that is distributed with the .MSI file that provides the instructions for the installation. In addition, the software parts could also be contained in CAB files that are stored within the .MSI file. It all depends on your preference based on the size of the bits that make up the software product. For example, a small program like Winzip Computing's Winzip compression tool could all be stored within a single .MSI file. On the other hand, a very large program like Microsoft Office 2003 will contain an .MSI file, several .CAB files and separate components as well since the bits making up this installation take up several hundred megabytes.

Within the .MSI file will be the installation database—a relational database that the Windows Installer service uses to perform the installation. The information in this database will be hierarchical in nature and will include:

- **Product** — This is the highest layer of the hierarchy. It usually identifies what needs to be installed, for example Microsoft Office 2003. The product is identified by WIS through its product code, or globally unique identifier (GUID) called the **product code**.
- **Features** — The product is composed of features. Features are units of installation that can be discretely selected during installation. For example, in Microsoft Office 2003, Microsoft Word, Microsoft Excel, Microsoft PowerPoint and so on, are all features and users can select or deselect them when installing Office. Features can also include sub-features. For example, in Microsoft Excel, the Help files are a sub-feature. The same applies to Excel Add-ins, Sample Files and so on. Each item that can be selected or deselected for installation is a feature or sub-feature. Features can be shared across applications. One good example is the Spell Checker in Microsoft Office. It is shared between all Office applications, but it is not automatically removed when a feature that uses the shared feature is uninstalled. For example, if you want to remove Excel from a system, but keep Word, Windows Installer would not automatically remove the Spell Checker. In fact, WIS will not remove a shared component until it knows that no other installed product requires it. That's because it tracks which product uses which shared features.
- **Components** — Features are made up of components. A component is a collection of files, registry keys, shortcuts and other types of resources (for example, an icon image) that make a feature work on a computer. As far as Windows Installer is concerned, components are single units that are identified with special GUIDs called the **component code** within the installation database. Because they are considered

as single, cohesive units, components do not share files or any other object. If two components on the same computer include the same file, both will maintain a copy of that file on the system. Treating application objects as components helps speed up the operation of Windows Installer because it limits the number of items WIS needs to keep track of. Components can be shared between applications, but if two applications need to rely on the same component, both must include it in its installation. When the second product is installed, Windows Installer realizes that the component is already on the system and does not re-install it. Instead, it adds a special counter called a refcount to the database to identify how many products use the component. The refcount ensures that the component is not removed until all the products that need it are removed from the system. Of course, because the identification of a component is based on its GUID, two applications sharing the same component must be sure to make use of the same GUID in order to have the component properly managed.

A good example of how a component should be structured is related to all elements that a product will want to store in the HKEY_Current_User registry key. All of these elements should be contained within the same single component because this way, when a new user tries to access the product, the resiliency features of WIS will automatically identify that these elements are missing from the user's profile and add them.

- **Key Paths** — Key paths are associated with components. Each component has a key path that the Windows Installer service associates with the component. Key paths identify whether or not a component is fully installed. This is either through a special file or setting included in the key path. Windows Installer uses the key path to determine the health of each component within a product. If a key path is missing or incomplete, WIS will trace it back to the feature it belongs to and reinstall the entire feature or sub-feature. This is the engine that provides self-healing for WIS installations. The time it takes to repair an application will depend on the number and size of the components within a given feature.

While some authoring Windows Installer setups choose to create one feature per component, it is not necessary to do so since a feature can be made up of multiple components. In addition, the same component can be used by multiple features. To ensure that self-repairs are not overly lengthy in time, authors have to balance the number of components they include in each feature with the number of features or sub-features they include in their products. For example, if all components are stored in one feature, any self-repair operation would result in a reinstallation of the entire application.

Figure 4 illustrates the installation dialog box for Microsoft Office 2003. As you can see, Office 2003 is made up of multiple features and sub-features. Users can discretely select each feature and sub-feature during an interactive installation. In addition, they can determine how the feature or sub-feature will be installed.

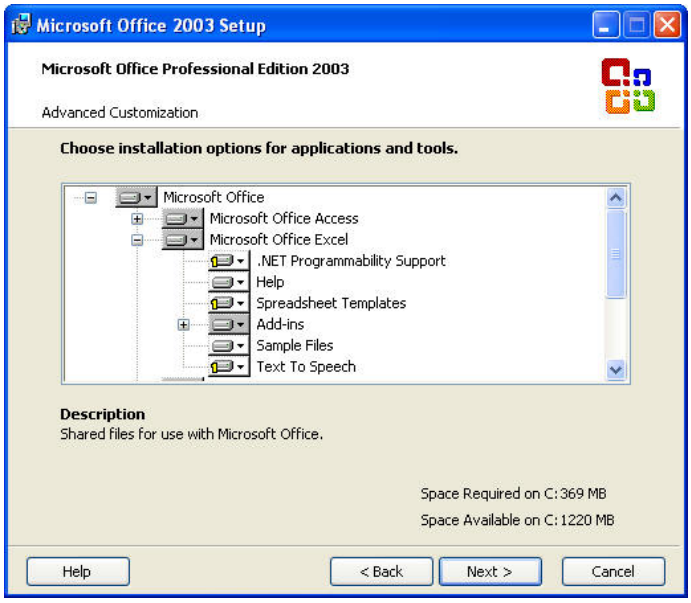


Figure 4: The Makeup of the Microsoft Office 2003 Windows Installer package.

Four choices are available for how one may choose to install each feature or sub-feature:

- Run from My Computer
- Run all from My Computer
- Installed on First Use
- Not Available

These options are illustrated in Figure 5. The first installs the feature, but not all of its sub-features. The second installs the feature and all of its sub-features. The third installs the feature only if you choose to use it while working with the product. The last is self-explanatory. The second option, **Run all from My Computer**, is often the best option to select since you are guaranteed that whatever you may need from this feature and its sub-features is available to you whenever you need it.

The third option is sometimes used in networks, but it provides annoying pop-ups of the Windows Installer dialog box while users are in the middle of working with a product. Not a good idea. In addition, this third option requires constant access to the installation source files. This is not very useful for mobile users who may not be connected to the network when they need the feature. While the second option may install unnecessary features and components, it is by far the option that gives the most pleasant experience to the user. For this reason, it is often the best feature to use. If you decide you prefer not to use this option, make sure that whatever is not installed, is set to **Not Available**, the fourth option, because this ensures that no installations need to take place while users are working with a product.

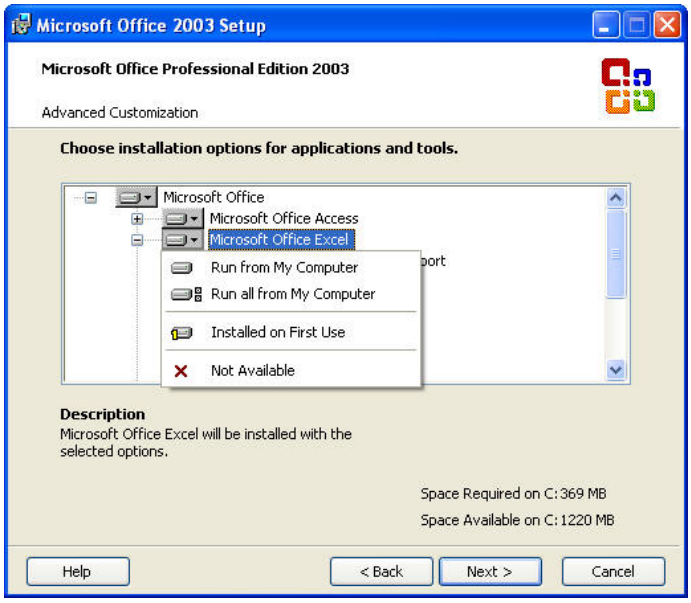


Figure 5: The Feature Installation Options.

The Windows Installer Database Structure

As mentioned previously, Windows Installer installations are based on the .MSI file that is included with the product setup. This file is nothing more than a set of installation instructions that are organized in tables within a relational database. Each table is defined by the **Windows Installer Software Development Kit** (SDK). This SDK includes an MSI database editor that has limited functionality. This editor is called Orca. In an organization that invests into an Enterprise Software Packaging strategy, you use much more comprehensive or commercial tools to view, edit and create MSI installation databases.

The tables found in the MSI database contain a series of information types within rows and columns. For example, one commonly used table is the **Launch Conditions** table. This table sets out the conditions under which an installation may or may not be executed. A good example of this is when you create a package for a product that is designed to run on workstations and not on servers. In this case, you would put in a launch condition that verifies the operating system onto which the package is being installed and if the OS query returned Windows Server 2003 or Windows 2000 Server, you would display an error message stating that this is the wrong OS and abort the installation process.

The tables that make up the body of an MSI database are the **sequence** tables. Sequence tables tell Windows Installer what to do during an installation and in which order it should b done. A common sequence of events is the verification of the Launch Conditions and then if they are met, the copying of installation files, the modification of the registry and the removal of temporary files such as those used for rollback.

There are three types of sequence tables, each tied to a specific Windows Installer service feature. The first is the **Admin** table type. Admin tables are used for administrative installations or installation of a product into a network share to create a remote installation point for the product. The second type of sequence table is the **Advertisement**. This table type is used to

advertise products and features. Advertised features and products are not actually installed until the user activates it by trying to use it. The final type is **Installation** and is the most commonly used sequence table because it controls how a product is installed. The Installation type can be used for either interactive or silent, background installations.

Along with each sequence table type, you have two associated sub-tables: **InstallUISequence** and **InstallExecuteSequence**. The first, InstallUISequence, is used for interactive installations and includes all of the dialog boxes that are displayed to the user during the interactive installation. The second, InstallExecuteSequence, lists the actual steps to perform during the installation. The InstallExecuteSequence table includes a column called **Actions** which includes a set of predefined actions that the Windows Installer service can perform. Some of these actions include:

- Check for execution requirements (Launch Conditions)
- Search for previous versions of the product in order to upgrade it
- Create folders
- Create shortcuts
- Install or delete files
- Install or remove registry keys
- Move or copy existing files to new locations
- Install, remove, start or stop Windows services
- Install or uninstall Common Language Runtime (CLR) assemblies within the .NET Framework
- Install or remove ODBC drivers and data sources
- Register COM classes or COM+ applications
- Modify environment variables

In addition, Windows Installer supports **custom actions**, or actions you define yourself. In support of custom actions, Windows Installer can execute VBScript or JScript code, run commands from the command line or call functions that may have been defined in a special dynamic link library (DLL) that you programmed. Custom actions are very powerful and add almost any installation action to the Windows Installer service.

Finally, another table that is commonly edited by administrators within MSI packages is the **Property** table. Properties can be used to define installation variables. They work much like environment variables do in Windows itself. For example, it is the Property table which will help you ensure that an installed product is available to only a single user or to all users of a computer. This is done through the ALLUSERS property which will tell WIS to either install product settings on a per user or per machine basis. Note that property names are case sensitive so allusers does not mean the same thing to WIS as ALLUSERS. Properties can be applied during installation in one of two ways. The first is through the MSIEXEC.EXE command,

but this command only works with *public* properties. The second is through another special WIS file type called a transform. Transforms can address any property.

Another useful property is ROOTDRIVE. By default WIS installs products into the drive with the most free space. If your workstation drives are split into more than one disk partition, for example, C: for system files and D: for data, and the D: drive has more free space than the C: drive does, WIS will automatically install programs on the D: drive. Using the ROOTDRIVE=C:\ property will ensure that your packages are always installed on the system drive.

System Disk Structure
There are very few justifications for splitting disks into multiple partitions in Windows today. Most of the administrators who split drives do it to simplify recovery. It makes sense in a way: if the system partition crashes, then user data will have been protected. In reality, this demonstrates a poor overall system protections strategy. Today, in a well structured organization, workstations should be recoverable within 30 minutes; user data should be protected through folder redirection and Group Policy; and system repair policies should be limited to no more than 10 to 15 minutes' effort—if the technician spends more than this amount of time on a system to try to repair it, the technician should simply move to a re-imaging of the PC. In this way, system repairs never take more than one hour. Because of these proper strategies, there is no reason to split disks. This will help improve Windows performance because it will have lots of room for a proper page file and for the storage of temporary files during operation. And, you won't need to worry about using the ROOTDRIVE property.

The few tables mentioned here are not a comprehensive list of all the tables available within a Windows Installer database. There are quite a few tables within this database structure. For example, other tables include:

- Application design
- Feature
- Component
- Feature components
- Directory
- File copy
- File
- Media
- Registry entry
- Installation procedure
- User interface
- Desktop integration
- Installation validation

You will not need to understand all tables unless you choose to author an MSI setup without the aid of an authoring or repackaging tool.

Windows Installer File Types
All of the installation options are available programmatically as well. This is where the different file formats for the Windows Installer service come into play. You're already familiar with the MSI format. This is the file that contains the installation database and can also contain compressed CAB files, application settings, and other resources that make up the package required to install the product. Windows Installer also uses other file formats to perform special operations during installation.

The second most prevalent Windows Installer file type is the **MST** or **transform** file type. The transform is a secondary file that is tied to the MSI database during execution to modify the behavior of the installation. One strong reason to make use of transform files is to adhere to the highly recommended practice of never modifying the MSI files you receive from any

manufacturer. These files follow a specific structure and include specific content that will be required when it is time to upgrade or even simply remove a file from the system. If you modify the internal contents of an original MSI, you may break its upgrade or removal capability.

Therefore, when you want to customize the installation behavior of any given MSI, you need to transform it by adding all of your custom changes into an MST. When you run the MSIEXEC.EXE command to install the product, you use appropriate switches to apply the transform during installation. This maintains the integrity of the original MSI file while allowing you to customize the installation to your own needs.

Transforms can include most any type of customization but the most common are:

- Identifying which features of a product should be installed.
- Determine if and how users interact with the installation. Most often, there is little or no user interaction.
- Identify which answers need to be provided to the setup during installation. This includes items such as installation location, product activation keys and the like.
- Identify which shortcuts should be created and where they should be placed. Get rid of special Internet offerings and other annoying bits from manufacturers' products.
- Include additional files such as corporate document templates.
- Identify which registry settings are to be modified and how.

As you can see, transforms are a useful way to modify the original MSI.

The next most common Windows Installer file type is the **MSP** or **patch** file. Patches are updates to the product that do not affect the **ProductCode** attribute within the MSI database. It may however, increment the **ProductVersion** for the MSI. When the ProductVersion is modified, the patch is usually large enough to be considered a service pack or, in WIS terms, a major update. When it does not affect the ProductVersion, the patch is considered a minor update.

Note: If you modify the source code for a deployed product, you will need to update or refresh the deployed installation. This is because the source files for a deployed installation may reside on a network share in support of self-healing. If you modify these source files without updating the deployed installations, self-healing will no longer work especially if the ProductVersion attribute has been modified by the patch. To reinstall the product on all deployed systems you should run the following command:

msiexec /fvomus name_of_the_package.msi REINSTALL=ALL

The switches used in the above command include the following functions:

f — fix or repair an application
v — run from the source and re-cache the local package
o — if file is missing or an older version is installed
m — rewrite all computer specific registry entries
u — rewrite all user specific registry entries
s —overwrite all existing shortcuts

As of version 3.1, you could actually use on /fv since all other switches are now the default repair behavior.

When you run an installation through the Windows Installer service, you actually create an installation database on the target computer. This installation database is then used to support long-term program viability features. Applying patches for software products that are integrated to the Windows Installer service means updating this installation database and modifying key components, often dynamic link libraries (DLLs) of the program.

While original MSI files are often transformed through MST files to customize their installation within corporate networks and adapt them to corporate standards, the MST does not modify the original MSI. Patches, on the other hand, modify the original installation database as well as key program components (see Figure 6). Therefore when patches are made available, you need to apply them both to installed copies of the product to update the deployed installation database as well as applying them to any administrative installation of the software you may have performed. This will ensure that any one performing future installations or repair operations from this administrative install point will install an updated version of the product.

Yet another Windows Installer file type is the **MSM** or **merge module**. The merge module is designed to allow you to include sub-products into your installation. For example, several products require a database to operate. In order to ensure that such a database is available, they will include a copy of the freely distributable Microsoft SQL Server Desktop Engine (MSDE).

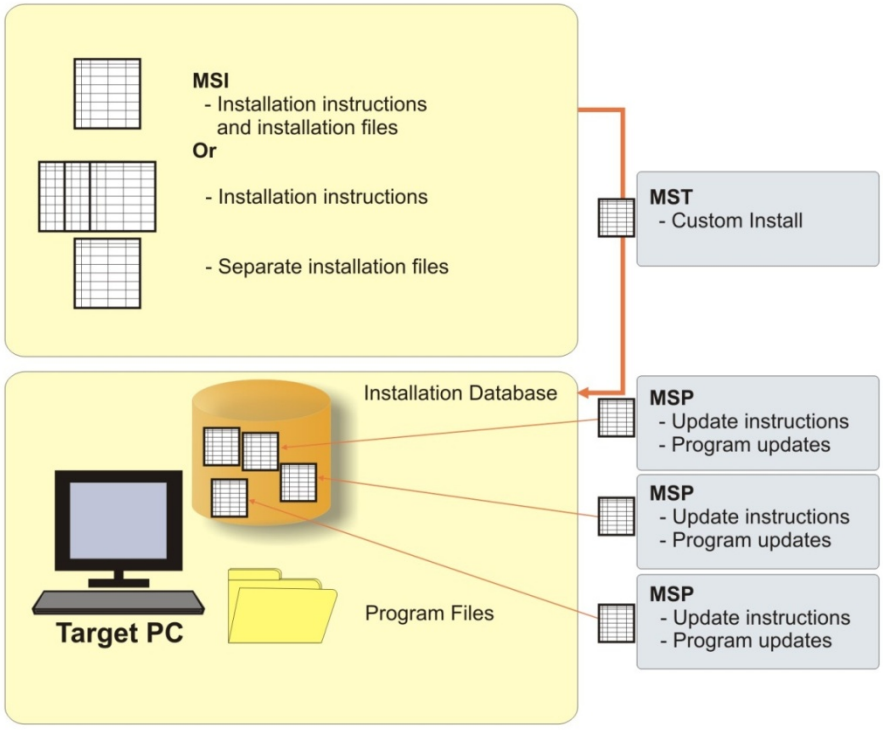


Figure 6: The MSP Patching Process.

These are the major file types used with the Windows Installer service but there are others. For example, when you just export the installer database out of an MSI file, you create an **IDT** file (Installer Database Tables). During the preparation of installer patches, manufacturers will work with **PCP** files. During the creation and/or preparation of an MSI package, you may work with **CUB** or package validation files.

And finally, when you integrate a Windows Installer package with Group Policy to deploy it to a user instead of a computer, Group Policy will create an **AAS** file which is an advertisement script. It is this script which is deployed to the user instead of the actual MSI. The script supports the automatic installation of the package once the user either clicks on the product shortcut the script creates or on a document that requires the product to open. The full list of file extensions used with Windows Installer is listed in Table 3.1.

File Extension	Purpose
MSI	Installation database and possibly installation resources
MST	Installation transformation instructions
MSP	Patch information to be applied to original MSI
MSM	Merge module to be integrated into MSI
IDT	Exported Installer database file
PCP	Patch creation file used during patch preparation
CUB	MSI package validation file
AAS	Group Policy advertisement script
CAB	Compressed file containing installation resources for a product

Table 1: Windows Installer File Extensions**.**

2.2 Managing the Windows Installer Service

As you can see, there's a lot to the basic Windows Installer architecture. But now that you understand that Windows Installer is a transactional service that is based on the MSI database, you're starting to see the value of such a service in a managed network. Since Windows Installer is a service, it needs to be managed like all the other components in your network. Microsoft has made it easy to work with and manage this service. As mentioned previously, much of the service administration is performed through the MSIEXEC.EXE command. But Microsoft has also given you the possibility to centrally manage the behavior of the service through Group Policy Objects (GPO) and Active Directory.

Working with Windows Installer Installations

Because it is a service, Windows Installer runs under the Local System account privilege. This means that it has the right to perform almost any installation operation even if users are in a locked down environment. Note that you may control if and how the ability to run installations with elevated privileges should be handled via Group Policy—refer to the AlwaysInstallElevated policy. There are several ways to start this service, but they are all related to a Windows Installer or MSI installation. Without the MSI, you cannot access the service. This means you can launch a Windows Installer operation with the following actions:

- Double-click on SETUP.EXE for an MSI-based installation
- Double-click on an MSI file
- Use the MSIEXEC.EXE command

- Use Add/Remove Programs
- Deploy an MSI product through Group Policy or through a software deployment tool that understands MSIs

When an interactive installation is initiated, it calls upon the Windows Installer application programming interface (API) to start the service and present the appropriate dialog boxes to the user. To do this, the service runs several processes, one in your user context with your user rights and permissions and others in the Local System account context. This is why it is best to deploy MSIs centrally: even if WIS has access to the Local System account, if your credentials do not allow you to perform installations, you won't be able to install a product interactively. On the other hand, if you deploy the installation with the appropriate settings the service will be able to perform the installation even if the logged on user does not have installation rights. In this case, the processes that run in the user context are only run to configure actual user settings and not to install the application. Remember that the user has complete control over the user profile so running in the user context allows WIS to properly complete installations. Because of this, this method creates **managed applications**.

One of the key elements of an MSI installation is context. You can work with Windows Installer to deploy applications on either a per user or per machine basis. Using the ALLUSERS=1 property during installation will tell Windows Installer to perform a machine-based installation or an installation that will configure settings within the All Users profile, letting any user who has access to the machine have access to the installation. In most if not all cases, you'll want to use per machine installations. That's because if you install an application on a per user basis, only the user who installed the application can remove it from the system because only that user has access to the user profile in which the application hooks reside. If multiple users installed the same application on a per user basis, then it cannot be removed until each user has uninstalled it. In managed environments, this can be quite a headache.

Note: The default installation that Windows Installer performs is a per user installation. That's because by default, the ALLUSERS property is not set and a user installation is the default behavior. Make sure that you change this value. As mentioned above, the value can be set directly within your own packages, can be modified through a transform or can be added to the command line while installing a product.

There are a couple of possible values that may be assigned to the ALLUSERS property and they can mean different things based on the rights of the individual performing the installation.

ALLUSERS = 1 (All Users)

If the user performing the installation is a standard user, this will return an error as an unprivileged user does not have sufficient permission to install an application for all users. If the user has admin privileges, the installation will be performed for all users (to the "All Users" profile directory).

ALLUSERS = 2 (Current User)

While you would expect this to always install just for the current user, if the user has administrative privileges it will be installed for all users even if this settings is specified. If you really do want to install just for the current user and have administrative privileges, this property should not be set at all so the default behavior of installing just for the current user may be realized.

During the installation, Windows Installer will back up any file it replaces in order to support installation rollback. These files are copied to a temporary folder, often the C:\CONFIG.MSI folder. Of course, once the installation is complete, it removes the files.

In addition, when it finishes an installation, WIS will copy the installation's MSI file and any associated MST file to a special folder called %WINDIR%\INSTALLER and rename it with a cryptic name. This allows it to manage the installed product directly from the local machine without necessarily requiring access to the original installation source. That depends, of course, on whether the installed resources were contained within the MSI or not.

Figure 7 outlines how the pieces of the puzzle work together during a product installation.

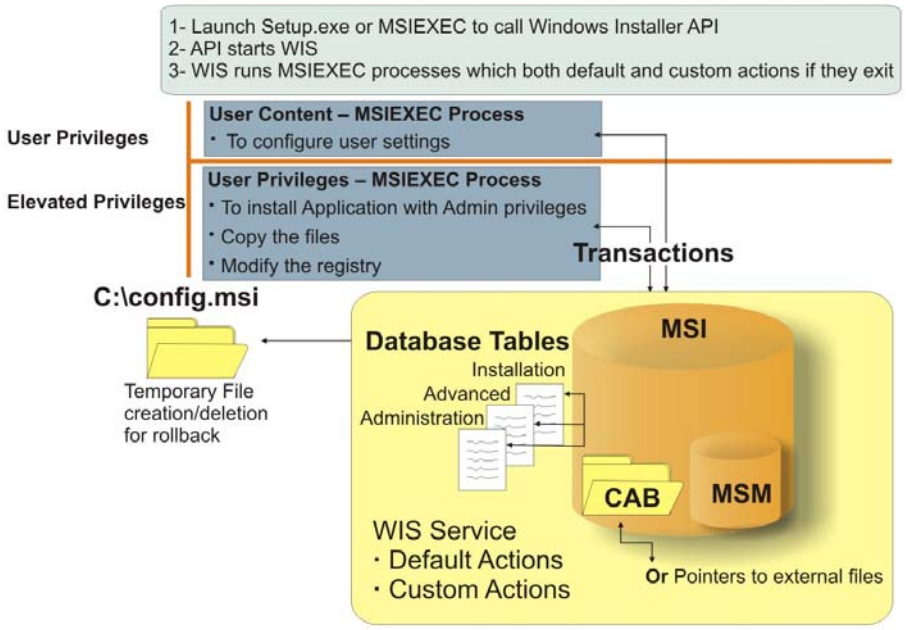
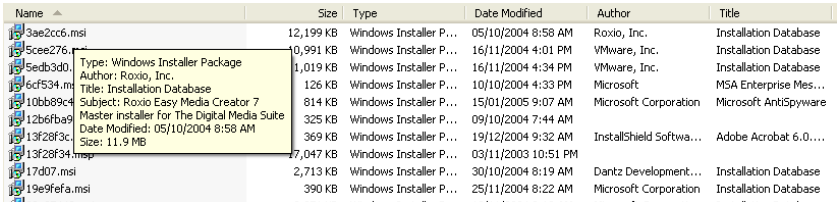


Figure 7: The Windows Installer Service and Product Installations.

Note: The %windir%\installer folder is a hidden folder by default. To view it, type in the address in File Explorer. In addition, you'll notice that the all of the file names are very cryptic. To view which product the file matches to, right click on the Explorer sort bar at the top of the details pane and add both Title and Author to the detailed view. This will show you which manufacturer and which product the file relates to.



Alternatively, you can simply hover over the file to see its properties.

GPO Settings and Policies

Corporate networks that run Active Directory can centrally control the Windows Installer service through Group Policy. If you don't have Active Directory, then you'll have to use another centralized management tool to control these settings or you'll have to do it through local policies. Once again, as you are in the process of putting in place an ESP, you won't be managing this service through local policies that have to be modified individually on each workstation in your organization.

There are quite a few policy settings for the Windows Installer service, especially after Service Pack 2. Table 2 outlines all of these settings and indicates some recommended settings for them. All specific settings for Windows Installer are found under **Computer Configuration | Administrative Templates | Windows Components | Windows Installer** or **User Configuration | Administrative Templates | Windows Components | Windows Installer**. Software deployment settings are located in either **Computer Configuration | Software Settings | Software Installation** or **User Configuration | Software Settings | Software Installation**. In most cases, you'll want to work with the Computer Configuration portion of a Group Policy because you want to avoid user-based software deployments as much as possible.

> **Note**: Networks running Windows XP Service Pack 2 and Windows Server 2003, but without Service Pack 1 for Windows Server 2003 need to add a special hotfix for Windows Server. This hotfix will allow you to open policies either from Windows Server or from Windows XP with no error. This is described in article number 842933 on the Microsoft Knowledge Base. It can be downloaded from here: http://support.microsoft.com/default.aspx?kbid=842933.
>
> In addition, to update your domain controllers with the .adm files that are contained in Windows XP Service Pack 2, you need to run the Group Policy Management Console (GPMC) from a machine running XP SP2 and open each of the policies you want updated. This automatically adds the appropriate .adm files to the policies.

Group Policy Setting	Purpose	Recommended Setting	Notes
Computer Configuration | Administrative Templates | Windows Components | Windows Installer			
Disable Windows Installer	Controls the use of Windows Installer	▪ Not configured in Windows Server 2003 ▪ In Windows 2000 Server networks, change to **For non-managed apps only**	This setting ensures only deployed software can be installed by normal users. This is the default in Windows Server 2003.
Always install with elevated privileges	Tells WIS to use system credentials to install software	▪ Not configured	When you have a proper MSI deployment tool in place, there is no need for this setting.
Prohibit rollback	Stops WIS from creating temporary files for rollback	▪ Not configured	The only reason to enable this setting is to save temporary disk space. Do not use this setting in either Computer or User Configuration because setting it in one automatically turns it on for the other.
Remove browse dialog box for new source	Stops users from browsing the file system when installing	▪ Not configured	The default behavior is sufficient in this case.

Group Policy Setting	Purpose	Recommended Setting	Notes
	features in WIS		
Prohibit patching	Stops users from patching WIS products	▪ Not configured	By default, only administrators can patch products.
Disable IE security prompts for Windows Installer scripts	Lets Web-based programs install without user knowledge	▪ Not configured	Applying this can be a very high security risk.
Enable user control over installs	Gives users elevated privileges during installations	▪ Not configured	Users should not have the right to install software except under special situations. In this case, there are better ways to give them these rights than this policy.
Enable user to browse for source while elevated	Gives users access to restricted files and folders	▪ Not configured	This would let users use the Local System account to access restricted files and folders during an installation. It is turned off by default.
Enable user to use media source while elevated	Gives users access to removable media during installations with high privileges	▪ Not configured	Users can access removable media during installations in their own security context. Since you want only per machine installs, do not enable this setting.
Enable user to patch elevated products	Gives users the ability to patch software in their own context	▪ Not configured	Patches should be delivered centrally to end users.
Allow admin to install from Terminal Services session	Lets administrators install applications when in a TS session	▪ Enabled	This affects only system administrators and lets them install software through Terminal Services sessions.
Cache transforms in secure location on workstation	Saves transforms in secure location on the desktop	▪ Not configured on Windows Server 2003 ▪ **Enabled** on Windows 2000 Server	Caching transforms into secure locations protects them from malicious tampering. This is the default behavior in Windows Server 2003. This also is key to per machine installs because transforms can only be used on the same machine when this setting is enabled.
Logging	Sets the logging level for WIS	▪ Not configured	Use this only when required. Logs are saved to the Temp folder of the system volume. While logging is available at the command line, for deployments via Group Policy this can prove a valuable feature. By default, logging is set to include status messages, warnings, error messages, start up of actions and terminal properties.
Prohibit user installs	Controls how user installs are configured	▪ Enabled ▪ Set to **Prohibit User Installs**	This setting will prevent per user installations and allow only per machine installations. **Note**: on Windows 2000, this setting disables all installs.
Turn off creation of System Restore Checkpoints	Protects user systems by creating restore points on Windows XP	▪ Not configured	This setting is enabled by default.
Prohibit removal of updates	Protects updates from being removed	▪ Not configured	By default, only administrators can remove updates.
Enforce upgrade component rules	Controls how upgrades occur	▪ Not configured	If this setting is enabled, some upgrades may fail because you will have to follow strict upgrade rules. Even in an ESP, it is not always possible to control how upgrades are performed because you do not

Group Policy Setting	Purpose	Recommended Setting	Notes
			always control the source code for the upgrades.
Prohibit non-administrators from applying vendor-signed updates	Controls how updates can be performed with Windows Installer version 3.1	▪ Not configured	By default, updates that are properly signed by vendors can be installed by users.
Baseline file cache maximum size	Controls the amount of disk space for the baseline cache for Windows Installer version 3.1	▪ Not configured	By default, Windows Installer users 10 percent of available free space for this cache. Change it only if you feel 10 percent is not sufficient for your needs.
User Configuration | Administrative Templates | Windows Components | Windows Installer			
Always install with elevated privileges	Tells WIS to use system credentials to install software	▪ Not configured	When you have a proper MSI deployment tool in place, there is no need for this setting.
Search order	Tells WIS where to search for installation files	▪ Not configured	By default, WIS searches the network first, then removable media, then the Internet.
Prohibit rollback	Stops WIS from creating temporary files for rollback	▪ Not configured	The only reason to enable this setting is to save temporary disk space. Do not use this setting in either Computer or User Configuration because setting it in one automatically turns it on for the other.
Prevent removable media source for any install	Controls if users can install software from removable media	▪ Enabled	This setting stops users from being able to install software from removable media. In a proper ESP, software should be installed from the network only. Even if it is enabled, administrators can install from any location.
User Configuration | Administrative Templates | Control Panel | Add or Remove Programs			
Hide the "Add a program from CD-ROM or floppy disk" options	Lets users add programs from removable media through the Control Panel	▪ Enabled	This setting is used in conjunction with the "Prevent removable media source for any install" setting. Note: You might want to simply enable the "Hide Add new Programs page" as this will stop users from adding programs through the Control Panel.

Table 2: Group Policy Settings for Windows Installer.

Make sure you apply the settings from Table 3.2 to your production network and that you include testing computers with these settings in your packaging lab. This will let you test as a user under full user conditions when preparing packages. As for software delivery through Group Policy, it is recommended that you only use the Computer Configuration portion of the GPO because you want to ensure all installs are on a per machine basis.

Software Restriction Policies

To further protect your systems from unwanted software installations, you should make use of Windows' Software Restriction Policies (SRP). SRPs are designed to help control the execution of code within your network. SRPs rely on four different rules to determine if software can execute in the network. These rules include:

- **Hash rule**—a hash is a special identifier that is generated by performing calculations on the binary elements of a file. Because of the way the hash is calculated, no two hash rules are the same. It is also impossible to reverse the process to find the originating data. To use hash rules, you need a hash-generating program.
- **Certificate rule**—a public key certificate that is included in both the SRP rule and in the MSI packages. This is often the easiest way to use SRP because it is easier for you to control certificates, especially since Windows Server 2003 includes the ability to manage an internal public key infrastructure (PKI).
- **Path rule**—one of the simplest SRP rules because it simply states which paths are acceptable for software installation. Be careful if you use this method because if you allow *\softwaresource* for example, anyone can create a program that makes the C:\softwaresource folder and run program installations from there. The best way to use this rule type is to implement a distribution structure based on the Distributed File System (DFS) which can present the same installation source to all sites through domain-based DFS shares[3].
- **Internet Zone rule**—this is based on the zones perceived by Internet Explorer. This method is slightly more risky because once a zone is allowed, any installations from this zone will work.

Rules are applied in the order they are listed here. Often the easiest way to implement SRPs is to combine both a certificate rule with a path rule based on DFS. Because domain-based DFS shares use the domain name in the universal naming convention (UNC) rather than the server name, the same path can be used anywhere in the network.

If you decide to use certificate rules, you'll want to pre-deploy the certificates you will use in your packages. That's because when you deploy a package with an untrusted certificate, the user will have to accept the certificate before the installation can proceed. If on the other hand, you have pre-deployed the certificate, then the installation can proceed uninhibited. Fortunately, Windows Server 2003 supports certificate auto-enrollment. This means that users will not even be aware of the need for or the issue of certificates for software deployment and installation.

You might also consider the following. Administrative installations of MSI packages may change the nature of the package so it is always best to install a certificate in the package *after* it has been installed to the administrative location. Commercial MSI packages may also already include digital signatures. In this case, you can add the vendor's certificate to your SRP rules. If you modify the package once deployed, it needs to be re-signed. Finally, make sure your certificates are managed properly and accessible to anyone who needs to sign packages that are ready for deployment.

---

[3] For more on DFS and how it can be used in an ESP, see chapter 7 of *Windows Server 2003: Best Practices for Enterprise Deployments* by Ruest and Ruest, published by Osborne McGraw-Hill in 2003, ISBN: 0-07-222343-x.

When you set up SRP, you'll need to first generate the SRP objects in the Group Policy you will use to manage them. These objects are not generated by default within GPOs. Next, determine how it will be enforced. It is best to enforce to all software except libraries (for example DLLs) and to all users. This way, administrators are not affected. You'll also want to configure how trusted publishers will be evaluated. It is a good idea to re-verify certificates before allowing the publisher to be trusted. This way you won't allow publishers with outdated certificates to be trusted.

Source List Management

Another aspect of MSI package deployment that you need to take into consideration is the installation source list. When you deploy an MSI package, Windows Installer needs to maintain the ability to access the original deployed package source for several reasons. One of the main reasons is package self-healing. During the self-healing process, Windows Installer has to connect to the original installation source to reinstall each feature that has a missing or damaged key path. If the original source is no longer available, WIS will ask the user to provide it with an appropriate location. This is definitely not something you want users to face because they have no idea where these files should be. As you know, many of them will still attempt to resolve this situation themselves before they think to call the Help Desk.

Another example is the product upgrade. In some cases, you may upgrade a product that requires access to the original source installation of the previous version in order to remove it properly. A third example is the installation of companion products. For example, if you install a third-party grammar checker in Microsoft Office, it might require access to the original Office installation files to add features that were not originally selected during install.

All of these situations can cause an installation to fail if source lists are not properly managed. Of course, the arrival of Windows Installer version 3.1 mitigates the impact of these situations because version 3.1 at least can use the locally-stored MSI file (in the %WINDIR%\INSTALLER folder) to patch and upgrade products, but it is still very important to maintain constant source lists. Windows Installer manages source lists through a special property in the MSI database. It will try all available locations in this property before prompting the user with a request for a valid location. In addition, the way WIS searches for valid locations is controlled through the search order GPO setting in user configurations. By default, WIS searches the network first, then removable media, then the Internet. The key to source list management is to make sure sources are available before WIS needs to verify removable media.

There are two ways to deal with this issue. The first is to design a proper package delivery infrastructure in your network. As mentioned above, the best way to do this is to use domain-based DFS shares. The DFS service lets you create a UNC which is in the form of *\\domainname\share* instead of *\\servername\share*. The advantage is that with a domain-based DFS share, there should be no reason why the share name should ever change. This lets you modify the share targets to your heart's content without ever impacting users or installations. That's because the domain-based DFS can point to multiple targets that can be in different locations. In addition, the Windows File Replication Service (FRS) can replicate data from target to target to make sure the contents of all targets are identical.

While the first version of Windows Server 2003 includes an FRS that only replicates entire objects—copying a whole file even if only one byte has changed—the next version, R2 supports delta compression replications, replicating only changes to existing files. This makes the DFS replication (DFSR) engine even more powerful and useful for this scenario.

The second prong of this approach is to make sure that the source lists you include inside your packages include all possible locations for a package. For example, domain-based DFS is wonderful for any system that is connected to the network, but what happens when the package is on a portable that is no longer connected to the network? Once again, users would be prompted and this situation would even be worse since there would be no way a user could fix this problem short of having the installation CD with them. One way to solve this issue is to create a hidden partition on the hard disk of portable systems and copy all packages to this location prior to their installation. That way the location is available in the event of a problem with a product while the user is traveling.

Of course, this means that you need to maintain these special locations, but that is part of your software deployment infrastructure design and operation. Also, the vastly expanding default disk size on portables makes an approach such as this quite viable today.

> **Note**: If you have already deployed packages and you have not managed source lists properly, you have probably run into some issues with these installations. There are several ways to repair source lists in deployed products. One is using features that are potentially provided with your deployment tool. For example, if you are using Microsoft Systems Management Server (SMS) 2003, you can modify source lists after the fact for per machine product deployments. Another example is Altiris Software Delivery Solution which can repair source lists for both per machine or per user installations. A third way is to use scripts to modify the list. Finally, you can simply redeploy a properly managed package. This also gives you the opportunity to rectify other potential problems you may have introduced before you knew better (such as per user installs instead of per machine installs).

2.3 New Features of Windows Installer 4.0

Windows Installer received a significant facelift to run with Windows Vista. As you may know, Vista heralds a whole series of changes and modifications in terms of both user access and system protection. The most significant changes in WIS 4.0 are related to:

- Compatibility with Restart Manager
- Compatibility with User Account Control
- Compatibility with User Account Control Patching
- Support for Windows Resource Protection

Most existing MSI packages will run on Windows Vista, but it is always best to update your own packages to run with this version of WIS to make sure you have the latest built-in capabilities and compatibility.

Compatibility with Restart Manager

In order to avoid user disruptions as much as possible, Vista includes a new feature, the Restart Manager. By default, Vista will rely on Restart Manager to stop and restart applications rather than stopping and restarting the system. It actually saves the state of applications and temporarily closes them to prevent the need to restart. The system is restored after the installation is complete without a restart.

Applications that are compatible with WIS 4.0 will include a new MsiRMFilesInUse dialog that will automatically link them with the Restart Manager's capabilities. For applications to run properly with Restart Manager once installed, they must include the new RegisterApplicationRestart function. Both can be added as a transform to packages that are not designed for WIS 4.0.

Note: More information on Windows Installer 4.0 and Restart Manager can be found at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/using_windows_installer_with_restart_manager.asp.

Compatibility with User Account Control

User Account Control (UAC) is a new security feature of Windows Vista that is designed to let all users run with standard user privileges, even if you are logged in with an administrative account. Each time an action requires administrative privileges, UAC requests authorization from the user. The difference lies in how it does this. When logged in as an administrator, UAC simply requests you to allow or deny an action. When logged in as a standard user, UAC requests the name of an account with administrative privileges and its password to proceed.

Because of its integration with UAC, administrators can rely on WIS 4.0 to install all applications as managed applications. Managed applications are automatically installed with elevated privileges and are stored into the HKEY_Local_Machine registry hive, which is the same as using the ALLUSERS=1 property. Once an application is registered as a managed application, it will no longer prompt users or administrators during installation.

If applications are not registered as managed, then standard users will require over-the-shoulder credential assignment or asking someone else to fill in the user name and password for the installation to complete.

Note: More information on WIS 4.0 and UAC can be found at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/using_windows_installer_with_uac.asp.

Compatibility with User Account Control Patching

Vista's UAC supports the ability to patch applications without requiring elevated privileges, but to do so, applications must be digitally signed. Several other conditions are required:

- The application must have been installed by WIS 3.0 or higher.
- If the application was installed on Windows XP, it must have been done with removable media—CD or DVD—otherwise it will not work. Note that this restriction does not apply to applications installed on Vista.
- The application must have been installed for all users or per machine.
- The patch or the original package must include the MsiPatchCertificate table which in turn includes the digital certificate for the patch.

More conditions must be met, but suffice it to say that when patches pare properly prepared for Vista, they will install under standard user privileges. This is something you should always aim for in your patches.

> **Note**: More information on patching with UAC can be found at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/user_account_control__uac__patching.asp.

Support for Windows Resource Protection

Microsoft has renamed Windows System File Protection to Windows Resource Protection (WRP) in Windows Vista. WIS 4.0 is integrated to WRP in the following manner:

- If system files are contained within a package, WIS skips its installation and logs an entry into the log file and continues the installation. This is different from Windows 2000 and XP as WIS would call on SFP to install the file for it.
- WRP protects both files and registry keys. As with files, if WIS encounters a protected registry key in the installation, it skips it, logs a warning in the log file and moves on.

There is more to the integration, but for administrators preparing software packages, it is important to know that WRP does not allow WIS to update any protected resource.

> **Note**: More information on WIS and WRP can be found at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/windows_resource_protection_on_windows_vista.asp.

In a way, this lifecycle is very similar to the software lifecycle, or the history of the software products you choose to use in your network. That's because they both begin with a request; someone somewhere decides they need to have or use a specific product, a product that will provide the particular features they need. In the packaging lifecycle (see Figure 8), this request can stem from a variety of sources, but most often, it stems from the user community.

In many cases, these requests are informal—a manager decides a product is required to better the productivity of their team; some user tells a manager they can't do without a given product; IT decides it is time to upgrade or change one of the core products in the network. Ideally, you should be using a tracking system that automatically ferries this request through the proper channels, locating the budget required for the acquisition or the development project if no commercial product responds to your needs, providing approval for the purchase or project launch, and delivering the product to be packaged to your doorstep. At the same time, this tracking system can let users or rather, requesters know of the status of their request and the expected delivery date for the product on their desktop.

Next comes the discovery. Here you begin the actual packaging process by examining the product to be packaged. How does it behave when installed? Are their prerequisites for its operation? Will it run properly on your operating system of choice? Does it conform to given Windows standards? These are the types of questions you need to answer and document before you can move on to further stages. One of the most important questions at this stage will be: is the product designed to work with the Windows Installer service or not? The answer to this question will greatly influence how you proceed in the next stages of your packaging lifecycle.

Figure 8: The MSI Packaging Lifecycle.

A key step in the discovery process is research and review of existing documentation sources for the application installation. Help may be found in locations such as a README.TXT file, included documentation, or the vendor's website (FAQs or knowledge bases). Unfortunately this information can often be difficult to uncover. One place where such information can be found for most applications is the AppDeploy Package Knowledge Base. This knowledge base

is a community contributed resource of information on the automated installation of applications on a product by product basis with a separate living document available for each version of a product installation. Details include:

- **Command Lines** — command line syntax for installation and removal of the software. May include available command line arguments and/or public properties.
- **Notes** — shared information on how to handle the automated installation of the software.
- **Virtualization** — shared information regarding how to work with the software in sequencing or creating a virtual deployment package for products like Altiris SVS and Microsoft SoftGrid (Softricity).
- **Terminal Services** — details on how to address problems working with this software in a Terminal Services session as with Citrix shared environments.
- **Related Links** — links to official and unofficial information regarding the deployment of the software.
- **Security Lockdown** — information regarding which files, directories and/or registry entries must be opened to standard users for operation in a locked down environment.

It should be noted that AppDeploy.com has forums for questions and discussions on this subject, but the Package Knowledge Base is set aside for the sharing of facts. The most commonly reported method of package development (repackaged, transformed, scripted, etc.) as well as a simple difficulty rating is voted on by members which can provide you with a quick starting point to judge how difficult an application you are about to face.

> **Note**: The Package Knowledge Base may be found at http://www.appdeploy.com/packages.

The next step is what most people consider the packaging process itself. This is where you prepare the automation of the installation and configuration of the product you intend to deploy. If the product is compatible with Windows Installer, you can customize it by creating a transform manually or by capturing the installation and configuration settings in a file that may be applied to the original product installation to modify its behavior according to your needs and requirements. If the package is not compatible with Windows Installer, you'll want to convert its legacy installation into one that will work with this service. In this case, you can capture the installation and configuration settings into a special file that will serve as a new installation executable.

Once you've created the initial package, you need to test it to ensure that it behaves exactly as you expect it to. There are a number of different tests you can perform at this stage: automated or silent installation test, pull installation (manually launched on the workstation or server from a network location), push installation (automatically deployed with your systems management software), uninstallation, and so on. But system tests are not the only tests you require. More often than not, software packagers find themselves preparing a product for deployment that they've never heard of or have very little experience with. This means that it is difficult for you to fully understand if the configuration you devised for this product is properly designed or if the eventual user will approve of this configuration. To validate the configuration, you'll need to involve an experienced user in the testing process. Their role will be to ensure that the product behaves as it should once deployed. This is normally referred to as acceptance or integration testing and is often formalized to identify the expert users who perform it as software owners, people who will be responsible for more than just testing, but also for recommendations on product evolution once it is in formal use in the network.

Another form of testing is conflict detection. It was only a few years ago that we began to realize the impact of "DLL Hell" or the impact of trying to make a multitude of products from different vendors, developed at different times behave properly on a single system. In fact, DLL Hell became so prevalent that Microsoft finally decided to implement a single installation standard for software products in Windows: the Windows Installer service.

While this service has been rightly hailed as a savior by many systems administrators because of its many features for the support of the coexistence of misbehaved products on a single system, it is not the be all and end all of conflict resolution. Take, for example, the integration of two products on a system: product A and product B. Product B includes components that are not compatible with product A, yet both must be installed on the same system.

The solution? Convert both to an MSI installation to integrate them to the Windows Installer service if they don't already support it. Thanks to the magic of this service, incompatible programs can coexist and operate on the same system. Should product B's components damage product A's, Windows Installer's self-healing capabilities will automatically repair the damaged product the next time it is launched, so long as Windows Installer has access to the source installation files. But you could find yourself in a DLL Hell Loop: When product B is launched, it breaks product A, but fixes itself, when product A is launched, it breaks product B, but fixes itself and so on. This may be because each program uses a different version of the same DLL or requires the same registry keys with different settings. Nevertheless, you certainly don't want your users to see Windows Installer launch each time they start a program on their system.

That's why conflict detection and resolution are still required. Despite the fact that Windows Installer can handle conflicting situations on the fly, you want to control the behavior of packages in your network. This means ensuring that products are well-behaved when delivered. Detection is handled by comparing the package you are working on with the others you have deployed (or a subset of those deployed) and can even handle a snapshot of your baseline system image to ensure conflicts with Windows itself is accounted for. Resolution can

be handled a number of ways and is discussed in more detail later in this chapter when discussing Conflict Manager.

Once all testing and conflict detection is done, you need to perform a final quality assurance on the package. Ideally, the person performing this final QA activity will be different from the person who originally packaged the product. This will provide you with a better and more thorough verification. Don't forget to complete all documentation about this package at this stage. Too many organizations try to save time by leaving documentation until after the product is deployed only to find out some critical component was missed and they have no means to find out how to repair it.

So now your package is ready for release. At this stage, users begin to hearten because your package tracking system has announced to them that they will soon see the package on their own desktop. All they have to wait for now is deployment. Your release process should fit smoothly with your deployment system, automatically integrating the package into the source package store that your software deployment team will use as the source for deployment.

Now that the package is out, you feel that your job is done. Unfortunately that is not so. *The packaging lifecycle does not end with deployment.* That's because today, it seems that released software looks and feels like Swiss cheese and must constantly be patched to maintain its operational consistency and protect those who have chosen to use it. Patches, hot fixes and service packs are a fact of life and will have varying impacts on your packaging process depending on when in its own lifecycle you decide to deploy a product. If you deploy the product after it has been out for some time, you will need to integrate its patches and possibly service packs to the package before you deploy it. If you have decided to use a new product in its infancy, you will have to deploy patches for the product once it is in use in your network. In both cases, you will have to continue deploying hot fixes and other patch types throughout the lifetime of the package in your network.

In addition, you'll be constantly deploying new applications and updates on target systems that already include installed programs. This means you'll always need to take the installed applications into consideration whenever you test either the new program or the new update.

Then, once the package as reached the end of its usefulness to users or once the package has become obsolete, you'll need to retire it from the network. Retirement may also mean replacement if the business function the package fulfills is still required. If you plan to replace the package with a newer version of the same product, then you face an upgrade, once again a feature that is supported by Windows Installer.

3.1 To Package or not to Package?

While Windows Installer offers strength and resiliency for product installations, unfortunately, it is not the be all and end all of all product installations. In fact, there are some key products which should not and cannot be packaged through the Windows Installer service. They include:

- Service packs, hot fixes and some system extensions cannot be repackaged with WIS. Good examples of this are core Windows system component updates such as Windows Media Player or even the Direct X system. Along with the service packs,

these components make low level changes to the Windows system and therefore cannot be supported by Windows Installer.

- System File Protection (SFP) components are also excluded from WIS. As you know, components that are included in the SFP are automatically protected by the operating system and only the system can update files under this protection. Currently only Microsoft has the ability to modify files at this level.
- Special packages that are already included in deployment kits. For example Internet Explorer comes with its own Administration Kit (IEAK) which lets you package and deploy IE within your network. This type of product is best left to its original state because packaging it in WIS might break something that is automatically taken care of in the IEAK.
- Device drivers and network protocols are special components that must verify a system's configuration extensively before installation. It's not that WIS can't do that, it is mostly that the changes brought about by these installations are at such a low level that it is very difficult to package them into a WIS package. A good example is Adobe Acrobat. Because this product installed actual printer drivers, it took Adobe several years to modify its installation to the MSI format. Before this, many organizations invested heavily into the repackaging of Acrobat sometimes with little or no success. Now that it is an MSI, it is much simpler to work with.
- Finally, never repackage commercial MSI products. Remember that to modify these packages, you need to use a transform or MST and apply it at installation.

If however, your only mode of deployment is through Active Directory and Group Policy, you can still use Windows Installer to help deploy these types of components. Windows Installer may be used as a wrapper that may use custom actions to turn the installation into an MSI. Windows Installer doesn't actually perform the installation, but it does perform the delivery of the package and launches the SETUP.EXE that is required to perform the installation.

> **Note**: To be fair, it is important to mention that using existing command-line arguments to install a product that is not in MSI format preserves the vendors provided installation logic and is supported by the vendor. Repackaging has many benefits, but it also adds complexity. If a problem with a repackaged installation cannot be reproduced in an interactive installation the vendor is very unlikely to provide support.

4. Best Practices for Using Windows Installer

Use the following best practices when customizing your packaging environment.

- Take the time to carefully review the Windows Installer concepts covered in this chapter. They will prove to be your guideline for package preparation from now on.
- Categorize all of your software packages into one of three categories: Native WIS Software, MSI-integrated Corporate Applications, and Repackaged Legacy Software.
- Make sure you use the latest version of the Windows Installer service.
- Learn the ins and outs of the msiexec command so you can access Windows Installer features through the command line.
- Maintain your workstation lock-down status at all times. Use Windows Installer features to help maintain this environment.
- Work with the Windows Installer service as much as possible, but learn when to make exceptions. Keep your exceptions to a minimum at all times.
- Try to limit your feature installation to Run all from My Computer or Not Available. Avoid using Installed on First Use to provide a better experience to your users.
- Use per machine installations everywhere. This means using the ALLUSERS=1 property either inside the package or on the command line when installing packages. This will always create managed applications.
- Always use the ROOTDRIVE=C:\ property or better yet, design your workstations with a single disk partition, the C: drive.
- Always create and apply mst files to customize commercial msi packages.
- If you create your own msi files, make sure you store all user data into a single component.
- Make sure all of your msi files use unique product codes. This will ensure that msi products can coexist on the same machine.
- Make sure you update all deployed packages when you modify the package by applying a patch.
- Make sure you apply the proper GPOs to control the Windows Installer service in your network.
- Use Software Restriction Policies to control which msi packages are installable in your network. The two best methods are Certificate and Path rules, often used together.
- If you use certificate rules for SRP, make sure you set up an auto-enrollment PKI before you deploy packages that are digitally signed. This will avoid placing this burden on users.

- Make sure you manage source lists properly. In fact, you should be using a domain-based DFS share strategy for package installation sources.
- Cache packages locally on portable systems.
- Adapt the MSI Packaging Lifecycle to your own environment and integrate it into your corporate packaging policy.
- When using WIS 4.0, make sure your packages are integrated to the Restart Manager.
- When using WIS 4.0, make sure packages are installed as managed applications. This will avoid upgrade issues in the long run.
- When using WIS 4.0, make sure that patches are digitally-signed and that the certificate is in the MsiPatchCertificate table. The best way to do this is to add it to your custom msi template so that it is automatically added to any new msi you create.
- Verify your WIS 4.0 packages to make sure they do not include any resources protected by WRP.

Now you're ready to use Windows Installer to its best abilities.

> **Note**: Details on a video on the subject of **Repackaging Best Practices** may be found at http://www.appdeploy.com/video.