realtimepublishers.com™

The Definitive Guide™ To Windows Software Deployment

NEW BOUNDARY TECHNOLOGIES

Chris Long

Introduction

By Sean Daily, Series Editor

Welcome to *The Definitive Guide to Windows 2000 Software Deployment*!

The book you are about to read is not only the definitive technical resource for information on Windows 2000 software deployment, but represents an entirely new modality of book publishing. The founding concept behind Realtimepublishers.com is the idea of providing readers with high-quality books on today's most critical IT topics—at no cost to the reader. Although this may sound like a somewhat impossible feat, it is made possible through the vision and generosity of corporate sponsors such as New Boundary Technologies (formerly Lanovation), who agree to foot the production expenses for the book and host it on their website for the benefit of their website visitors.

It should be pointed out that the free nature of these books does not in any way diminish their quality. Without reservation, I can tell you that this book is the equivalent of any similar printed book you might find at your local bookstore (with the notable exception that it *won't* cost you $30 to $80). In addition to the free nature of the books, this publishing model provides other significant benefits. For example, the electronic nature of this eBook makes events such as chapter updates and additions, or the release of a new edition of the book possible to achieve in a far shorter timeframe than is possible with printed books. Because we publish our titles in "real-time"—that is, as chapters are written or revised by the author—you benefit from receiving the information immediately rather than having to wait months or years to receive a complete product.

Finally, I'd like to note that although it is true that New Boundary Technologies' Web site is the exclusive online location of the book, this book is by no means a paid advertisement for them. Realtimepublishers.com is an independent publishing company and maintains, by written agreement with the sponsor, 100% editorial control over the content of our titles. However, by hosting this information, New Boundary Technologies has set themselves apart from their competitors by providing real value to their customers and transforming their site into a true technical resource library—not just a place to learn about their company and products. It is my opinion that this system of content delivery is not only of immeasurable value to readers, but represents the future of book publishing.

As series editor for the Windows IT series at Realtimepublishers, it is my *raison d'être* to locate and work only with the industry's leading authors and editors, and publish books that help IT personnel, IT managers, and users to do their everyday jobs. To that end, I encourage and welcome your feedback on this or any other book in the Realtimepublishers.com series. If you would like to submit a comment, question, or suggestion, please do so by sending an e-mail to feedback@realtimepublishers.com, leaving feedback on our website at www.realtimepublishers.com, or calling us at (707) 539-5280.

Thanks for reading, and enjoy!

Sean Daily

Series Editor

Copyright Statement

Chapter 1: The Foundation of Software Deployment

Spouses of auto mechanics are the first to admit they have the least maintained cars. Often this isn't an idle complaint—it's true. What they mean more than anything else is that their car isn't maintained as well as it should be considering there's an auto mechanic in the family.

I would imagine the same holds true for plumbers, carpenters, lawn maintenance professionals, and software developers. Hah! Caught you off guard didn't I? I admit the last one may be localized to only my household, but it's an example I can share with you. Often, when someone is good at something and they do it for a living, they don't have the time to spare or the inclination to apply that talent within their own lives. Call it human nature. After banging on a computer all day, I'm not eager to jump on the computer when I'm at home.

Last year my wife and I purchased a nice computer. Given all the computer books and computer training classes available, I've been impressed with the computer abilities of an eight-year old and his six-year old brother who have no access to these resources.

With minimal guidance, they've managed to enter my email address into every conceivable online contest, download cartoon screensavers, install desktop themes (usually involving spaceships and aliens), browse the Internet, play online games, and even let me know how much gaming systems are going for on eBay.

The obvious result of all of this tweaking is that user interface design has certainly improved over the past 10 years. These boys also managed to cripple the printer (repeated paper jams) and generate inconsistent hardware errors (the recordable CD drive is no longer reliable). Not to mention all the disk space these activities consume. Amidst the grumbling, I'm trying desperately to rationalize why my own home computer is in such a mess.

Typically, none of these problems have been big enough to disrupt my weekend. After all, I do most of my work in the office. Only when the family computer comes grinding to a near halt does maintaining the home computer escalate to the "A1" (with a bullet) priority on my To Do list. The squeaky wheel gets the attention and all of that.

After a few months of listening to these complaints, I am eager to reduce the impact on my weekend. I've since created separate logon accounts for the boys, the printer became off limits for all activities except printing, downloading software became an "On Approval" only action, and the recordable CD drive was replaced with a newer model. Sure I haven't solved all of all the problems, but the ones I have fixed certainly reduced my workload and I now have more time to enjoy the new DVD player.

Getting to the Point

What does all of this have to do with software deployment? Good question and I'm glad you've stuck with me. Like my attitude toward the home computer, anything to do with getting an application onto the user's computer has historically been an afterthought. It's the leaky brake line to our successful automobile repair shop. It's the crabgrass in our own yard to our wildly growing lawn-mowing business (sorry, I couldn't resist).

For software manufacturers, software deployment is a necessary evil to reach the ultimate goal of creating software. After all, software deployment has nothing to do with the problems the software was originally written to solve. It's a by-product of creating the software. It's a customer's need for convenience. Ah, therein lies the rub.

Software manufacturers' lives aren't made easier by software deployment—it makes customers' lives easier. In fact, software deployment is the one aspect of software creation that is completely driven by the customer. While each revision to a software application stems from competitive market analysis, internal feature set creation, and customer feedback; software-deployment improvements are motivated by customer pressure. Typically, customers reduce the cost of deployment by pushing back on the software manufacturer. The software manufacturer in turn, puts in extra effort to make its application easy to deploy.

What then is software deployment? The answer to this excellent question conveniently serves as a great introduction to this book. A software program is first formulated as a result of a need—a potential customer need. From there, the program is pitched to a company's senior staff. If approved, the idea is sent to marketing to verify basic assumptions such as; will enough people purchase this software to make the development costs worthwhile?

If marketing finds that a need exists, the idea is sent to the product team to create a software specification. Development and quality assurance (QA) folks use this document to create and test the software. An install program is created, boxes are made, shrink-wrap is applied, and the software hits the streets. Software deployment encompasses everything the customer goes through to disseminate the software after it has been purchased. By defining what software deployment is not, we've defined what it is.

This description is important and the way it's described is also important. Software manufacturers know what the customer will do with their software; but they don't know how the customer will arrive to the state in which they can use the software. How the customer gets to that point is software deployment. Later in this chapter we'll come up with a more realistic model, but for now this is the path we're going down.

What Is Software?

Don't groan, but an important part of discussing software deployment is making sure that our definitions are well understood. For this reason, I want to sidetrack a bit and define software. The definition we're going to derive is pretty much what you and I think software is. The only addition is we're going to define software in terms that are convenient for a software-deployment discussion.

Originally, software was monolithic. By that, I mean the executable was self-contained: It used few, if any, external resources and no executable cared whether any other executable existed.

A lot has happened over the past 20 years to change that definition: More people are using computers than ever before. More non-technical people are using the computer. The software market is more competitive than ever before.

In fact, the sheer number of people using the computer today for the bulk of their work has created a whole new economic classification: The use of the computer has increased worker productivity to the point that it impacts the nation's economy. That's simply amazing.

But this productivity wouldn't have developed as quickly if software manufacturers hadn't improved the software-creation process. In addition, we certainly couldn't have become this productive without a high-level of connectivity. These two culprits—rapid development and higher connectivity—are the primary generators of the software-deployment issues we face today.

Rapid development is a term that spawned from a significant change in software development. Instead of a monolithic executable, software today is a collection of software. Third-party vendors contribute a great deal to the software-creation process. This collaborative effort is a strange paradox: On one hand, this effort lets software manufacturers plug in functionality that would be prohibitive to create internally. On the other hand, as a result of this collaborative effort, third-party software that should have reached the end of its life is still shipped and active because a third-party software manufacturer can retract or modify its software only through the application's creator.

Higher connectivity plays a role in software deployment by bringing a higher level of conformance from user-to-user. With a larger exchange of data, software applications share a larger amount of how data is processed. This idea ties into componentized software. Why should I reinvent how to access a specific database if I know someone's software component can do it for me?

Software Is a Collection of Software

The phrase "software is a collection of software" is a euphemism for componentized software. Through published interfaces and a construction cleverly called *dynamic linking*, any library can talk to any other library. In fact, a strong argument could be made that today's software development efforts are primarily spent gluing components together rather than creating new components.

An excellent example for this argument is the Windows operating system.. With the advent of the Windows application programming interface (API) set, developers could leverage common code. This common code is provided by a core set of dynamic-link libraries (DLLs) that ship with Windows. These DLLs make Windows-centric tasks, such as re-sizing, moving, opening, and closing windows convenient. These DLLs aren't limited to graphical functionality—virtually anything done in a Windows application has to go through this core set of libraries.

At the same time Microsoft released Windows, the company was pushing new technologies and new functionality. These upgrades were also made available through a collection of libraries. Since these libraries didn't ship with the original operating system release, they were termed *redistributables*, and software vendors were allowed to ship these redistributables with their software applications.

Today, bundling redistributables with the install program of software applications is expected. Open Database Connectivity (ODBC), Microsoft Data Access Components (MDAC), Visual Basic Runtime, and Microsoft Common Controls are just some of the redistributables available today. The current list of Microsoft redistributable files numbers in the high hundreds. Third-party software vendors are also in on the act. In fact, install programs commonly include third-party redistributable software. For example, Adobe Acrobat and Macromedia Flash are two widely installed software redistributables.

Software Connects to Software

The linking of a Windows application to its underlying libraries is complex. Figure 1.1 illustrates just how complicated things can get. This figure shows how libraries provided by different vendors rely on the presence of other libraries for correct loading. A missing library generates a system error message. An incorrect version of a library can lead to subtle runtime errors. Contrast this figure to the days of DOS, when an application sat directly on top of the operating system.

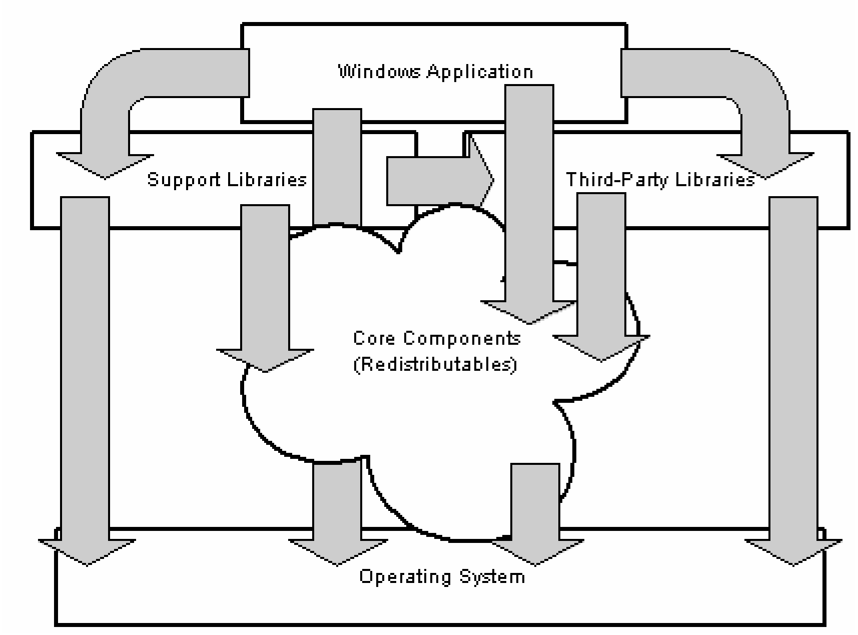


Figure 1.1: Through library linking, applications strongly depend on the operating system and support files.

In addition to redistributables, there are always updates to the operating system. These updates consist of hotfixes, maintenance releases, and service packs. These updates usually update the underlying Windows core DLLs.

Just to keep things relatively sane, each DLL was usually accompanied by a numerical version (for example, 4.1.23.6). This version number is usually accurate; however, editing the version resource is the responsibility of the developer. This version number is used during the update process to make sure newer files aren't overwritten. Did you catch the two *usuallys* in the previous sentences? Well, when these tasks aren't done, things get really messy.

Linking

The software manufacturer divides the development effort into a variety of tasks. This division of labor is done not only from an architectural point of view but also for a nice division of resources. Lisa and Donna can create the computing engine, while Jason and Jesse can develop the database query system. In each case, the functionality results in multiple support libraries. The separate libraries come in handy when there's a spin-off project that also needs a computing engine or a database query system.

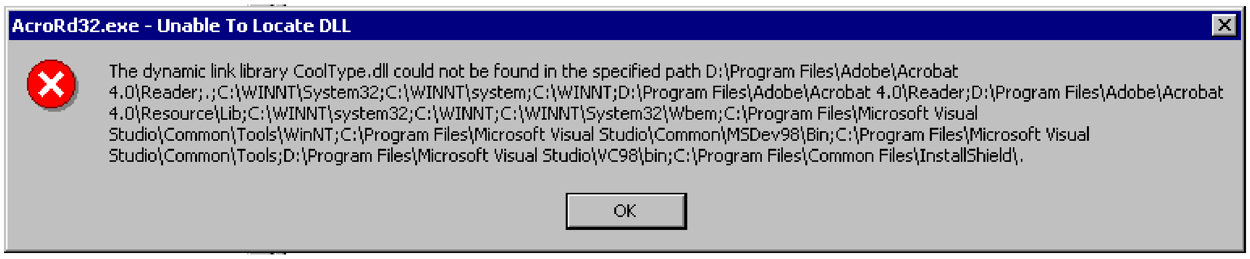
The arrows in Figure 1.1 illustrate the reliance the entire application has on external resources. Of course, software deployment would be fairly straightforward if we could always guarantee the correct library is available when it's needed. Unfortunately, such isn't always the case.

The first problem is that determining the complete list of libraries required by an application is an extremely difficult task. The reason this task is so difficult lies within the application itself. An executable can use a function provided by one of these DLLs in two ways. The first is to actually embed the function inside the executable. This is called *static linking*.

Static linking is much like bringing your own bed on your vacation. Sure, it's comforting to know that it's there, but you're taking on a lot of overhead. If you know there is a bed where you're going, you could take the leap of faith and leave yours at home. Alternatively, if you apply the bed analogy to *dynamic* linking, you'll wait until you need to sleep, then you'll expect the bed to be underneath you when you lie down. Therein lies the problem—sometimes there is no bed and sometimes it's not the bed you expected (a bed of nails versus goose down).

From a software-deployment point of view, things get even a little bit trickier. Not only is there dynamic linking, but there are two types: implicit and explicit. Implicit linking means that a list of libraries an executable requires to run is located within the header of the executable. This arrangement is useful because when the executable loads into system memory, the operating system will track down the necessary libraries. Explicit linking means the application will attempt to load the library when the application needs the file.

The error that occurs as a result of a missing library depends on the type of linking. If an implicitly linked library is missing, the error will be immediately obvious because the operating system won't allow the application to launch. I imagine you've seen the error before, but I've provided Figure 1.2 to jog your memory.



```
The dynamic link library CoolType.dll could not be found in the specified path D:\Program Files\Adobe\Acrobat
4.0\Reader;.;C:\WINNT\System32;C:\WINNT\system;C:\WINNT;D:\Program Files\Adobe\Acrobat 4.0\Reader;D:\Program Files\Adobe\Acrobat
4.0\Resource\Lib;C:\WINNT\system32;C:\WINNT;C:\WINNT\System32\Wbem;C:\Program Files\Microsoft Visual
Studio\Common\Tools\WinNT;C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin;C:\Program Files\Microsoft Visual
Studio\Common\Tools;D:\Program Files\Microsoft Visual Studio\VC98\bin;C:\Program Files\Common Files\InstallShield\.
```

Figure 1.2: The infamous Unable to Locate DLL error message.

There are very good reasons to statically link versus dynamically link to a library. While static linking guarantees the correct API call for the exact version—after all it's embedded within the executable, dynamic linking reduces the physical size of the executable. A key reason for componentized software, and thus dynamic linking, is so software applications can benefit from updated components. For software deployment, understanding the tradeoffs of static and dynamic linking is important.

There are pros and cons to implicit and explicit links. For example, I'm using Microsoft Word to write this chapter. On a whim, I loaded Word into a profiling utility. Because almost all Windows applications dynamically link at some level, I was curious about the extent of implicit versus explicit linking. Table 1.1 shows the number of implicitly linked versus explicitly linked DLLs.

Implicitly Linked	Explicitly Linked	
MSPANDB.DLL	AGENTMPX.DLL	NTLANMAN.DLL
KERNEL32.DLL	ATL.DLL	NTMARTA.DLL
ADVAPI32.DLL	BLNMGRPS.DLL	NTSHRUI.DLL
COMCTL32.DLL	CLBCATQ.DLL	OBALLOON.DLL
GDI32.DLL	CSCDLL.DLL	OLEAUT32.DLL
MSO9.DLL	CSCUI.DLL	RICHED20.DLL
NTDLL.DLL	DNSAPI.DLL	SAMLIB.DLL
OLE32.DLL	LINKINFO.DLL	SECUR32.DLL
RPCRT4.DLL	LXATSTRN.DLL	SYNTPFCS.DLL
SHELL32.DLL	LXATUI.DLL	VERSION.DLL
SHLWAPI.DLL	LZ32.DLL	WDMAUD.DRV
USER32.DLL	MPR.DLL	WINMM.DLL
WINSPOOL.DRV	MSGR2EN.DLL	WLDAP32.DLL
WINWORD.EXE	MSI.DLL	WS2_32.DLL
ACTIVEDS.DLL	MSSPELL3.DLL	WS2HELP.DLL
ADSLDPC.DLL	MSVCRT.DLL	WSOCK32.DLL
CRYPT32.DLL	NETAPI32.DLL	MSO9INTL.DLL
MSASN1.DLL	NETRAP.DLL	WW9INTL.DLL
MSLS31.DLL	NETUI0.DLL	
OLEPRO32.DLL	NETUI1.DLL	
USP10.DLL	NTDSAPI.DLL	
W32TOPL.DLL		

Table 1.1: Implicitly linked versus explicitly linked libraries.

If you've followed my discussion to this point, you know that the implicit linked entries are complete; however, since I didn't test every bit of Word's capability, the explicitly linked column is probably incomplete. In this example, there are about twice as many explicit as implicit linked libraries. I'll provide more information about implicit versus explicit files in later chapters.

You can download Dependency Walker, excellent profiling tool, from http://www.dependencywalker.com. This utility is freeware made available by Steve Miller, a Microsoft developer who has devoted an enormous amount of time to keeping this utility up-to-date with the latest Microsoft operating system releases. A version of this software ships with Microsoft Developer Studio.

Software Breaks Software

Microsoft has always stored its core libraries within the system folder under the primary Windows folder. Microsoft has attempted to maintain this folder as a repository for the operating system's files; however, nothing prevents software manufacturers from using this folder. In fact, manufacturers place their applications' shared files into this folder because the operating system automatically searches the Windows and Windows system folders when an application needs a library.

This functionality has added responsibility for an application's install process. In addition to file transfer, the install program must configure the application to sit on the operating system. I compare this responsibility to a matchmaker. The install program must take two unacquainted entities and introduce them in such a way that they take an instant liking to one another. As in real life, things can get complicated.

The operating system goes through a well-known search order t to find all the libraries needed by an application. Let's say my application starts from a unique location. It makes a request for a Microsoft Common Control. In this case, let's say the library is ComCtl32.dll. This file usually resides in the Windows system folder and contains some common Windows controls, such as the spin box control and IP address control. (I've selected this control because of its widespread use in Windows applications.) In response to my application's request for ComCtl32.dll, ideally, the operating system finds this file within the Windows system folder, loads the library, and it's ready for my use. During this process, a couple of things can go wrong. First, what if the library isn't in the Windows system folder? The operating system then peruses the rest of the search order. If it's not found through this process, a system dialog box will be displayed saying the system file could not be found.

Second, what if the operating system finds the file, but it's not the right version of the file? There are a couple of options at this point. The file may work just fine if the functionality my application requires is contained in the library. If it's not, any number of errors could occur depending on what is in the library and what my application intended to do with the library.

The biggest question is how could the library be the wrong version? Let's assume a best-case scenario: The install program made sure the correct version of the file resided on the system when it installed the application. Then, another install program came along and overwrote the existing ComCtl32.dll with an older version. My application is definitely broken.

But there are other ways things could go wrong. What if the existing ComCtl32.dll was overwritten with a newer one that wasn't completely backward compatible or wasn't compatible with the functionality my application needed? My application is still definitely broken. What if a second ComCtl32.dll was installed in a different location, but closer in the system search path? Same thing. The wrong version would be loaded before the one my application needs and my application would be broken.

DLL Hell

As you can see, there are a number of ways these shared DLLs can be manipulated to create havoc for Windows applications, thus the term *DLL Hell.* A classic example of DLL Hell goes back a few years and involved a file named WinSock.dll. This file, while a standard name, was implemented in drastically different ways by TCP/IP stack vendors. (This example takes place in the days before the WinSock.dll came bundled with the operating system.) For an install developer, a physical search of the hard disk was required before a new stack could be laid down. Any chance that a different WinSock.dll could be found had to be removed. Extraneous WinSock.dlls had to be renamed to avoid a collision. These requirements prevented problems from the install side, but nothing prevented another install program from laying down a different and incompatible library.

DLL hell is the primary reason that locked-down systems exist. In fact, large corporations commonly have rather draconian processes in place to keep the operating system shared libraries at a consistent version. These processes range from limiting what the end user can install to physically relaying the Windows system folder when the computer boots.

There are trade-offs with this approach. The more locked-down or standardized the computer, the less flexible it becomes for the user. Then again, the more flexible the computer is for the user, the more likely that version conflicts will occur. It costs money when things break. How much a company is willing to spend is a function of the corporation's motivations. Figure 1.3 graphs these tradeoffs.

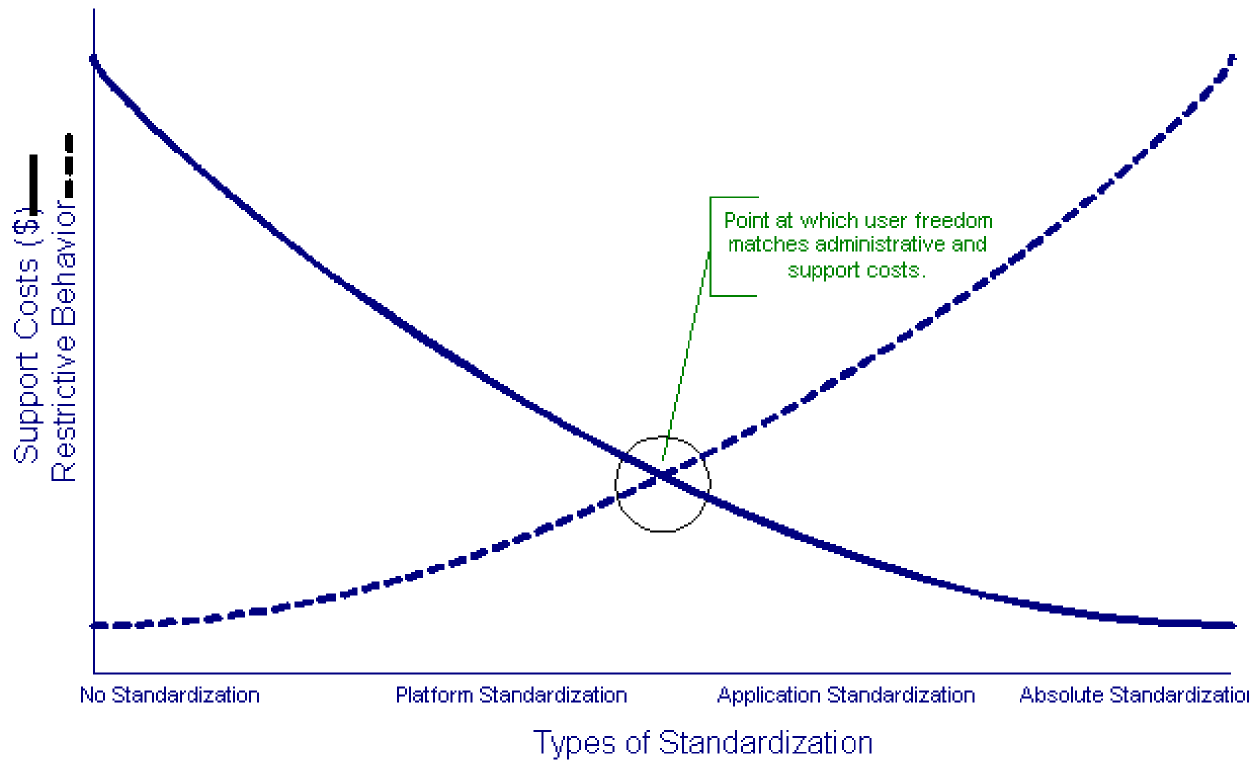


Figure 1.3: The tradeoffs between restrictive behavior and freedom to modify.

At some point, a happy medium exists. Most corporations recognize that the computer user requires some latitude. This balance is different for each environment, so the tradeoffs aren't always the same. In other words, the point at which user freedom matches administrative and support costs moves horizontally in this figure.

As a result of DLL Hell and the behaviors of some rogue install packages, a gradual state of system decay occurs after the first application has been installed. In today's world, computers will eventually lose functionality without careful monitoring. This decline is primarily a result of the difficulty in tracking and resolving conflict libraries.

Software Evolves

Microsoft provides two mechanisms to help reduce the impact of DLL hell: Windows File Protection (WFP) and side-by-side sharing of libraries. Both features are completely functional in Windows 2000 (Win2K) or later.

WFP ensures that core operating system files aren't inadvertently overwritten. This functionality is accomplished through the operating system. WFP monitors a select group of files in the Windows system folder. If any of the prescribed files are overwritten, they're replaced with the originals on the next reboot. This feature requires a DLL cache location where the original files are stored.

Side-by-side sharing allows applications to use a specific version of a file even if a different version resides in the Windows system folder. This functionality is accomplished by installing the application-specific version of the file locally with the application. A data file indicates that the local copy of the system file should be used.

Introducing Software Producers and Consumers

Up to this point, I've been calling the people who make software the software manufacturers. This terminology is a bit cumbersome and not entirely accurate because not everyone who provides software is the creator of the software. Along those same lines, "manufacturer" has some connotations that implicate more niche responsibility. By going up a level, and using the term producer we'll certainly capture all guilty parties.

☞ A software producer governs the creation of the core software system.

The same holds true for the computer user. This term is a bit too specific to be useful in many of our discussions. In some cases, it implies the user directly purchased the software or directly deployed the software to the computer. By using the term "consumer," we can capture the responsible audience.

A software consumer governs the purchase and deployment of the software.

Motivations and Responsibilities

In one of my favorite movies, the hero is being chased across a barren planet by a rock monster. The hero is in communication with the ship and asks for help to outwit the monster. The calm response, "What's his (the monster's) motivation?" Of course, it's funnier in the context of the movie, but the question is useful to this discussion.

If we understand the motivations of the software producers and consumers, we'll come closer to not only understanding software deployment but also the problems that litter the current software-deployment landscape. After we truly understand the problems, we can begin to look for solutions.

Software Producers

Let's begin with the software producers. They've got to ship software while schedules and resources are tight. If the software doesn't ship soon, the window of opportunity is lost and market share shrinks. Between product requirements and feature creep (the continuous addition of last minute "must have" functionality), just getting the software built, let alone tested, is an almost insurmountable task.

This daunting task is further complicated by a reliance on third-party components. Will the latest release of their software be compatible? Will they ever fix that one last outstanding bug? Were they being completely honest with respect to bandwidth limitations? Has the legal department finally signed that agreement? These are just a few of the questions that software producers deal with on the race to release.

After the software is released, the motivation of the software producer turns to its true goal—making money. One way to not make money is to release software that generates a lot of support calls. In fact, the ideal software release involves many sales calls, no support calls, and a lot of happy customers. The closer a software release gets to this ideal, the more profitable it will become. Just producing the software isn't sufficient for the software producer to be successful. A successful software producer must be motivated to lower the cost of supporting their software.

Software Consumers

The software consumer wants software that is easy to deploy and requires little maintenance overtime. In the eyes of the software consumer, keeping the software up to date through maintenance releases, hotfixes, or some other producer-initiated update process is the producer's responsibility.

The consumer is well aware of the cost of owning software, which isn't simply the initial purchase cost. This expense consists of all the additional costs associated with owning the software, including the cost to invest in better hardware to run the software, the cost associated with the loss of productivity when the software doesn't work according to expectations, the expense of additional training required on non-intuitive software, and the cost of negotiating license or site agreements.

Installing Software

Crucial to a software producer's success at lowering support costs is the ease of the software's install process. This point is an important aspect of software deployment. After all, the early models of software deployment were predominately installing the software.

A deployable software package contains the physical application files and the embedded knowledge to not only transfer those files to the user's computer but also configure the computer such that the application is usable. Ideally, the file transfer and configuration is done in such a manner that all other applications still work.

File Transfer

File transfer is the cornerstone of software deployment. If the deployment process can't handle getting the application binaries from the media to the user's computer, all hope is lost.

In the early to mid 1980s, most applications were typically on the order of a handful of diskettes. As applications grew larger, the size of physical media also increased. Early versions of Microsoft C shipped on less than five diskettes. Later versions (in the early 1990s) took 24 diskettes. Unfortunately this number of diskettes fell within the disk-failure rate and generated a substantial number of support calls related to failed media. Don't forget that phone support was free at that time.

Compression and joining algorithms were the hot commodities. Typically the application resided in a single folder structure. Compressing this folder and splitting the compressed file across the diskette media was the media-creation process. The application was extracted into a single folder on the user's computer. From almost the beginning, some file-integrity checking was performed.

All reasonable efforts were taken to reduce the number of diskettes. After all, shipping on one less diskette lowered the overhead and provided one less reason for media failure. In those days, the install program was little more than decompressing and joining software.

Often the ReadMe.txt file that accompanied the first diskette gave directions about any additional configuration. Usually, this configuration process involved a manual edit of the autoexec.bat and config.sys files. The end user would reboot the system manually.

During this period, large-scale distribution entailed little more than placing the image on the network. End users launched the image from the network to install locally. For smaller distributions, the system administrator had to visit each computer and install the software directly, especially if the install required any changes to system files.

The file-transfer process today has experienced little change. Most install packages compress the application files into a single image. Although this image can be split across physical media, this process isn't often used because the space required by the distribution image is well within the space available on a CD-ROM. In addition, compression algorithms are more efficient and provide better file-integrity checks.

File transfer usually encompasses removal of the application. In the pre-Windows days, uninstalls were virtually non-existent. Only within the past 5 years has removal of application files become commonplace and within the past several years that a true application uninstall has been around.

Configuration

Configuration is the other half of the application-install process. After all, we've long ago left the days when simply transferring the files to the user's system was the only action necessary for the software to be usable.

Autoexec.bat and Config.sys Configuration

When the install software had the smarts to edit files directly, autoexec.bat and config.sys were the first files to benefit. The first several releases of Windows didn't impact the install process substantially because most applications at the time were DOS based. The ability to edit these files gave the install program more responsibility over the install process. As Windows grew in popularity, file editing became a requirement.

And with this added responsibility of file editing came opportunities to fail. Because the operating system provided little more than reference documentation about the install process, software manufacturers were left up to their own devices when the time came to create the install program. Although there were a few dominant install tool providers, the install programs themselves ran the range from excellent to downright dangerous. For example, the PATH environment variable commonly exceeded its allotted length or for system drivers to be replaced without the user's knowledge. Discovering these shortcomings was all part of the learning curve for install developers.

System.ini and Win.ini Configuration

Like modifying autoexec.bat and config.sys, editing the initialization files for the Windows layer was commonplace in the Windows 3.x days. Although editing of config.sys and autoexec.bat required a system reboot, editing Windows initialization files required a restart of Windows itself. Today it's hard to remember back when Windows was distinct from the operating system.

Today these files are no longer used in the operating system. As the Windows layer permeated the operating system, the system registry replaced these files.

The Advent of Windows

With the discussion we've had so far, you can see the distinct line between pre-Windows and post-Windows software deployment. That line occurs with the introduction of a graphical user interface (GUI), which in turn coincides with the widespread acceptance of Windows 3.1x. The Windows' GUI has driven the current software-deployment trends, so I'll delve into this area a bit deeper.

Windows' GUI

The large-scale acceptance of Windows 3.1 was the turning point for software deployment. Windows 3.1 made the computer easier to use for non-DOS users. The graphical interface was made easier to navigate with the addition of the mouse.

The Windows interface opened new possibilities for developers. File extensions could be linked to applications. Double-clicking on a file could launch an associated application. Drag-and-drop functionality supported embedding of application objects. Graphics could be added directly into documents. What you see is what you get (WYSIWYG) became the buzzwords of software.

The graphical nature of Windows drove the graphical interface of the install process. This fact may seem trivial at first, but it opened doors to an entirely new user: the novice. The novice user rapidly accepted working on a Windows platform using a Windows application. The graphical nature was typically non-threatening. Designed correctly, the software install program provided safe default choices. Configuration changes, which historically had been left up to the systems administrator, were now handled automatically.

Shared Files

All those shared components that a software application requires are lumped under the generic title of shared files. Not only do these files have to be updated based on version number, but (on Windows) the installation of these files requires an update to an internal shared DLLs count.

The concept of shared files and the internal counter is a construct that first appeared in Windows 95. This functionality was an important step in maintaining system state. Previously, shared files were always left on the user's computer.

If you've ever seen the Windows system folder of a Windows 3.1 system, you know what I'm getting at. Literally hundreds of files were left orphaned on the computer with absolutely no way to determine whether they were actually still in use. Adding to this frustration was the fact that disk space was a fairly expensive commodity.

While the Shared DLLs count is a big step forward, it's not a complete solution. Because the install process updates this counter, the count can easily become inaccurate. In the worst-case scenario, a shared file could be removed prematurely.

The System Registry

The system registry truly came to life in Windows NT. Although a variation existed in Windows 3.1, it didn't exhibit the depth of system reliance that NT and later versions of Windows 9x exhibited. The system registry is the repository for hardware and software configurations—for individual user accounts and for the computer itself.

The vast amount of energy in deployment is expended on updating the system registry, especially for computers with multiple user accounts. The registry maintains a profile for each local account, so any software that requires initialization of specific user profiles has to be installed multiple times—once for each logged-in user account. This requirement is true even if the files go to the same physical location. The reason is the user's registry data needs to be instantiated.

Current trends in application design attempt to make this process easier by relying on the software application to populate the user registry data on first launch. In this case, not only are all user accounts taken into consideration, but all future users are as well.

The system registry is centralized storage for ActiveX controls. Each ActiveX control creates a series of CLSID registry entries during a process called self-registration. By invoking self-registration, the ActiveX server registers each of its controls. Amongst these registry entries is the location of the ActiveX library or executable.

Uninstalls

As Windows grew in acceptance, users developed some demands of their own. Key among them was the ability to uninstall an application. As a result, uninstalls gradually became commonplace (granted, it took 5 years). They even became quite good; however, it took some work. Completely uninstalling an application from a computer is not an easy task. It's not simply a matter of removing the application and it's configuration—it's a matter of returning the computer to a known working state without the application or its configuration present.

Beyond the Install

Not surprisingly, the install process alone doesn't completely define software deployment. To be truly representative, software deployment takes a bigger view. Let's take a stab at a preliminary definition of software deployment. After the discussion so far, I'd go with something like the following: Software deployment is the integration, installation, and support of software in a known environment. This definition includes how the media is shipped, the mechanism used to distribute the image, the conformance testing that occurs before large-scale distribution, the process by which adequate license counts are maintained, the decision of who gets the application and which parts, how the product will be supported internally, and so on.

The audience is a strong function of software deployment. Clearly, deploying software in a large corporation is different than in a small company. Technology is also a function of software deployment. A company that uses a software-management system is better suited for deploying software than one that doesn't. A company that can support the latest deployment features is one step ahead of one that can't.

Software deployment relies heavily on experience. Success relies on not repeating mistakes made by others as well as knowledge of the operating system. Success in software deployment requires an understanding of how Windows applications interact.

Figure 1.4 illustrates the role software deployment takes in the software-creation process. The user assumes the process and expense of software deployment. However, software manufacturers can make this process less painful.

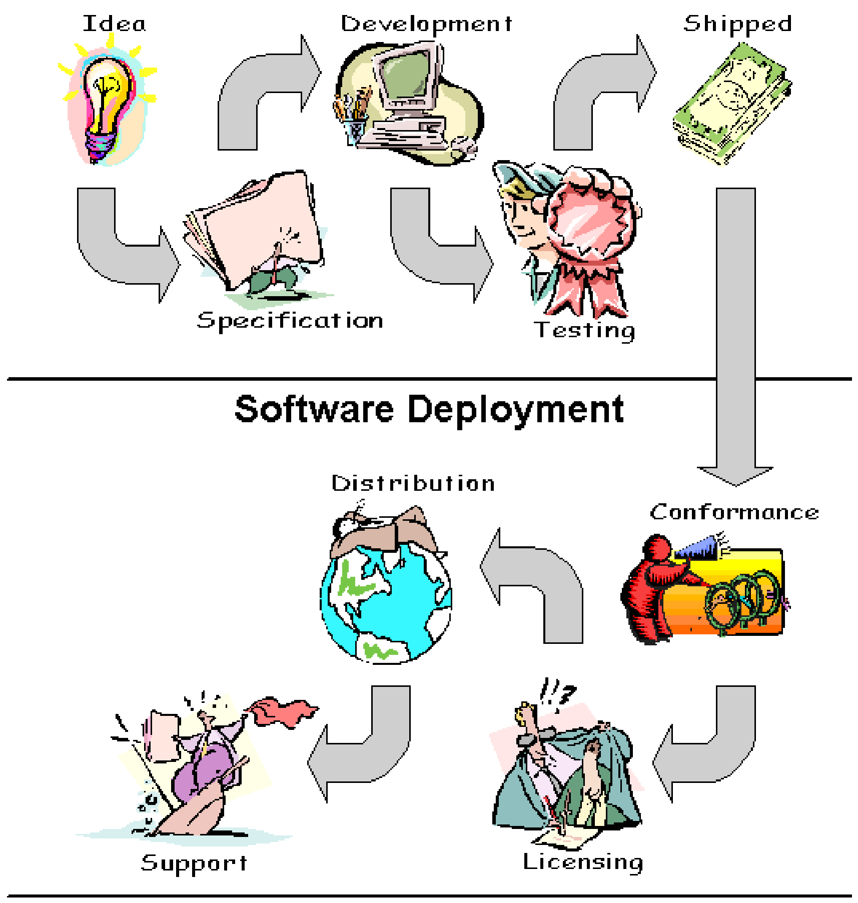


Figure 1.4: Portion of development cycle that encompasses software deployment.

The Evolution of Software Deployment

As this figures shows, software deployment is a big area. But this hasn't always been the case. Given the Windows timeline, software deployment for the bulk of Windows history has been concerned with simply installing the software. Only over the past few years has the area of software deployment matured. Originally, getting your software to work on a user's system was the biggest goal. After you got that to work, the goal became getting your software to work with existing applications. Once we were happy with that process, the issue became removing your software. And of course that was followed with removing your software without destroying anyone else's software. The rest of the software-deployment areas were picked up along the way.

Initially, software deployment was little more than getting a single executable onto the computer. Those were the early days of DOS, and most users were either technically savvy or had direct access to systems administrators. There was no GUI to the operating system, and applications were launched from a DOS prompt. The install process was fairly straightforward. It involved decompressing files from a removable media to the user's hard disk. To put you into the right mindset, these were the days of 5.25-inch floppies (300Kb and 700Kb!). Any configuration was handled through a ReadMe.txt file. Most users were technically knowledgeable to make changes to autoexec.bat and their config.sys files. For the most part, that's all that was required of software deployment.

A DOS-based delivery mechanism was more or less predominant through the late 1980s. The advantage was its simplicity. The primary disadvantage was its limited functionality—there aren't many things you can do from within a batch file. In addition, a DOS batch file didn't lend itself to mass distribution.

Within the past 5 years, software deployment has made great strides. Today's large-scale distributions involve thousands of computers—all administrated remotely. Management software can remotely deliver virtually any standard software package to a client. Remote administration is a foregone conclusion.

Repackaging

An excellent example of the importance of software deployment is the growth of repackaging. For about every three to five install programs written from scratch, there's a corporation repackaging of the same software. Why? Because the company's software-deployment requirements demand it, and repackaging someone else's software is cheaper then using the existing install program. There are strong advantages to repackaging software: additional software can be bundled, components can be removed, the user interface can be suppressed, and file conflicts can be tuned to the client's environment. By repackaging the software, these users can exert fine control over the deployment process. In some cases, repackaging leads to better integration with their software-management tool.

The evolution of software development closely mimics my situation at home with the computer. Initially, software was thrown to the end user with very little forethought. As the Windows operating system matured and penetrated more of the corporate environment, how software was installed, accounted for, and maintained became a concern for the software companies.

Those within the software industry know all too well that the critical gate to shipping software is the install program. Typically, it's the last on the list and sits on the back burner until shortly before the ship date. Seasoned developers have been known to run when the topic of the install program is broached. To be honest, this reaction is understandable. There's not much glamour in writing an install program. If you're successful and the install program is perfect, you won't get any kudos. After all, that's what it's supposed to do. However, if the install program is buggy or exceeds the allotted time to complete, the world crumbles, schedules slip, and project managers camp outside your office.

There is a distinct lack of enthusiasm when it comes to writing the install program. This lack of eagerness may be due to the high profile that comes with taking on the task. In moderately sized project teams, most software developers are assigned key areas of responsibility. Most have limited impact on the user interface of the application. If bugs are generated, testing can continue around the affected area. Rarely is the entire application crippled. The install program, however, is critical. One bug can cause the software not to install, and a bad bug can cause the user's system to be corrupted. Given those scenarios, which end of the software do you want to work on?

Figure 1.5 illustrates the explosion of software-deployment functionality within the past decade. For the bulk of the 1980s, a simple file transfer would do the trick. Within the past 10 years, customers have been much more demanding about how software gets onto their computers and what happens while the software is on their computers.

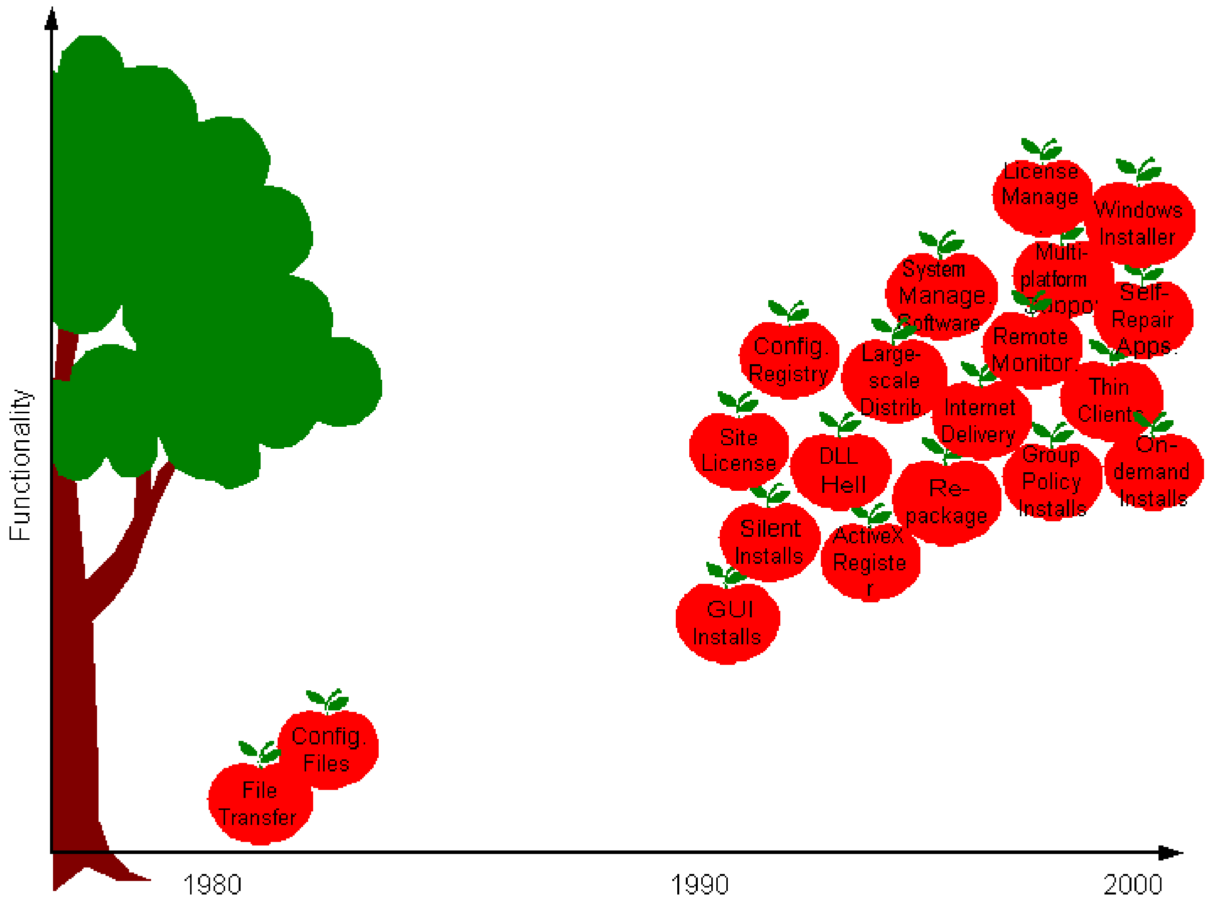


Figure 1.5: A timeline of software deployment's evolution.

The Growth of Software Deployment

Software deployment started out as little more than install development. As I previously mentioned, the install-development process was primarily compressing the application, splitting it across physical media, and transferring the files correctly to the user's computer.

As the operating system grew and customers demanded more, the install-development process matured. As the number of Windows computers grew within corporate environments, distributing software internally added more requirements to the install process.

This trend continues today. The advent of system-management software to oversee the distribution process has saved corporations money.

Large-scale Deployment

The entire discussion to this point culminates in large-scale deployment. Starting from such humble beginnings as DOS batch files, through the GUI, including the bells and whistles of Windows 9x and NT configuration, we arrive at the landscape of today—large-scale deployment.

The true test of software support for software deployment is large-scale deployment. Rolling out software to thousands of client computers is not a trivial task. Sure, clicking Next through a single install seems easy enough, but no one wants to repeat an attended install for thousands of computers—even a handful can be a test in persistence.

Beyond the annoying aspects of attended installations, systems administrators are concerned with what the install process will do to the existing and working computers. Is the install process going to break the stuff that is already working? Is the software application going to require a system update? Is the system update going to break working software?

For a large number of installations, systems administrators want to be able to roll out the software in a way that minimally impacts their day. They would prefer an install package that supports an unattended mode, can be remotely installed, and works with their system-management software.

Software deployment has come a long way since its DOS roots. The fact that most software applications can be deployed through system-management software is testament to this evolution.

The process of deploying to a large number of computers is separated into four tasks:

1. Conformance Testing—Verifying the software application will play nicely with existing software.
2. Licensing—Ensuring there are sufficient licenses for the expected user base.
3. Distribution—Getting the application out to all applicable users.
4. Support—Dealing with the inevitable.

Conformance Testing

Left to their own devices, new applications will interfere with existing applications. A process to eliminate, or at least substantially reduce, this interference is required in a productive environment. This process costs money. In fact, there are cases in which the cost of implementing a software release is prohibitively expensive.

When productivity gains exceed the costs associated with implementing the software, the software consumer is seeing a positive Return on Investment (ROI). Consumers need to be assured of a positive ROI before updating to new software is even considered. A large part of that assurance is conformance testing in which the new software is tested in small rollouts to detect incompatibilities. An investigation into incompatibilities determines how the software will be deployed. The likelihood for incompatibilities is largest when shared files are encountered. In many cases, the original install delivery method isn't the one used internally. There are a couple of reasons for performing conformance testing, but the main reason is to adapt the existing corporate computer's environment to accept the new software.

This adaptation process is primarily spent updating libraries or configuring the system to reduce the amount of file sharing. Deriving the conformance testing is the most time-consuming task. Commonly, conformance testing of one version of a software package wraps up just as the next release hits the streets.

Licensing

A large concern for software producers is ensuring each copy of their software is legitimate. Anti-piracy measures are easier to enforce within corporate environments than for individual consumers. Current deployment schemes usually include some method of accounting for license schemes.

Licensing implementations vary from the restrictive to liberal. A restrictive licensing scheme usually involves some method of verification before the application is usable. This verification might be as simple as a hardware-software handshake along the lines of parallel port dongles to client-server verification.

An example of a liberal licensing scheme is a site license. In this model, a software producer grants unlimited use of an application within a corporation.

Of course, there are licensing schemes that fall between these two extremes. Rolling licenses allow a fixed number of software users at one time. This implementation requires metering software to maintain a user count. When the number of users matches the license count, additional users are restricted.

Distribution

The true test of software deployment is distribution. After conformance and licensing issues have been worked out, the last big hurdle to jump is getting the software onto the user's computer. Typically, this process involves an investment of resources to design and repackage the software application.

Repackaging is done for a number of reasons:

- Provide a silent or unattended install method
- Move to a supported distribution scheme (for example, IntelliMirror)
- Remove conflicting shared files or resources
- Add or remove optional parts of the software application
- Bundle other software packages or augment the software application with internal components

Support

Support in the case of software deployment is largely concerned with implementing updates to existing software. Certainly software training is part of the investment that a corporation makes when rolling out new software, but it's not strictly part of the software-deployment cycle.

The Software-Deployment Life Cycle

Up to this point, we've seen that software exists on the user's computer almost as a living organism. There is growth and there is reduction. Adding software and removing software changes the environment for existing software.

Figure 1.6 illustrates the possible paths of software deployment. We'll be using this life cycle flow throughout the remainder of this book. It's important to understand that true software deployment must embrace each aspect of the life cycle.

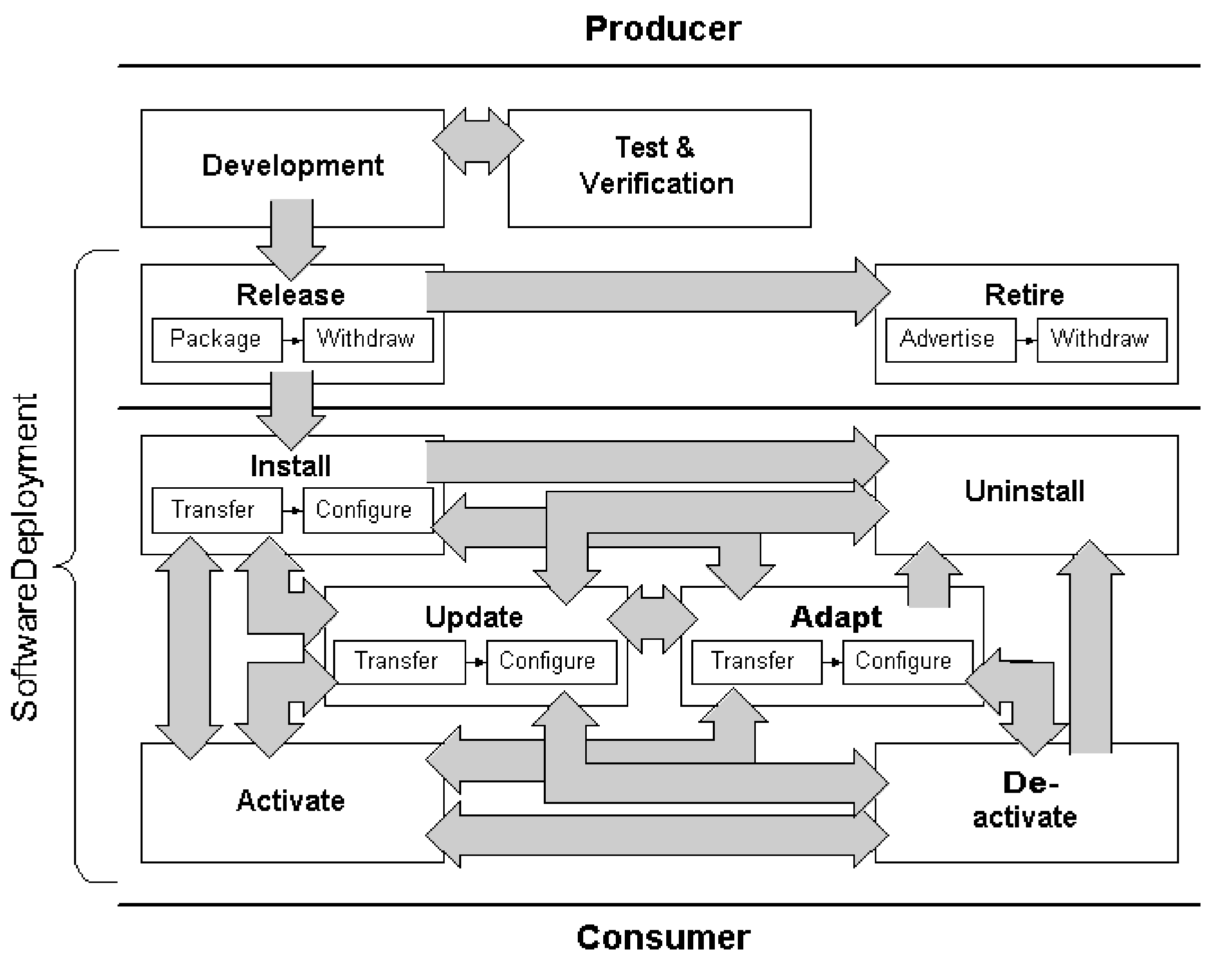


Figure 1.6: The complete software-deployment life cycle.

As Figure 1.6 shows, software deployment isn't just installing software. It's the transactional relationship between different software states.

Release

As the figure illustrates, the release stage is a producer concern. It's also the connection between software development and software deployment. The successfully released package embodies not only the requisite application files but also the embedded knowledge of the install and configuration process.

A release occurs for any change in the original package. Changes include maintenance releases, hotfixes, and any other change from the original software state.

Retire

Much like release, retiring software is a producer concern. When software is retired, it's no longer available for further deployment. More than likely, it's still maintained at many locations, but the producer would ideally like to stop it from propagating further.

Install

We've talked about the install process, but now we can see its role in the software-deployment life cycle. The definition of the install process remains the same as earlier. It embodies the knowledge of instantiating the software application on the computer. Part of this knowledge is evaluation. The other part is configuration. An interesting way to think of the install process is as an upgrade when the application doesn't exist.

Uninstall

When the software is no longer needed, uninstall provides the mechanism for removing the software from the system. In our model, we'll assume that uninstall is a complete and calculated task. After uninstalling the software, the remaining applications must retain their current state. Uninstalling should remove the application files and any resources that are no longer shared.

Along these lines, an effective uninstall examines the current state of software dependencies and constraints and removes the targeted software in such a way as to not violate the state of dependencies and constraints. In our model, an install isn't a simple matter of undoing everything that was done during the installation phase. Uninstall may first require a de-activate process.

Update

The update process modifies software that has been previously installed. By its nature, an update isn't as complex as the install—it involves a subset of the original install process.

In our model, the complete update requires a de-activate process, then the update occurs, and finally the application is re-activated. This set of steps may not be required and, depending on the software, may even be a hindrance.

Adapt

A crucial process to software deployment is the adapt process. During this process the system undergoes a new configuration by modifying the current state of one or more software applications.

Distinguishing an adapt from an update is important. An update modifies a specific software application, whereas, an adapt process modifies any number of existing applications. Normally, an adapt is a corrective measure.

Activate

The activate process is the connection between the dormant and active application. It's the mechanism used to indicate the application's functionality is required.

De-activate

The de-activate process is the connection between the active and the dormant application. Yes, you could say it's the opposite of activate. And you'd be correct. A de-activate request initiates the shutting down process of the application. This process would be required prior to an uninstall and may be necessary as part of an update.

Summary

In this chapter, we took a walk through the history of software development. The evolution of software design played a large part in the maturing of software deployment. In fact, the evolution of software development was a precursor to the current software-deployment life cycle.

Certainly, software deployment has matured with the proliferation of the Windows operating system. In turn, large corporations are highly motivated to trim costs of distributing software within their organizations. There are also other issues at play. As we'll see in the upcoming chapters, current trends in software design, such as multi-platform support and distributed computing, make understanding the software-deployment life cycle critical to the success of the software industry.

Many of the issues we've discussed have risen from pressure corporations have placed onto software manufacturers to make their software easier to distribute and maintain. We can see this situation by understanding the motivations of both software producers and consumers.

In Chapter 2, we'll take an in-depth look at the questions large-distribution consumers must answer in the process of deploying software. The answers to these questions drive which software applications are selected for internal use and how the deployment process is implemented.

Chapter 2: Conformance Evaluation

Whew! Conformance evaluation are big words for such a simple concept: How willing are you to install or upgrade software? For you and I as individual software users, the answer to this question revolves around cost, need, and expectations. Companies releasing software internally must ask the same kind of questions on a much larger stage.

If we're interested in a software package and its minimum requirements exceed those of our 3-year old home computer, how likely are we to upgrade our computer to use the software package? Odds are not good if the need for the software package is half-hearted. We're more likely to upgrade a personal computer to move to the latest version of Microsoft Office than to Wing Commander. But it's a function of disposable income and other non-related priorities.

If you spend a few minutes thinking about the software you've purchased, you'll probably realize that you invested more than the simple cost of purchase. You've spent the time installing the software onto your computer. This process may have involved clearing up some disk space and removing some older software. The install program may have involved tinkering with your system to get the install program to work—installing an updated Internet browser for example. More than likely, you also spent some time learning the software—either through the manuals or online Help. We can safely say that you've developed a type of relationship with your new software.

If we look at an investment of time, energy, and money into our software as a relationship, conformance evaluation is the courting phase. It's the period of time you spend thinking, evaluating, and price shopping before you purchase software. In this chapter, we'll address the issues of distribution and support of the software.

As individual software user's, we have it relatively easy. Software consumers that represent large volumes of users—in the order of thousands—have a very difficult task. The process is pretty much the same, but the scale of investment is several orders of magnitude greater and the costs of success and failure are also a lot larger. This task is what the task that we're going to focus on in this chapter. Even though the quantity of software packages is larger, the concerns of large-scale software consumers are identical to those of individual users.

Large-volume software consumers start by determining the level of need for the software package. They need to understand if the applicable computers meet the minimum requirements for the software. They also need to demonstrate that the investment of time and energy to distribute the software will lead to a cost savings. These factors and more need to be considered, evaluated, and answered before the software-deployment process can start.

In the remainder of this chapter, we'll follow the trail of software deployment. We'll look at everything from evaluation to installation to support. This trail will lead us through the following phases:

- Acquisition—Incorporates the process of verifying whether the software is necessary and achieving a plan of deployment.
- Distribution—Includes delivery, maintenance, and reporting of the software packages.
- Support—Encompasses the responsibility of maintaining the existing software in a productive state for the end user.

The remaining sections deal with these phases in much more detail. After reading this chapter, you'll have a complete idea of the software-deployment process. More importantly, you'll know which questions and tasks need to be resolved before beginning the task of deploying software.

Acquisition

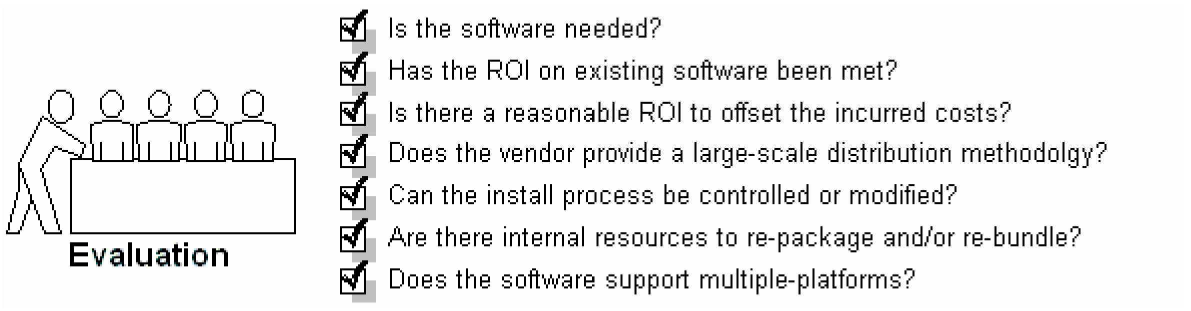
The process of acquiring software isn't as simple as you'd first think. Not only is there a fair amount of work to do before the software is purchased, but there is also an awful lot of work that must be done after the software is inhouse. In any event, the results of the acquisition phase of software deployment determine not only if the software rollout will occur but also, generally, how the rollout will occur.

Let's start at the beginning. Acquiring software is a three-step process that involves an evaluation of need, testing for conformance to standards, and finally a retooling of the delivery mechanism:

- Evaluation—Determines whether the software is necessary in the working environment.
- Testing—Involves testing for hardware and software compatibility.
- Tooling—Incorporates reworking the existing deployment package.

Evaluation

The goal of evaluation is to determine whether the software is necessary and to estimate the cost of implementing the software-deployment process as well as the Return on Investment (ROI). At a minimum, the cost of implementing the software should be offset by its usefulness. In many cases, the actual deployment infrastructure (how the software packages will be distributed) will already be in place. Work done at the evaluation level determines whether the software is worthwhile and how deployment should proceed by answering the questions that Figure 2.1 presents.



- ✓ Is the software needed?
- ✓ Has the ROI on existing software been met?
- ✓ Is there a reasonable ROI to offset the incurred costs?
- ✓ Does the vendor provide a large-scale distribution methodolgy?
- ✓ Can the install process be controlled or modified?
- ✓ Are there internal resources to re-package and/or re-bundle?
- ✓ Does the software support multiple-platforms?

Figure 2.1: Check list of questions for the evaluation process.

Determine Whether the Software is Necessary

The first step is to verify the usefulness of the software. Do you have preexisting software that can do the same job? How about upgrades or extensions to pre-existing software? After all, if you've already got something on the computer that can suffice, the process of deploying software wouldn't be necessary, and a software extension or upgrade would be less complicated.

The need for the software is based on the problems it's expected to resolve. In some cases, the software may be a requirement, for example, to view or edit proprietary files. In other cases, the software may automate tasks that were previously handled by a time-consuming process. The list could go on, but the central point is that the end goal of deploying the software should be clearly defined.

Calculate the ROI

If current software can't meet the current need, and the need is imperative to accomplishing a corporate goal, the next step involves calculating the cost of deployment. The cost of the software plus the complete cost of deployment (installation, configuration, training, and support) is the investment cost. The money saved by using the new software is the benefit cost. Initially, the investment cost of the software will exceed the benefit cost of owning the software. Over time—the life of the software—the benefit cost should exceed the investment cost.

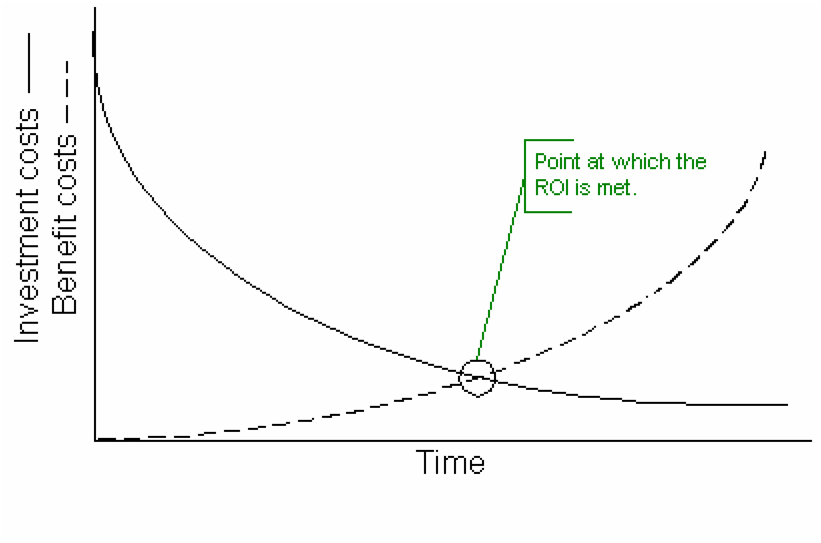


Figure 2.2: The point at which the software deployment effort is financially worthwhile.

In Figure 2.2, the ROI is met at some point after the software has been successfully deployed. In this figure, the complete investment cost is spread out over a period of time. As time goes by, the benefit costs offset the investment cost. While this is a simplistic model, it upholds the basic tenet of software deployment: Deploy only the software that will exceed its ROI. Conversely, don't replace software before it ROI is reached. If the need or the longevity of the software is uncertain, a careful reexamination of the software need is warranted.

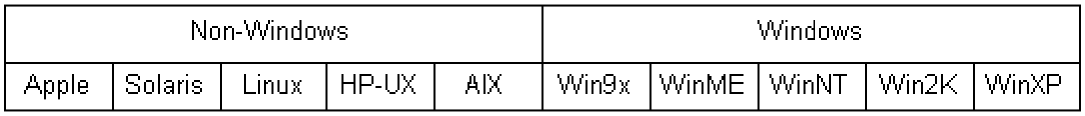
Determine Who Will Require the Software

Once a positive ROI has been calculated, the next step is to determine who requires the software. This number may be fixed initially, but as teams grow or the need becomes more widespread, additional copies of the software may be necessary. At this point, a worthwhile task is to associate the software with specific groups or departments. Some deployment methodologies make deploying across a group of users substantially easier than deploying to specific users.

Some corporate environments structure their system user groups specifically by software requirements. Although this structure covers the majority of cases, keep in mind that there will undoubtedly be exceptions. Deciding how to cover the exceptions should be part of the overall software-deployment strategy.

Develop a Plan Based on Operating System

Now that the need and users have been identified, the next step is to determine the computer system that these users have. All computer systems can be separated into two broad groups: non-Windows and Windows, as Figure 2.3 shows. This separation has as much to do as market share as anything else. Clearly in your work environment one operating system will dominate. Within the non-Windows group there are two distinct types: Apple OS and UNIX variations—although with the latest release of the MacOS, which embeds a FreeBSD variant of UNIX, even these types are becoming blurred.



Non-Windows					Windows				
Apple	Solaris	Linux	HP-UX	AIX	Win9x	WinME	WinNT	Win2K	WinXP

Figure 2.3: Classification of operating systems.

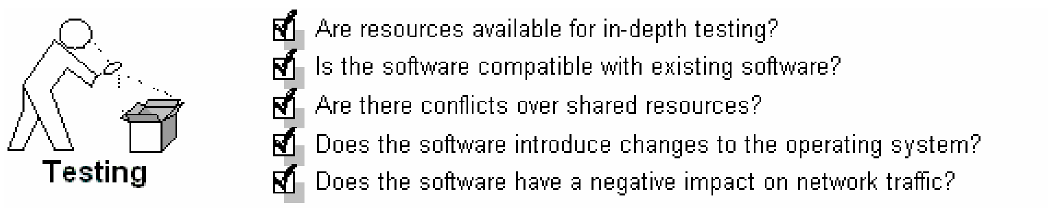
With the exception of software-management systems, there is no best-case solution that deploys to all operating systems. Typically, you'll find yourself having to come up with similar strategies with different implementations to support a heterogeneous platform base.

We'll get to software-management systems later on, but for large software deployments system-management systems offer a cost-effective way to transparently cope with software deployment on heterogeneous systems.

Compatibility Testing

Now that we know the software is required and it's cost effective to deploy the software, the real work begins. Making sure the new software plays nice with the existing software is an important part of software evaluation. Without thorough compatibility testing, introducing new software could be disastrous with effects ranging from intermittent software anomalies to complete operating system failure.

In some cases, compatibility testing extends to the hardware. A complete test would include a range of computer types. Not only would this testing cover brand-to-brand systems but also client computers, server computers, and laptops. If the software has stringent video requirements, for example, testing the software with a large range of video cards and monitors may be in order. If the software has large memory requirements, testing the software on typical corporate systems with a large range of memory capacities would be worthwhile. Figure 2.4 provides a list of questions for you to answer during the testing process.



- ☑ Are resources available for in-depth testing?
- ☑ Is the software compatible with existing software?
- ☑ Are there conflicts over shared resources?
- ☑ Does the software introduce changes to the operating system?
- ☑ Does the software have a negative impact on network traffic?

Figure 2.4: Check list of questions for the testing process.

Compatibility testing starts by defining the typical user computer configuration. In cases in which computers are heavily standardized, this process can be straightforward. In environments in which computers are freely modified, this process is not only complicated but also impossible to test for complete compatibility.

The typical configuration should include standard hardware and software combinations. The boundaries of the configuration definition should be realistic. After all, a theoretical matrix of testing could yield hundreds of test scenarios when in reality there might be 20 or 30. The constraints of the configuration definition should be based on the group of users for whom this software is targeted. After all, it doesn't make sense to include those computers that aren't applicable for the software deployment. When setting up the initial test scenario, be sure to use the most constrained security settings most likely encountered. This setup will force incompatibility issues between security settings and the examined software. You might need to revisit the exact security settings for the application to work effectively. An example is access to the specific registry keys on a Windows system or read/write privileges to data files.

Table 3.1 illustrates a sample matrix, which can help to ensure all possible configurations have been thoroughly tested. It's also an excellent reference document for the support folks. Replicate this table for each OS as appropriate and compile a similar table for the hardware requirements.

Existing Software	**Test 1**	**Test 2**	**Test 3**	**Test 4**
Microsoft Office	✕		✕	
Corel Suite		✕		
WordPerfect				✕
Netscape Navigator 5.0	✕		✕	
Netscape Navigator 6.0		✕		✕
Internet Explorer (IE) 5.0	✕			
IE 5.5		✕		✕
IE 6.0			✕	
MDAC 2.1	✕		✕	
MDAC 2.5		✕		✕
NT Service Pack 3 (SP3)	✕			
NT SP4		✕		
NT SP5			✕	
NT SP6				✕
Status (Success/Failed):				

Table 2.1: Sample configuration-testing matrix.

Not shown in this table, but just as important, are the variations introduced by notebooks and laptops versus desktop computers. While less of a factor today, older models may have more of a discrepancy. Older models of notebooks and laptops have distinct hardware—different video, smaller hard drives, and less memory—which may not be compatible with the testing software.

Create a Reproducible Test Environment

In practical terms, configuration testing starts by creating a reproducible test environment. At a minimum, this environment would be a single computer whose hardware could be reconfigured to match the results of the hardware-matrix table. In addition, the operating system itself could be reinstalled multiple times—even for multiple operating systems.

The importance of reproducibility cannot be overstated. The ability to quickly and accurately reproduce any single test scenario is imperative. Although hardware changes would require physically adding or removing components, software changes could be instantiated through replication software. There are a number of software packages that capture a system's binary state and store it to a data file for later use. For a range of configurations, these system image files may require large amounts of disk space, although burning the image to a CD-ROM is certainly an option for smaller test scenarios.

A well-known imaging software package is Symantec's Ghost. You can find this software at http://www.symantec.com.

Some failures in reproducibility are a result of subtle differences in hardware. Just because two computers have the same model number and external case doesn't mean they're identical. Some hardware manufactures will substitute similar internal devices when original sources become scarce. Opening up the computer and logging its exact hardware contents may be worthwhile.

The reproducibility of tests is important for a number of reasons. First, as a sanity check, reproducible tests are useful to verify issues or validate later assumptions. Second, for software support, they're ideal to quickly reproduce a client's configuration locally even though the actual problem computer may be hundreds of miles away.

Once a test environment is created, the process is quite simple. Set up the hardware and software for each of the individual matrix checks. Install the to-be-deployed software, and test. After a single matrix test, you'll quickly see why this phase of software deployment is the most time consuming.

DLLs and Shared Resources

Remember the discussion about static and dynamic linking of libraries from the Chapter 1? Well, bad news. From that discussion you'll see why the entire functionality of the new software must be exercised. In addition, the entire functionality of the pre-existing software (from the software matrix table) must also be fully exercised. To avoid DLL Hell, which I also discussed in Chapter 1, you can rely on a sufficiently thorough software matrix to uncover any version incompatibilities.

In addition to these two very important items, there are other things to consider. The bulk of these considerations have to do with shared resources. Shared resources include file extensions, user-document workspace, ActiveX controls (through self-registration), ODBC databases and drivers, fonts, shortcuts, and so on.

A classic example of a shared resource is file extensions. In a Windows–operating–system environment, applications can be linked to specific document types. The document type is identified by the file extension. On most systems, the extension .txt is associated with Notepad.exe. This association means that double-clicking a document called Readme.txt will launch Notepad with the selected document active. Although this association isn't usually an issue, when two applications vie for the same extension or any shared resource for that matter, you'll have to make a decision about which one wins.

A classic example is Web browsers. When installed, Internet Explorer and Netscape want to be associated with .htm and .html extensions. However, only one application can be associated with an extension at a time. Another example, are common picture extensions such as .gif, .jpg, and .bmp. These extensions are usually associated with the same viewing software. The same holds true for video data files with extensions .avi and .mpg.

Another often-forgotten test point is uninstalling applications. I'm not specifically talking about the new software—any software will do. Sometimes a resource that is shared is inadvertently removed during the uninstall process. This removal has the nasty tendency to break the software that is staying behind. This kind of event should be noted for latter conflict resolution.

Software-Automation Tools

For some companies, investing the time and energy into software-automation testing tools may be worthwhile. Although these tools can be expensive and time consuming, they compensate by being reusable. In this case, you create an initial automation test script to verify compatibility of the new application. You run the new automation test in tandem with all the existing tests. After the application is accepted and deployed, its automated test is saved and used for compatibility tests for future software. Over time, the savings in test time and resources will be substantial.

The Results

Are you going to be able to achieve 100 percent test coverage? Absolutely not. But the closer you get, the better you'll be able to predict the success of the software-distribution process; not to mention the reduced number of support issues, which should be enough incentive to gain as much test coverage as possible.

If you've created a thorough hardware and software matrix, most of the work is done. Any issues should be adequately documented for later conflict resolution. The testing and verification phase is time consuming and tedious, but it's in no way challenging.

The results of compatibility testing determine the configuration steps necessary to use the software. What you'll end up with is a hefty document that specifies all the detected conflicts. The next step is to gain sufficient resolution on each of the conflicts to proceed with the deployment effort.

Now that you've seen what compatibility testing entails, you may be interested in hearing more about standardization. After all, the closer all computers are in pre-existing hardware and software, the easier compatibility testing will be. The smaller we can make the hardware and software matrix tables, the less compatibility testing needs to be performed.

Standardization

As we saw in the previous chapter, there is a way to reduce the amount of work required during the evaluation phase. You could simply lock down users' computers. Restricting what users can change on their computers drastically reduces the differences between systems. And, because compatibility testing is a function of standardization, testing for compatible software is easier.

The basic premise is simple: the effectiveness of software deployment is largely dependent on having an accurate picture of the target platform. In fact, follow me closely here, software deployment is a cost-reducing benefit that is directly related to the cost-reducing functionality of computer standardization. Although that fits nicely in the sentence, it may leave you scratching your head.

At the extreme level of standardization everyone has the exact same computer—hardware and software included. Keep in mind that this discussion is completely theoretical. If you can buy that premise, you can easily see that deploying software in this kind of environment is very simple. By completely removing the matrix table, compatibility testing is limited to a single computer! Before you roll your eyes, this description is exactly the scenario for the home consumer—that's you and me. The only luxury the home consumer doesn't have is testing the software before installing it.

Range of Standardization

Standardization doesn't even have to be so clear-cut. There is a full range from completely open systems to locked-down environments.

For starters, computers can follow a platform standard. In this case, each computer abides by defined standards. These standards limit the computer to a list of available operating systems, client computers, laptops, servers, and network communication devices.

Computers may be required to follow specific application standards. In this case, each computer abides by defined software standards. Typically this standard is a list of version-specific, vendor-specific applications with standards established for specific users and user groups. Application standardization includes setting up purchasing policies and procedures as well as monitoring and managing application distribution, installs, and updates to assure that compliance is maintained.

Enforcing Standards

Enforcing standards is accomplished in a couple of ways: managed environments or locked-down environments. A managed environment keeps users from making changes to their systems, such as introducing unauthorized software or changing settings that may cause conflict with other system resources. In addition, a managed environment controls the ease of use of the desktop, providing a common set of applications and access for groups of users or individuals. In this manner, users are presented with only the tools they have been trained on and need for the job. This environment assures that changes are managed.

A locked user environment is a limited version of the managed user environment. It precludes users from changing settings and installing unauthorized software. It is different than a managed user environment in that it is machine specific and is local to the device. It is not typically managed or synchronized with a server profile. The primary reason for a locked user environment is that it makes a predictable target for software distribution.

Conflict Resolution

Okay, after a couple of months of testing, we've got a stack of incompatibilities. They could range from the serious to the trivial, but we still must sort them out and arrive at workable solutions. In every case, additional work is required to discover the exact cause of the conflict. Maybe an existing application breaks when the new software updates to MDAC 2.6. Does the new software absolutely require MDAC 2.6? We don't know and we'll have to investigate further.

For each incompatibility, we'll have to attribute the incompatibility to a piece of software. From there, we'll need to clearly define the exact problem—having imaging software helps greatly in reproducing the problem. Then a couple different solutions will have to be applied and retested.

During this process, you'll benefit from playing with each of the software packages in different configurations. You may discover the incompatibility is caused by an optional component of the software. Removing that component removes the problem and resolves the conflict. However, every incompatibility may not be so easy to resolve.

Let's take a deeper look at some of the standard issues. The most common incompatibility is versions of system files. This incompatibility usually involves a software package installing a new core component. This new core component may or may not be required. Depending on how the install program was written, a developer may have just grabbed the latest and greatest from Microsoft's Web site and slapped it into the install package.

The conflict-resolution process involves a series of tests. First, replace the new core component with the oldest possible component for your environment. Once again, test each software application for complete functionality. Chances are you'll replace one incompatibility for another. Keep updating the core component—perhaps while the original core component wasn't sufficient, a version in between resolves the incompatibility.

Part of the conflict resolution may require retooling the install package. This requirement is especially necessary for some incompatibilities: If you remove shared resources during an install, retooling the install package is the only solution.

If the incompatibility is less serious, a simple runtime script run before or after the install program may be a remedy. Of course, this solution would imply that the existing install program meets all other requirements.

When the work is done, you'll have documented each conflict resolution with an applicable fix. Starting the compatibility testing again with the documented conflict resolutions is an excellent way to verify the software-distribution process. The results of the conflict-resolution phase determine the configuration steps necessary to use the software. Additional lab testing verifies the fixes. This testing is where the imaging process from compatibility testing will come in handy.

In the cases in which nothing works, a reevaluation of the software need may be warranted. It may be simply that existing software must be removed for the newer software to be installed. The complete documentation set should be made available for the support team.

Tooling

In virtually every case of conflict resolution, changes must occur within the software-install process. Because the software application typically already has an install wrapper, retooling or recreating the install program is a necessary part of software deployment.

The conflict-resolution process may all come down to the simple fact that the install package doesn't conform to the deployment objectives. In this case, an install program that conforms to those objectives must replace the existing install program. Figure 2.5 illustrates the questions to answer during the retooling process.

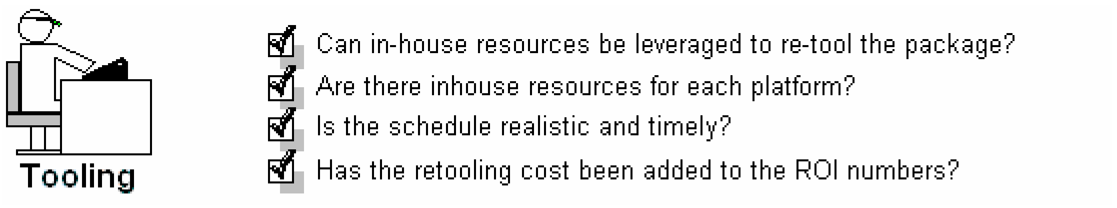

Tooling
✓ Can in-house resources be leveraged to re-tool the package?
✓ Are there inhouse resources for each platform?
✓ Is the schedule realistic and timely?
✓ Has the retooling cost been added to the ROI numbers?

Figure 2.5: Check list of questions for the tooling process.

Don't get me wrong; retooling a software application isn't limited to conflict resolution. The following list provides several other just as important reasons to retool:

- To resolve incompatibilities and conflicts
- To bundle additional software
- To add or remove components within the distributed software
- To eliminate or customize the user interface
- To facilitate a proprietary software-distribution mechanism
- To provide additional install configuration

Although resolving conflicts and removing the user interface are the most common reasons to retool, it may also be a way of bundling additional software. This bundling could include other software applications or simply proprietary extensions to the software. Instead of using three separate installs, you could retool them all into a single package.

For example, suppose that you're delivering Microsoft's PowerPoint. By creating your own install package, you can include corporate templates and sample presentations. You could also choose to remove unnecessary PowerPoint components and create corporate-specific shortcuts.

The useful thing about retooling is that you seldom have the same constraints that the original software manufacturer did. When the original install program was created, the range of supported computer configurations was quite large. Retooling for your own environment is usually a more manageable task. For example, although the original software may have supported all flavors of the Windows operating system, your inhouse requirements may be just to support Win2K.

There are two methods of retooling: repackaging and creating an install program from scratch. Although creating an install from scratch is functionally the same as repackaging, the term repackage is associated with a specific install-package creation process. Thus, we'll stick with that usage here.

Repackaging

Repackaging is the process of extracting an application from one delivery vehicle and inserting it into another. Repackaging involves capturing the changes made to the computer while the original install program is running. (Although this functionality is what is advertised for the sake of simplicity, every repackaging tool on the market compares the changes before and after an install is performed to generate a package and doesn't actually capture the changes made to the computer while the install is running. I elaborate this point in Chapter 3.) There are specific software packages that provide this functionality for you automatically. Some even go so far as to recreate the install package automatically. Of course, you'll want to get into the package to make your modifications.

A classic use of repackaging is to remove the user interface from the install program. In this case, perhaps the original install program didn't provide an unattended or silent install. By tooling our own install program by repackaging the software, we can create a suitable solution.

There are limitations to repackaging: Repackaging software detects and replicates only changes to the system. Thus, if a file is already on the machine on which the repackaging task is being performed, the file will not have changed during installation and won't be included in the repackage. Therefore, the file will be missing if pushed to a machine that doesn't already have the file installed.

In addition, there are some technologies that the repackaging system is not familiar with—on Windows systems, configuring Internet Information Server (IIS) for a virtual directory or Web server is a common limitation.

For the most part, the fundamental limitation to repackaging is that it doesn't capture the logical evaluation process of the original install. Some repackaging software provides more options to you than other repackaging software—shopping around and comparing features will help you greatly. The next chapter deals exclusively with repackaging.

Create an Install Program from Scratch

Alternatively, you can write the install program from scratch—a much bigger task than repackaging. In fact, if at all possible, repackaging is the preferred option. Writing an install program from scratch, although giving you greater flexibility, incurs the most responsibility.

A very rough time estimate comparison makes repackaging the quicker option. In fact, a typical repackaging process may take the most part of a week, including testing. Creating an install package from scratch on a moderately difficult application may take the bulk of a month. That estimate also includes testing. Creating an install program from scratch is discussed in detail in Chapter 5.

Lab Testing

After retooling the software install package, thorough testing will have to be done to ensure proper installation and configuration. After all, you don't want to introduce new problems in the process of resolving existing ones. There's nothing like success, and lab testing is a way of determining how close your newly tooled package is to success.

Distribution

Now we've done it. We've determined that the software application is important. We've resolved all outstanding software conflicts. We've even retooled the install package for the software—making install-specific changes along the way. The next step is to derive a strategy to get the install package to run on users' computers.

There are a number of ways to accomplish this goal. They range from making the install image accessible on a networked drive to administering the install process through a software-management framework. Which method you select is a function of cost and your final objectives. In this section, we'll spend a little bit of time talking about each one. Along the way I'll point out the pros and cons of each method. Figure 2.6 outlines the key questions you should answer for the distribution process.

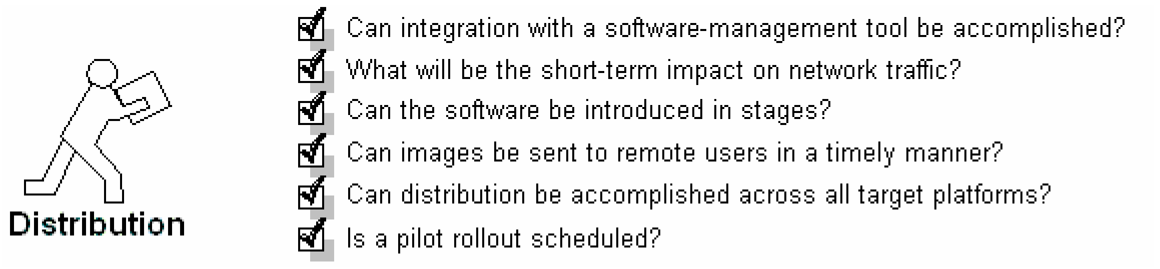

Distribution
Can integration with a software-management tool be accomplished?
What will be the short-term impact on network traffic?
Can the software be introduced in stages?
Can images be sent to remote users in a timely manner?
Can distribution be accomplished across all target platforms?
Is a pilot rollout scheduled?

Figure 2.6: Check list of questions for the distribution process.

There are some key components to the distribution process that are secondary to the physical distribution of the software:

- Provide warehousing support (catalog and store the package for delivery)
- Verify software-package integrity
- Report success and failure status of distribution attempts

Within this section, we'll also touch upon the topic of software-management environments. These are prepackaged software bundles that automate much of the software-deployment process. I'm introducing the topic here because this software is a must-have solution for large corporations. Chapters 6 and 7 go into much more detail.

Image Delivery

Delivering a software image can be done in a number of ways. The image could be a physical CD-ROM that is passed around the office. A link could be made on an Intranet Web site. The software could be installed directly to the computer without user intervention. Or an email could be sent to each user identifying the software location and requesting that the user install the software.

There are pros and cons with each delivery method. In this section, we'll talk about some of these tradeoffs. You should be able to get a general idea of the software-distribution mechanism appropriate for your environment.

The primary factor in determining your distribution method is the method that's currently in place. The second important fact is the number of computers that require the software. While my prioritizing doesn't do the "right solution" any benefit, I'm assuming that the implementation you're dealing with is correct for your environment.

If a distribution mechanism is already in place and working effectively, there's not much else to consider. Software-management systems are expensive and time consuming to implement. If your company has made such a selection, there were very good reasons for its selection. There's not much else to do but learn how the environment works. However, distributing software in a fresh environment is primarily a function of the number of computers that require the software.

Independent of how you choose to deliver the install package to users' computers, there are primarily two delivery modes: push and pull. A push install delivery mode is one in which the server drives the delivery, whereas a pull delivery mode is one in which the user (or client software) initiates the install action.

Push Delivery

By far push delivery is the preferred method of choice. It ensures that the distribution process is started for the targeted system and can usually record a success or failed status. A push delivery can also make sure that a valid license exists for the targeted computer and keep track of the complete license use of the software.

There are many remote options available. In general, the targeted computer must be powered on (although most system-management software and hardware options provide a remote boot option). Usually, push installs are accomplished at night when most users aren't active on the computer and network traffic is minimal.

In addition to the standard mode of delivery, a push install can also advertise the availability of the software application. In this model, the initial advertisement is pushed to the computer, but the actual installation of the corresponding software is done through a pull install. Advertisement is supported natively in Win2K and Windows XP; however, the mechanism for implementation is well documented and could be extended for use with other operating systems. There are two forms of advertisement: assigning and publishing.

Assigning

Assigning the availability of an application gives the appearance that the application is completely instantiated on the user's computer. All shortcuts are present, all self-registered actions are present, and all file extensions exist. Basically, everything exists but the application itself.

Assigning is useful when an application isn't required by a user, but is available when needed. The strength of assigning is that it doesn't consume disk space until some request for functionality is made.

When a request for use is made, the operating system makes a request to the Windows Installer service. The Windows Installer service tracks down the source media image and initiates the installation of software files. Note that this process occurs silently behind the scenes. To the user, there is a delay between the selection of an advertised application and the actual launch of the application. During the delay, a status dialog is displayed to show activity.

Publishing

Publishing is a subset of assigning. In this case, the software isn't directly available through the user interface. Instead installation is triggered when other software makes a request for the software. Similar to assigning, when a request is made, the operating system makes a request to the Windows Installer service and the install proceeds as the previous section describes.

Pull Delivery

A pull software install is the easiest to implement. It's also the least expensive. It typically makes a network, URL, or intranet location available to the user and relies on the user to trigger the install process. This process has the drawback that it's difficult to force the user to install the software. A much needed security patch may languish because the user fails to trigger the installation. In addition to not being able to enforce installation and configuration management, a major reason for not implementing pull software installations is security. By default, users often don't have sufficient rights to perform software installation.

A variation of the pull install is to create an entry in the boot process. In a Windows environment, you could create an entry in the boot process through the RunOnce registry subkey, a logon script, or the Startup folder. The next time the computer boots up, the install process is kicked off. While the original entry in the boot sequence was made remotely, the fact that it's still client-initiated (in this case by the reboot action) makes this method a pull delivery. This method suffers from the usual pull delivery drawbacks.

Pilot Rollouts

An important facet of distributing the software is testing the distribution methodology on a representative sample group. The point of the pilot rollout is to test our assumptions about the target systems, the install package, the bandwidth, and so on. Not to mention that recovering from disaster is easier when it affects a small number of computers.

The first requirement of the pilot rollout is to find a good variety of target computers, groups, and users. This variety allows the rollout process to be exposed to the largest number of distribution circumstances. If remote and mobile users are a concern, they should be included in the pilot rollout.

If a pilot rollout detects a substantial drain on network resources, distribution to intermediate deployment servers may be in order. A deployment server has the effect of offloading network traffic through different connection points. In software-management terminology, these servers are known as software distribution points (SDPs). Another alternative is to stagger the software distribution so that different parts of the company receive the software at different times.

Any modifications that need to occur as a result of the pilot rollout should be documented and included with compatibility-testing and conflict-resolution documentation. The support process requires this kind of documentation to resolve issues after the final distribution.

A final part of the pilot rollout is to test accessibility of all groups to the software package. This testing includes distributions that occur through firewalls, virtual private networks (VPNs), and dial-up connections.

Software Management

A good software-management tool distributes and installs software. It compliments these functions with a sweeping range of support modules. These may include reporting, license monitoring, and network monitoring to name but a few. The clear advantage to using software-management tools is the automation and black box approach to software distribution—I mean distributing a software package to a user may be a simple click-and-drag action—the underpinning of what is really happening is not apparent.

Software-management software comes in several forms. It can be a focused software-deployment solution or it can be bundled within a larger enterprise-management framework. The distinction is that software deployment is a subset of the enterprise-management framework—typically a plug-in module. The larger enterprise-management framework includes many functions in addition to software deployment. We'll go over the details in later chapters. This section introduces the terms and concepts commonly associated with the software-deployment side of software management.

At a basic level, the role of software-management software encompasses the following areas:

- Inventory—Keeps a record of what is currently installed and by which user
- Delivery—Provides a means of getting the install package to users' computers
- Licensing—Keeps a record of in-use versus available licenses
- Reporting—Creates a variety of status reports
- Warehousing—Physically provides storage space for the distribution images
- Managing components—Performs a limited automated support role for the distributed software

The primary goal with this type of software is to provide a centrally managed location for the bulk of software-distribution tasks. Centrally managed implies ease of administration and clearly defined status of installed software and pending software releases.

Inventory

A software inventory is an up-to-date listing, preferably electronic, that contains detailed information about the client, network, and server software installed within an organization. Inventory information is used for troubleshooting, license compliance, and strategic planning.

Ideally, the information should also be automatically collected, updated, and maintained. An ideal inventory would include license terms and conditions, date of acquisition, user names and location, system installation details, maintenance agreements, usage monitoring, history, and other relevant data.

Without a software-management tool, software inventory would be limited to keeping track of purchased software. For small companies, this inventory could be a spreadsheet or hard copy. In this case, the inventory is manually updated and contains minimal information about the software. The objective is to meet minimum requirements to avoid breach of license challenges, and provide upgrade proof and volume purchase leverage.

A more advanced software inventory includes regular inventory of software from purchasing through distribution. At this level of detail, the inventory process would be used with asset-tracking and system-discovery tools.

Delivery

Of course the delivery aspect is the most crucial to our focus. In this case, software-management tools typically provide their own version of user and group profiling. This functionality allows software to be deployed based on policy management.

By creating a new user and associating the user with a specific computer and appropriate user groups, the corresponding software is automatically deployed. The software delivery vehicle in most cases is an unattended push installation to the client machines. Because of the extensive reporting, success or failure is often automatically indicated to the administrator.

My personal favorite is the ability of these tools to automatically schedule software deployments based on network traffic. When network traffic falls below an administrator-specified amount, software distribution is kicked off. If for whatever reason, network traffic picks up to a point that exceeds the administrator-specified threshold, software distribution is temporarily halted. This ability to monitor the network and automatically schedule software releases is truly impressive.

Licensing

Licensing is a key component of software distribution. The bulk of software-deployment tools support some form of integration with licensing mechanisms. The advantage to having licensing centrally managed is that the software-distribution administrator has immediate record of license status. Typically license status is part of the built-in reporting process. Licensing functionality includes software metering to manage the use and control the number of licenses. Many solutions also support the dynamic balancing of licenses across licensing servers.

Reporting

With built-in reporting, systems administrators have immediate access to the success and failure of software rollout. Additional reporting functions include auditing and management reports as well as network status and loads over time.

Warehousing

Warehousing is the physical storage of distributable software images. It usually incorporates the added benefit of license integration. In this case, not only does the software store the software-media image but also lets you know how many licenses are currently used and how many remain.

Policy Management

Policy management allows administrators to assign individual users to specific groups. Simply by belonging to a managed group, the user has access to job-appropriate software. Users can become members of multiple groups if their job function overlaps or to make software distribution simpler.

Manage Components

The manage components feature of a software-management tool provide automatic self-healing of distributed packages and catastrophic system recovery. Self-healing is usually provided by monitoring the client systems and keeping a record of the install status. A significant change in the client computer status would trigger a redeployment of the affected software.

Catastrophic system recovery is similar but on a larger scale. In the case of software failure, the recovery entails a reimage of the system followed by redeploying all subscribed software applications.

The manage components feature additionally tracks and manages software compliance according to specific rules. Events that fall outside these rules trigger a report to the systems administrator.

Finally, all managed software is included in regular reports. These reports usually include such details as software type, revision, user name, user number by software title, and a record of valid software installs versus failed installs.

Support

After a software application is installed to users' computers, there is an inherent responsibility to monitor the software to make sure it continues to function correctly. Added to this responsibility is that of educating the user about the software.

In addition to these two important tasks, support is the central repository for the process history of the software. The results of compatibility testing and any conflict-resolution processes should be documented and made accessible to the support team. This kind of information goes a long way to quickly resolving implementation issues as they arise.

The support facet of software deployment encompasses the following roles:

- Ensure software and computer integrity through file management.
- Manage the software-licensing schema.
- Resolve software issues through the documented deployment process.
- Serve as a repository for hotfixes, maintenance packs, and upgrades.
- Manage deactivation and decommissioning of out-of-date software.

Figure 2.7 provides a check list of questions for the support process.

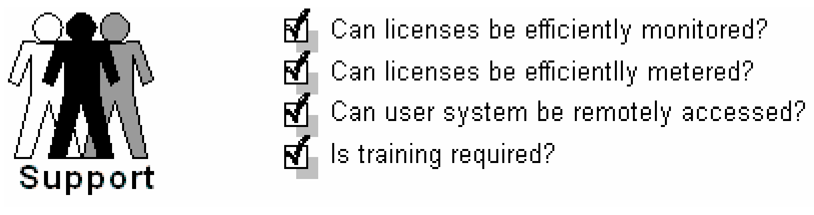

Support
Can licenses be efficiently monitored?
Can licenses be efficientlly metered?
Can user system be remotely accessed?
Is training required?

Figure 2.7: Check list of questions for the support process.

Software Repair

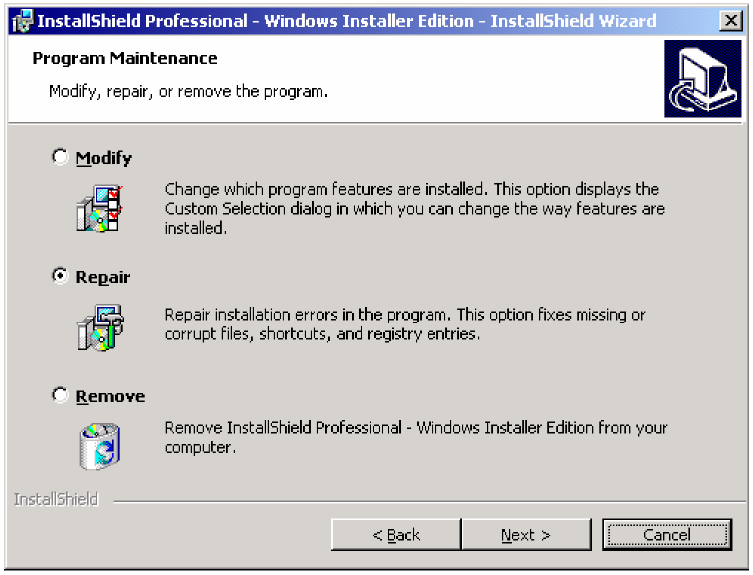
An interesting aspect of software support is repairing software that becomes damaged. The key is how to implement the check and subsequent repair before the damage resource is encountered by the application. In a locked-down scenario, a record of acceptable system files is compared against those on users' computers. Using a bit-pattern detection scheme, the files are compared on a case-by-case basis. When an invalid file is encountered, its acceptable equivalent is pulled from a central repository.

With Windows 2000 and XP, the operating system itself protects system files through Windows File Protection. Whenever the operating system detects an attempt to replace any of these special system files, a corresponding restore sequence is initiated to restore the system to its acceptable state.

While this process works fine for system files, it doesn't help the application binaries out too much. Here again the Windows operating system is on the forefront. Through runtime resiliency, the Windows Installer service can detect missing, corrupt, or the wrong version of key files or other installable resources. When corruption is detected, the Windows Installer service restores the appropriate component (technically this component may entail far more than an individual file; especially if repackaged, the entire application may be seen by the Windows Installer service as one big component)and associated installable resources from the originating source image.

Short of operating system intervention, there are limited avenues to incorporate true runtime software repair. In this case, repair amounts to warehousing the installable media images from which the user can initiate the repair process through the application install. This process works because the newer Windows install programs mark the system as the application is installed. On a subsequent running of the install program, maintenance mode is triggered in which case the user can elect to modify the current application status, repair the application by reinstalling the application files and configuring the system, or to simply remove the application.

The process has become fairly standard on the Windows operating system. A typical sample of the maintenance mode detection is shown in Figure 2.8.


InstallShield Professional - Windows Installer Edition - InstallShield Wizard
InstallShield
< Back
Next >
Cancel

Program Maintenance

Modify, repair, or remove the program.

( ) **Modify**

Change which program features are installed. This option displays the Custom Selection dialog in which you can change the way features are installed.

(●) **Repair**

Repair installation errors in the program. This option fixes missing or corrupt files, shortcuts, and registry entries.

( ) **Remove**

Remove InstallShield Professional - Windows Installer Edition from your computer.

Figure 2.8: Sample maintenance mode dialog box with Repair option selected.

License Conformance

Just because we've created an implementation process for our software, doesn't mean we can freely distribute it. After all, there may be a fixed number of licenses allotted and all are assigned. In this case, either software must be removed to make a license available or additional licenses must be purchased.

A key aspect to support in the software-deployment process is to ensure that the company maintains license conformance. As soon as all licenses are consumed, an alert should be triggered to investigate the software's status.

In some cases, license metering may be in affect. For this unusual situation, a fixed number of applications can be in use at a specific time. As soon as that limit is reached, any other instances of the software can't be run. Usually, the user will be asked if he or she wants to wait for a license to become available or to try again later. A job within the support team may be to automate the availability of these licenses.

The type of software license has a lot to do with how much flexibility is available. Some software is licensed per user. Thus, the software can be installed any number of times; however, only the assigned user can use the software. Other software is licensed per computer. This licensing scheme means that the software can be installed on only one computer—no matter who is actually using it. Another form of software license is a site-license. This license method is the most flexible arrangement and allows any number of installations as long as each user is within the same company.

Understanding the scope of the licensing arrangement is imperative to staying within conformance. It may also provide latitude in distributing the software.

A few years ago, I ran into a global company that circulated its licenses from time zone to time zone. When the employees in New York were done for the day, they would uninstall the software application. At the beginning of the day, those in the Belgium office would install the software—thereby consuming the licenses made available by the uninstalled software in New York. This process would continue around the globe, eventually leading back to the New York office. Keep in mind that this practice may be expressly forbidden by some software licenses.

System Recovery

Although system recovery isn't typically part of software deployment, it's worth mentioning here. Because the support mechanism already houses the distribution images, taking on the role of system recovery makes sense.

System recovery is slightly different than software repair because system recovery is the singular case of catastrophic software repair. In this case, the operating system as well as the computer applications has become inaccessible.

Ideally, system recovery is accomplished by storing a base image of the computer. Here again standardization becomes an important component. If there were a fixed (and hopefully small) number of standard systems, the system-recovery mechanism would have to warehouse a few images.

Based on the base computer image and the group that the user is a member of, all software (including the operating system) could be distributed through the standard software distribution channel. Excluding hardware failure, this process would quickly restore the computer to a working state. In conjunction with regular system backups, a minimal amount (if any) of user data would be lost.

Decommissioning Software

In much the same way that the software can be pushed onto a computer, it can also be removed. This removal is often done when upgrading existing software or simply removing software that is no longer useful. Software could also be removed because a specific licensing scheme has expired. In this case, instead of remotely pushing the software, a remote uninstall is performed.

As I previously mentioned, potentially, trouble can arise when removing software. Shared components or software that relies on the software being removed are common reasons not to remove software—or at least to do so very carefully.

Summary

Through this chapter, we've defined the fundamental basis of software deployment. With this information, you should feel comfortable starting a deployment effort. Perhaps you've even managed to put together a list of questions to present to the management and the deployment teams. Keep in mind that a thorough evaluation of the software and the distribution process goes a long way to ensuring an effective deployment strategy.

Chapter 3: Repackaging

In the first two chapters, we discussed the general background of software deployment, how to justify getting software, and methods of distributing the software. If you're reading this book with the thought that it will show you the one single method of getting your application packaged and delivered, this chapter will start to dispel those thoughts rather quickly. Very simply put, the "best" method of deployment depends on several factors:

- OS
- Size of the deployment
- Required timelines
- Your preference, skill level, and comfort zone with a deployment method
- The needs of the application to be deployed
- Security considerations

This chapter is going to focus on packaging (or repackaging) an application for deployment. In addition, I'll give you a list of best practices that includes testing your package and lightly touch on the deployment mechanisms you can use. Chapter 4 will cover the deployment methods in greater detail. For this chapter, the term *repackaging* refers to the process of getting a software application prepared for deploying to your environment. How to handle MSI, Wise Installer, and InstallShield-type applications will be explained in a later chapter.

Why delve into the details of packaging before you finalize the deployment methods? Although this process might seem a little backwards, if you are familiar with the methods and requirements of packaging software, deciding on the deployment method is easier. To determine which deployment product or method to use to get the job done, you need to understand how the new deployment mechanisms work with packages.

Are you new to software deployment? Reading this chapter might cause you to wonder whether anyone could do this task. Bear with me while we go through the details. Take a second and think back on how you became a Windows technical expert instead of just a Windows user. Most likely, your interest in the OS got you digging deeper than the average point-and-click user. By learning the details, you were able to fix things most folks couldn't, and you could troubleshoot problems because you understood what was happening under the GUI front end. You don't always think about the details and most likely don't use that knowledge when you do normal tasks. When was the last time you thought about all that was actually happening when you installed the OS on a new system? When a problem occurs, such as when the NIC fails to initialize, your knowledge of the details help you determine whether the *Could Not Connect* error message is because of a bad password, a hardware-level error, or a security restriction. The same holds true for software deployment. You need to understand what is going on behind the scenes to understand what the front end might (or might not) be doing.

Thoroughly confused and worried now? By the end of the chapter, you will find the whole process to be fairly logical and, with the newer deployment packages available in the market today, you might find the process enjoyable. OK, if not enjoyable, at least tolerable.

Repackaging Background

In the early days of software deployment (before 1990), you basically were a developer who focused on distributions. To ensure the application was deployed properly, you needed to know the details of every file and setting on the application, the interaction of the file(s) with the target OS and with other applications, and how to recover from any condition met along the installation path. As software applications increased in installation complexity, packaging an application for distribution took longer and longer. Current software-deployment software has made great strides in easing repackaging, and you don't have to be a software developer to perform the repackaging task. Of course, if you don't use one of the commercial deployment packages and opt for your own methods (spray scripts, clock-activated batch files, email, and so on), you will need to perform all the functions the commercial software does for you.

Do you need to repackage all applications? To some degree, you most likely will. The answer depends on how much adjustment the basic package needs so that it will fit in your environment. Remember what the core issue is: You are taking a piece of software that was (most likely) created to be installed on one PC at a time with the user of the system providing input along the way and distributing that piece of software out to your entire target population in a manner that supports your standards (hopefully, while providing a means for you to remotely administer the software).

No matter which method you use, there are basic steps that you need to follow. Your deployment method might skip through a step or two, but knowing the functions will help you understand the process better.

These are suggested best practices. If time, money, personnel, and resources were not an issue, these steps would prevent distribution-related problems (leaving only unforeseen application problems). Tailor the steps and processes to fit your environment and needs.

Repackaging Process

A note before we get started: Do you know what is in your environment? Having a detailed understanding of your environment is critical to ensuring that the applications are rolled out properly. I don't mean that you need to have an inventory of every single machine, but you do need to know the environment you are dealing with. The level of detail of your documentation is up to you. At a minimum, you should know:

- OS version(s)
- Service pack(s)
- Browser version(s)
- Major production applications already installed (for example, which version of Office is installed)
- Hardware (for example, which CPU, how much memory, hard drives)

Be sure to tailor your repackaging efforts toward what you have in your environment. Regularly updating your environment list (inventory) is a good idea. The update schedule depends on how often your environment changes.

The following steps provide an overview of the process that we're going to follow:

1. Ensure that your test environment is ready and will fit the needs of the rollout.
2. Talk to the rollout team (defined later in this chapter) to define their expectations.
3. Create a specification that clearly defines the project goals and seeks to resolve any outstanding issues.
4. Access a computer whose sole function is to create and verify the repackage process.
5. Create a repackage image.
6. Verify that the package meets the objectives of the rollout team.
7. Ensure that the final rollout specification documents the target environment, defines the boundary conditions, and provides assumptions about the rollout process.

Each of these steps is discussed in more detail in the sections that follow.

The Test Computer

No matter which deployment method or packaging process you decide to go with, the most important aspect of the repackaging process is your test computer. Let's spend a few minutes talking about what it is and what it's not.

A key contributor to your success is the test computer, which is simply a computer at your disposal for testing. Because of its special nature—it's one that you can actually damage with little downside—it shouldn't be used for any other purpose. The test computer is not a development platform, and it's not the computer you use for email or administrative work.

The test computer doesn't necessarily have to be fast, and it doesn't necessarily have to be new. It must, however, be within the hardware requirement constraints for the deploying package and be representative of your installed hardware base. The test computer is the fundamental verification step for the repackaging effort.

This test computer will go through several states during the repackaging verification process: clean, baseline, and working. Each of these states is fairly simple to understand. From the clean computer, we'll build the baseline computer state. From the baseline computer state, we'll build the working computer state. Each is a state progression in the repackaging process.

How many test computers do you need? Can you do the entire process properly with only one test machine? If you re-image and start from a clean machine each time, a single machine will work. However, the process would be easier if you have one machine that is used to create the deployment package and one (or more) systems to test the deployment process. That way, if you see issues with the package, you can go back to the system used for creating the package, make adjustments, then send the package back out to the other test machines. However if your budget and resources are limited, you may need to work with only one machine.

A *clean* test computer is one that has the core OS installed, all drivers are present, and any required general network settings have been made.

Starting with a *clean* test computer gives you a lowest-common denominator to work with and build from. Consider it the cornerstone platform of your testing, as it will give you the ultimate testing flexibility to start with.

A *baseline* test computer is built on a clean computer and has had service packs and any required patches applied and user accounts, group policies, and any other *existing* applications installed that you are likely to see in the end-user environment. Consider the baseline system a clean system with the updates applied to meet your environment needs.

A *baseline* test system is built on the *clean* system and should be representative of your environment. At this level, you will begin your testing. By applying different baselines, you can configure your clean test system to conform to various end-user needs. Although we all fight for standardization across the company, reality is another matter. The baseline computer from Accounting is not the same as the baseline computer for the Warehouse group even though they may be the same computer hardware.

A *working* test computer has the software application installed and functioning properly. It is the final and desired state of the testing process.

When you end up with a *working* computer after your test deployment, give yourself a pat on the back and enjoy lunch! Once the test environment is established, the next repackaging project is easier as you have the test environment set up already. The first time you create the test system(s) and the repeatable testing stages may require an investment of time that will see payoffs immediately. The best practice is to do initial tests against a system that you have no regrets about wiping out.

Creating a Clean State

In its clean state, the test computer is pristine and untouched by any extensions other than those provided by the OS install program. The clean machine in its initial state will be the starting point of every software-deployment effort.

The clean computer is the one with the oldest version of the targeted OS installed, and that OS is the only thing installed. If you're supporting multiple OSs, the clean computer will have to be reset for each OS. At this point, a reasonable question is "Why use the oldest version of an OS?" It's a very good question.

By oldest OS, I am talking about the oldest version for the level of OS that you will be deploying to. Don't develop or test against a Windows 95 system if your target systems are going to be solely Windows NT. If your target OS is Win98, and you have Win98 and Win98SE in your environment, you would target the Win98 version to ensure that your application will work for all versions. Be sure to confirm that the application works against both versions of Win98 (as it keeps the surprises down), but the initial target should be the older Win98.

The purpose of repackaging is to replicate the same final goal as the original install program—deliver and configure a working instance of the target software application. From our discussion in Chapter 1, we know that the software application has a hefty number of dependencies on the OS. By targeting the oldest version of the OS, we're forcing the install program to maximize its file transfer (by delivering more updated system files) and exercising the most post-install configuration.

A quick example illustrates this point. Imagine that we build our clean machine with the latest version of the OS and all the latest service packs. We then create our software repackage. The package we've just created will assume that every system it touches is going to have the same system files and settings as our clean machine. Because we can't guarantee such is the case, the package will undoubtedly fail because every system won't necessarily have the latest service packs installed. The purpose of the test computer is to repackage for the worse case scenario.

> ✎ Many of the newer snapshot products take into consideration potentially missing files and modify the install package accordingly. Thus, effectively ensuring you get all the files you need no matter what was on the initial packaging system. As a matter of best practices, a good habit is to use the older OS to test against.

If you haven't done so already, plan on gearing up your test computer. Make sure you format the disk and reinstall the entire OS from scratch. Be sure to use your company's standards for the configuration or use the typical installation of the OS. Spend an extra few minutes to make sure the system is completely functional and representative.

> ☞ If you haven't already done so, here is a chance to document a new workstation build process. Include in the document the steps, options, parameters, and settings used to create the image, and you will have a repeatable process documented for later use by you or others.

Once the OS is installed, the next step is to install the minimal set of drivers to get the computer in a useable state in your environment. Once all the necessary drivers are in place, the next step is to complete all network and Web browser settings. Be sure to update your Web browser to your environment's minimum level. Make sure that you reach any external locations that are representative of your environment.

I advise against making any custom settings at this point. The reason for this advice will be clearer soon, but for now don't make any mapped drives or set any other user preferences. Once the computer is in its clean state, reboot, and verify that everything is working correctly.

Creating a Good Image

Once the clean computer is set up and working, you've overcome a major hurdle. In fact, you might agree that this process isn't something you're too eager to repeat. And we don't have to. There is software that can capture the current state or image of the computer. Typically, this image is stored as either a duplicate image or a data file. This image can be recalled to reset the computer to the desired state. That's exactly what we need to do with our clean computer.

If we can store its state, we can get back to this clean computer state quickly, easily, and reliably. Depending on how many applications or environments we need to repackage, we might have to return to this state quite often.

There are several imaging software applications available. A little bit of research on the Internet should get you up to speed quickly on the various features and how the different imaging applications compare. Two common imaging software products are Symantec's Ghost and PowerQuest's Drive Image. Both are excellent packages and are well suited for our needs.

A good place to start your search for imaging software is http://appdeploy.com/tools/imaging.shtml. The AppDeploy Web site has collected the names of popular imaging software packages. A comparison of these packages can be seen at http://appdeploy.com/comparisons/imaging/index.shtml. Keep in mind that you should follow up with the software providers themselves for the latest features and changes.

Regardless of the imaging software you choose, the software performs the same task: storing the current state of the computer so that you can restore it later. Create a bootable CD-ROM with this data file, and you can quickly reset a computer back to its original binary status. Simply amazing and a must-have for installation testing!

Typically, the imaging application is launched, and you proceed through a series of menus to start saving the image. Figure 3.1 shows the menu sequence to begin saving the current disk image. The current image can be stored to a previous image (similar to drive cloning) or to a data file. The data file can be stored on a variety of media types. Usually, a network drive or CD-ROM is preferred.

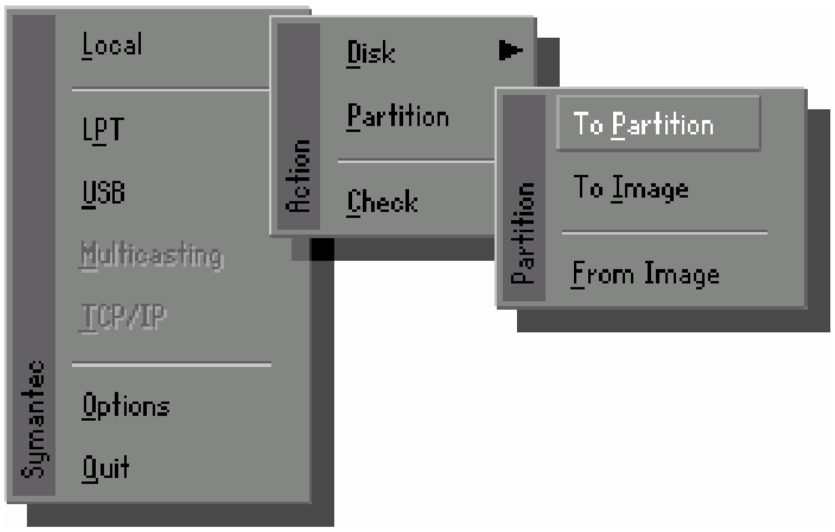


Figure 3.1: The initial menu sequence in Ghost.

When we're ready, we can use a boot disk to restore the disk image back to its clean state. Figure 3.2 shows the image selection from within Ghost. Note that multiple images have been collected and that the images use a .gho extension.

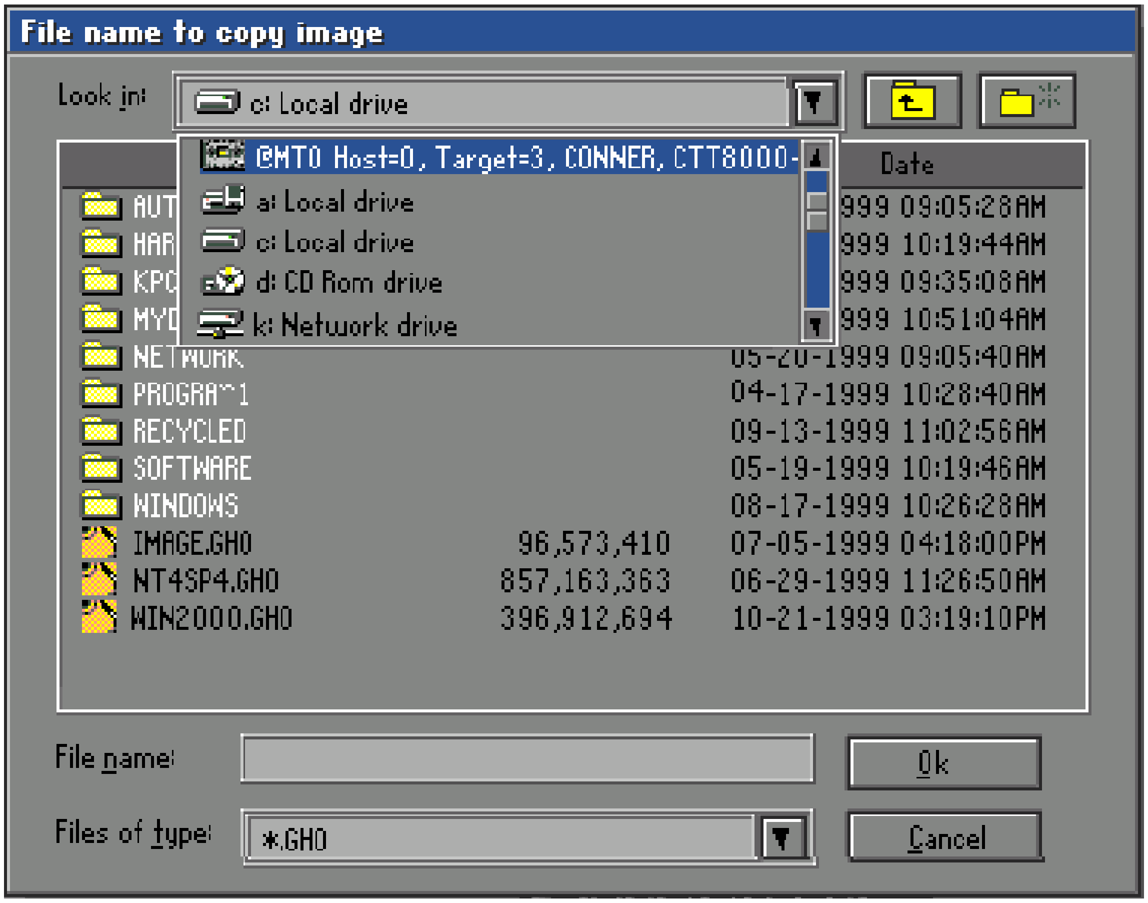

```
File name to copy image

Look in:  c: Local drive

@MT0 Host=0, Target=3, CONNER, CTT8000-
a: Local drive
c: Local drive
d: CD Rom drive
k: Network drive

                                              Date
AUT                                        999 09:05:28AM
HAR                                        999 10:19:44AM
KPC                                        999 09:35:08AM
MYD                                        999 10:51:04AM
NETWORK                               05-20-1999 09:05:40AM
PROGRA~1                              04-17-1999 10:28:40AM
RECYCLED                              09-13-1999 11:02:56AM
SOFTWARE                              05-19-1999 10:19:46AM
WINDOWS                               08-17-1999 10:26:28AM
IMAGE.GHO           96,573,410        07-05-1999 04:18:00PM
NT4SP4.GHO         857,163,363        06-29-1999 11:26:50AM
WIN2000.GHO        396,912,694        10-21-1999 03:19:10PM

File name:                                  Ok
Files of type:  *.GHO                       Cancel
```

Figure 3.2: The restore selection in Ghost.

No matter which imaging software you use, keep in mind that the image is unique to a specific hardware configuration—down to model numbers, versions of video cards, amount of memory, and so on. The odds are good that the image will not work on a radically different computer configuration.

One utility that I haven't yet mentioned is the Microsoft Sysprep utility. Sysprep takes into account some hardware variances and adjusts accordingly. Check out http://www.microsoft.com/WINDOWS2000/techinfo/reskit/en/Deploy/dgcb_ins_izyl.htm for the specifics.

In my office, I have a series of clean machines with ghosted CD-ROMs for each OS. You can typically reinstall an entire OS in less than 30 minutes. Images range in size from less than 60MB (for an NT client) to more than 250MB (for Win98). Your image sizes may vary significantly based on OS and selected options.

For my personal use, I opted to print out a complete file listing from the contents of the Windows and Windows System folders. Key data included with this file list includes version number, time and date stamps, and file size. Storing this information in a spreadsheet can be invaluable when you are tracking the origination of a specific system file.

Some imaging software can also be used to deploy applications or provide other advance deployment functions. Check with the software manufacture to obtain exact features and behaviors.

Creating a Baseline State

The baseline state of your computer gets its name from the fact that it represents basic system requirements for your environment and the software application. The baseline state is different from the clean system because what you've added may have different requirements from other software that you're packaging. By having the two states, we can always go back to the clean computer (through the imaging software), and build a different baseline for a different software application.

We get to the baseline by adding software most like the configuration of the targeted users. If you know the application has software prerequisites, make sure that they are also installed and functional. These prerequisites may include software such as Internet Explorer (IE), service packs, and so on. Also make sure that the system settings suit your needs.

The first place to start is to gather all the associated executables. Grab your environment's most up-to-date binaries. Although the temptation is always there to use the latest and greatest of all the executables, if they are not in general use in your environment, you are only asking for problems. Why test your application on NT 4.0 with Service Pack 6a (SP6a) when all your users are still on SP4? Of course, if the application will be deployed after your user base has upgraded to another service pack, test on the new service pack. Many large IT shops keep a depository of all their in-use executables to help ensure everyone stays at the same revision level of software.

For OS service packs and security patches, go to Microsoft's Web site and get copies. Catalog all the executables and document the installation order—this documentation is especially important for hotfixes and security patches. Some hotfixes and security patches depend on being installed in a specific sequence due to dependencies. Having the installation order documented will prevent those annoying dependency errors.

Once we've gathered all the executables in a single location, we have several options to consider. We could install all the executables onto our clean computer and re-image the disk. This process would give us a second image completely prepared to the baseline level. Clearly this option is convenient. The downside is that extracting a specific executable or replacing one executable with another may be difficult, depending on the tools available to you.

Alternatively, we could create a batch or script file to sequence the appropriate executables onto the machine. Another option would be to combine the executables into deployable packages. These options have the same downside as the first option. However, if you're certain that some executables will always go together, this option becomes a little more convenient, although it adds time to the imaging process. Finally, we could use snapshot software such as Prism from New Boundary Technologies (formerly Lanovation) to create snapshots and restore the baseline image as needed.

As you can see, the method needed to create the baselines on a recurring basis is up to you. Decide based on how fast you need to get back to your baseline, which modifications might be needed, and which option you feel the most comfortable with.

The next step is targeted toward the user settings. These should include joining the proper domain, network settings, user permissions, group policies, and so on. In a way, this step is very much like setting up the computer for a new employee. Be sure to include any mapped drives that are either available on the typical user computer or are convenient for repackaging. This process can be achieved by creating a series of scripts. These scripts should be well documented and easily located.

You may also want to make sure the system settings are useful—set the Windows Explorer shell to display file details and clear the *Hide files of known extensions* check box. Don't forget to map any drives you may need to access. You can also perform these steps through scripts. If so, they should be well documented and easily located. Remember that the settings will be enforced for the specific user ID that you are using (or setting them for).

However you decide to implement the automation of arriving at the baseline computer, make sure that the process is well documented and that the source executables are easy to locate. You should be able to return to the baseline as quickly and painlessly as possible. More than likely, the baseline definition will change regularly. You should plan on this change, and make it easily supported in the automation process. This step is where your documentation detail and choices of applying hotfixes, drivers, service packs, and updates will come into play.

If you are in a time or resource crunch, at the minimum, create a clean image and a fully loaded baseline image using disk imaging software (such as Ghost or Drive Image). This image will allow you to get to a testable state quickly and easily.

The combination of clean computer and baseline means that we also have instant repeatability, which is important if you're chasing odd behaviors and transient problems. If your package works on the baseline computer, and your user computers are very close to this configuration, you'll have most of the bases covered.

The final package will be a combination of a well-defined computer platform, the OS(s), system packages (service packs, hotfixes, security patches), well-defined partner applications, the distribution package, and possibly custom scripts for miscellaneous customizations. This process is intrinsically linked to each step in between. Aim for reproducibility at every step of the way by carefully documenting what was done and why it was done.

Ask Questions

Now that you have the test environment set up, you can start on the actual package to be deployed. Before you send out the next solve-all software package, you need to understand what is expected of you and the package. To that end, identify the *rollout team*. Although you may be the only person doing the actual work, you need to identify members of the rollout team. The members can be active participants or just contacts that you can send questions. Some suggested members include

- The developer or vendor
- Project sponsor
- A representative of the target end users
- Any technical staff required to reconcile company procedural issues
- A technical writer/documenter (if you are lucky enough to have someone else do the work)

In some cases, the team may consist of only you and maybe one other person. Or the team may become an entity by itself. The size of the team depends on the culture of your company and the project needs. The point is that you need to have the contacts in the right departments to ensure that the distribution is done properly based on the negotiated expectations of the project owners. Notice the word "negotiated" in the previous sentence. Part of your job is to set the appropriate expectations.

Believe it or not, initially, every member of the rollout team will have different expectations about the repackaging process and what the final package can and will do. They may also have varying views about how difficult the repackaging process will be. Some will picture the new package as doing lots of neat stuff. Others may see it as a necessary evil. Your first task is to capture each team member's expectation and generate a common view. Then you need to share that view with the team in a manner that balances schedule with deliverables and functionality with realism. This process is the team interview.

The team interview is best accomplished by starting with a brief questionnaire. The questionnaire isn't meant to merely gather information; it also alerts the remainder of the team to important issues early in the development cycle. Too often the repackaging process isn't given the consideration that it should have early enough. The questionnaire obtains preliminary information from the project team and, by virtue of the questions raised, informs the rest of the team.

Common questions contained within the install questionnaire range from the simple (who are the decision makers) to the complex (additions, subtractions, and package modifications). In some cases, the questions are to officially position the package-creation process. In others, the questions are relevant to how the package works. The following questions and descriptions, which Table 3.1 lists, are a standard set of topics that you may want to consider using.

Question	**Information Application**
Who's on the rollout team?	Identifying the names and levels of accountability is an important first step.
When is the package required?	If you're working against a tight schedule, some customizations might be dropped for the initial rollout.
What are the objectives?	Is the objective to simply remove the UI? Is it to restrict user selections? The objectives should clearly state the end users' expectations.
What should be removed?	Are there pieces of the software application that should not be included in the final package?
What should be modified?	Are there modifications that need to be made? This answer may cover topics such as customizing Start menu entries or adding a link on the desktop.
What should be added?	Is there additional software that should be added to the final package? Is this package going to bundle several distinct software applications into a single package?
Are there any target computer hardware requirements?	Is the application a simple batch file or is the process a complex installation that requires a minimum hardware level to be dealt with? If it is complex, the standard computer platform should be clearly specified. This specification should include

	processor types, available system memory, name and model of video cards, and so on as needed.
What are the target computer OS requirements?	Typically, a package isn't deployed to just any OS. Each OS can have wildly varying modifications—from maintenance packs, hotfixes, and core components (MDAC, Visual Basic runtimes, and so on) to security patches. These should be included within the rollout specification.
What are the coexisting applications?	Unless you're deploying a single application to a brand new installation of the OS, you'll need to know exactly what software should be expected to work with your package. The list should include the release version of the software as well as any updates.
What is the deployment method?	Because you're packaging the application, you might have to take into account how the final package will be deployed to its final destinations. Some deployment software suites have additional requirements. Additionally, some packages can be further optimized.

Table 3.1: Questions for your rollout questionnaire.

To see how the questionnaire will benefit you, picture the following scenario: You have the envious job of rolling out a complex office productivity package (such as Microsoft Office XP). The goal of the distribution is to send it to the entire company, and because you use the product all the time, you know what to customize and how to make it do things properly. Why bother with questioning the rollout team?

You want to be thorough, so you get the rollout team together and discuss the distribution. In the course of this meeting, you learn that

- The folks in Garage Maintenance don't want the Presentation application.
- The Accounting group wants additional templates added to the Spreadsheet portions.
- Half the Sales folks are still running on laptops with only 64MB of RAM.
- Every department has different storage locations for saved files, and they want their default locations to appear automatically.
- The Technical Lab needs a special dictionary.

Suddenly the rollout isn't so simple. To provide the functionality required for each department, you need to devise the rollout process. Thus the need for repackaging the application has been determined by the questioning process.

The questioning process can take only a few minutes or it might take weeks, depending on the size and culture of your environment. Don't skip this part of the process. Always get the basics answered or you will find yourself doing the job again after the first deployment. Keep track of your questions, and develop your own question chart for future use.

Document

Once you've got the results of the rollout questionnaire, you're ready to compile the first pass of the rollout specification. You'll be updating this document throughout the course of the package-creation and deployment process, so you'll benefit from being as thorough and descriptive as possible. For the first release, you should turn the results of the questionnaire into prose. Simply write down the view of the package goals as described by the team members. Be sure to resolve any outstanding conflicts or at least make a note within the specification to point out these contradictions. You might even want to create an outstanding issues list to keep track of any issues needing resolution.

Your specification should contain a conversational discussion about the package. Describe what the package does and how the package does it. You should also provide a list of files (with versioning information), registry edits, and file changes. After all is done, not only will this specification be a handy reference for future changes but also an invaluable tool for training the folks in phone support. The beauty of most recent software packaging applications is their ability to provide the file and registry detail information to you in an automated format, saving you hours (or sometimes days).

The level of documentation will depend on your working environment. Documentation is a step that is normally avoided or skipped if you can get away with it. Be careful, don't skip it completely. If you are the sole deployment design engineer, distribution tech, application support, Help desk worker, and chief bottle washer, your documentation may be a few paragraphs saved to a central location. If your organization hands off each step to other people, the documentation becomes more important. In either case, get your ducks (and data) all in a row before you get rolling.

Preparation

Historically, taking the time to prepare to meet your goals is time well spent. Preparation ensures that you've thought of all the possible scenarios, you've bounced ideas off other people, and in general, had a chance for the right solution to stand on its own. If you've reached that level of clarity, chances are you're ready to go. If you haven't quite arrived, this section should help you. We'll look at the different angles of tackling this distribution model.

Research the Existing Application Media

Once the document has a consensus of approval, look at the installation image created by the software company (the source CD-ROM, for example). Document what the image contains, the format and type of the existing install package, and the contents of each subfolder. Try to assign a reasonable purpose to each file and folder. In many ways, the key to repackaging starts by fully understanding what you're starting with.

A good reason for this extra work is that many common install packages ship with an internal mechanism for reducing the UI. Some go so far as to ship with full support for reduced or unattended installations. Install programs that can be malleable are typically referred to as original equipment manufacturer (OEM) packages. Checking with the original software manufacturer could save you lots of work, and your company could save a lot of time and resources by simply using the already-provided OEM interface.

There is a long-time standard set of install-development providers. The primary among these are InstallShield and Wise Solutions. Although both companies produce proprietary schemes for installation packages, they also provide development environments for Microsoft Windows Installer.

If the install package is a single file, it's probably a self-extracting, self-starting package. Launching the file causes the internal image to be extracted—often to the system temporary folder. Once the install program gets to the Welcome dialog box, navigate to your temporary folder, and look around. Cleaning out the temporary folder first may help.

Getting familiar with the basic application image now will help you out later during the testing and troubleshooting phase. Remember Murphy's Law: If something can go wrong, it will, and at the worst possible time. No repackage effort will run perfectly the first time, and if the errors reference a file name, it would be best to understand if that file is part of the basic application, a system file, or a file that was replaced by the application install.

For example, if the target modifications to the software application are the addition or removal of files, and the application is using InstallShield for the native install, using InstallShield's tools may be a quick and easy solution. If you've determined that the media image includes an InstallShield package, you stand a reasonable chance of having the package deploy silently.

InstallShield is not the only media packager to allow silent installs. An option to perform silent installs is quickly becoming a de facto standard in the software industry. Most of Microsoft's major applications provide some method of silent install, as do most applications installed using Wise Installer.

As with any repackaged distribution package, the unattended installation should be tested on a number of different computer configurations. An unattended install that works on one system may not work on another. A major weakness of InstallShield's unattended install is that the dialog box sequence must occur identically on all computers. Something as normal as a reboot request on one system causes the unattended install to abort when running on a computer that doesn't prompt for a reboot.

In the next chapter, we'll go through these deployment tools in much more detail. Until then, if you're interested in more information, InstallShield, Microsoft, and Wise Solutions all provide additional content on their Web sites. As most Web site content changes regularly, your best bet is to aim your browser at each vendor's home page and follow the relevant links. InstallShield's Web site is located at http://www.InstallShield.com. Wise Solution's Web site is located at http://www.WiseSolutions.com. Technical information about Windows Installer can be found at http://msdn.Microsoft.com (search for Windows Installer).

Script Solutions

In some cases, you might find that while the unattended installation works, there are still some smaller tasks that need to be accomplished. Maybe some added customizations need to be done, a notification flag needs to be set on a remote server, or possibly added security needs to be set. Creating a script file may be just the right addition. Scripts can be as simple as a DOS batch file—although some DOS batch files are by no means simple—or as complex as a full-fledged VBScript or JavaScript.

Alternative scripting languages are available. Most of these are suit-to-taste options. Each scripting language has its own strengths and weaknesses. When it comes to selecting a script language, pick one that will efficiently do the job and still be maintainable, extensible, and—more than anything else—easily reusable.

A key factor in deciding on a scripting language is the availability of support and freeware scripts. For popular scripting languages such as VBScript, many common distribution solutions already exist. The following is a list of the most common scripting languages: Perl, JavaScript, WinBatch, KiXtart, Python, VBScript, DOS, ScriptIt, Tcl/Tk, and Rexx. Keep in mind that many of these scripting languages require a runtime environment that must pre-exist on the user's computer. Solutions such as VBScript and JavaScript that are built into Microsoft's Windows Script Host (WSH) are popular solutions because the runtime environment already exists on a substantial number of computers.

☞ Additional information about scripting languages is available on the Internet. A simple search for scripting or the name of the scripting language in which you're interested will turn up many useful sites. There are a number of sites that cater to scripting. I suggest you try some of my favorites to get started (these are in no special order):

http://www.desktopengineer.com

http://www.appdeploy.com

http://www.winscriptingsolutions.com

http://www.microsoft.com/scripting (redirects to MSDN location)

http://cwashington.netreach.net

Creating a Working State

As you can see, just to get to the point at which you can start the "real" work of creating the new package takes a lot of effort. That's why I pointed out earlier that you should really understand the goals of the repackage process. Being able to definitively state why a repackage is necessary is also important. By the time you get to this point, both should be perfectly clear.

We've got the test computer in its baseline state. The next step is watching the native install program and capturing all of its actions. That is, we need software that records what the install program installs in a way that is useful to create the final distributable install package. That is a carefully worded sentence. There are several implications. First, some of what the snapshot software observes as newly installed may not be relevant to the application-install process. Second, some of what the install program updates or installs may not be relevant across all OSs or computer environments. Finally, other events that aren't performed by the install program may need to be added to our repackage.

Before there were system snapshot tools, much of the system detection was done by comparing the differences between files. For example, if I wanted to observe what happened on a specific Windows computer, I would perform a DOS dir command on the root disk. The results of this command would be placed in a simple text file. This listing is a very small part of the complete file and folder dump.

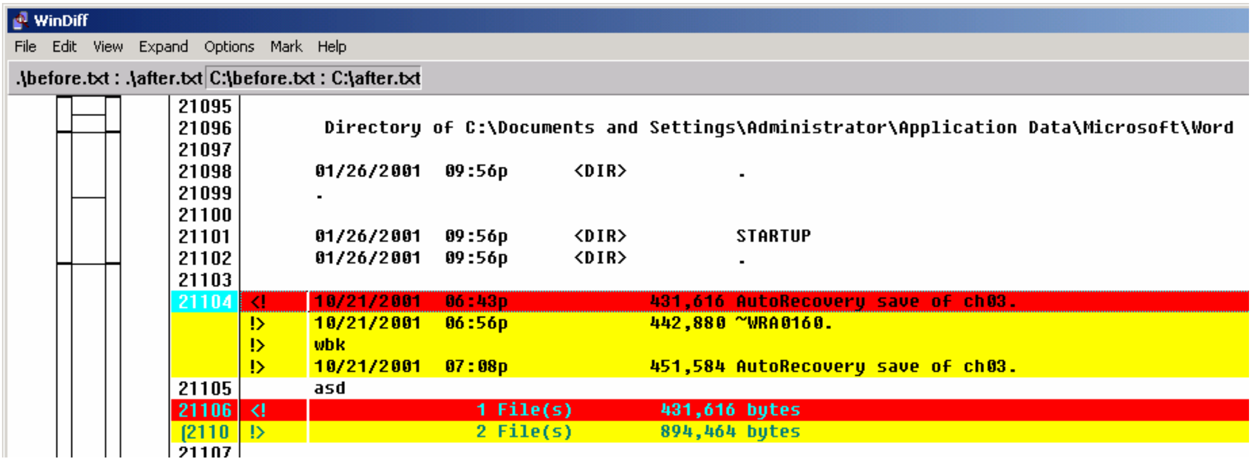
By using regedit (part of the Windows-included utilities), I could do the same thing in the system registry by selecting the root of the registry and selecting Export Registry File from the Registry menu. This action prompts me for a filename and a location to store the exported registry settings. By the way, this file format is the same file format that can be double-clicked to merge the contents back into the registry.

Double clicking the .REG file will cause the system to import the file into your registry. In the best scenario, this action imports only data that you already have. However if the .REG file is from another computer, you could see disastrous effects. Even to the point of wiping out your system registry settings and making your system think it is another computer entirely (not that I've ever done that, you understand). Disconnect the .REG association or rename the exported file to a .txt extension and save yourself potential pain.

Realistically, I've covered a large percentage of the snapshot territory with these two actions. If I now repeat the two actions after installing the application, I'll have four files: before and after snapshots of the file and folder structure along with before and after snapshots of the system registry.

Using any kind of utility that can compare two files, I can view the changes to my system between the previous events. And that's exactly what I've done. As Figure 3.3 shows, I used the WinDiff tool that ships with Microsoft Developer Studio (it is also available in the NT resource kit). The changes are highlighted so that they are easy to spot, which is especially important when comparing the differences between large files.

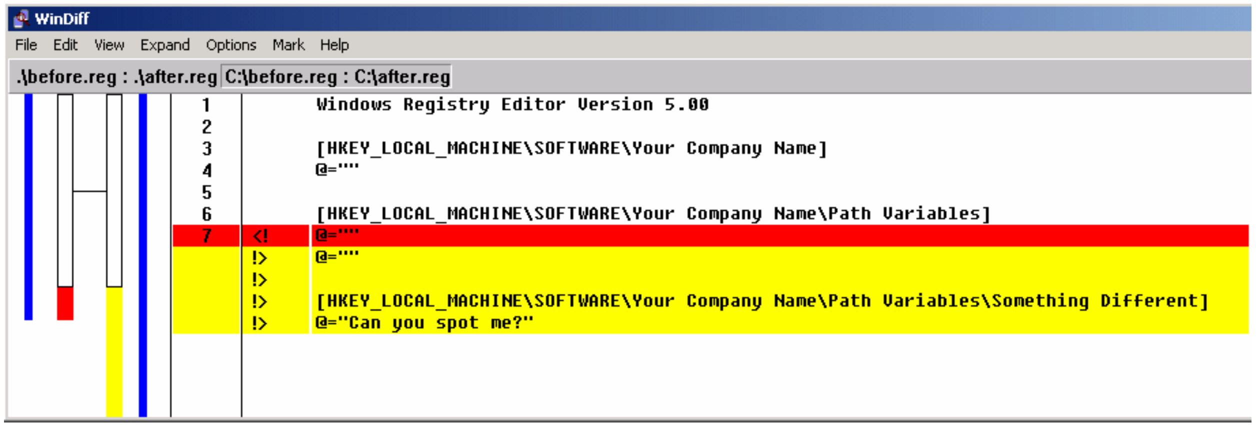
The difference in text files revealed that I had saved my document and that Word's AutoSave feature updated the cached image of my document. There were some adjustments to temporary files and the snapshot files themselves showed up.


WinDiff
File Edit View Expand Options Mark Help
.\before.txt : .\after.txt C:\before.txt : C:\after.txt

```
21095
21096        Directory of C:\Documents and Settings\Administrator\Application Data\Microsoft\Word
21097
21098       01/26/2001  09:56p      <DIR>          .
21099       .
21100
21101       01/26/2001  09:56p      <DIR>          STARTUP
21102       01/26/2001  09:56p      <DIR>          .
21103
21104  <!   10/21/2001  06:43p             431,616 AutoRecovery save of ch03.
       !>   10/21/2001  06:56p             442,880 ~WRA0160.
       !>   wbk
       !>   10/21/2001  07:08p             451,584 AutoRecovery save of ch03.
21105       asd
21106  <!                  1 File(s)        431,616 bytes
(2110  !>                  2 File(s)        894,464 bytes
21107
```

Figure 3.3: The snapshot process made simple—the results of the file comparison.

For the registry, I intentionally added a registry key, as Figure 3.4 shows. The OS seems to always be up to something unless you want to catch it doing something. Be aware that WinDiff works with ASCI files. Attempting to compare Unicode files will continually generate a *Different in blanks only* message.


WinDiff
File Edit View Expand Options Mark Help
.\before.reg : .\after.reg C:\before.reg : C:\after.reg

```
1        Windows Registry Editor Version 5.00
2
3        [HKEY_LOCAL_MACHINE\SOFTWARE\Your Company Name]
4        @=""
5
6        [HKEY_LOCAL_MACHINE\SOFTWARE\Your Company Name\Path Variables]
7   <!   @=""
    !>   @=""
    !>
    !>   [HKEY_LOCAL_MACHINE\SOFTWARE\Your Company Name\Path Variables\Something Different]
    !>   @="Can you spot me?"
```

Figure 3.4: The snapshot process made simple—the results of the registry comparison.

This exercise has been rather light, but I wanted to demonstrate that there are very easy ways that you can detect changes to your computer. The example process was elemental but important. It was also time consuming. You weren't here, but waiting for WinDiff to capture the differences between two 34MB files was quite the yawner. Fortunately, there are several tools available to not only automate this process but also to generate a redeployable package.

Snapshot Software Options

Before we get started on the nuts and bolts of repackaging, let's take a look at the different types of snapshot software. There are two basic groups:

- Strictly snapshot tools
- Snapshot tools that create an install project

Strictly Snapshot Tools

This first category of snapshot tools simply observes and reports the changes that occurred on the system since the last time the tool was run. This functionality is very much on the same order as what we did earlier. Of course, anytime software can compare and present the differences for us, so much the better.

Two tools that fall into this category are both commonly available and free! One is a tool called InCtrl5. It can be run in two-phase mode, which means you launch it to take the first snapshot, then launch it again after changes have been made to the computer for the final snapshot and comparison. Or you can run it in standard mode. In standard mode, you specify the install program to launch and monitor.

InCtrl5 has been around for several years. This is a *PC Magazine* program written by Neil J. Rubenking. As of this writing, you'll find a downloadable version at http://www.zdnet.com/downloads/.

Another strictly snapshot tool is part of a larger software package. It's called VeriTest-Rational Install Analyzer, provided by Rational Software and VeriTest. The Install Analyzer is a small part of a larger tool that tests software applications in a distributed setting. What's especially encouraging about the Install Analyzer is that VeriTest uses it to qualify Windows applications for the Microsoft product logo.

Install Analyzer is an excellent snapshot utility. Unfortunately, it runs on only Win2K. It's part (actually a very small part) of Rational Software's TestFoundation for Win2K. As of this writing, you'll find a downloadable version at http://www.rational.com/products/testfoundation/index.jsp.

Whereas InCtrl5 limited a snapshot to the immediate before and after images, Install Analyzer allows you to compare any number of snapshots. For example, you could create a snapshot from the base OS, followed by a second snapshot of the installation of IE, followed by a third snapshot of the installation of RealNetworks' RealPlayer. With these snapshots you could compare the first and the third, the second and the third, and so on. The ease and flexibility in comparisons can be incredibly useful.

The tool is very easy to use and generates very readable output. The output is slanted toward Microsoft logo compliance, and I think that is a good thing. For example, it will identify installed files that don't have registered file extensions.

The Install Analyzer watches seven critical areas of software configuration and provides comments on any violations. Most of these critical areas are Microsoft-recommended best practices and can be considered violations of logo compliance guidelines:

1. Install to Program Files by default, do not attempt to install files protected by System File Protection on Win2K, and ensure correct uninstall support.
2. Provide 32-bit components.
3. Do not read from or write to Win.ini, System.ini, Autoexec.bat, or Config.sys on any Windows OS based on NT technology.
4. Ensure non-hidden files have associated file types, and all file types have associated icons, descriptions, and actions.
5. Classify and store application data correctly.
6. Do not place shortcuts to documents, Help, or uninstall in the Start menu.
7. Kernel-mode drivers must pass verification testing on Win2K.

Although these tools are useful, they don't take us all the way to a deployable package and have their limitations. Let's take a look at some tools that do take us to a deployable-ready package.

Snapshot-to-Project Tools

These tools follow the snapshot process with the creation of a deployment project. A project is an editable version of the final package. Normally, the project is opened within the repackaging software. This project can be customized and altered before you create the final distribution package. Typically, the customization will deal with file placement, file removal, path variables, and so on. As of this writing, the heavy hitters in this department are

- Prism Deploy by New Boundary Technologies
- AdminStudio by InstallShield
- Wise Package Studio by Wise Solutions (creates a Wise Scripting project)
- Wise for Windows Installer by Wise Solutions (creates a Wise Windows Installer project)

The questions to ask yourself as you're going over the feature lists are the basics: What is the final format of the deployable package? Does it have the option to create a Windows Installer database file? How can I deploy the package? Is additional software required on the end user's computer to support package deployment? Does the installation of the snapshot software have a small and unobtrusive footprint? How much control is allowed over the package creation? Can the package be customized once it is created? Can the package be uninstalled in an easy manner if necessary?

These tools provide the ultimate in flexibility. We'll take a look at these tools as well as several others in the next chapter.

☞ A rather lengthy list of distribution packages exists on the AppDeploy Web site. For more information about a complete list of tools refer to http://appdeploy.com/tools/distribution.shtml. Make sure to check with the manufacturers for the latest features and availability.

Snapshot

After reviewing the tools and selecting the one appropriate for our environment, we're almost ready to go. We have the clean computer set up and a baseline has been installed. The next step is to take a snapshot of the installation of the targeted software application and review the package results. This process is actually fairly simple, as Figure 3.5 illustrates. The hard part is making sense of the final package to ensure it's complete and doesn't contain any spurious system changes.

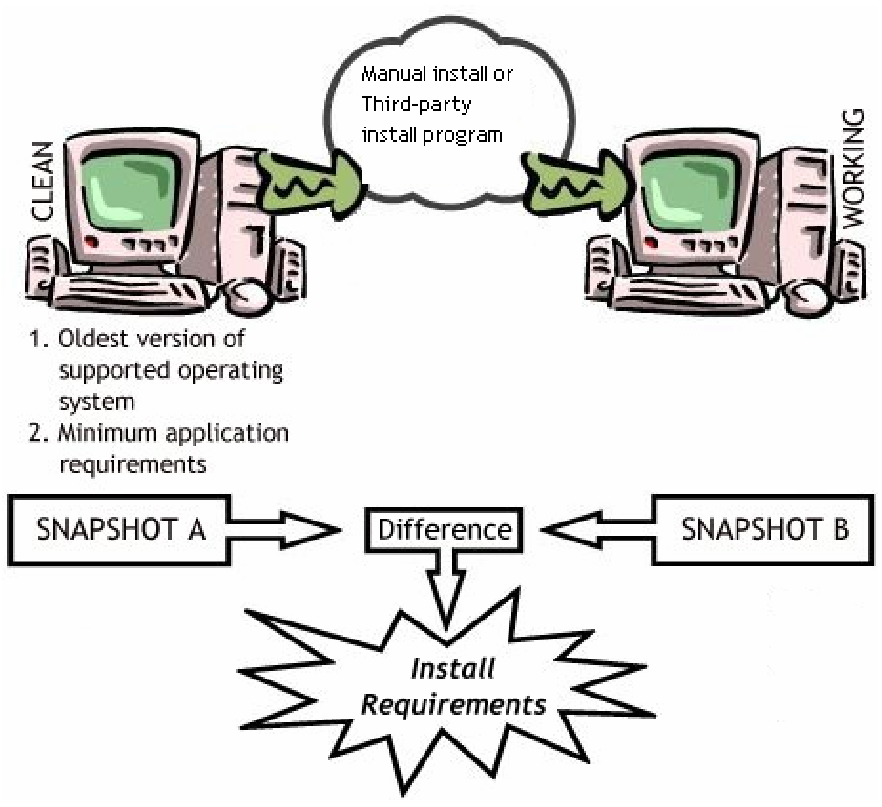

Manual install or Third-party install program
CLEAN
WORKING
1. Oldest version of supported operating system
2. Minimum application requirements
SNAPSHOT A
Difference
SNAPSHOT B
Install Requirements

Figure 3.5: The snapshot process is a before-and-after record of the original install.

The first step is to install the snapshot software onto the test computer. The one thing you may want to make sure of at this point is that the snapshot software doesn't contaminate the system. Updating any system file or setting should be considered poor form in the snapshot business. After all, the objective is to capture the contents of the original install package without changing the environment itself. The better snapshot software will remove references to itself in the final report and prevent that contamination issue from being replicated.

Once the snapshot software is installed, close any unnecessary applications and shut down any unnecessary processes and NT services. Depending on the function of the target system, standard NT services that may be safely shut down are World Wide Web Publishing services, IIS Admin service, and FTP Publisher. Getting the system to a truly inactive state reduces the inclusions of unrelated system modifications into the snapshot package.

When you're ready, fire up the snapshot software and install the targeted software application. Once you're done, you need to first make sure that the application is working, then evaluate the newly created package contents. Don't be too surprised if you need to tweak the computer a bit to get the application to work. Make sure these post snapshot changes are reflected in the package itself. The changes should be picked up in the second snapshot.

Don't fall into the assumption that the native install program is going to do everything correctly or even do everything that needs to be done to get the application to work. You may be pleasantly surprised to find that sometimes your repackage is better than the original install program.

Don't proceed to the final snapshot until the application is working as expected. If the application doesn't work yet (don't laugh, it happens), don't finish the snapshot process. You need a fully working application for this step to be worthwhile. Ask around for help. It all must work because that's what you're going to repackage. If the application is broken in the snapshot, it will more than likely be installed that way. Expect to spend some time on this step. Be sure to document as much of the process as possible. You may have to do it again.

Once the application is working, the last step is to take the final snapshot. The difference between the first and last snapshot is what your package will replicate. The tricky bit is figuring out how to do it. Most OSs have a lot of stuff going on behind the scenes. Your snapshot will invariably capture some of it. You should have the system as quiet as possible. Also, keep in mind that the more the system was mucked with to get the application working, the more noise you'll have in the snapshot. You'll need to spot the difference.

As we'll see, sifting through the snapshot data is an important part of the design process. Including the wrong files or not including the right ones is only part of the concern. Making sure the configuration process is correctly performed is the other part.

Package Evaluation

Okay, now we've taken a snapshot of the system to see how the application is installed. What do we do with all of this information? We need to verify that the results should be part of our package. Keep in mind that although these snapshot tools are many times better than they used to be, they are only tracking system changes and incorporating them into a deployable package.

There is much black magic in this process. Rarely will you get a snapshot that is noise free. Let's spend a few minutes talking about some of the stuff that you'll see and what it means. Or in some cases, what it doesn't mean.

First, the golden rule of viewing snapshot results is "Just because it shows up in the snapshot doesn't mean you created it." This rule is closely followed by the silver rule of viewing snapshot results: "Just because it shows up in the snapshot doesn't mean it needs to be there." These rules will go a long way to helping you sort out the snapshot contents. Let me explain.

The snapshot gives you the end result of changes to the system. It doesn't tell you how the changes were made or who made them. You'll need to do some research to figure out the differences. Let me give you a simple example—self-registration. In the snapshot output, you might see a ton of registry entries that look something like the following example:

```
[HKEY_CLASSES_ROOT\
      CLSID\
      {EAB841A0-9550-11CF-8C16-00805F1408F3}\
      InprocServer32]
Value Name: (Default)
Value Data: C:\\WINNT\\System32\\thumbvw.dll
Value Name: ThreadingModel
Value Data: Apartment
```

You're looking at the end results of a DllRegisterServer call from a self-registering file. You do not create these registry entries; you simply need to make sure that the appropriate file is marked as a self-registering file when you link it to your install project. Most good snapshot tools will capture and identify these files for you. When you see a bunch of registry keys like the ones in the example, simply look for the value data associated with the InprocServer32 key. This value data will give you the name of the self-registering file.

Unfortunately there are more examples of snapshot data that are created as a by-product of some other event. You may have to do some homework to figure out the differences. Table 3.2 shows some other examples that are not created manually but through a transient process.

Example	**Description**
Temporary files	There is a system path referred to as a Temp folder. Many applications create files as temporary storage within this folder. You can normally exclude any files that are created in this area.
Type libraries	Very similar to the CLSID key example except that type libraries will have a TYPELIB string in the registry key. Registering a type library file creates these entries.
ODBC drivers and Data Source Names (DSNs)	These are registry entries that are created through a SQL Windows API call—namely to instantiate the driver if it doesn't exist and to create or modify the DSN, including a reference count on the drivers themselves.
NT services	These also show up in the registry, but are created or started through the Service Control Manager (SCM) Windows API calls.
Fonts	Fonts are copied to the Fonts folder (a well-known path), then added to the system through an AddFontResource Windows API call.

Table 3.2: Examples of snapshot data created by a by-product event.

There are plenty of others that behave the same way. Carefully sift through the registry snapshot data looking for more. The snapshot software you are using may take into consideration most of the transient issues for you, a careful review will show if any slipped past..

Next are file edits. These can also be by-products of other events, particularly if the file resides within the Windows or Windows System folder. There are a number of Control Panel applets that still rely on the settings of system initialization files for their settings. Depending on the snapshot tool that you're using, you may not capture the file change or what was changed. You may have to run several snapshots to detect content changes within files. A good example is an internal file modification of INI or text files. Unless you've explicitly identified a specific file to be monitored, the original snapshot would miss the internal change. All that you would know is that the file was changed. On a second snapshot, the file can be explicitly monitored and the specific change could be captured. Knowing that an INI or text file has been updated with, say the local time zone variables, will help in troubleshooting and configurations for different regions.

Finally, there are files that are added to the computer or simply updated. You'll first need to distinguish between files that are created versus files that are installed. Clearly, if it's a binary file, the odds are good that the file was installed. Text files and initialization files may be created based on configuration data detected on the computer. While this possibility may be rare, it's worth keeping in mind. One example would be the ATL.DLL. There are two versions of this file: an ANSI version for Win9x and a Unicode version for NT and Win2K. You'll need to make sure that you get the right one. There may be other files that fall into this category. Thus we have a need for a snapshot on each target OS. Be aware that some applications place numerous files on a system. For example, tracking the entire file list for an Office 2000 install will be difficult. The snapshot software you use will assist this effort by providing the list of files installed.

If you're not sure whether your application needs a specific file or uses it, don't include it. Make a note of your decision by updating the specification. The reason is simple. The install will go through a testing phase. During this testing cycle, the install program will hit a large number of system variations. If one of those test scenarios uncovers the need for the file, then include it. Detecting a missing file is easier than detecting an unnecessary file. Sometimes shipping an unnecessary file can cause problems on the end user's machine. You'll have enough problems; you don't need to create any more. However, snapshot tools such as Prism Deploy can automatically handle changes to INI and other special text files, so if you're using these tools, you don't need to know whether a file was created or updated. Some tools go so far as to merge the changes to, for example, INI files on target systems. Let's go through a list of the more common gotchas.

UI

One of the common problems in creating a deployable package from snapshot software is that the information prompted for on the install dialog boxes is now hard-coded into the package. A good snapshot tool will provide a way around this problem. Many vendors provide a mechanism for replacing paths and hard-coded values with runtime equivalents.

For example, software applications commonly ask for a serial number. Unless you've obtained a generic one-size-fits-all serial number, you'll be hard-pressed to get unique serial numbers into a single package. The same is true for passwords, user information, and destination paths.

Another interesting point is the license agreement dialog box. There are legal ramifications to not showing this agreement in the deployment process. Has the legal department signed the documents that confirm this process?

Capturing each dialog box in a screenshot and adding it to the rollout document is a very good idea. Every user-provided piece of information should be accounted for within the deployed package.

To create a completely silent install process (one in which the user is not required to provide any interaction), solving the serial number issue is one example of where added scripts or runtime variables could be used.

System Files

Almost every install program bundles at least a few system files. Although the snapshot tool will capture upgraded files, it may not automatically know which version replaced the file. Here again, you get what you pay for. Good snapshot utilities will allow you to modify the deployment package to account for overwrite-by-version file replacements.

With good snapshot utilities, you don't need to track down the DLL versions as the software will provide some variation of that ability for you. Knowing how to do the research is important. Remember Murphy's Law? Inevitably, you will be doing an emergency distribution at 3 A.M. and have a problem with one file that will need to be researched.

Additionally, some install programs don't follow the Microsoft guidelines for redistributing files. Here's your chance to go one better. Identify each executable, DLL, and ActiveX control (.ocx) file by its manufacturer. Whenever possible, track the file to its associated redistribution agreement. From this agreement, you'll know how the file was meant to be distributed. There are a significant number of Microsoft files, for example, that cannot be redistributed in the raw. They are part of a bigger package that should be installed as a layer—or all at once.

For Microsoft files, there is another method of tracking down where the system files come from—the Microsoft DLL Help database. This database is a searchable repository for all Microsoft's system files. It's a very handy way to track down the versioning history of a specific Microsoft file. As of this writing, you can find the database at http://support.microsoft.com/servicedesks/fileversion/dllinfo.asp?fr=0&sd=msdn. If this link doesn't work, go to http://msdn.Microsoft.com and search for DLL Help.

Although this part of the process sounds like a lot of work (and it is), remember one goal of this chapter is to show the best practices in an unlimited-resource world. Reality will make the decision for you as to how deep you will need to go to check files.

Internal Logic

This gotcha is the big one. The snapshot software has no way of knowing why the install program made the changes that it did. The snapshot software can't hear the install program thinking; the software only sees what the install program does. As a result, snapshot software won't see the evaluation for disk space or the check for system prerequisites such as SNMP and file and printer sharing. Other things that trigger different actions are a pre-existing version of the same application, competitive upgrades, or the presence of specific hardware.

In these cases, you'll need to be very careful about how you deploy the package. Remember that the package assumes every computer it encounters will look very similar to the test computer. If the repackaging of the install program prevents those checks from running, using a pre-install script or, if the deployment package allows it, some form of variable checking can be done to ensure that the package install logic is followed.

Does that mean that if your target machines are slightly different from your test machine you won't be able to run the package? Not really. If your test environment is close to your target deployment machines, good third-party tools will be able to handle heterogeneous environments. Prism Deploy and Microsoft's Sysprep are two examples of tools that will make allowances for system differences. The point to remember is to keep test-system and package-creation processes as close to your target environment as possible so that the original package install logic will install all the proper components.

Conflict Resolution

Often you'll have two packages that require different versions of the same file. In some cases the file conflict may be severe—each application needs its own specific version of the same file. Other times, a little bit of testing is all you need to determine which version of a file needs to be shipped. A common misconception is that you should always ship the latest version. Resist this temptation; I'll explain why.

The goal of an install program is to place the application on the OS with minimal problems. Thus, we don't want to change anything that already exists if we don't have to. Lots of applications rely on core components. The odds are good that a number of machines that your install will encounter will already have some version of the conflicting core component. They just may not have the latest version. But hang on a second. If your application is completely functional with the older version, should you even ship the latest? Especially when you consider that the latest version may break pre-existing applications that are also getting along fine with the older version. The answer is a resounding No! You should ship the version of the conflicting core component that your application will work with that is already out in your production world. If the core component already exists on the target system, you're golden. It's already there, so the install program doesn't install it. And if it's not installed, the odds of breaking anything are reduced.

So once you know a conflicting core component must be shipped, spend some time tracking down the exact version required. You'll save yourself and the support team some headaches. You'll save your company some money. Several companies are looking into the conflict-resolution issue and are working to solve it for us. We'll cover them in more detail in a later chapter.

Getting a Profile

Did you know that there is an indispensable tool to determine all the files that are required by an application? By all the files, I mean statically linked and dynamically called. What are these you may ask? Well, let me take a few minutes to explain.

The Windows OSs are composed of a collection of DLLs. It's this collection of DLLs that comprise the Windows API set. The complete list of files runs into the hundreds. This list includes files such as KERNEL32.DLL, GUI32.DLL, SYSTEM32.DLL, and so on. These are the very files that make the OS what it is. Naturally, a great number of applications take advantage of these libraries. Some applications may require these files to be located before the application even loads. At other times, the applications are loosely coupled to these same system files. The applications will load but become unstable when they go to use the library, and it's no longer there. The first example is called *statically linked*. This functionality is accomplished by linking to the library file when the application is compiled. Another way of linking to the file is at runtime. This linking is often referred to as *dynamically linked*. The application loads the system file while the application is running and makes a call to an exported routine.

Not only does your application link to files, but the files the application links to may link to other files. This process of a library calling another library calling another library is referred to as *chaining*. Figure 3.6 illustrates just how complicated this process can become. You may be surprised that a fairly small inhouse application (6 to 10 files) may require a support structure of other libraries totaling close to a hundred. (We've seen this figure in Chapter 1, but I'd like to show it again.) Take a look at the figure for a general idea of what I'm talking about.

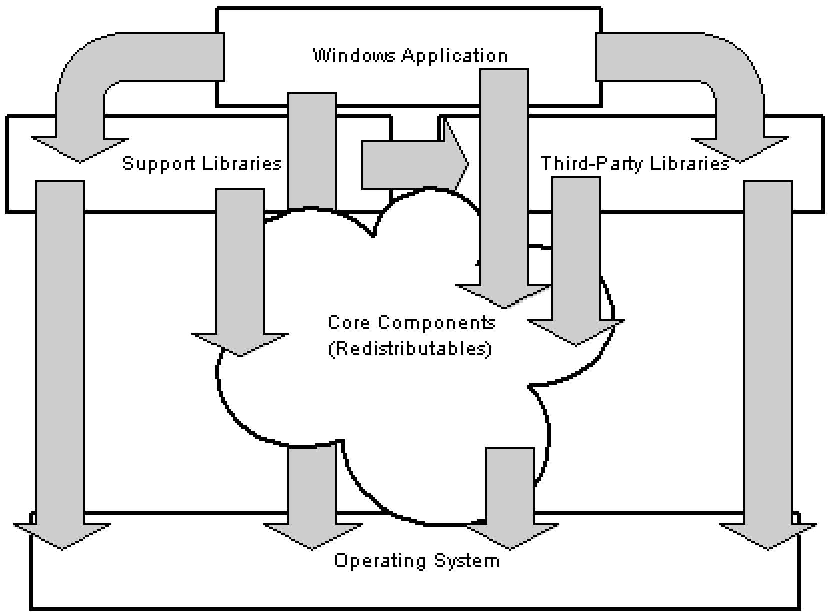


Figure 3.6: Each Windows or application library has a heavy dependency on other files.

In this figure, you can see by the arrows that all the libraries are related. The application files require support files and perhaps third-party files. In turn, the support files may require the third-party files. At this level, all may require specific core components. Sooner or later, every file requires functionality exposed by the OS libraries. It truly is a messy world.

You may be asking yourself why this dependency matters. Well, it does in an interesting way. In the process of creating the package, we'd like to know the complete list of files that an application may need—at any time. While there are a number of tools that can determine which files are statically linked, determining which files the application loads as it is running is quite a different matter.

To make this determination, the application must be launched and watched. Once the application is loaded, it must be thoroughly exercised. Every bit of functionality must be used to monitor all the files that the application requires. This process is called *profiling*. There are a number of tools that will profile for you, and you may even find that you're packaging software is one of them. A publicly available tool called Dependency Walker provides a list of all linked files—dynamically (implicit and explicit) or statically linked.

Dependency Walker is publicly available from http://www.dependencywalker.com. Steve Miller provides this tool.

Keep in mind that the results are only as good as the amount of functionality exposed by running and operating the application. You may find that the software test team is better at getting complete results than you are. In fact, because they are continually running and testing the application, they'll be able to create and maintain a more complete list than you can.

Once you've successfully profiled the application, you'll have a huge list of files. You'll need to sort this list into something meaningful. The files listed can be split into two broad groups: ones you can redistribute and ones you can't. Your first task is to sort out which ones are which. Redistributable files include

- Inhouse files (the original software creator creates and builds)
- Microsoft core components (ODBC, ADO, RDO, IIS, OLE DB, MFC, VB, and so on)
- Third-party files (SQL Anywhere, Java Runtime Engines—JRE, JRun, Crystal Reports, and so on)

Non-redistributable files include

- System files (part of the OS)
- Debugging files

There are a number of ways to filter out these files. Let's take a look at a couple of tricks that you can use. The first is to extract the company name from the resource of each file. The easiest way to do so is to right-click the filename and select Properties from the secondary menu. From the subsequent tabbed dialog box, select the Version tab. On this tab, you'll find a number of informational parameters about the file. If it says Microsoft Corporation for *Company Name*, the file is either part of a Microsoft core component (redistributable), a Microsoft non-redistributable file, or a Microsoft system file. Not much help, but we can delve deeper. If the *Company Name* refers to anyone else, you'll need to explore the redistribution agreement provided with the software documentation. For large numbers of files, writing a VBScript is a quick and easy way to perform this task.

Let's explore the Microsoft files a bit further. The easiest way to sort out the files is to put the Microsoft Developer Network (MSDN) to work. For the past couple of years, Microsoft has provided a DLL Help database that provides versioning information about all the Microsoft files. This reference is located at http://msdn.microsoft.com. Once there, select Support from the right-hand vertical menu, then select DLL Help Database from the drop-down menu. As of this writing, you can find the database at http://support.microsoft.com/servicedesks/fileversion/dllinfo.asp?fr=0&sd=msdn.

Enter the name of the file in question, and you'll get a list of all versions of that file. For a particular file, select more information to see which Microsoft product shipped the specified file. Unless you've had some experience working with these files, determining which files are redistributable and which ones are not is still difficult. The general clue is the core component string. In this context, this core component is not the same as the component we've been talking about. If you see this string associated with a file several times, you may suspect that it's part of the OS. The surefire way to know is to search the MSDN library and knowledge base for more information. After poring over a couple of developer articles, you should get the general idea in a hurry.

The following list summarizes some of the basic rules of redistributing conflicting core components:

- Install the oldest supported versions of core components—don't ship the latest and greatest if your application doesn't require it.
- Take a snapshot of the oldest supported minimum requirements for the system (hardware and software).
- Keep core components on the user's system—make sure they are excluded from the uninstall. We'll cover this rule in the next chapter.
- Install core components only if they are required and not just to be safe.
- Use the specified redistribution mechanism suggested by the developer of the core component. The specified method might provide an easier means of updating the component.
- Place third-party redistributables in recommended locations. Check the application documentation, your install notes, or the application's Web site for help in determining the recommended location.

New Boundary Technologies' Dependency Walker is a useful tool if you have the time and desire to learn the ins and outs of each file within the software you're deploying. If you're time constrained and have some critical software, such as a security update, that you need to quickly deploy, you can rely on a good snapshot tool and you won't need to concern yourself with these details.

Test

Testing your final package is the catchall for problem packages. Initially, a test may be as simple as resetting the test computer back to its baseline status and deploying the package. Once you're comfortable with this environment, pick one or two likely candidates. These initial candidates should not be critical computers or contain irreplaceable data. If the package deploys satisfactorily and the pre-existing software applications are not adversely affected, you can expand the test.

The first real test of the deployable package is the pilot rollout. The point of the pilot rollout is to test the assumptions about the target systems, the install package, the bandwidth, and so on. Pick a good representative group for the rollout—different computers, groups, and users. Make sure the representative group has the normal access rights you expect users to have; don't do a pilot rollout to administrators that have full access to the local system or servers, or you may not discover a lurking permissions problem. Take the extra effort to make sure that these are applicable users. Using a group that has little if any interest in the application doesn't make sense. A little bit of variety is good for the pilot rollout. Try to test a little bit at a time. As you gain confidence with packages, you can expand the variety. If remote and mobile users are a concern, you may want to include a few in the pilot rollout. If the testing or pilot rollout process uncovers any modifications, they should be documented and included with compatibility-testing and conflict-resolution documentation.

Repackaging Is Easy, Right?

We've covered a lot of information in this chapter and you may be feeling rather intimidated by the process. Think back to the beginning of the chapter where we talked about getting you familiar with the process even though you may not use it all. For most cases, the actual repackaging process has become easier.

Many of the steps we discussed are full of details you may not use on a common repackaging effort. It's when you hit those trouble spots that the added data will come into play. Take for example the section on profiling. It covered great detail about how to review conflicts and decide about core files. Do you need to do that for each package? No. Software is available to review the conflicts (for example, New Boundary Technologies' Prism Conflict Checker) and can ease the burden, resolving many of the issues for you. Knowing what an automated process is doing will help you in confirming the results.

With the current level of repackaging software, many of the details we covered in this chapter are handled automatically, and the results are presented in various formats. Knowing what to do with that information can be the difference between success and failure in getting your package out. As you will see in the following example, getting a package out is a logical procession of steps.

A Repackaging Example: Macromedia ColdFusion 5

Let's take some time to go through a real-world example of creating a deployable package. I've selected Macromedia ColdFusion 5 as the application to be packaged. It is by no means a simple application and serves to test the boundaries of our snapshot software.

The native install program for ColdFusion 5 was written with InstallShield 6.2. What makes this example even more applicable is that it does not use the InstallShield silent install (.iss) option. For our example needs, this process will show us all the modifications the install program will do.

Clean Requirements

The clean requirements for ColdFusion 5 are a 32-bit Windows OS, excluding Win95, Win98, and Windows ME. I've elected to go with NT 4.0 Server. Fortunately I have a test computer with an image of the required OS. I'm also using Symantec's Ghost 2001 for imaging software.

Baseline Requirements

The baseline requirements for ColdFusion 5 are a Web server, IE 5.0 or later, and the latest IIS security patches. Because I'll be using IIS, the security patches are especially important.

Preparing to Create the Package

For this exercise, I've elected to use New Boundary Technologies' Prism Deploy. This package does everything that I'll need for my package. It provides a snapshot environment with all the right smarts. I can edit the package after it has been created, and within Prism Deploy is a deployment environment. (We won't get to the deployment environment in this chapter, but we'll certainly be using it in Chapter 4.)

To this end, I've installed Prism Deploy onto my test system. I used InCtrl5 to watch where the application files ended up and to ensure that no system files or registry changes occurred that could contaminate my image. Because the vendor has taken the existence of the repackaging software on the system into account, it has been specifically designed to place all files into its own folder and does not update or install any shared or system files.

Once installed, the first thing to examine is the rules.ini file. This file defines the boundaries for the snapshot process. By editing this file, we can limit which registry keys are examined, which folders are excluded from the snapshot, and so on. A complete list of the default parameters defined by the rules.ini file follows.

Ignored folders:

- %RECENT%
- %TEMP%
- %WINSYSDIR%\CONFIG
- %WINSYSDIR%\NtmsData
- TEMPORARY INTERNET FILES
- \RECYCLED
- \RECYCLER
- _RESTORE
- %WINDIR%\Security
- %WINDIR%\Debug

Ignored registry keys:

- Software\LANovation
- Explorer\RecentDocs
- Explorer\StreamMRU
- Explorer\Streams
- Explorer\DesktopStreamMRU
- Explorer\DesktopStreams
- UuidPersistentData
- HKEY_LOCAL_MACHINE\SYSTEM\ControlSet
- HKEY_LOCAL_MACHINE\SYSTEM\Clone

- HKEY_LOCAL_MACHINE\Clone
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\PendingFileRenameOperations
- HKEY_LOCAL_MACHINE\SAM
- HKEY_LOCAL_MACHINE\SECURITY
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography
- HKEY_LOCAL_MACHINE\HARDWARE\ResourceMap\PNPManager
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\***\Control
- HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer

Ignored files:

- 386SPART.PAR
- PAGEFILE.SYS
- WIN386.SWP
- SYSTEM.DA
- USER.DA
- NTUSER.DA
- AVCONSOL.INI
- NTUSER.DAT.LOG
- SYSTEM.DAT.LOG
- WININIT.INI
- CLASSES.DAT
- SHELLICONCACHE
- SCHEDLGU.TXT
- PICTAKER.LOG
- METABASE.BIN
- *.SFTCH

Ignored file types:

- (Default setting)
- Hidden Files (No)
- System Files (No)
- Read-only Files (No)
- Temporary Files (Yes)

- Compressed Files (No)

Registry keys as counters:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Shared DLLs

Registry keys whose contents are merged:

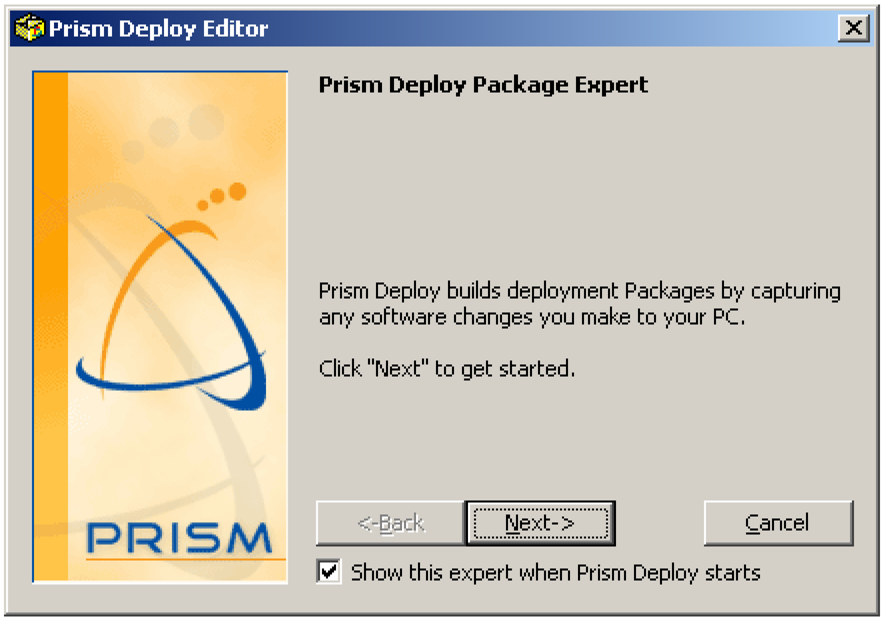
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
  Session Manager\Environment,Os2LibPath
  Control\Session Manager\Environment,Path
  Control\Session Manager\Environment,PATHEXT
- HKEY_CURRENT_USER\Environment,
  Os2LibPath
  Path
  PATHEXT
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
  Services\Eventlog\Application,Sources
  VirtualDeviceDrivers,VDD
  Control\ServiceGroupOrder,List
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\NetworkCards\*\NetRules\Block
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\*\

  Linkage,Bind

  Linkage,Export

  Linkage,Route

The rules.ini file is editable, so these values can be modified or appended. You'll find the rules.ini file directly from the Start menu, within the Prism Deploy shortcut folder. In this example, I've decided to go with the default settings. Launching Prism Deploy Editor starts the process, as Figure 3.7 shows. Prism Deploy uses a series of wizard dialog boxes to gather the initial information.


Prism Deploy Editor

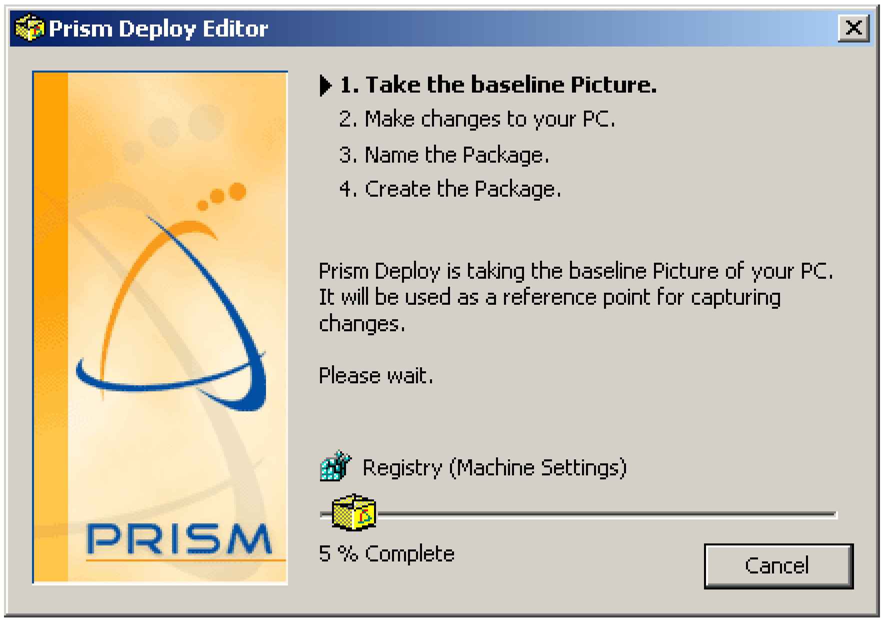
Prism Deploy Package Expert

Prism Deploy builds deployment Packages by capturing any software changes you make to your PC.

Click "Next" to get started.

Figure 3.7: Introduction to the Prism Deploy wizard.

The first dialog box is an introduction to the snapshot process. I elected to continue through the wizard expert. The next dialog box shows a progress gauge as the initial snapshot is performed, as Figure 3.8 illustrates.


Prism Deploy Editor

1. Take the baseline Picture.
2. Make changes to your PC.
3. Name the Package.
4. Create the Package.

Prism Deploy is taking the baseline Picture of your PC. It will be used as a reference point for capturing changes.

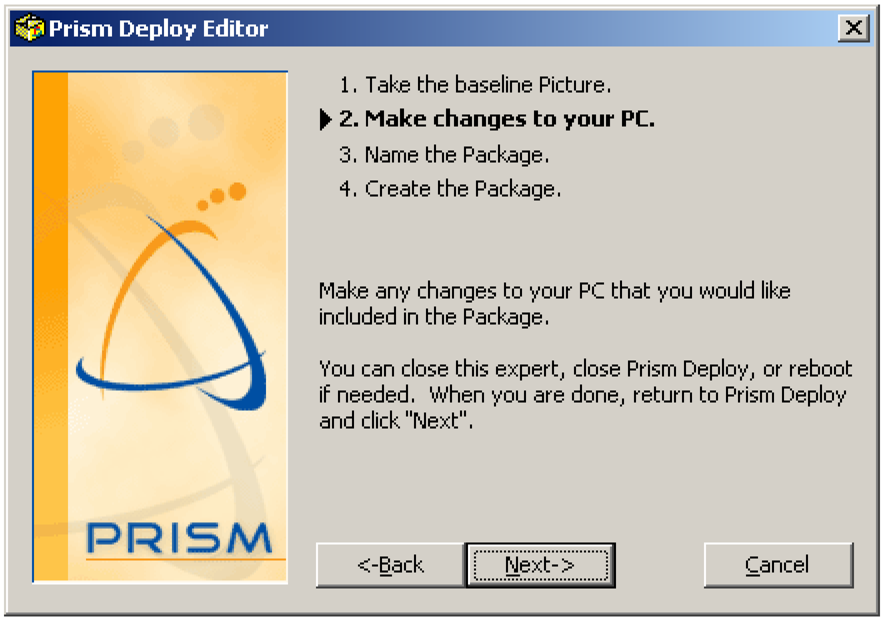
Please wait.

Registry (Machine Settings)

5 % Complete

Figure 3.8: The initial snapshot is performed while this dialog box is displayed.

The registry, computer folders and files, as well as some file changes are all stored for a later comparison with the second snapshot. After the initial snapshot is finished, you can exit Prism Deploy or leave it running. In this case, as Figure 3.9 shows, I've elected to exit Prism Deploy.



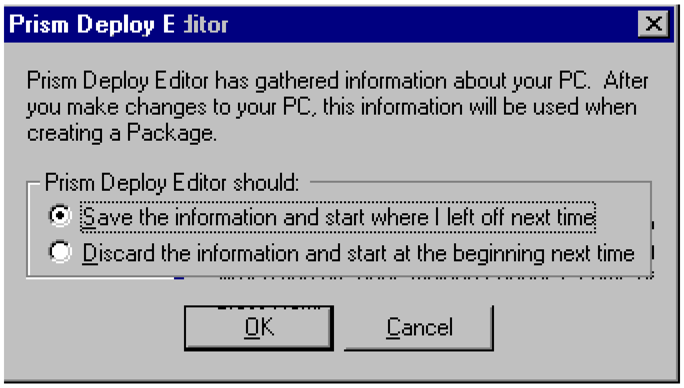
1. Take the baseline Picture.
2. **Make changes to your PC.**
3. Name the Package.
4. Create the Package.

Make any changes to your PC that you would like included in the Package.

You can close this expert, close Prism Deploy, or reboot if needed. When you are done, return to Prism Deploy and click "Next".

Figure 3.9: At this point, we can exit the snapshot process and start the target install program.

Upon exiting, a prompt confirms our selection to save the detected information. Figure 3.10 shows this prompt. This snapshot will be saved and used to analyze changes by comparing the second snapshot.



Prism Deploy Editor has gathered information about your PC. After you make changes to your PC, this information will be used when creating a Package.

Prism Deploy Editor should:

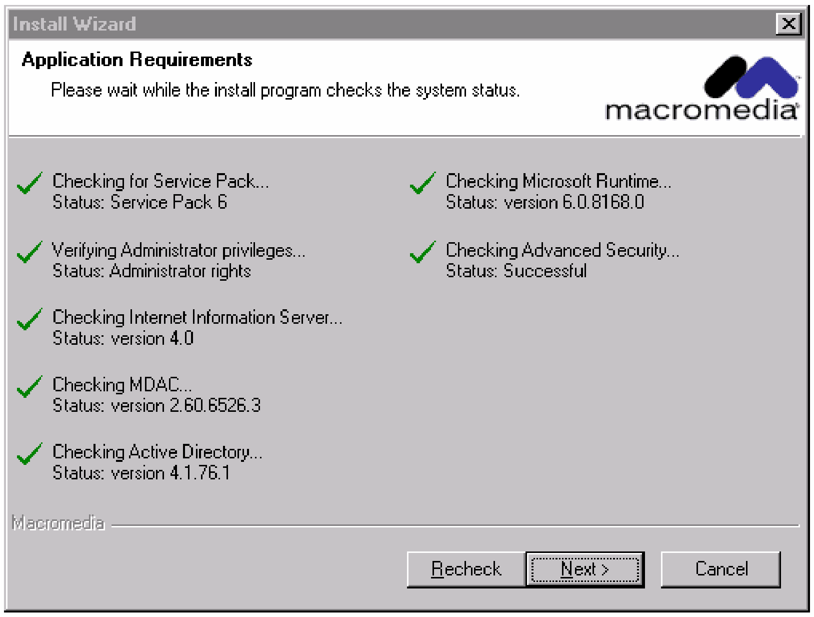
[✓ Save the information and start where I left off next time]
Discard the information and start at the beginning next time

Figure 3.10: While exiting the wizard expert, we elect to have the detected information saved.

Once the first snapshot is finished, the ColdFusion 5 install can be started

Normal Install

The ColdFusion 5 install is initiated from a CD-ROM. The first dialog box, which Figure 3.11 illustrates, displays the system status as a number of system prerequisites with checks. Although not shown in this figure, the ColdFusion 5 install program detected older versions of MDAC and the MFC runtimes.



Install Wizard

Application Requirements

Please wait while the install program checks the system status.

✓ Checking for Service Pack...
Status: Service Pack 6

✓ Verifying Administrator privileges...
Status: Administrator rights

✓ Checking Internet Information Server...
Status: version 4.0

✓ Checking MDAC...
Status: version 2.60.6526.3

✓ Checking Active Directory...
Status: version 4.1.76.1

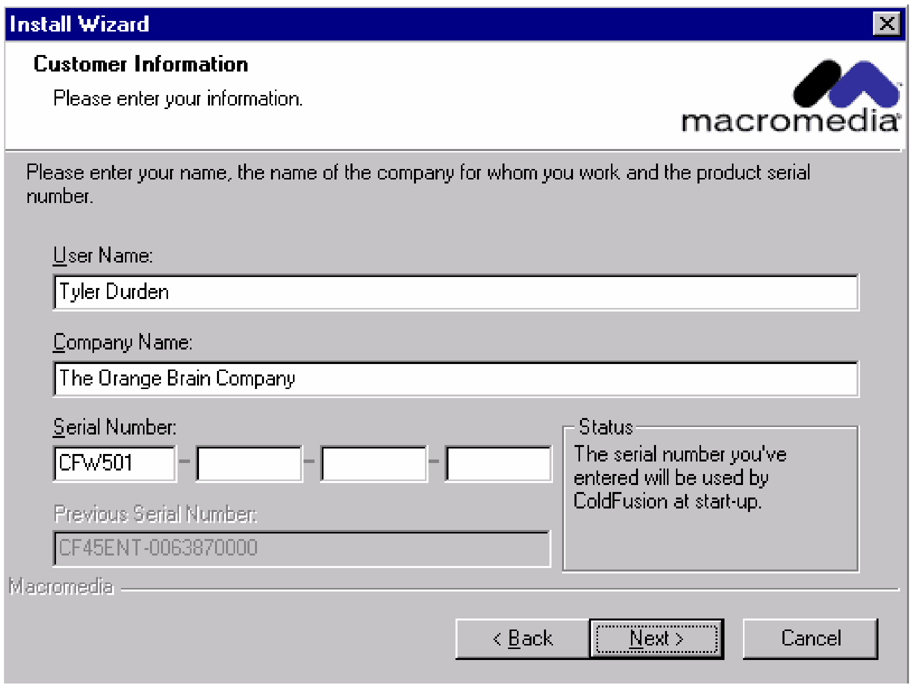
✓ Checking Microsoft Runtime...
Status: version 6.0.8168.0

✓ Checking Advanced Security...
Status: Successful

Figure 3.11: The ColdFusion 5 install program queries for a group of application requirements.

In each case, the appropriate redistributable was installed and the computer rebooted. By looking on the CD-ROM image, you can see that ColdFusion 5 ships with the appropriate installs for the prerequisites. After two reboots, the baseline system meets the minimum application requirements and the ColdFusion 5 install continues.

Select Next through the standard welcome and license dialog boxes. The next dialog box in the sequence that is worth looking at prompts us for customer information, as Figure 3.12 shows.



Install Wizard

Customer Information

Please enter your information.

Please enter your name, the name of the company for whom you work and the product serial number.

User Name: Tyler Durden

Company Name: The Orange Brain Company

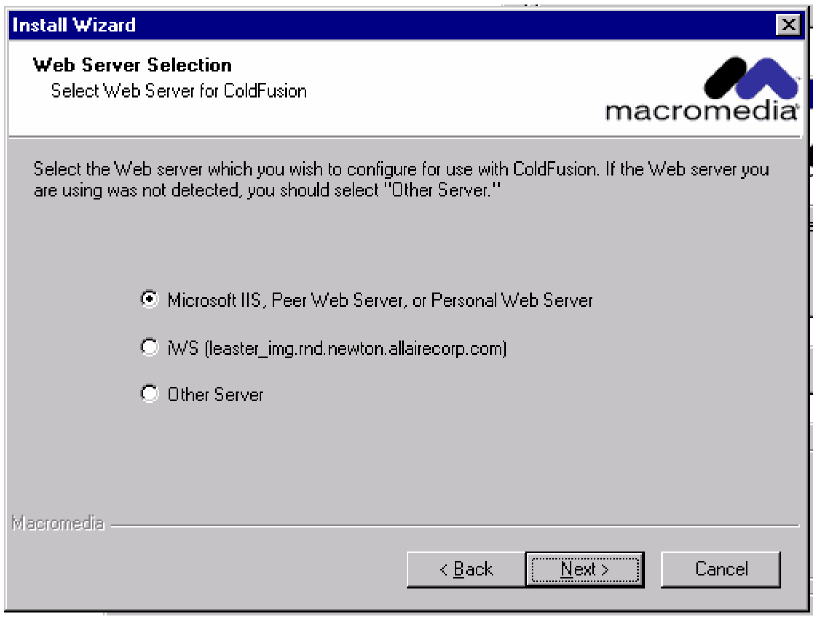
Serial Number: CFW501 - - -

Previous Serial Number: CF45ENT-0063870000

Status: The serial number you've entered will be used by ColdFusion at start-up.

Figure 3.12: A customer information dialog box prompts for, among other things, a serial number.

The information in the previous dialog box represents some of the issues to resolve in the package. The user and company name can be resolved by replacing them with generic values. More than likely, the individual serial number will have to be replaced with an OEM equivalent. The most common solution is to use a corporate-wide serial number for all installs. If this option isn't available, an alternative will have to be pursued. If the serial number is stored directly in the system registry (for example, if the application is an upgrade), you could write a script to update this value on a per-user system. In the case of our example tool, Prism Deploy allows you to substitute a variable for the hard-coded serial number. The variable can read a specified registry value and write the value during the package installation without requiring any additional scripting. As Figure 3.13 shows, the ColdFusion 5 install program queries for existing Web servers.


Install Wizard

Web Server Selection
Select Web Server for ColdFusion

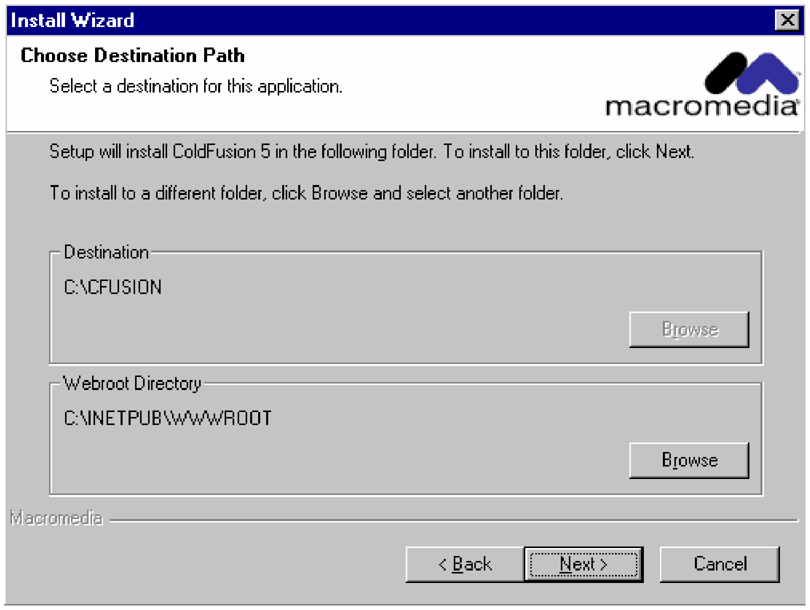
Select the Web server which you wish to configure for use with ColdFusion. If the Web server you are using was not detected, you should select "Other Server."

[✓ Microsoft IIS, Peer Web Server, or Personal Web Server]
iWS (leaster_img.rnd.newton.allairecorp.com)
Other Server

Figure 3.13: Multiple Web servers were detected by the install program.

In this case, Microsoft IIS and Netscape iPlanet were detected. In our distribution model, we are more than likely expecting to find a single standard Web server. Making the appropriate selection is important to a consistent rollout.

The ColdFusion 5 application installs files not only to the application destination but also to a location accessible by the Web server. On the dialog box that Figure 3.14 shows, both paths must be provided. The Webroot directory is unique to the Web server that was detected.


Install Wizard

Choose Destination Path

Select a destination for this application.

Setup will install ColdFusion 5 in the following folder. To install to this folder, click Next.

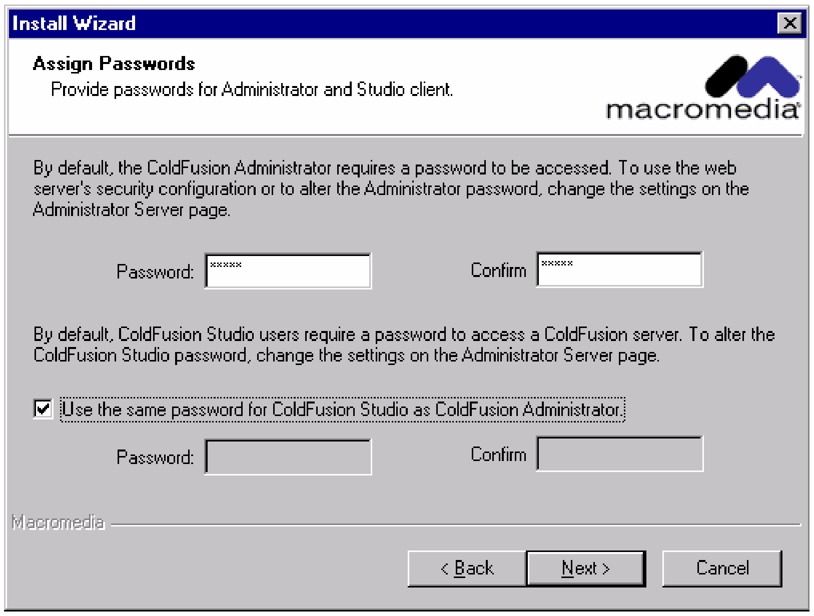
To install to a different folder, click Browse and select another folder.

Destination: C:\CFUSION

Webroot Directory: C:\INETPUB\WWWROOT

Figure 3.14: Two path locations are requested.

The ColdFusion 5 application server requires a password for the ColdFusion Administrator. A second password is required for the ColdFusion client tool, ColdFusion Studio. The dialog box that Figure 3.15 shows gathers both passwords.


Install Wizard

Assign Passwords

Provide passwords for Administrator and Studio client.

By default, the ColdFusion Administrator requires a password to be accessed. To use the web server's security configuration or to alter the Administrator password, change the settings on the Administrator Server page.

Password: ***** Confirm: *****

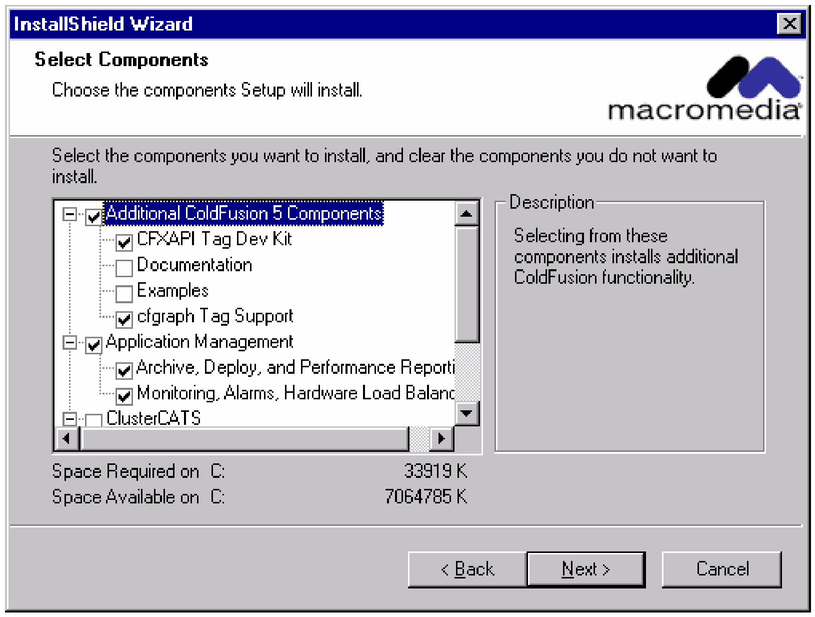
By default, ColdFusion Studio users require a password to access a ColdFusion server. To alter the ColdFusion Studio password, change the settings on the Administrator Server page.

[✓] Use the same password for ColdFusion Studio as ColdFusion Administrator.

Password: Confirm:

Figure 3.15: The ColdFusion Administrator requires the user to provide a password.

A key issue to resolve for almost all install programs is which components should be installed. More than likely, this question will be resolved from the rollout team and will be included in the rollout document. Figure 3.16 shows the dialog box in which you must make this selection.



Select the components you want to install, and clear the components you do not want to install.

- [✓] Additional ColdFusion 5 Components
  - [✓] CFXAPI Tag Dev Kit
  - [ ] Documentation
  - [ ] Examples
  - [✓] cfgraph Tag Support
- [✓] Application Management
  - [✓] Archive, Deploy, and Performance Reporti
  - [✓] Monitoring, Alarms, Hardware Load Balanc
- [ ] ClusterCATS

Description

Selecting from these components installs additional ColdFusion functionality.

Space Required on	C:	33919 K
Space Available on	C:	7064785 K

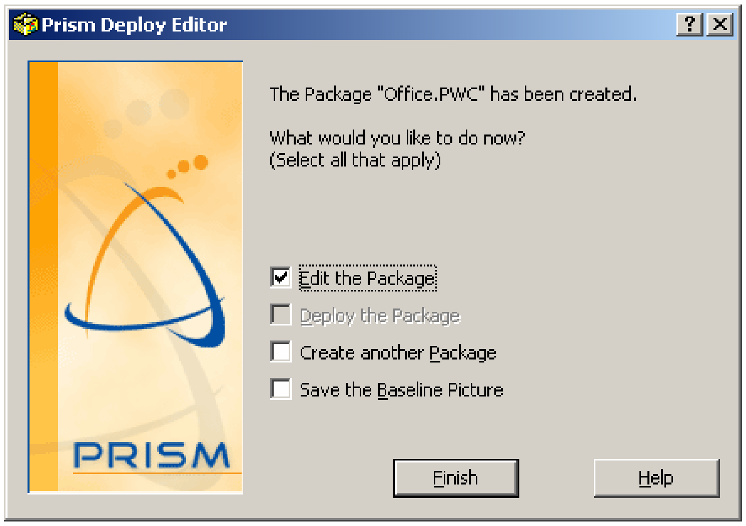
Figure 3.16: Component selection is a key dialog box that almost all install programs provide.

After the remaining dialog boxes (Confirm Selections, File Transfer, and Finish), the application is installed. The ColdFusion 5 install required a reboot. After the reboot, I launched ColdFusion 5 to verify that it was installed correctly and that all is well.

Snapshot Process

After the install is finished, I launched Prism Deploy again. It automatically detects that it's in the middle of a system snapshot, and we're prompted to continue. After selecting Next, the final snapshot is performed and a comparison is made. Afterward, the comparison is used to create the ColdFusion 5 package

As Figure 3.17 shows, an opportunity is given to edit the package. At the same time, we can save the snapshot results as a ColdFusion 5 baseline. I've elected to edit the package.



The Package "Office.PWC" has been created.

What would you like to do now?
(Select all that apply)

- [✓] Edit the Package
- [ ] Deploy the Package
- [ ] Create another Package
- [ ] Save the Baseline Picture

Figure 3.17: After the snapshot comparison is done, we're given the opportunity to modify the package.

This selection launches the Prism Deploy Editor. From within this application, I can evaluate, change, and fine-tune my ColdFusion 5 package. Before we begin that process, however, let's take a look at what was captured in the snapshot comparison. Figure 3.18 shows the Prism Deploy Editor with the ColdFusion 5 package snapshot comparison. The package is separated into a number of sections, as shown in the left pane. The horizontal toolbar provides quick access to a number of useful features.

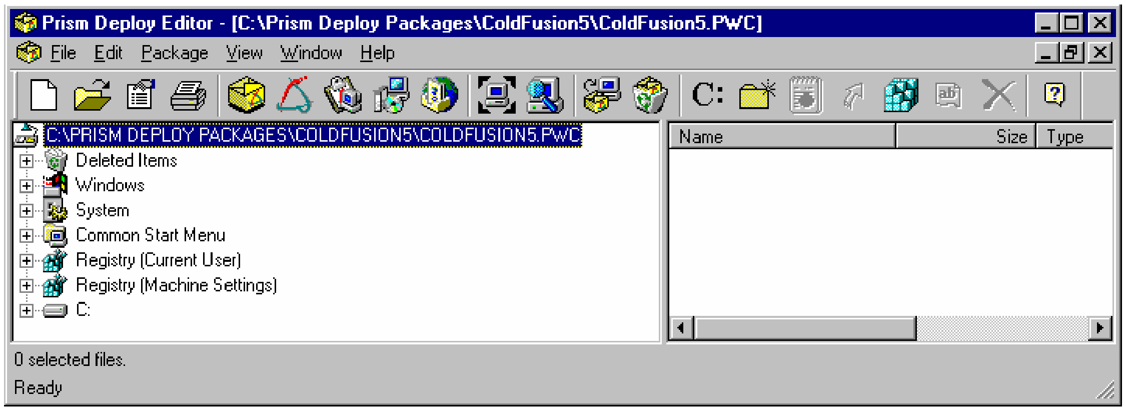

Prism Deploy Editor - [C:\Prism Deploy Packages\ColdFusion5\ColdFusion5.PWC]
File Edit Package View Window Help
C:\PRISM DEPLOY PACKAGES\COLDFUSION5\COLDFUSION5.PWC
Deleted Items
Windows
System
Common Start Menu
Registry (Current User)
Registry (Machine Settings)
C:
Name
Size
Type
0 selected files.
Ready

Figure 3.18: The Prism Deploy Editor is where we'll perform the modifications to our package.

From the Prism Deploy Editor, elements in the package can be removed, added, or modified. As we go through each of the package subtrees (in the left pane), as Figure 3.19 shows, a decision can be made on which values should stay and which should go. If any values are unknown or unclear, you should research them to fully understand where they came from.

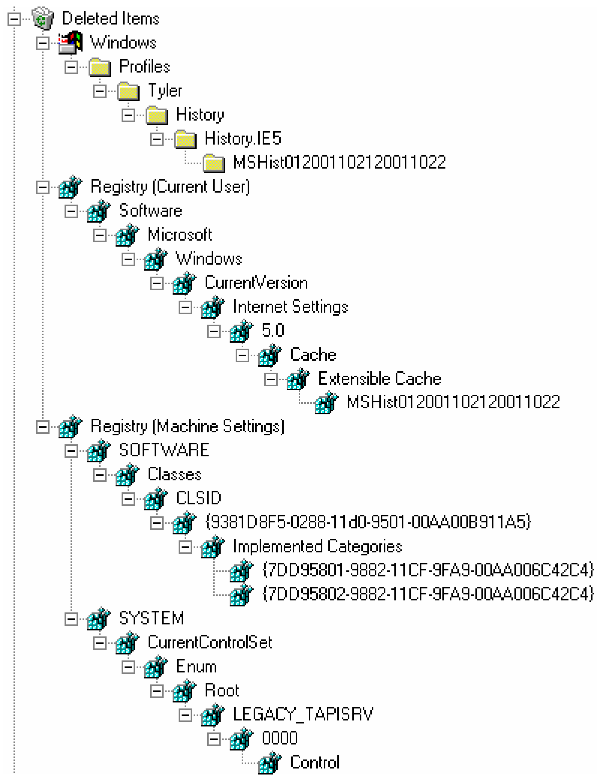
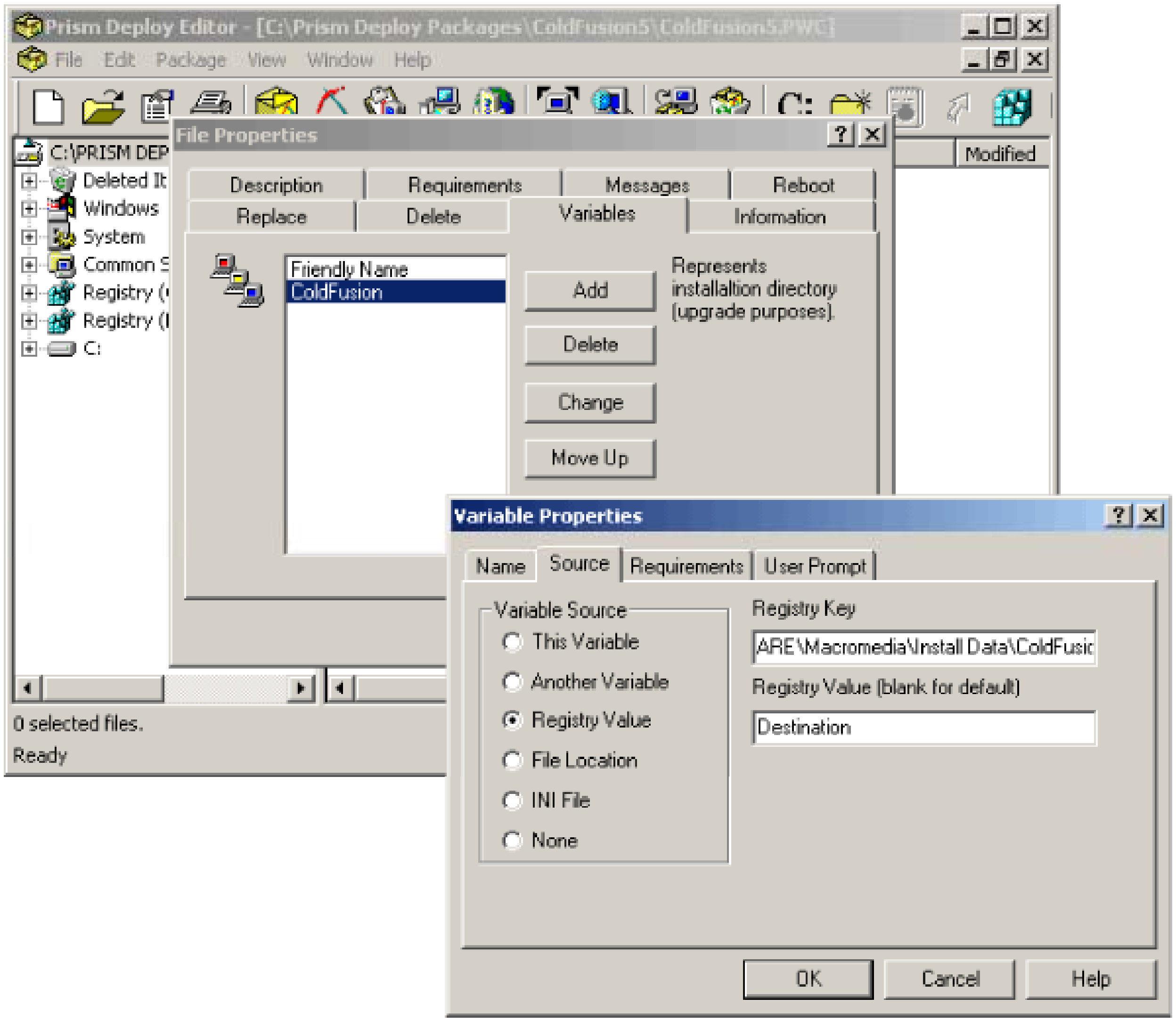


Figure 3.19: This list of deleted items shows up in our package.

There is a fair amount of noise in the everyday operation of the OS. The snapshot comparison will invariably determine that some of this noise is relevant system changes. A key task in creating the package is removing the unnecessary (and potentially damaging) system changes.

Package Modifications

To modify the package, Prism Deploy offers some very nice features, such as adding folders and files and abstracting certain hard-coded values as more malleable variables. This feature allows us to distribute the package and fine-tune certain values based on the target computer. Figure 3.20 demonstrates turning the installation directory into a variable. This feature is especially useful when you are upgrading software but you're unsure where the software is installed on each target PC. Prism Deploy can scan an existing registry value and change all embedded paths within the package to reflect each machine's installation directory. This feature enables one package to work for new installations as well as upgrades. A Prism Deploy package changes itself for a diverse environment all with little or no scripting.



Friendly Name
ColdFusion

Represents installation directory (upgrade purposes).

0 selected files.
Ready

Variable Source
- This Variable
- Another Variable
- [✓ Registry Value]
- File Location
- INI File
- None

Registry Key: ARE\Macromedia\Install Data\ColdFusic
Registry Value (blank for default): Destination

Figure 3.20: Turning an installation directory into a variable.

After removing the extraneous changes, Prism Deploy gives you the option of adding functionality. For instance, you can specify how to replace files that already exist on target systems, you can make Prism Deploy as loud or silent as you like (you can add before, after, and during prompts that display to end users), you can set reboot properties, you can define prerequisites (such as OS and disk space checks), and you can substitute Prism Deploy variables to define areas such as the serial number inclusion, user environment data, and system details. The package is now ready to test.

Comments on the Final Package

With Prism Deploy, an accurate distribution package was created. Some minor modifications were necessary to the package, such as the serial number issue identified earlier. Because the changes generated by the MDAC and MFC runtime installer were picked up, separate installable packages could be created.

Summary

Repackaging is a big process. The intention of this chapter was to provide you with adequate background information so that you would understand the scope of the work. Much of this chapter was spent looking at simply preparing the test computer for the package process. We also looked at a number of tools to help you get started. The key is to understand the process and become familiar with the available tools.

Don’t forget to keep the install specification up-to-date—especially through this crucial stage. You'll face a number of people asking questions about what the package does or doesn't do. Having a complete and accurate specification at the ready demonstrates your commitment and knowledge of the package process. As an appendix item, you may want to include the entire snapshot and profiling data. When problems arise, you'll find them a handy reference.

The final step is iterative testing. I recommend verifying core logical pieces as they are implemented. “Package, Debug, Test, Repackage, Debug, Test” is a good mantra and will save you time. The list below summarizes the steps we've gone through to get to this point. Your mileage may vary, but I believe you'll find it a solid starting point:

1. Create a base rollout specification.
2. Run the native install program and incorporate the results into the specification.
3. Interview team members and submit the first pass of the rollout specification.
4. Take a snapshot of the application on a clean machine (for each OS) with the required baseline.
5. Profile the application for linked libraries.
6. Process the snapshot and profiling results into the distribution package.
7. Update the rollout specification.
8. Test and iterate.

Chapter 4: Methods for Deploying Software

You now have a package ready to deploy. Your application is tested and installs on all your test machines; its functionality works just as you planned and there are no incompatibility problems with the multitude of applications on your test machines. The job is done, right? Oh, wait. You have to get the packaged application *deployed* somehow.

Like everything else about computers, there is a multitude of ways to ensure your workstations get the application. The choice is yours. This chapter will discuss various methods of deployment, the pros and cons of each method, and provide some practical tips.

Choosing the *method* of deployment is just as important as the actual deployment. Although you might usually send out all your applications using a commercial deployment-software application such as New Boundary Technologies' Prism Deploy or Microsoft's SMS, circumstances for a particular deployment might require using an entirely different method. For example, if you need to load a small application on only two workstations that are right next to you, manually deploying the application is easier than any other method.

Having a regular method established lets you package your applications to a known set of policies and standards. Understanding the needs of the deployment can assist you in making the proper choice of methods. Do you follow the company standard or is the manual method the way to go this time? The answer has to be evaluated for each application deployment.

History

In the early stages of modern computing, the Big Iron (mainframe) world used a form of basic software deployment. All the processing ran on the mainframe, and the terminals only had to deal with the input and output. Everything was centralized and the only processing that was done on the terminals was simple presentation. As the Little Plastic (PC) world came into the business environment, the mainframe world began to offload some of the processing onto the "smart terminals." As PCs came into greater use and were accepted by the mainframe staff, application installs were all done on a one-by-one basis with the original media. There were administrators in the IT shop that did nothing but run from location to location with a bag of 5¼" floppy disks and install the requested applications. As the PC world evolved into a larger distributed environment, the expense of doing manual installations mandated that an alternative solution be found to drive those costs down.

One method was to rely on the users to do the install work. With a desktop OS such as Windows 3.1 and a networking solution such as Novell in place, the users could connect to the network, map a drive to a single server location, and follow written procedures to install a package that resided on the network. As the applications evolved in complexity past VisiCalc (remember that one?), the level of computer knowledge needed to install applications quickly surpassed what users were willing to become skilled at. Before the installers in use today were around, installing applications often required making manual adjustments such as customizing INI files—not exactly the user-friendly method we have come to expect from software installs. Network administrators started looking for other ways of getting the software out to the end users in a reliable manner.

Network administrators began creating scripts and repackaging the applications into various methods they were familiar with. This method was met with varying degrees of success, depending on the skill of the packager, the deployment environment, and the software application. There were few commercial deployment methods in wide use in those days, so deployment styles in one company were rarely compatible with other companies' styles. Without the standards, a common practice was to develop a custom way to deploy applications. This situation created a nightmare for software companies as they started looking at how to write their applications to be "network aware."

Early attempts by software companies focused on sending only a few files or executables to the workstation and running the application on the server. The PC world was trying to emulate the Big Iron mainframe environment by centralizing everything and sending a minimum of the application to the workstation. As network use and application complexity increased, the world of 8088 and 286 machines quickly flooded the Token Ring 4-megabit network. A solution had to be found.

One of the first companies to work on solving that issue was New Boundary Technologies with its LAN Escort product. The product was one of the first successful commercial deployment packages to focus on enterprise environments.

The Parts of a Deployment Methodology

By d*eployment methodology* or d*eployment software*, I'm talking about the process of taking the packaged application, getting it out to your target machines on schedule (if there is one), and confirming that it installed properly on the target systems. Let's break this process down into basic parts, some of which are review at this stage:

1. Prepare the package for deployment (which we discussed in Chapter 3).
2. Identify the *targets* (the machines that will receive the deployment) and the method of grouping the targets (*User-based* and *machine-based* are two examples of grouping.)
3. Identify any target-impacting requirements. Reboots or user authority are a few common issues that can impact the user.
4. Identify any deployment time requirements. Does the application have to be there NOW, by the start of business in the morning, or simply made available for users to get when they want it? Can the application only be installed within a dark window (a time of opportunity that does not impact the users)?
5. Determine online/offline status of the targets. How will the target machines get the application if they are offline?
6. Communicate to users. Is the application going to be visible to end users? If so, they need to know about it.
7. Review the deployment status. Did the software get to the targets you wanted? Did it get on the machines properly without issues?

Choosing a Deployment Method

Let's get started with a discussion by looking at the various deployment methods available. We'll talk about using Sneakernet, scripting, email, Web, image inclusion, and third-party solutions, including Prism Deploy, SMS, and IntelliMirror.

Sneakernet

Sneakernet is a tongue-in-cheek term for a manual deployment. The name came from a description of the primary transport mechanism of the data: your sneakers. You grab your install CD-ROM or floppy disk, your glasses, and the caffeine drink of your choice, then put on a good pair of sneakers, walk to the target machine, and manually install the software application one machine at a time. Anymore, this method is mostly used for machines that are not reachable from a network or for an installation of a standalone package to one or two machines. Obviously, this method is not the most economical nor does it lend itself to much consistency in the install method, as mistakes (called *finger-checks*) are common.

Paying an administrator to sit at a machine while an install process chugs away on the CD-ROM is expensive. If the install program for the application requires input or choices to be made, the process will need to be detailed out in a document so that each install is done properly, not at the whim of the installer. Even with a document in hand that speculates every single option, the administrator will make normal finger-checks or will find a better configuration method, and suddenly the target machine installation is not the same as other machines. This inconsistency may require a revisit to the machine(s) for adjustment of the application settings, which only adds to the overall expense.

Carrying a copy of the application around with you might be the preferred Sneakernet method because CD-ROMs are hard to damage if you take reasonable care in their use, but damage does occur. In addition, there are instances that might prevent their use. If the target machine does not have a CD-ROM drive or it fails during your install, the cost of Sneakernet just went up. Assuming you do not have network bandwidth issues, mount the CD-ROM on a share point and read the CD-ROM from another machine. The less the source media is moved around, the longer it will survive.

As with normal packaging, be sure to put the source media in a safe location. A drawback of Sneakernet is that if you lose the one CD-ROM you have for an application, you must buy another copy to reinstall on a single workstation if you are forced to reimage the machine.

The main advantages of Sneakernet are the ease of the install and the potential low level of skill needed to perform the install. You could basically take a beginning IT person, hand them the Office XP CD-ROM, give them a document to follow, a list of workstations to visit, and set them off to the task. Assuming the person does not make mistakes and works steadily, and the machines are close to each other, four or five machines could be installed every hour. Of course, with a mass deployment method, several hundred machines (or more) could be installed with the Office XP application in the same time frame.

A positive aspect of Sneakernet that can not be replicated by any mass deployment method is the face-to-face interaction you can have with the end user. The communication possibilities offered while the software churns its way onto the workstation goes a long way to removing the Faceless IT image we all fight. Talking to the user, explaining what is happening, doing some training, addressing a few of the user's computer issues, getting feedback, finding out how he or she uses your systems, or just sharing a joke or two is often worth the time to do a Sneakernet install.

However, this interaction comes at the cost of removing users from their workstations while you're performing a Sneakernet install. While the install is going on, users are not able to do their work. So to help reduce that impact, the installer tries to schedule a convenient time that does not impact the user as much. Maybe the install is done during lunch, scheduled breaks, or outside the normal work hours. This translates to the installer having to work longer hours or adjusted hours, and eliminates the interaction benefit I previously mentioned.

Will you ever use a Sneaker distribution? Sure. For example, if your CEO wants the latest version of Quake installed on his or her machine, creating a deployment package and sending it out with your standard mass deployment methods would take a lot longer than if you walked over to the CEO's machine with the CD-ROM and did the install. If 20 or 30 users were suddenly asking for the latest Quake version so that they could play with the CEO, Sneakernet is no longer the method to use. Table 4.1 compares the pros and cons of Sneakernet.

Pros of Sneakernet	**Cons of Sneakernet**
Quick to install and does not require repackaging.	Using the application install may not conform to your standards and could cause conflicts with other applications
Good method for one-off installs to local machines that you can physically access.	Remote silent deployments are not possible.
Possibly cost effective for a very small number of installs.	Even with detailed install documentation, differences in the install between machines are common, possibly resulting in different application configurations.
Provides user-interaction opportunities.	Losing the source media (CD-ROM or floppy) may prevent re-installs.
Usually requires minimal training of the technical staff for the installation task.	Very labor intensive for more than a handful of installs. Expensive when viewed from a cost of labor point of view.
	Support issues may result if a repair is needed. Remote repair is obviously not available.

Table 4.1: Pros and cons of Sneakernet.

Sneakernet Recommendation

As you can see in Table 4.1, Sneakernet is best used in one-off situations in which support and repeatable distribution are not issues. Sneakernet is not a recommended method for mass deployments.

Simple Scripting Methods

Scripts work. Scripts are great. Scripts are a pain to maintain. Scripts are the next step up the evolutionary ladder from Sneakernet. For the remainder of this chapter, the term "scripts" is defined in a generic way: WSH, C##, VB, or other programming languages are all lumped into this category. For this discussion, we are defining scripts as anything that you write in the scripting or programming language of your choice to facilitate a deployment.

Those who have been in the IT industry for several years remember when scripts were the only alternative to Sneakernet. Distribution scripts were created in the language most preferred by the script creator. The script would do one particular deployment, and rarely were the scripts generic enough to do complete rollouts of other applications. As many administrators faced the fact that Sneakernet would not be the best way to support a growing distributed environment of PCs, scripts became popular.

Scripting deployments is not a dead practice. For some businesses, scripting is their only option as a result of budgets or size of the deployments. Some form of scripting might also be used to enhance aspects of deployment if you use mass deployment software. Using scripting as the sole deployment method requires you to provide all the mechanisms to a deployment:

- Sending a package over the network.
- Triggering the workstation install (not to be confused with packaging).
- Providing a confirmation of delivery and install.
- Reporting on the deployment success or failure.

If we assume that only scripting is used to deploy an application, the process normally becomes very specific and not easily transferred to other applications without editing or modifications. Those edits and modifications can take as long or as much work as it took to build the original script. The process might also have a heavy dependency on other components and utilities, so the script might require copying a requisite utility to the target machine. This requirement will cause the install to expand in size. Starting to see why scripting is not always the best method?

The success (or failure) of scripts depends on your skill with programming. If API calls, CLSID, and DLL registrations are all familiar to you, scripting the deployment will be within your abilities. Admittedly, those are extreme examples of scripting and not all deployments will be that involved, if the install package is done properly and the source files are available to the target. When you script for deployments, remember to take into consideration some of the issues discussed in Chapter 3 about repackaging:

- Drive differences—is the directory on the C or D drive?
- File location variations
- How to reach offline machines when they come back online
- Any potential OS variances such as service pack levels
- User permissions for the install—the user might not have authority to install the application
- File version conflicts
- Is there a way to make sure certain files that are in the package you are deploying merge changes into files on targets (like INIs) without replacing them?
- Is there a means to back out of a failed deployment without breaking other software?

By using quality deployment software, these issues mentioned are taken care of for you.

One common use of scripting is to include the deployment mechanism in the logon process. By updating the NETLOGON directory and having the logon.bat file call the application, it is possible to have users install a simple application when they log on. Although this option sounds like an easy solution, it has major pitfalls for large applications:

- The user might be delayed in signing on until the install completes.
- The size of the install might be too large to download at logon. You should assume the worst case scenario and plan on a large number of your users hitting the NETLOGON directory at the same time. The impact to your network could be detrimental to the bandwidth, which would lengthen the install time, causing a longer impact than you originally planned for.
- Network bandwidth is not controllable with the logon method.
- If the application requires reboots, users are impacted. Their systems will take longer to become stable, which will delay them from beginning their day.
- Will users have the permissions required to complete the install? As an example, if a registry entry needs to be added or modified, and the user has restricted permissions to the local system, the user security context might prevent the application from fully installing.
- The user might decide not to wait for the logon process to finish and power off the machine (which is a common action). In frustration, the user might move to another machine and try again. If you use the logon method, communicate to users what is going to happen and how long to wait.

If you need to adjust a registry entry or map a common static drive letter or reconfigure a printer setting, NETLOGON scripting is effective. Just don't use this method for large applications such as Office XP.

A primary principle in programming is to document the program in such a way that anyone can go back later, read the documentation, and be able to follow the logic flow. Every programming or scripting how-to manual you read states the same principle. Scripts are programs and should be documented either externally to the script or with enough remarks internally to explain the logic flow. Ideally, if a deployment script were fully documented, it could be cut into modules and used in other deployments. In reality, often the script will be pieced together to solve the deployment project of the moment, not documented properly, and even the writer will have difficulty a year later trying to remember exactly what happens inside the script. If the script writer no longer works for the company, another person will have to put in the time to get up to speed on any complex deployment processes the script is doing.

Could a script be created generically enough to launch packaged applications without customization? Sure. Will it work in all cases? That will be up to the skill of the scripter.

The types of utilities that can be included in your scripts to assist in the deployment are nearly limitless and growing in number on a daily basis. Cruise the Web and look for shareware or freeware utilities. If you are sharp at programming, creating your own utilities should not a problem.

To help you find specific function utilities, check out the following Web sites: http://download.cnet.com/, http://www.zdnet.com/downloads/, and http://appdeploy.com/downloads/.

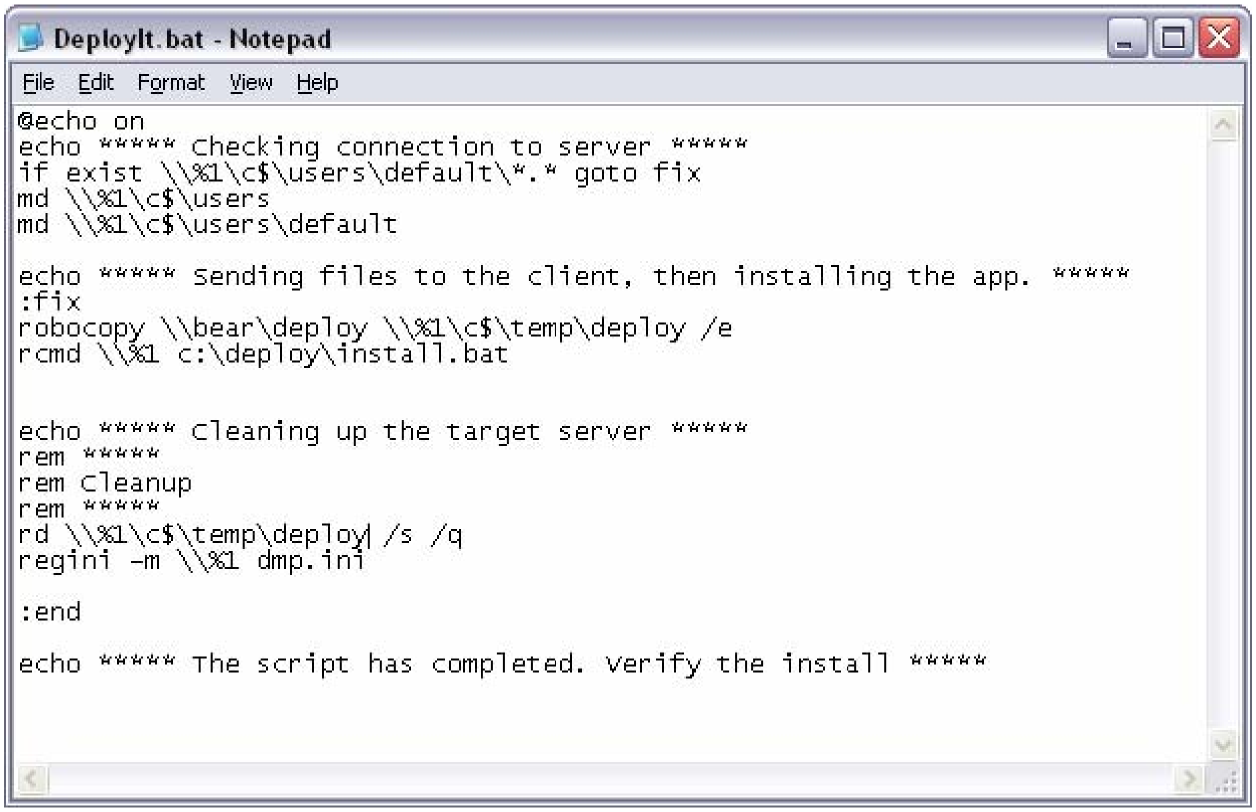
The scripting abilities inherit within the Windows OS can provide you with a simple deployment utility. Using the Browser data available on your network, a few IF statements and the resource kit utility RCMD.EXE, you can send out a simple spray script that will run a command on a target workstation in the security context of your user ID. I can already hear all the comments about how other utilities or methods are better. These comments might be true—this instance is shown as an example about how to script a simple deployment method.

Listing 4.1 shows a script with functions that are pretty basic. The resource kit Help file will give you the details about RCMD, but you should know that RCMD requires a service to be installed and running on the remote machine, and it is quirky in how it handles itself. Use it for simple things.

```
@echo off
net view /domain:%1 | findstr \\wrk > %1.txt
for /f "tokens=1" %%i in (%1.txt) do DeployIt.bat %%|
```

Listing 4.1: StartIt.bat

StartIt.bat, which Listing 4.1 shows, is a spray script that scans a domain and searches for any machine starting with \\WRK. The results are handed off to the batch file that Figure 4.1 shows.



```
@echo on
echo ***** Checking connection to server *****
if exist \\%1\c$\users\default\*.* goto fix
md \\%1\c$\users
md \\%1\c$\users\default

echo ***** Sending files to the client, then installing the app. *****
:fix
robocopy \\bear\deploy \\%1\c$\temp\deploy /e
rcmd \\%1 c:\deploy\install.bat


echo ***** Cleaning up the target server *****
rem *****
rem Cleanup
rem *****
rd \\%1\c$\temp\deploy /s /q
regini -m \\%1 dmp.ini

:end

echo ***** The script has completed. Verify the install *****
```

Figure 4.1: DeployIt, which is called by StartIt.bat from Listing 4.1, does the actual work.

Table 4.2 compares the pros and cons of scripting as a deployment method.

Pros of Scripts	Cons of Scripts
Highly customizable method of deployment.	Not easily transportable to other application deliveries.
Depending on the skill of the script creator, may be quick to create.	Which programming language is used is determined by the script creator and might not be supported by others.
Provides one method to reach workstations that do not have other deployment methods installed on them.	Maintenance of the script might be difficult depending on creator's documentation of the process and the complexity of the script.
Can utilize other utilities to perform functions.	Might depend on other utilities that might not be on the workstation.
The logon process can trigger actions for you.	If used at logon, the startup time could be expanded and might require a reboot (or two) before the user can start to work.

Table 4.2: Pros and Cons of Scripting.

Scripts Recommendation

Scripting has its place and should be used if needed. It can be used to enhance the deployment process, but should not be used in large-scale deployments of moderate-to-complex software applications. Keep the focus of scripting a deployment method to solving small issues or enhancing the more enhanced versions.

Email

Email deployments are pretty simple. Once the package is created, you either send out an email telling the users where to get the application or you send it to them as an executable type of file. Many of the newer deployment packages provide a link method for inclusion for using email as a means of sending the packaged application. Alternatively, you can use the manual email method, which is used as a quick and dirty means to get things out without using more enhanced methods. The following steps show one method of doing a manual email deployment:

1. The application is packaged and prepared as previously discussed.
2. A central location that the end users have access to is chosen as a distribution point for the package. The location can be a Web page or a Uniform Naming Convention (UNC) location (the *\\servername\sharename* format).
3. An email is sent to the targeted users with instructions about where to get the installation. Any configuration actions that the user needs to perform are detailed in the email.
4. The user connects to the detailed location, and either downloads or triggers the install.
5. The application is installed.

There are other variations to email deployments, depending on the inhouse programming skill in your organization. Some examples that work rather well include

- Using Microsoft Outlook, the deployment email can also trigger a Web page that does the install for the user automatically. With this method, the email notification acts just like a virus (gasp) to download the new application.
- An executable is sent in an email message to install an application. If you are rolling out a new deployment software client and have disconnected machines to reach, this method works great. By sending the deployment client executable to the user of the remote machine (such as a laptop), the user installs the client. There after, you can use the deployment software to reach the client. So you use one method to activate another method that is more reliable.

Sounds simple, right? Everyone is on email, so why not use it all the time? The deployment mechanism is already on the targeted machines. All that is needed is for the user to click on a UNC and run the install. So what is the big problem? The primary point of disruption in the deployment are the users. They are required to do something. Thus, the deployment success is at the whim of the users. If they have the time, understand the instructions, do not experience difficulty, and have the security context to run the install, they might do the install when they get the email.

With all the virus problems email clients have been experiencing in recent years, the message from the IT world has been how bad attachment executables are. We are constantly telling the users not to click on executables. Email and anti-virus programs are beginning to automatically quarantine attachments—sometimes the user never sees the attachment or the email. Some users haven't gotten the message yet, but most are following the safety procedures.

Now along comes a deployment from the IT shop with an email that says their attachment is OK and that the users need to execute it immediately. If the users follow the mantra that has been oft repeated in recent years, they will call the sender (you or me) and ask if we really sent the email and if the attachment is really safe. If only one user out of four contacts you, that is still a large number of calls. Figure maybe 10 to 20 percent of the users will just flush the email and not read it. Suddenly the distribution method is getting low on the success rate.

Success rate is determined by the number of users that get the application within a determined time period. If you are sending out a new company logo in .JPG format, getting it to all the users in the next week or two is acceptable. If you are updating a virus definition file during an emergency, nothing less than 100 percent of your targets need to have the file installed within the next hour.

Most email programs have a read-receipt option that sends a notice back to the sender when the email has been read. Short of doing a scan to every targeted machine, a read receipt is one method to confirm that the user got the distribution email. That is, if the user sends the read receipt back to you (Outlook XP gives you the option to not send it).

You could have the installation program send a notice back to you that it installed. Either to a tracking file somewhere or through an email to a collection email box. Either way, you need to do some comparing of the return statements and the target list to see where the holes are. For large deployments, confirming the distribution could take as long as installing the application takes. We'll cover distribution confirmation later in this chapter.

Now consider support: How do you perform repairs or fixes or updates to the deployed application? Most likely, you and the user will spend time (hours or days) on the phone working through any problems. Fixing a problem might require giving users Administrative access to areas you prefer to keep locked down or having them deal with system file issues without the source files. See the problems that can be generated? If you choose to use an email deployment method, very carefully weigh the support issues and costs and decide if they are worth it to you.

When is email distribution a good method to use? Some examples include

- If the end target is not on any other distribution method.
- If the distribution needs to take place in a small shop (and some large ones, sadly) in which the budget is not available to purchase a more robust distribution method and the users are geographically dispersed.
- If you need to send an installation to a small group of users. (Think of email distribution as an electronic Sneakernet.)

If you are rolling out new deployment software and need to install a client piece on workstations but have no other method of getting the client piece out there, email distributions might be an answer to your problem.

An Example Email Deployment

Let’s step through how to set up a simple email deployment. The requirements of the deployment include

- The application installation is considered optional for the user. The users can choose to install the application or not. This setup makes it easy on you when it comes to verification.
- The application is fairly small in size, so network-bandwidth utilization is not an issue, meaning the install can be done during the busiest part of the business day by multiple users without negatively impacting the performance of the network.
- The application does not require user interaction during the install (a silent install), and does not require customization afterward.
- The application can be installed without requiring elevated user authority. If your workstations are locked down and most of your users do not have Administrative access to the local machine (or the domain), you want to ensure any distributions you perform can be done under the restrictive access rights of the user.

You’ve packaged the application, tested it, and are satisfied that it can be deployed without issue using the email method. Let’s use the UNC method. Decide on a central location for the users to obtain the install executable—one that can handle the expected worst-case load and that everyone has access to. By worst case, assume that all the users that get the email will activate the install immediately in the middle of the busiest part of the business day.

If your environment has multiple domains, make sure someone in the other domain(s) are not restricted by trust relationships to the server you are using. If your domains do not trust each other and you don’t have a common access location, you will have to set up the distribution to be pulled from each domain, then detail the different locations in your email.

Create a directory and a share for the users to connect directly to. Remember to take into consideration any OS restrictions in the names you use. You want the share name to be short and simple without a lot of special characters. The directory name itself can be a little more descriptive, but don't get carried away.

Because this share will be accessed by everyone, the share permissions will be left at the default level of Everyone: Full. The directory permissions will be used to safely protect the application data. By leaving the share permission to Everyone: Full, you can ensure that all the users will be able to connect to the data. In addition, leaving the share permissions to Everyone: Full eliminates a possible problem: If this deployment was only going to a particular group, restricting the share to only that group in addition to using the directory-level permissions would help to keep others out.

To prevent accidental modifications of the data by anyone who doesn't have Administrative access, the directory will be set to Everyone: Read Only. This setting will allow the end user to grab the data without someone accidentally overwriting or deleting a file. Don't laugh, it happens.

Now it is time to craft the email for your users. The email should be simple, direct, and provide enough information to prevent a flood of support calls. Remember that most people like it simple, follow the KISS principle—Keep It Short and Simple. If the email is too detailed or wordy, the email might get stuffed into that "do-it-later" file and never get acted on. Put the details into a separate file or location so that those users who want that level of information can get it.

Because users have access rights to the directory, all they have to do is click the link in the email and they are sent to the location where the install begins to run. Table 4.3 shows the pros and cons of the email deployment method.

Pros of Email	**Cons of Email**
Simple to implement.	Depends on the user to activate the deployment and is unreliable for time-critical implementations.
Utilizes the existing email infrastructure in your environment.	Security concerns are high.
Can be used to deploy client software for a better deployment management system.	Attachments might be hidden by the email or virus-protection program. Outlook XP has a setting that hides potentially dangerous file extensions (.EXE, .BAT, .CMD, and so on) and might interfere with the user seeing the attachment.
Is an improved version of Sneakernet.	Tracking the install requires added programming.
	No native support within the deployment method. Requires additional utilities or human interaction.
	No network-bandwidth control options.
	Requires added utilities for management of the deployment (remote control, tracking, error corrections, and so on).
	Attachments might have mailbox size limitations.

Table 4.3: Pros and cons of the email deployment method.

Email Recommendation

Email deployments are one step above Sneakernet methods and inherit the same limitations. If you are faced with budget issues that prevent the purchase of a true deployment system, this method is better than Sneakernet, although only by a slim margin. Email deployment is versatile for clients that are not online all the time. With the virus concerns about attachments, notifying your users ahead of time will be critical.

> I've gone into some added detail in explaining the email method. Although the discussion fell into the email section, the same principle and issues apply to any of the manual deployment methods: scripting, email, and Web. There is a lot of work that needs to be done with each deployment and not all of the work can be easily transferred to future deployments.

Web

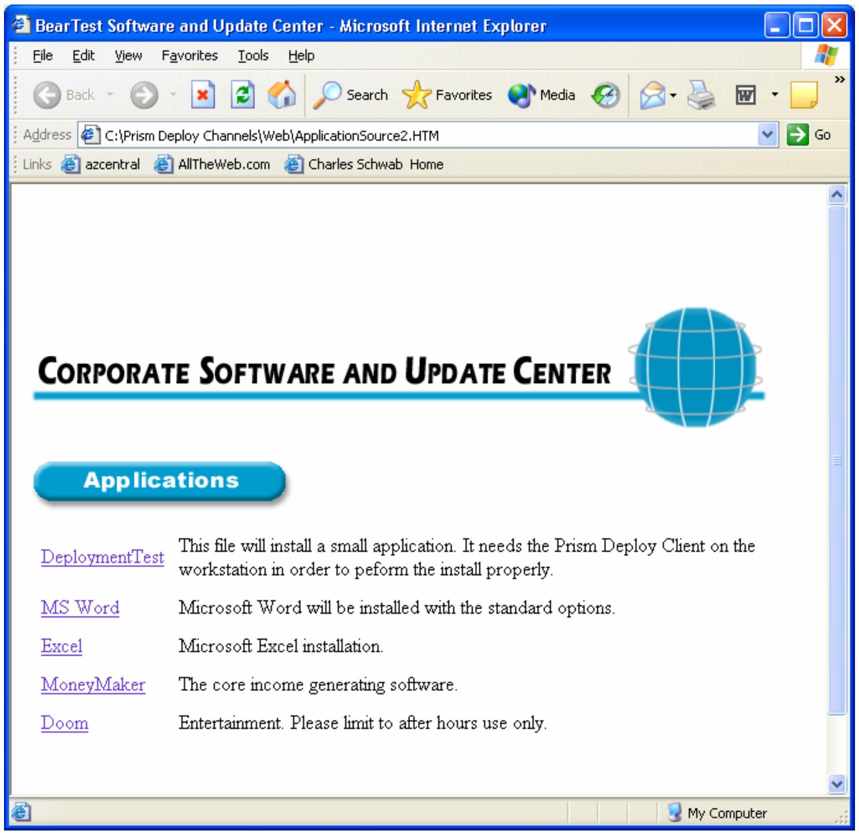
Another alternative to having a UNC central location is to post an application to a Web page and allow the users to obtain the applications from the HTML link. Using a Web page offers several advantages over a simple UNC location deployment:

- Using the browser and Web security features, you can provide added security to the application install. This security can ensure that only approved people install the applications.
- Applications can be displayed in groups. Arranging the displays according to business unit or functionality might make it easier for the user to locate the desired application if you have a large list to choose from.
- Installation auditing can be enhanced. You can track who is obtaining the applications, when they did so, and which computer and IP address they were sitting at.
- Additional logic can be applied to the installation process. For example, when the user clicks on the Web link, instead of just firing off the installation process, a Web screen could ask for the cost center to charge the licensing fee to and send an email to the manager of the cost center for approval. With the proper back-end programming, the Web-based process could even insert the accounting data into your accounting program automatically.
- Some level of network bandwidth control is provided by the browser and Web server. A direct file copy (or UNC trigger) does not allow for retries on poor connections. Most browsers will retry bad connections for missed packages and don't have the sending timeout problems that a direct UNC pull would experience.
- You can hide the true location of the files—not only to keep the knowledge secret (which you may want to do), but also to provide an easy method for moving the source files to another location if you need to. Changing the pointer on a Web link is easier than having to send out emails and notification to everyone explaining that the source location has moved.

- With proper security applied, the Web page can be made available outside of the corporate firewall so that users can grab the data from the Internet without having to become authenticated to the corporate infrastructure. If your user community cannot get to your corporate LAN, they could most likely get to the Internet and reach a Web address. This setup is obviously dangerous, so be sure about your security before you turn out the page to the outside world.
- Added text can be displayed. UNC locations are only directory and file names. A Web page can explain any company procedures associated with the install and provide a description of the application. If users need to do other configuration tasks, those can be shown at the same location. All this information can be provided in a nicely displayed page instead of relying on a README file the user may not see or open.

Using a Web deployment is similar to the email method we discussed earlier. A central Web location for installs will become quickly known and eventually be a part of the "common corporate knowledge." This knowledge makes it easier for you, as the users will automatically go to the central page for any installs they want. This setup translates to fewer calls to you.

Depending on your knowledge of Web design and security, you can get very fancy with the installation pages. Some front-end enhancements I've seen include conducting an automatic inventory on the workstation, confirming phone numbers and locations of users, sending confirmation emails, and updating a central database if the application needs it. Figure 4.2 was created using the Web Expert in Prism Deploy Editor.


BearTest Software and Update Center - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Prism Deploy Channels\Web\ApplicationSource2.HTM Go
Links azcentral AllTheWeb.com Charles Schwab Home
CORPORATE SOFTWARE AND UPDATE CENTER
Applications
My Computer

DeploymentTest	This file will install a small application. It needs the Prism Deploy Client on the workstation in order to peform the install properly.
MS Word	Microsoft Word will be installed with the standard options.
Excel	Microsoft Excel installation.
MoneyMaker	The core income generating software.
Doom	Entertainment. Please limit to after hours use only.

Figure 4.2: A sample Web page with links to application installs.

Table 4.4 compares the pros and cons of the Web deployment method.

Pros of Web	Cons of Web
A central location can be established that becomes part of the company "common knowledge" for where to grab software.	The installation normally requires some initial action by the end user. Thus, it is user dependent to complete the installation.
Software can be segmented by any categories you want: departments, software category, project needs, and so on.	Web programming knowledge is required.
Front-end scripts can be applied to the Web page to assist in tracking, license issues, and cost.	The success of the deployment depends not only on the skills of the application packager, but also on the skills of the Web page designer.
The install location could be opened to users outside of the firewall.	Anything exposed to the Internet is a potential security risk and must be monitored.
Many commercial deployment software solutions are beginning to provide a Web-based method.	
The Web browser or server can provide an added layer of security.	
Moving the install source location can be done behind the scenes without impacting users.	

Table 4.4: Pros and cons of using Web deployments.

Web Recommendation

Web-based deployments are a step up the evolutionary ladder. The controls, tracking, increased ease through deployment agents, and Internet access makes it a versatile method. If an existing intranet infrastructure is in place with internal Web sites, adding a deployment location keeps the costs low. Although not as full scoped as the more sophisticated deployment methods (which we will discuss shortly), this method is the better of the types we've discussed so far because of the increased flexibility and functionality of the Web front end.

Image Inclusion

Image inclusion deployment is a great method to ensure that everything on the target workstations will function without issues. For this discussion, *image* refers to the snapshot of a workstation that includes the OS, drivers, service packs, and applications needed to take a machine from out-of-the-box dumb to fully functional on your network. So *image inclusion* is the deployment method of taking a new application and adding it to the workstation build. Anyone that uses the updated image will have the new application on the target workstation. This method is destructive to any workstation the method is applied to. The overview of the steps for an image inclusion deployment includes:

1. Creating an image of a workstation that includes the OS and drivers.
2. Reformatting the hard drive on the target workstation.
3. Loading the new image.
4. Making the normal minor user adjustments.
5. Marking your install as complete.

Many IT shops use this type of method to some degree. All the applications work together properly, the workstation has the latest drivers and service packs, the user gets a freshly imaged machine without fragmentation of the hard drive, and you know the software level of the workstation. The process offers great benefits to the network administrator and the integrity of the infrastructure.

The actual process of the install is easy—not the creation or the processes involved in the deployment, but the install. Because of the pre-work involved in an image-inclusion method, deployments are normally bundled up to include several applications or updates, then rolled out at one time. Due to the effort involved in creating the image, the wipe and load, and the impact to the end user, such a deployment on an enterprise-wide scale is normally done only a few times a year.

An added advantage is the removal of any “non-approved” software that the end users may have loaded on their workstations. Because the image process will completely wipe out the hard drive and reinstall everything, any application the user may have loaded that could conflict with the approved software you are rolling out will be gone.

The negatives also need to be considered. The biggest drawback is the impact to the end users. Their machines are being cleaned out and new applications are being loaded. No matter how carefully you plan on saving the multitude of user customization parameters, the end user will have some parameters that aren’t saved or a critical document will be lost. Not to mention the potential hassle of learning new versions of their favorite software and learning to work with new applications, all at the same time.

Because adding a new application requires updating images and the roll-out process, application deployment is subsequently slow. Getting a new application out will take more coordination and work than any of the methods we’ve discussed so far. With all the effort involved, new installs or updates will be held until there are enough to justify the work.

One other issue you will need to address is support for the applications. How do you remotely fix, repair, or reinstall an application when the user manages to damage it beyond simple patches? Do you have to re-image the user’s workstation because most of Microsoft Excel was deleted by accident? Or is the Sneakernet method applied for the situation? How do you handle remote users? Each shop has to arrive at a solution to these questions. Many solve the problem by

- Creating a remote install for the software. This could mean there is double the work involved in getting an application ready for deployment because you have two methods to plan for.
- Using the Sneakernet fix.
- Sending the machine to another location for repair, if the machine is remote from the administrative support staff.
- Using a remote control product such as Netopia’s Timbuktu or Symantec’s pcAnywhere could be used to take control of the workstation and run the repairs.

Should you even consider image inclusion? Take the scenario of an OS upgrade that includes new versions of several applications along with one or two entirely new applications. Creating an image with the new OS, new drivers, all the applications, and any other customizations is a good method of getting everything out there at one time.

If your company installs (or re-installs) workstations regularly, image inclusion might be a good solution. Including the applications that everyone uses in the base image would ensure that newly imaged machines are setup to go. Assuming that you use a mixed deployment environment, you could utilize your standard deployment methods to install the rest of the applications. Deciding whether this method is feasible for your environment will depend on the flexibility of your standard deployment method. If you use Web-based deployments, for example, using a combined method might work depending on the mix of image inclusion and after-image installs and who has to click on the Web site to download the applications. If your standard methods include a pull-type deployment that automatically loads applications based on group membership, image inclusion might be overkill. As with all decisions, testing in your environment is needed to help you decide.

Recent years have seen advances in the process of creating and deploying images. Multi-cast, PXE, Symanctec's Norton Ghost, CD-ROM, and other methods are becoming common. Although the pain of deploying the image is easing, the end-user impact and the potential for lost data is still a major issue. Use imaging with caution. Table 4.5 compares the pros and cons of image inclusion.

Pros of Image Inclusion	**Cons of Image Inclusion**
Easy to get several applications rolled out at one time.	It is difficult to provide fix or repair support for individual applications unless another method for installing applications is available.
A known status of the workstations is in the environment.	Any customizations made by the user as well as critical files might be lost. Third-party imaging products are evolving to address this issue.
The potential for conflicts with unknown applications is removed because only approved software is included in the image.	The process is time consuming.
	Creating the deployment process, scheduling with the impacted users, and after-imaging support can be labor intensive.

Table 4.5: Pros and cons of image inclusion.

Image Inclusion Recommendation

Using image inclusion as a means of deploying software is primarily effective for providing a starting point for new workstations (or re-imaged ones). Using it as the sole method is costly, support restrictive, and slow. It is best used in combination with other methods.

Third-Party Solutions

OK, now we get into the meaty deployment methods. Remember that deployments consist of getting the package created, getting it out to both online and offline users, and ensuring that the package properly got to your targets. There are tools out there that give you more and more control on all aspects, giving you one-tool-for-all type functionality. Except for the image-inclusion method, using any other deployment methodology that we've discussed so far requires some or all of the following:

- Inclusion of utilities to do controlled transfers, data compression, or other actions.
- Batch scripting or programming to ensure the installation gets the proper data.
- A custom reporting method of some sort that will report on the success or failure of the installation.
- Custom front ends (such as a Web page).
- End-user participation to activate the process.

The beauty of most third-party tools is the ability of the product to provide these functions automatically without additional work on your part. This area is where the cost of the deployment software starts to pay off—the reduced labor costs in getting an application rolled out the first time, and the ability to reuse the same methods for subsequent deployments.

For the Web, scripting, and email methods we talked about so far, the labor cost does not end with the initial deployment. Your labor costs are still very high for each subsequent deployment. Although the infrastructure is in place, the deployment preparation and the manual requirements to trigger the installation are always there. Add the support issues each method incurs when attempting to resolve problems and you can see where the startup costs for the previous methods might be low but the upkeep and support costs are high. And remember that each of the previous methods require end-user participation to make it work.

Commercially available deployment packages are just the opposite. Their startup costs might not be the lowest when compared with the scripting method's startup costs. However, their subsequent costs drastically decline. The deployment software usually provides means for support while individual applications can be rolled out easily and quickly. Methods for dealing with online and offline machines are usually taken into consideration. The major cost savings come in the labor, support, speed, and reduced end-user interaction.

Let's get the major hurdle addressed right off the bat. Third-party tools cost more than any of the options we've discussed so far *in terms of startup*. Scripting and email deployment methods might not have any startup costs, as the required infrastructure is already in place in your company. The primary advantage to commercially available products is the cost savings they provide after the first deployment (if not during). Many third-party products can recover the startup costs of obtaining the software within a few mass deployments once the deployment mechanism is up and functional.

Some of the deployment systems are very complex and require support from vendors and consultants to be installed. While your staff is learning the ins and outs of the new deployment software, vendor consultants might be involved (onsite or online) to assist in the first few deployments. Of course, none of these support options are cheap. And consultants' presence doesn't guarantee that the first few rollouts will go smoothly. Remember that when you are deciding on purchasing software, you need to figure in all the labor costs of the alternative methods for each deployment, not just the setup.

☞ Go to http://www.appdeploy.com/ for the purchase price of many deployment applications. Use that data as an estimate; the actual costs for your company might be different.

Deployment software uses three main methods, push, pull, and a combination of the two. Let's explore each method.

Push

Push technology refers to the deployment process being instigated at the server and moving *to* the client. The client is just a receiver, and the server decides when the client gets the software. If most of your clients are online all the time, a push-type deployment will work for you, as the client will be available when the server sends out the instruction package.

Pull

The client instigates the deployment process and the server acts as a storage depot in the pull method. If your target machines are not always online, this method works as it allows users to gather the needed applications when they are online. If the client is not online when the server sends out the application, the client might not get it. If your clients move around a lot and need their applications to be installed at the new workstation, a pull method allows the client piece to obtain the software on demand and not require a manual trigger action from a server. This setup translates to easier support for you.

Combination

A combination deployment method uses the best of both worlds. It allows you to send out deployments from a central location and ensures that remote or moving clients get the application(s) when they come online.

Trying to determine which method a deployment software uses can be difficult. In all cases, the server portion has to define the package properties and make the data available to the clients. The client needs to determine from the server where to grab the instruction information. The instruction information tells the client what to do with the package: install, remove, update, repair, make it available to install, or make it mandatory to install. Check how the install is handled when the client is offline and which piece instigates the deployment to give you an indication of the method used. Additionally, deployment software can be split into two general categories: standalone and suite.

Standalone

Standalone deployment software does primarily that: deployment. It may or may not include other utilities such as inventory, programming interaction, or packaging functionality. Its primary function is getting the software application out there.

Suite

These are the big deployment packages. They include lots of other functionality in addition to deployment capabilities. As a result of the complexity of the suite type, their initial setup may be more involved than *standalone* types and may require a steeper learning curve to get a good understanding of all the bells and whistles.

Suite software is generally more expensive than standalone software and may require the addition of other hardware to support the infrastructure. If you want to deploy your applications and you are not interested in additional functionality such as inventory, there is no need to pay the additional costs of a suite.

Deployment software takes care of the basics that you had to deal with using any of the previously mentioned methods: Network bandwidth considerations, scheduling a deployment time, back outs (or roll backs), and insuring the application arrived are all dealt with by the product. Suddenly, your job just got easier! Every vendor has different thoughts about and designs for how to get your applications out. Most are GUI-based and have some sort of command-line options so that you can script the process. With a command-line trigger, you can use the deployment software with a Web-link trigger. By using the central Web page method we discussed earlier, a user clicks the application link and fire up a pull trigger that starts the deployment software running. Such a combination can give you the benefits of both methodologies: a Web front end to do any added functions and the standard deployment process so that you don't have to redesign what is all ready out there.

Deployment software needs to have some method of identifying the target workstations. A manually created machine list, a separate database entry, AD connectors, user group membership, or machine group membership are common methods of creating a target list for the deployment. The more the software interacts with your existing environment, the easier it will be for you to get your applications out.

Situations in which departments move around a lot and their machines stay can pose a problem for distributions. You will need to keep up on the changes somehow. With Win2K, you can use AD to keep track by moving machines into the proper resource group. Then your distributions could be machine-based to the resource group. If your distribution method allows dynamic distributions and is connected to AD, new machines will get the applications as they move into the resource group. When machines are moved out, the application could be removed or left active if you choose.

If you don't use AD or LDAP-type databases, a common method to identify target machines is to use a *machine.txt* file. The file contains the targets to be addressed in a format that the software wants. The deployment software reads the file into itself and uses that data as the UNC locators for the targets. Once the targets are identified, the deployment software can include the list in an internal database of some sort.

In much the same way, the deployment software should be able to read the contents of the NT account database or AD to some degree. This ability allows for some form of group-based subscription. Each vendor will have a specific method for how it obtains the data. Let's step through a user ID-based process. For this example, all members of the Accounts Receivable (AR) group will get Excel on their workstation (client):

1. You set up groups within NT for all domain members.
2. Within the deployment software, you define the application to send out and who it goes to. Anyone in the AR group will get Excel.
3. When a member of the group logs on, the client piece (workstation) of the distribution software contacts the controlling server and identifies the NT groups the client resides in. This process can happen from the client end (Userid: Joe; Groups: AR) or from the server. If from the server, the client would send up Userid: Joe, and the server would query the NT account database for what groups Joe resides in.
4. The controlling server then instructs the client piece where to find the Excel application.
5. The client workstation identifies that Excel is not loaded, then contacts the location specified by the controlling server. Excel is then loaded to the workstation.

As you see, the distribution piece finds the user groups and tells the client workstation where to obtain the needed application. This scenario is an example of the pull method because the installation is controlled by the client.

We'll deal with enterprise-wide distribution issues in Chapter 6. For now, assume that the deployment software has a method of obtaining the targets and uses that information to send out the applications.

Let's look at two third-party deployment software offerings: New Boundary Technologies' Prism Deploy and Marimba's Desktop/Mobile Management.

Prism Deploy

Prism Deploy is a standalone deployment system. Once the target workstation has loaded the client piece, the clients *subscribe* to *tasks* to obtain the software you want to deploy. When the client *subscribes*, the client begins to process the data as you have designed (update, remove, install, and so on). The application-installation package is termed a *task*. The combination of the deployment targets and tasks is called a *distribution channel* or *channel*.

Prism Deploy includes utilities for packaging as well as the deployment process. The packaging process was detailed in Chapter 3. With the Prism Deploy Console, you can effectively ship out your packaged applications to your targets. The true power of the product is the integration between the packaging, conflict checking, and deployment features. Before a client can participate in a channel, the client program needs to be installed. Prism Deploy Console can directly attach to any NT, Win2K, or Windows XP client that you have administrative access to and install the bits. For Win9x clients or machines that are not on the network, the install can be packaged into an executable file that will run on the local machine. That executable can be delivered through methods such as AD group membership, logon scripts, email, Web links, or Sneakernet. Prism's client installs the PrismXL Service, which has the ability to elevate install privileges so that you can automate deployments in locked down environments.

Once the client piece is loaded, the target machine will contact a server that you have specified as the distribution point and perform the action you have packaged into the task. Using a very efficient compression and delivery method, the makers of Prism Deploy have found that the need for a check point restart for even the largest files is not really necessary with their product. Checkpoint restarts help ensure a file continues from the interrupted point instead of starting at the beginning all over again. In New Boundary Technologies' tests, Office 2000 was downloaded over a 56k dialup line in approximately 17 minutes with their transfer method. Given that level of efficiency and compression with such a large application, normal tasks can be downloaded to the client within expected normal connection times. So the remote user that only connects to your corporate network a couple minutes a day to check email will get the tasks in that time frame.

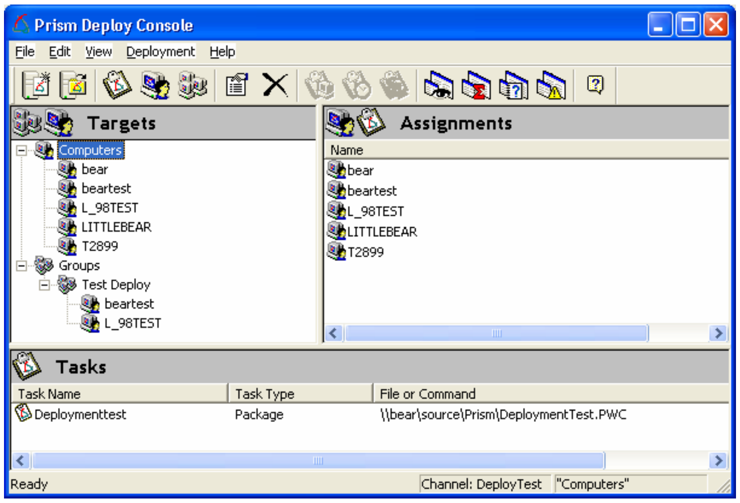
Installing Prism Deploy is a simple matter. Once you identify a system as the deployment server, insert the CD-ROM and follow the prompts. A few minutes later, the Prism Deploy Console is ready to go, and the wizards pop up to step you through the initial processes. No other infrastructure is needed to begin the deployment of your tasks. Once your package is created and defined as a task, you define the target computer or group and set the time for the installation. After the clients are installed, they contact the proper channel and activate the task.

Just how easy is the process? I was very skeptical to say the least. As a test, I asked an entry-level systems administrator to install the software and send out a simple batch file to my test computer. The batch file did a *net send* to the computer it was on that said "Hi." Nothing fancy or complicated in the package as the test was focused on the deployment software functionality. Less than 2 hours later, with only one request for a reboot on the targeted machine, a pop up message arrived. I was impressed. Using only the manuals and CD-ROM that comes with the application, the junior systems administrator was able to set up the infrastructure on his test system, create a Prism Deploy install file, create a task, add my test computer, remotely install the client on my test system, and distribute the batch file in less than 2 hours. Of course, that overall time would be extended depending on the complexity of the package to be installed and the customization that would be needed. As evidenced by the example of my junior administrator, the issues of costly integration and consulting that I talked about earlier, will most likely not be a consideration with Prism Deploy.

One principle with Prism Deploy is that it does not need to conduct any kind of inventory of the target systems prior to deployment. Because Prism Deploy can perform some prerequisite checking on target systems, including resolving hard-coded path statements and other variables, you don't need to determine what software is resident on the targets nor do you need to create a complicated backend database. Some other deployment solutions place this step into the setup process, which can add complexity and time to rollouts.

Administration is done from the Prism Deploy Console. It presents an overview of the channel with the computer targets, assignments, and tasks displayed in a concise method. The channel functions are available on the menu and the toolbar.

As you can see in Figure 4.3, the Target window shows the individual computers that were added to the channel, the group called Test Deploy, and the two group member computers (beartest and L_98TEST). A simple Add was done to insert the computers, using both the Direct and File methods of installing the clients. Creating the group was as simple as clicking the group icon (the icon of three computers). Populating the group was done by dragging the name of the computer to the group name.



```
Targets
⊟ Computers
    bear
    beartest
    L_98TEST
    LITTLEBEAR
    T2899
⊟ Groups
  ⊟ Test Deploy
      beartest
      L_98TEST
```

Assignments

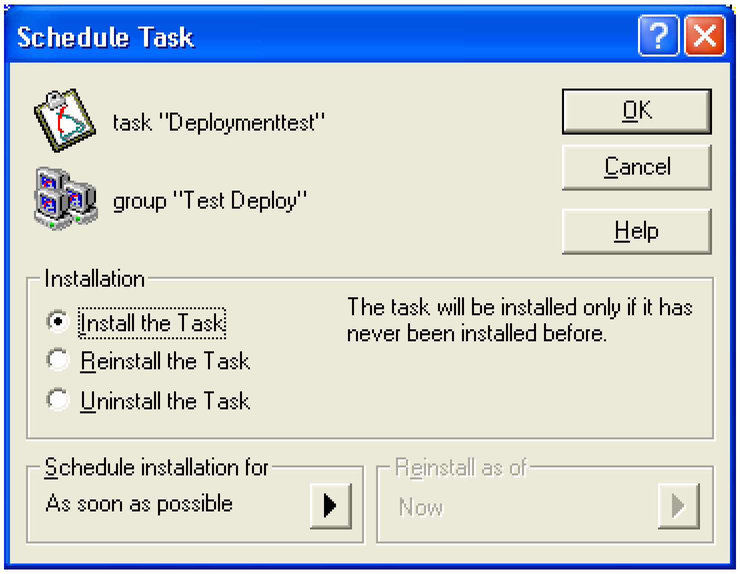
Name
bear
beartest
L_98TEST
LITTLEBEAR
T2899

Tasks

Task Name	Task Type	File or Command
Deploymenttest	Package	\\bear\source\Prism\DeploymentTest.PWC

Figure 4.3: A screenshot of the Prism Deploy Console showing populated Targets and Tasks.

A test deployment package had been previously created and was added to the Tasks list. Figure 4.4 shows the window in which the Task is created. From here, you can install, reinstall, or uninstall tasks. The default scheduling is to send it out ASAP, but you can adjust that as needed. After a task is scheduled, it appears in the Deploy Console (which Figure 4.3 shows).



task "Deploymenttest"

group "Test Deploy"

Installation

Install the Task [✓]

Reinstall the Task

Uninstall the Task

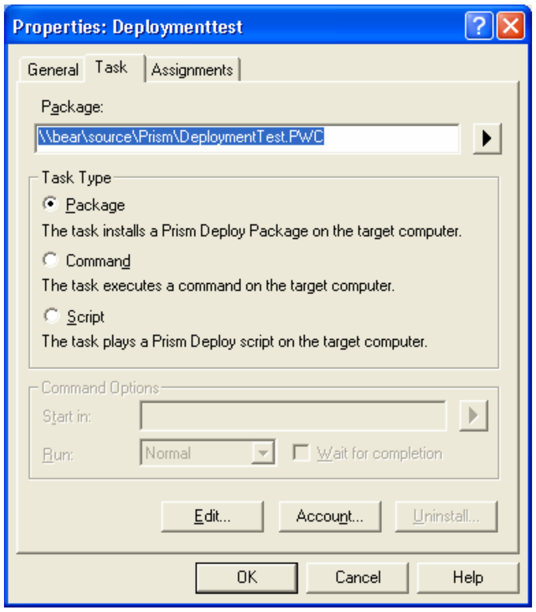
The task will be installed only if it has never been installed before.

Schedule installation for: As soon as possible

Reinstall as of: Now

Figure 4.4: Setting the properties for the Task.

After the task is defined, checking the properties from the Prism Deploy Console brings up the window that Figure 4.5 shows. This window gives you the opportunity to check the parameters and assignments. A Prism task represents a package (created with Prism's editor), a command, which can launch a program outside of Prism, or a Prism Script. A Prism Script can, for example, string together a series of Prism packages or be used to perform other checks on targets systems.



Package: \\bear\source\Prism\DeploymentTest.PWC

Task Type

Package [✓]

The task installs a Prism Deploy Package on the target computer.

Command

The task executes a command on the target computer.

Script

The task plays a Prism Deploy script on the target computer.

Command Options

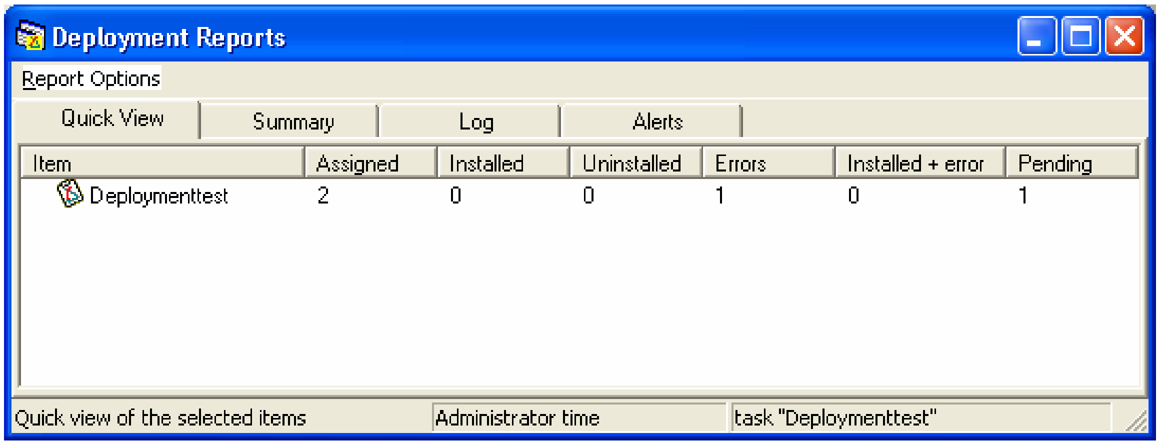
Start in:

Run: Normal

Wait for completion

Figure 4.5: The properties of a scheduled Task.

Getting a status report is a snap. You can access the Deployment Reports, which Figure 4.6 shows, from the console. Clicking the tabs presents a different view as needed. In the figure, the Quick View shows that the Task is assigned to two targets; there is one error and one pending deployment.

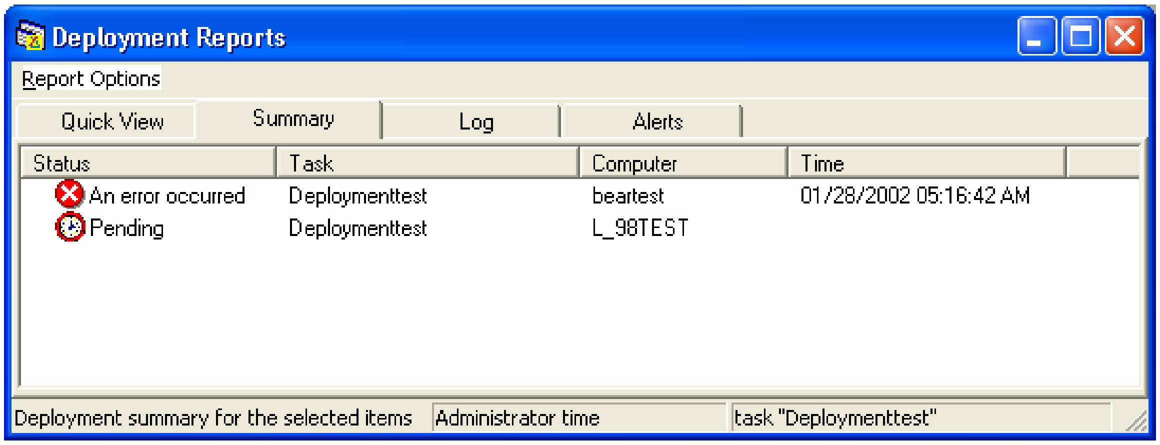


Item	Assigned	Installed	Uninstalled	Errors	Installed + error	Pending
Deploymenttest	2	0	0	1	0	1

Quick view of the selected items | Administrator time | task "Deploymenttest"

Figure 4.6: The Deployment Reports window.

Clicking the Summary tab shows a little more detail about the Task. Figure 4.7 shows beartest experienced an error and L_98TEST is pending the deployment. In just a couple of mouse clicks from a single console, the deployment status is available.



Status	Task	Computer	Time
An error occurred	Deploymenttest	beartest	01/28/2002 05:16:42 AM
Pending	Deploymenttest	L_98TEST	

Deployment summary for the selected items | Administrator time | task "Deploymenttest"

Figure 4.7: The Summary tab displays a little more detail about the deployment status.

Prism Deploy emphasizes ease of use without sacrificing power. With a few mouse clicks and a couple of drag and drops, a deployment was created and the status was available. In fact, for the test deployment, the entire process was done from a laptop running Windows XP and the source files were located on a Win2K server.

Prism Deploy does not require a complicated infrastructure—the clients can be offline or online and can obtain packages from HTTP, FTP, or UNC locations. This functionality combined with the product's ease of use make Prism Deploy a good standalone package.

You can download an evaluation copy of Prism Deploy at http://www.newboundaries.com/download/prismdeploy/_prismdeploy_down.htm.

Marimba Desktop Management

Now we step up the complexity a bit. Marimba Desktop/Mobile Management used to be called Marimba Castanet. If you come across any documentation referencing Marimba Castanet instead of Desktop/Mobile Management, they are talking about the same product, just different versions.

Marimba Desktop/Mobile Management is a pull-type deployment suite that uses a Java-based client. Applications are packaged as *channels*, copied to the *Master Transmitter,* which sends the data to *Repeater Transmitters* for delivery to the *tuners* on the clients. Inventory data is stored in an Oracle or Microsoft SQL Server database. Targets are defined by a machines.txt file, a connector to the NT account database, AD connectors, or LDAP inclusion. Administration is done through GUI-based utilities or command lines.

A *Master Transmitter* is the primary server. It connects to the database, provides all the administrative functions, houses the source files of the channels (applications), and is the connection point for the Windows or directory-services connectors. The Master Transmitter can service the clients but is happiest letting a Repeater do that duty.

A *Repeater Transmitter* or simply *Repeater* is a mirror of the Master Transmitter and acts as a distribution point. The Repeater hands all decisions and data up to the Master Transmitter for processing and only services the clients after the clients contact them.

Tuners are the client piece that is installed on the targets. The tuner provides the interface to the service on the client and handles the channels depending on the individual specifications of the channel. Every client that is part of the Desktop/Mobile Management complex needs a tuner or it cannot participate. The tuner can be installed from an executable file that is delivered through any normal delivery method: Group Policy Object (GPO), CD-ROM, Sneakernet, Web link, or email.

Channels are the packaged application and instruction file. A channel is HTTP-based and will be referenced by the URL name of the Master Transmitter. If, for example, your channel sends out WinZip from the Utilities directory and the Master Transmitter is called TopServer, the URL will look like http://TopServer:5282/Utilities/WinZip. Because the client tuners will get the data from a Repeater, instead of having the URL change for each Repeater, the same URL will apply. The Repeaters don't impact the URL name and will act as a mirror of the Master Transmitter for the channels. This makes it easier to handle.

Marimba Desktop/Mobile Management is HTTP based so NT security does not play a large a role in deployments. Users can go across domains that they normally would not have access to due to the HTTP capabilities. Notice in the URL in the previous paragraph that a port number (5282) is used. As long as the proper ports are open to the client across any network firewalls, the client can get the channel. Marimba Desktop/Mobile Management incorporates HTTP security and some security of its own to ensure only proper access is granted.

One piece of trivia. The standard port number used by Marimba Desktop/Mobile Management is 5282 to access the channels. If you look at a phone keypad, 5282 spells out JAVA.

Marimba Desktop/Mobile Management uses checksum verification for file transfers. When you package a channel, you always package a full channel as if it had never been sent out. The clients that will only be getting an update to the channel get only the checksum differences not the entire file. So a fresh install might get 10MB and an update might get only 1MB. With the checkpoint restart, if a file download fails at the 8.99MB mark of a10MB file, when a Tuner reconnects, it picks up at the 8.99MB point and continues instead of starting all over again. This feature is helpful for users that are only online for a short time each day on a slow link. It might take several days, but they eventually would receive the entire channel. Only after the channel is fully downloaded and the checksum verified is it installed.

Being a suite, other utilities are offered in the software. Inventory, packaging, and subscription are some of the offered modules within the product. Each module requires a license to function. Administration is done from an Administrator Tuner that connects to each client.

You can get a demo CD-ROM for Marimba Desktop/Mobile Management by going to http://www.marimba.com/products/change_management/literature-product_demos.html

Marimba Desktop/Mobile Management requires resources to properly set up the infrastructure. The pull product works well for remote users and the Repeater infrastructure is good for a large number of users in a dispersed environment. Evaluate it carefully to determine whether the product will fit your needs. Table 4.6 compares the pros and cons of the third-party tool deployment method.

Pros of Third-Party Tools	**Cons of Third-Party Tools**
Automates many of the details of delivery that were manually created in other deployment methods (such as scripting, Web, or email).	More expensive than email- or Web-based methods. Start up costs may be outside the reach of your budget.
Utilities within the deployment tool can ease your packaging and verification woes.	The quality of support is vendor specific.
The vendor is there to support you if you experience problems or have questions.	The deployment software might be strong in one area and weak in another. This imbalance might force you to supplement the software with added utilities or create your own.
The startup costs are quickly recouped on each delivery when compared with manual deployment methods that require designing the process each time a deployment is setup.	There is a learning curve involved in learning the new software.
	There might be a delay in vendor support to changes in an OS, such as service pack updates or new Windows revisions.
	The implementation time frame might take several months before it is fully functional. The requirements for extensive vendor support and consulting needs to be reviewed.

Table 4.6: Pros and cons of third-party solutions.

Third-Party Tools Recommendation

Overall, third-party deployment software is highly effective. The days of scripting all your own deployments is quickly fading as the advantages that are packaged into the commercially available deployment methods are far better and are constantly improving. Whether third-party deployment software is better than what Microsoft can offer is a decision that you will need to evaluate. I strongly recommend that you switch from the manual methods and into the proper vendor-supported method.

Microsoft

For those folks that don't like to use third-party applications for major systems functions, you can always use the Microsoft solutions: SMS or IntelliMirror. With the exception of SMS, all the other Microsoft options come on the Win2K Server CD-ROMs.

✎ There are other ways to create MSI packages other than using Microsoft utilities. VERITAS' WinInstall LE is just one third-party tool that works well to create the MSI-needed format.

SMS

SMS was one of the early suite utilities that tried to do it all. It has gone through several revisions with the current release at SMS 2.0. A newer version code named "Topaz" is expected in early to mid 2002.

SMS 2.0 is primarily a combination technology that utilizes a central server (a primary site server) and secondary site servers as distribution points for the clients. The primary site server is the SMS equivalent to a Primary Domain Controller (PDC). It houses the connections to the SQL Server database, ensures the SMS domain is intact, handles communications between the secondary site servers and the SQL Server database, and is the communication focus location for all the SMS administrative tools.

The secondary site server acts as the client access point (CAP) and the distribution point for the packages. The clients contact a secondary site server for the SMS information they need, and if the secondary site server is configured as the distribution point, the clients will obtain the packages from it. A primary site server can act as a secondary site server but a secondary site server can NOT act as the primary site server (due to the SQL Server connection issues). The secondary site server can act as the CAP, distribution point, and logon server. Or these services can be offloaded to other servers.

In previous versions of SMS, the site servers needed to be on domain controllers. At the time, this method ensured the SMS servers obtained the proper data for authentication needs. SMS allows the ability to relocate many of the site server functions to other non domain-controller servers to help ease the workload, but most installations kept the services all on one site server, at least initially. You can guess the problems that would develop: the domain controller's became overloaded very quickly with all the SMS functions, the NT domain duties, and possibly home and profile duties. SMS 2.0 site servers can be installed on any type of server: PDC, BDC, or member server (sometimes called a standalone server). This provides much needed flexibility.

There are several tools within the SMS suite: Inventory, software metering, remote control, and network diagnostic tools. None of the tools are second rate, often approaching or exceeding the level of standalone quality tools. The Network tool is one of the better non-dedicated diagnostics tools, providing great sniffer capabilities. The Inventory utility is detailed—providing system information as a list of executables it finds on the target machine—and keeps this information up to date automatically. The discovery of the executables is dynamic without the need for a central rules file to compare against. The display of the software titles is by company, making it easy to locate unauthorized software.

All the SMS data resides on SQL Server 6.5 Service Pack 4 (SP4—or later), either on the same server or on another server. There are always discussions about which method is better: Move SQL Server to a separate server or keep it resident on the SMS primary site server. Depending on your environment, budget, and the size of the server (and memory) you could go either way. Real world experience shows that several thousand clients and secondary site servers can be part of an SMS domain in which the primary site server houses a combined SMS and SQL environment.

The canned reports and functions within the SMS/SQL complex are OK to get started with. Your skills with SQL can be used to create added reports through Crystal Reports. Microsoft stresses that you should not do any modifications to the SQL data without using the SMS API methods; otherwise the company might not be able to assist you with database problems.

SMS works, as you would expect, with the Win2K infrastructure. SMS lets you advertise applications, works with MSI, comes with an aptly named installer called SMS Installer, and has the ability to elevate the install privileges in case the user does not have the authority level needed. The SMS Installer falls into the category of scripting tools discussed earlier. The client piece is a 32-bit application that gives you the ability to do mandatory installs or set up the application as an optional component for the end user to decide on. SMS also provides functions such as inventory and software metering.

With all that SMS gives you, it is a complex solution. This is not a product that you can install and get running fully in a few hours. Aside from the setup issues of the server infrastructure and learning the packaging requirements, understanding how to trace the flow will keep you busy for awhile.

When you trigger a job (a package) for deployment, SMS converts the instructions into different file types and moves the data from one directory to another, often changing the names of the files and the file types in the process. These actions are all done on the primary site server. Once the package is sent to the secondary site servers, the same process happens while the package instructions and data are moved to the proper directories. The clients check into special files to obtain any instructions they need to act on, then go to other directories (or possibly other servers) to obtain the data. Remember that the packages are all compressed and need to be uncompressed at each destination, so you need to allocate room for the decompression. SMS is poll-based for each event, meaning that it triggers actions according to a clock tick. SMS 2.0 has improved greatly the time delays from the pre-2.0 days, but it is not an immediate trigger, and you still have to wait for the clock to tick. Those clock delays are variable depending on how you customize the SMS infrastructure. Remember to take them into consideration when you plan deployments.

Understanding the job flow, data conversion, and possible break points are where SMS expertise is needed. Anyone can set up a packaged job for deployment using the MMC interface. Getting the package set up properly and being able to troubleshoot where problems may have occurred are the true tests of an SMS administrator's skill.

SMS 2.0 has received mixed reviews—administrators either hate it or love it. Talk to anyone that has used any version of SMS and there will be horror stories that can easily turn you off the product. In all fairness, listen carefully to the stories. Is the core issue truly the fault of SMS or the fault of the package, end users, or any other number of circumstances outside the control of SMS? As with any software tool, SMS has issues and problems. Just keep an open mind about what you hear.

There are a number of Web sites and news groups that talk about SMS. As a starting point, try http://www.microsoft.com/smsmgmt/default.asp

http://appdeploy.com/tools/sms2

http://www.myitforum.com

On your newsgroup server, check for microsoft.public.sms.* for several different groups.

If you are considering SMS, download the SMS 2.0 Reviewer's Guide at http://www.microsoft.com/smsmgmt/exec/revguide.asp. Microsoft has a nice online demo that gives a quick overview, including how to activate a job for distribution at http://www.microsoft.com/windows2000/demos/mod03.htm.

Table 4.7 compares the pros and cons of SMS.

Pros of SMS	Cons of SMS
Created by Microsoft, support issues between the OS and SMS are easier to address.	Complex and potentially labor intensive to administer and support.
Several utilities come with the suite.	The expense of the startup is not only for the software but the hardware needed for the site servers.
Supports most common packaging formats.	

Table 4.7: Pros and cons of SMS.

IntelliMirror

In a Win2K environment, you have the option to use IntelliMirror for deployments. IntelliMirror provides three primary functions:

- User data management
- Software deployment
- User settings management

Although all three functions are important, the area we are interested in is software deployment. The main premise behind IntelliMirror is the ability to have applications follow a user around to different desktops. Although MSI is the preferred format of the installer files, normal SETUP.EXE-type installs will work under most Group Policy distributions. With IntelliMirror, applications are either *published* or *assigned*.

A *published* application is available to the user to install. It is not automatically installed when the user signs on. If the application is needed, the user goes to the Add/Remove Programs applet in Control Panel and clicks the name of the application. If a file is opened that needs a specific published application, a form of installation called *document invocation* takes over and installs the application. Getting the published data to the user requires you to set up a GPO. To do so, you must have AD and Win2K or Windows XP rolled out to all computers and servers.

An *assigned* application is available to the user in the Start menu, but is not installed. It is merely an icon for the application. After the icon is clicked, it activates a background service called Windows Installer that processes the request and performs the MSI installation. Although the overall process sounds simple, there is some background work that needs to be done to enable the features properly. After the initial design work is done, setting up future deployments through IntelliMirror is easier. Table 4.8 shows the pros and cons of using IntelliMirror for deployments.

Go to the Microsoft Web page and download the IntelliMirror scenarios MSI file. It contains six scenarios and a good white paper that details exactly how to set things up, plus it provides batch files to assist in setting up GPOs on your domain so that you can test the processes outlined.

Pros of IntelliMirror deployments	Cons of IntelliMirror deployments
Deployments are principally done by Group Policies.	The "just in time" method of install can cause a delay to the user that may be unacceptable depending on the situation, line speed, and hardware platforms.
Applications have the ability to follow the user to different workstations.	Requires the Windows Installer service to be running.
Repairs are available if properly setup through MSI installations.	Requires that you roll out Win2K or Windows XP to all workstations and servers and that AD be rolled out to all users.
	Offers no reporting mechanism to ensure deployments were successful.

Table 4.8: Pros and cons of IntelliMirror deployments.

As you would expect, Microsoft has a lot of data available on its Web page regarding IntelliMirror (http://www.microsoft.com/windows2000/techinfo/howitworks/management/intellimirror.asp).

Microsoft Recommendation

Microsoft products are well known. The primary advantages with using them are the close tie-ins with the OS and the large number of other administrators using the products. The Microsoft label does not mean that the products are better or worse than others. SMS and IntelliMirror work. Will the costs of the startup and the functionality they provide be the proper choice for your environment? You will need to evaluate and compare against the other offerings.

Rollouts

You've packaged, tested, decided on your deployment method, and are ready to roll out the package. Now it is time to decide on the rollout process. Will you phase it in controlled steps or mass deploy and trust that your package won't cause issues? I recommend that if the application allows it, always go with a phased approach until you are comfortable that everything is right.

Being naturally paranoid of causing wide-scale problems that will keep me slaving over a hot keyboard for hours (or days), I'd rather discover that an application causes an unexpected problem with a small group of machines or users instead of the entire company. Something about having my name prefaced with negative sounding adjectives by everyone in the company does not really appeal to my sense of well being. However, if a dozen or so users are complaining about an application blowing up, well, those are the breaks of being on the "cutting edge" of applications within the company. Surprisingly, I've gotten people to volunteer by making it seem like a special perk to get an application first. It tickles the ego to get something before anyone else does.

Next time you're with a group of techies that deal with software distribution, stir up the conversation a little by saying something like "I believe the ONLY way to rollout applications is to use the phased method." Or claim the mass deployment method is better. Then watch the discussion heat up. The point is that there is no one single way to do it in all cases. Table 4.9 provides discussion points for the different perspectives.

Phased Approach	Mass Deployment
Takes longer to roll out applications.	The roll out is completed quickly.
There is more overhead in tracking who has an application and who does not.	Everyone gets an application at the same time so there is no added deployment tracking.
Errors, unknown interactions, and problems can be found and resolved before they impact numerous systems.	If any negative issues are discovered, they impact the entire deployment group—a serious situation if the problem prevents the company from making money.
Post deployment clean up is normally smaller by the time the final phases are run, as any clean up processes are discovered during the smaller phases.	Post deployment can be labor and time intensive, depending on the severity and degree of clean up due to the size of the roll out and learning curve.
A roll out can be halted at anytime, stopping the application from reaching everyone.	Once you pull the trigger, the application is going out and may be difficult to halt or roll back.
Support calls are manageable due to smaller volume.	Support calls may inundate your Help desk due to the magnitude of the rollout.
For simple rollouts, a phased approach is usually not needed.	For simple rollouts, a mass deployment might be the most effective method.
Some applications may have tie-ins with other processes or back-end systems that prevent a phased approach from working.	A mass deployment may be the only option if the back-end processes or systems can't be segmented.

Table 4.9: Discussion points for using a phased approach or a mass deployment method.

For mass deployments, scheduling the rollout is easy. Everyone gets the application by your deployment method at the same time. You don't need to deal with which groups are getting it and when; you follow your normal deployment processes and send the application out. Are you shipping out a new True Type font package? Mass deployment would be the method of choice to ease your burden and get the package out to the enterprise. A phased approach requires that you take several steps to roll out the application. How many steps are needed depends on:

- The complexity of the package.
- The number of targets in the rollout.

- Any added processes, configurations, or back-end modifications that must be made after the rollout has reached the targets.
- Your comfort with the rollout.

For most rollouts, using a three-phased approach works effectively. The first phase is sometimes called the *pilot* group because it is small, maybe 5 percent of the total target distribution. The second phase is a little larger, maybe 10 to 25 percent of the total. And the third phase would be the remainder of the rollout targets, assuming you are comfortable and have not found problems. The number of phases is adjustable. For the majority of your rollouts, using three phases will work. For complex deployments such as rolling out Office XP, increasing the phases to have a smaller number of users in each phase would be best. Adjust the number of phases as you see fit for the situation.

I keep mentioning the phrase "if you are comfortable." I'm referring to the level of success in the smaller rollouts and your impressions of how that rate will translate to larger rollouts. It is an educated guess on your part in assuring the impact to the end users will be positive. The users may be severely impacted as in the case of a service pack update or new word processing package, but they should not experience a failed system to the level that a reinstall of the OS is the only resolution. If your smaller phases are successful and you forecast the same level of success in the wider scale, your comfort level is high. If you don't like the package, have to work late off hours and weekends to apply back-end processes and need to cancel your long-planned vacation to ensure the critical rollout will work from the end users point of view, your comfort level is less than comfortable. Sadly, personal comfort does not come into play in the software distribution world. If the same conditions apply but the package fails in the initial phases, your comfort level is low, and the deployment needs to be questioned.

Pilot Users

The first phase of a rollout is sometimes termed a *pilot* phase. The first phase is a small group and is used to confirm that all your testing, planning, and designs are going to function as you wanted. There are surprises at times, no matter how hard you try to prevent it. For this reason, it is good to select your pilot users carefully.

The pilot group should be representative of the eventual rollout group. Don't rollout to your other network administrators that always sign on with their administrator accounts if the application is going to the mailroom that has restricted access rights on their accounts. You won't find the problems until they hit the mailroom.

Talk to the pilot users beforehand. They may or may not have volunteered for the phase, but they should still be made aware of what is happening. If you can, a face-to-face meeting or phone call to the group is a nice touch and goes a long way toward ensuring you get good information back. Let the pilot group know exactly what they will get and when, what to expect, what *might* happen if it goes wrong, what you expect them to do, and what to do if they see something strange.

When I rollout applications, I see the act of talking with that pilot group as a critical step for the success of the rollout. If the application is eventually going out to a large group, one small ego stroke you can give the pilot users is knowing that they not only get the application before anyone else but that their input is critical in helping to confirm the application should go out as planned. Letting them know how important their feedback is will get you that needed data.

Giving the pilot users the chance to become an expert on a new application before anyone else sees it is usually payment enough for dealing with any possible issues. People like to feel important and feel that they know a little more about something than anyone else does. Play up to that desire a little and it will make your deployments that much better.

There have been instances in the past in which a poorly created piece of software was rolled out to the pilot users and their input adjusted the rollout. In some cases, it cancelled the project. More commonly, it sent the project back for adjustments so that the software worked in the real world instead of the software designers' development lab. When it happens, make sure the pilot users know that their input had an impact. It will keep them on your side and wanting to do more.

Errors Are Found—Do You Continue to Roll Out?

Your pilot phase found some errors that were unexpected. As the deployment administrator, you must decide if the rollout is to continue or be delayed. I can't provide a solid answer to this question for you; there are too many variables (from political to technical) to deal with. I can offer up some points for you to ponder:

- How damaging to the system is the error?
- Is the error in the packaging, deployment, or software?
- What impact would the user or system experience if the error were not fixed right away?
- How hot is the project for the company? (What kind of pressure is there to get it rolled out?)
- If a delay in the rollout is needed, what are the procedures to follow in your company? Do you have the authority to call a halt?

If the error is going to severely impact the system(s) or negatively impact the users in completing their job duties, the only real option would be to halt the rollout until the error can be repaired or patched.

Ensuring the Deployment Worked

Getting the software out is one thing. Ensuring it reached the workstation targets and installed properly is another. Most commercially available deployment software has some form of delivery confirmation. It can tell you whether an application was delivered to the targets. Ensuring that the application installed properly is a little more difficult.

In an ideal world, every time an application was installed on a workstation, all the functions and features of the application would be tested to their fullest and confirmed that they work with all the other applications on the workstation. Obviously this scenario is not an option in the world that we are dealing in. In our hurry to get it done, we often forget, or opt to ignore, the need to confirm the deployment went as we desired. Instead we rely on the testing, which is why it is done, to find the road bumps. A lot of times the classic logic is used: If the phone is not ringing, the deployment went well.

In all deployments, you will find a small percentage (less than 1 percent) of errors that can only be assigned to the category labeled *what the heck?* Some simple steps can be used to reduce that number even further by proactively reviewing the deployment.

Review the rollout status in your deployment software. Investigate any targets that don't show a good completion status. Remember that depending on the software, you may be seeing the delivery status, not the status of the installation. If the installation failed to update the registry properly, the delivery status might show as a success because the install process completed and returned control back to the deployment software. The user sees the application as not functional, but your delivery mechanism shows it as good. You may find that some machines are constantly coming up with failures. These systems are good candidates for some proactive investigation to determine why they are having problems.

Prism Deploy deployment reports include both deployment status *and* errors such as *incorrect OS, user cancelled the operation*, and *error writing to the registry*. These reports indicate that an installation had a problem.

If your deployment software has an inventory option, use it to confirm the deployment. Trigger it to gather data on selected files that are critical to the application. Inventory the primary executable, the existence of a directory, a particular registry entry, or a flag file that you set.

You could also make use of the temp directory on the workstations. Create a C:\TEMP\Logs directory and set the applications to echo their install logs to the location. Have your installation process create a flag file that is only present if the installation went to the end. The flag file can be empty or have a short message or the date. It does not matter what is in the flag file, it needs to exist only so that your inventory modules can note the existence of it. The same function could be done by creating a registry entry.

Once your inventory module has gathered the data you are looking for, run a report to see which machines have the reference points you set. If you know what the targets were, a simple compare should show any that are missing.

Some Tools to Confirm the Deployment Went Out

Aside from using your deployment software and the tools that come with it, you can do some simple scripting with free utilities to determine whether the deployment reached its targets. Here are a few recommendations:

- Nearly any scripting language will provide some means for checking files and getting some data from the registry. Languages such as Perl, C++, and VB6 provide various methods. Choose the language you are most familiar with and use the hooks available.
- The Microsoft resource kits have some nifty utilities such as REGDMP, SCANREG, REGFIND, ROBOCOPY, and WINDIFF. Use these utilities to compare file dates, registry entries, or the existence of files. Check the Help screen that comes with each one to see the format.
- The Windows OSs contain different built-in utilities such as REG, DIR, IF statements (if exist \\machinename\c$\ntdetect.com set test=true), and the like. If you are not familiar with the choices, from a CMD prompt, enter HELP.

The Human Error Factor

Many deployment issues can usually be attributed to some kind of human error, but the problem is reported back as "the deployment software caused my system to blue screen." Issues with the package creation, damaged workstations, improper user interactions, machines turned off, or interaction with unknown software on the workstation are very common. Deployment software cannot resolve all those issues, no matter how good the software is. The deployment application depends on the intelligence of the administrator and the packager to resolve issues.

As an example, one of my favorite end-user complaints is the user that claims the deployment software rebooted his or her workstation in the middle of the day. When you research it, you find that a major application was rolled out that required a reboot. For example, a service pack update was sent out on Friday night with the intention that it would install over the weekend when no one was around. You get the complaint the next Thursday and find the user had the reboot on Monday. Checking into the logs you discover that the user had turned the machine off on Friday even after repeated emails telling everyone to leave their computers on. So, yes, the deployment software did install a piece of software that caused the reboot on Monday. Of course the issue of the user turning the machine off on Friday doesn't surface in the complaints. The error is the fault of the deployment software and it starts to get a bad name. The first statements that upper management folks hear are the original complaints.

It is also common for an error, or perceived error, to give the deployment team a bad name causing them to be skeptical of every system problem that follows. Taking care early on and avoiding this kind of reputation is well worth the extra time. True, if the application was packaged to check for delivery times and only run during certain hours, the previous scenario could be avoided. The point is that the problem is not the fault of the delivery mechanism.

Software deployment is not 100 percent about the technical aspects. You need to talk to your managers and make sure that they know what is happening and that the word gets sent up the ladder.

Remember that any deployment software is just that—a way to deploy software. Its primary job is to get the packaged application to a destination and trigger the installation. It is NOT a system that can correct faults or problems with applications, installs, or human errors in deployment targets. Never accept at face value the excuse that problems on a target machine were caused by the deployment software.

Summary

There are as many methods for deploying software as there are target environments. Once you find a solution that fits your budget, client needs, infrastructure, and corporate culture, you should set it as the standard. Any applications coming in should be packaged with the idea that your standard method will be the mechanism for distribution. But stay flexible. Blindly following a process without looking at the desired results can be burdensome. Maybe adding a central Web page for optional installs that tie in to your standard process is the best method for your organization. Keep an open mind and look for the opportunity to improve your methods; it will make your life easier. In Chapter 5, the issue of deploying to the remote user will be discussed.

Chapter 5 Deploying to Remote Users

The thought of getting a deployment out to a remote user can bring even a strong network administrator to the edge of a panic attack—especially for that first company-wide deployment when the project has a white hot spotlight of attention on it. However, this task does not need to be a difficult endeavor. Just like the slogan says No Fear—you've already got the information and as an administrator, you've got the attitude, all you need is the experience. For a remote administration, the principles that we've talked about in previous chapters still apply:

- A package is delivered to the target machine
- The install process is started

Physical distance is not a large issue. A target machine can be sitting next to you or half a world away, and the deployment method is basically the same. That is, unless you try to Sneakernet the install. I'm sure the company travel department and your finance administrator will have something to say about all the travel.

To clarify for the purpose of this chapter, the term *remote user* will be any user or workstation that is not within the immediate vicinity of the network administrator. The user in the next building is considered a remote user because the need to send applications to the user has the same hurdles as a user in a building 2000 miles away. Where the fun begins is dealing with machines, such as laptops, that move often or are not directly attached to the corporate LAN/WAN. Deployments to those systems are where this chapter is heading. As the laptop is on the extreme side of the remote user scale, I'll use them as examples in this chapter. After you get the laptop deployment issues resolved, the rest of the machines will be a snap. Grab a box of your favorite chocolate chip cookies, a couple of caffeinated beverages, and read on.

Generic Remote Issues

Focusing on the laptop issue (remember they are used as the extreme example), there are a couple of generic issues you must account for:

- Line speed
- Connectivity duration
- Windows security (domains, ACLs, and the like)

When a laptop user is on the physical premises for a full day and connects to the hardwired LAN, the laptop is running at your normal LAN speed and can be considered a local connection. No problems at this point. However, laptops spell mobility to the user, and these systems can also connect to your network at other speeds. Connections can range from a phone line running on a 56Kbps (or slower) modem to a DSL line running at 256Kbps+. Often, laptop users will connect with various methods in a single day. Take my own laptop connections as an example. Depending on where I am, the connection styles will vary greatly. It is not uncommon for my connection day to cycle from a Gigabit Ethernet hardwire to a 56Kbps dial up in a single day. Direct connect, VPN by way of the Internet, password access, and token security access are methods added to the mix. If your deployment method uses fixed network timing as a means of triggering an event, you can see how identifying the proper connection timing would be difficult.

☞ If your deployment method has the ability to disconnect users after a fixed clock time, be sure to set the time parameter long enough to support the remote user who is dialing in. This setting will prevent premature cancellation.

The following scenario is common: There is a requirement to get an application out to all your machines ASAP if not sooner. Of course, the application is not simple, and the package is a 15MB file. Dealing with the remote user who only signs on through a phone connection long enough to transfer his or her email and get email messages makes it difficult to get a large application loaded. If your deployment methods do not take such a case into consideration, your support will be viewed as poor by the users. To ensure the application deployment completes, you could force the user to stay online, have the user come in to a central location, or use any of the other methods we talked about in earlier chapters. Most commercially available deployment software will provide a good means to solve this issue for you. Transmission compression algorithms, checkpoint restarts, or smaller files all combine to get the data to the target in a quick and workable manner.

Administering applications for the remote user can be just as difficult. How do you determine whether an application is resident on a system if you can't run a report when you need to because the system is not online? Your packaged application includes reporting, so you will need to look at the archived data. Although this information isn't real time, it is reliable as of the report date.

Any reports you create stating success rates should always include a disclaimer line stating the as-of-report date. Not the date of the report creation, but the date the data was recorded. Status reports are only a snapshot of a time period.

Another challenge, domain security can prevent an application from loading. Laptops don't always have domain residency. Configuring them to a workgroup status is common (as is domain residency, true), which will provide added hurdles in getting your applications deployed. Windows security and domain issues will be discussed a little later in this chapter.

The primary needs of the remote user are the same as the local user that is hardwired to your LAN:

- Applications need to be deployed in a timely manner
- A report on the deployment status will be needed
- Application deployment needs to be administered

There are added issues involving the remote user that need to be considered in the solution formula. Obviously, not all issues apply to your environment and there may be additional issues unique to your situation:

- Remote users do not consistently log on to an NT domain or AD. They may only log on locally to the workstation that is either a resident of a workgroup, uses cached credentials, or changes domains often. Relying on domain resident logon scripts or GPO enforcement would not be effective in this case, as the security context and availability of the source location cannot be 100 percent guaranteed. Even locally attached machines can't be 100 percent guaranteed because users still have the annoying habit of turning off their computers.

- A department, office, or other business location may never hard-connect to the corporate network. They may rely on dialup connections (soft-connect) or a VPN solution via the Internet. VPN security and download times are a concern. Stuffing large files down the 56Kbps or 128Kbps pipe during business hours is not advisable if the users need the connection to make money. Bandwidth control that uses a percentage of the available bandwidth and adjusts itself is one solution. Just be careful that the bandwidth control does not extend the download time too much. Scheduling downloads for off hours is a common solution; just ensure that the targets stay online or you have a caching system at the target end of the pipe.
- The remote machine is at home. The system is used at home along with the kids' typing tutor program, the family book-keeping program, several different games, and an email program that is not on the corporate approved software list. The machine connects infrequently to the corporate network, and any application install could have difficulty with any of the other applications installed on the system. Your packaging process should address this issue for file and version conflicts. In this case, your enforcement of installing applications should be reviewed; do you really need to get applications to this user? If all you are concerned about is virus protection, it is easier to focus on protecting incoming data from the server side. Of course, you could also make it a corporate policy for only corporate assets to be used in connecting and any non-standard applications will be disabled or removed (if found).
- A user will only sign on for a few minutes at a time. The connection could be abruptly terminated at any time. Any download that is in progress may be interrupted.
- Mobile targets are hard to hit. For the machine that moves from one site to another, forcing the user to obtain all the data from the "home" site may be prohibitive due to line speed, a firewall, or other network restrictions. You will need to either wait for the target machine to come back to the home site or have a method to automatically change the source location on the fly.

Types of Remote Users

All remote machines or users are not the same. For my own sanity, I tend to split remote distribution targets into five main groups:

1. Common issues
2. Remotely hardwired machines
3. Remotely stationary machines
4. Roaming machines
5. Rarely connected

The common grouping handles most situations. Aim your overall deployment strategy planning at this group. As the target list becomes defined for a deployment, issues regarding deploying to the other groups (2 through 5) will come to light. Fine tune the distribution if needed to ensure the package is delivered smoothly to the entire target list. I'll focus the remainder of this discussion on the five groupings.

Common Issues

This grouping covers the issues common to all remote deployments. In the early stages of planning a software deployment when all the specifics are not known, plan to support this grouping. If the deployment can satisfy the common issues, chances are that the unique opportunities (a positive way of saying "problems") of the individual targets can be resolved. Whether you are going the custom delivery route or will use commercially available distribution software, you need to address the same concerns:

- Scheduling considerations
- Network transmission
- Deployment status reporting
- Connectivity
- Security
- User impact

Commercially available deployment software can provide most of the answers for you. Because the software vendor is in the business of software deployment to make money, they have a strong interest in addressing common problems and conduct testing to work out the bugs. At least that is the law of supply and demand. If they didn't address the issues, you wouldn't have purchased their software.

Scheduling

Scheduling is more than determining how soon the application gets out to the workstations. You also need to consider the end users and the business impact. Some of these issues were brought up in Chapter 4, and I will review them as well as some new concerns in the following section.

Is there a dark window when the user is not on the system and applications can be delivered as needed without interrupting the user? A dark window is a time frame agreed upon by all parties involved. All parties agree on this time as being a safe period. For example, the IT department and the business unit(s) agree every night between 2:00 AM and 4:00 AM and all day Sunday are periods when the computer folks can do anything to the computers (and infrastructure) without negative impact. This timeframe is when you do such tasks as forcing reboots, performing service pack updates, updating network drivers, doing backups (which may impact the network bandwidth), and update applications that the business unit uses heavily during the day. If a user is active during the dark window, the IT department is not held responsible if a reboot wipes out the unsaved report the user was working on for the past 5 hours. Be sure to remind users of the dark window on a regular basis so that everyone stays aware of the agreed upon time. Additionally, consider activating the logon time parameters in Windows; this feature helps to ensure that users are off during the dark window times.

Are there times when the workstation is doing critical functions and can't be mucked with? This timeframe is not a dark window situation. You need to know which times are critical to the end users and ensure that those times are not impacted. Because we are talking about doing things remotely, you can't see the target machines in person and may not be aware of what the user is doing at that point in time. Depending on the application and its user impact, your distributions can occur at all times of the day and night. You may choose to stage a large file to the workstations and activate the actual install at a later time. If the download takes over the available bandwidth during times when the business unit is heavily using the network, problems may ensue. Common times of concern are the beginning of the business day, heavy customer service hours, or defined account close-out times. You know the computing needs of your clients; make sure the deployment does not conflict with those critical times.

How do you trigger the start time of the application? The delivery mechanism will get the application to the target. Your deployment could kick off the package immediately upon transmission completion or it could wait for a scheduled update time. The start time needs to be addressed either by the packaging or the client portion of your delivery strategy. For machines that only connect for short periods of time, delivery may be done hours or days before the safe install time. Your trigger must take into consideration the variances to be met. This process works great if the target machines are online and turned on at the activation time. You also need to deal with machines (such as laptops) that could be offline or powered off at the activation time.

For example, suppose a cashiering application ties into a database server in your data center. The database server is undergoing an upgrade to a new version and a new ODBC on the client is required. The old clients will error out if they try to work with the new structure. Aside from having to deploy the update to hardwired machines that are online and powered up, you need to get the update out to your roaming laptop users who only connect to your network once or twice a week. To prevent needless problems, you stage the deployment to the target machines a few days in advance and set a trigger within the commercially available deployment software to perform the install on Friday at midnight. Thus, you get the data out to the users early but don’t trigger anything until later. If you don’t have commercially available deployment software, you can use the AT command (or JT from the Win2K resource kit) to schedule the activation of the client update.

Within the packaging, you also take into account the issues of late delivery. For the previous example, you would need to ensure the package kicks off the install if the trigger time has passed. As part of the initial package, you also worked to close the application if it was running, thus the reason for the midnight install. Applications can be closed remotely either through a command line specific to the application; using the NET STOP command for any NT, Win2K, or XP services; or through a combination of the TLIST and KILL commands, as Listing 5.1 shows.

```
net stop TestService
c:\util\kill TestService.exe
```

Listing 5.1:Simple application- stopping commands. The Kill command will stop all instances of TestService.exe.

TLIST.EXE and KILL.EXE are available in the Microsoft resource kits.

For those late installs, you can prevent the user from starting up the application after you stop it by renaming the primary executable. After the install has completed, rename the key executable back to the original name. Although this process might seem like overkill for a midnight update, you need to plan for the late installs that might occur when the user powers up their machine on Monday morning. Remember the earlier discussions about setting the end time or do-not-activate-after time? This is where it comes into play. We’ve all seen the fast-click users who start clicking applications or who set up their own startup scripts to launch applications fast. The timing between a startup and your package installation could be enough to have the user startup the application and mess up your install.

Another option for the late execution issue is to use the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce registry key to start up the trigger. Commands placed in the RunOnce registry key will execute once upon startup of the OS and clear out the key of entries. Remember the RunOnce command will activate the commands in the security context of the logged on user. Should you need higher authority levels, you will need to plan accordingly.

Don't confuse the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce key with the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run key. The Run key will cause the entered program to run each time the OS is booted.

This process prevents a never-ending loop of installs down the road. If your install depends on other tasks or functions happening prior to a particular time, use caution to not start the application early. Have the RunOnce key activate the trigger script only. The trigger script then compares the current time to the scheduled install time. If current time is after the scheduled time, the install runs. If the current time is less than the scheduled time, the trigger reinserts the RunOnce registry entry and terminates. This way, you are ensured to run the critical application prior to the user starting anything. Listing 5.2 and 5.3 show the commands for this process.

```
@echo off
regini runonce.ini
```

Listing 5.2: This line will call runonce.ini, which will insert the timer trigger into the RunOnce registry key.

```
hkey_local_machine\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
      test = c:\temp\TestTrigger.cmd
```

Listing 5.3: The registry format to insert the timer trigger into the RunOnce registry key.

We've now defined dark windows, start times, and business critical times. You need to make sure the installation does not creep into those critical times. Can you force an application to stop installing or stop processing at certain times? That is rather tricky. It is more of a timing issue on the startup. Looking at the issue backwards is the method to use. As we discussed in the last chapters, you should have a good idea of the time it takes for a package to install. Add a fudge factor of 10 percent or so, subtract the total time from the must-end-by time, and set the install drop dead time accordingly.

For example, if your application takes 30 minutes to install, and your dark window is from 2:00 AM to 4:00 AM, you want to ensure the installation does not exceed the 4:00 AM cut off time. A 10 percent flex factor is only 3 minutes, so round it to 5 minutes. All the clients are on fast connections, and the package is not large enough to worry about transmission delays. So at worst case, your application needs to be running by 3:25 AM to make the must-complete-by time.

Network Transmission

If the application is only going to remote users who are hardwired and online to the network at all times (such as call centers or corporate buildings), your primary concerns about the network will be the bandwidth considerations and the scheduling time of the deployment. However, if your deployment will go to remote users, be sure to identify the worst case transmission line to be dealt with. I don't mean to figure out the exact worst data line, general assumptions are good enough at this stage. Sending a distribution to a branch office at night through a dedicated 128Kbps link will ease the concerns into the same as the hardwired scenario. If it is to remote user dialing in, you need to understand the size of the files, the delivery method (compressed, checkpoint restart, or bandwidth controlled), and how to handle interrupted transmissions. Always plan on a transmission experiencing an early termination or a dirty phone line so that you're prepared when it happens. Commercially available deployment software confirms the transmitted file is intact prior to acting on the package. If you are scripting your own, at least do a simple compare of the file size between the original source and the file on the target. Checksum comparing is best in these cases.

You can find checksum utilities from any good utility site. Search for checksum at http://Download.cnet.com or http://zdnet.com.

Windows Installer also supports a checksum confirmation using the /f switch.

For the most part, the transmission concerns are out of your hands. You really don't have much control over what the local telephone company does with their lines. You can, however, plan how to handle the common file interruptions that are bound to happen. Comparing the file checksums, checking on segment completions, sending email to users about the deployment, and making the deployment a manual pull could assist in ensuring that the download completes.

Deployment Status Reporting

Deployment status reporting doesn't need much explanation. The primary issue is ensuring you get a status ASAP, hopefully while the user is still online. If not, you need the report on the next connection. Yes, most commercially available deployment software resolves this issue for you. If the collection point doesn't report back right away, a status is sent the next time the software client can talk to your collection point. If you are scripting your own deployment package, you need to get the data yourself.

The biggest determination is the level of reporting you require. Do you want simple success and failure codes? Or are you looking for confirmation-specific files or verification that registry entries have been properly inserted? If your software does not present the data you need, attach some logic to the process to pull the data and report it in a format you like. This format is up to you, and the possibilities are endless on how to create the report data.

Connectivity Durations

There are two types of connectivity durations: too short and long enough. Many users only connect to your corporate network long enough to transfer whatever data they are responsible for. Sales spreadsheets, timesheets, email, review of the corporate Web site, budget data, and maybe the latest joke .wav file or two. Then they log off and go about their business. The entire session could be completed in 10 minutes or so. In a day or so, they repeat the cycle. With short connections, it can be difficult to transfer application packages if they are large.

The long enough connections are a blessing. These users log on and stay online for hours (sometimes days) at a time. This timeframe gives you time to do all the automated administration tasks you would like to perform in much the same fashion as the hardwired remote user is dealt with. You have to deal with a dropped connection or two, but you know that the next session will be long enough to recover and continue. Obviously, the long enough connections are not a major concern. It is the too short connection folks that are your challenge. What you do to support the too short connection crowd easily translates to the long enough connection crowd, so I'll focus on the short crowd.

The delivery process will depend on your standard delivery mechanism. With the numerous and varied methods of deployments available, the answer is going to be just as varied. So the answers and examples I'm going to present to you are only suggestions, there are numerous other methods that will work just as well. Your choice depends on your environment and structures.

If your commercially available deployment software handles remote users, the task is not so bad. The client piece on the machine will have a trigger method of some sort: a timer that checks for a connection to a destination address every few minutes, a listener that waits for a broadcast poll from a server, or maybe a script to start up the commercially available deployment software client when the user activates the dial-up script. No matter the trigger, the end result is that the client piece talks to the deployment software and begins processing your instructions or the updates as you have scheduled. Installation then follows your normal processes. This easy method is another good reason to pick up good deployment software.

If your distribution is done through scripts, you will need to get creative. The final solution will depend on how your users contact the corporate network:

- If your users dial directly into the network using a modem-to-modem connection (not via the Internet), you can set up a trigger from either end of the connection.
- From the client side, you could set up a scheduled time for the computer to dial, connect to your network, poll for any updates, and disconnect. This option works best for the remote stationary client who is connected to a phone line at all times.
- If your users are contacting the corporate network via the Internet, you can tie in to the security application the users activate. A VPN or token synchronization appliance are two examples. In those cases, you can activate the deployment method from either end, though triggering it from the client might be easiest, depending on the host methods you are using.

Looking at the modem-to-modem connection, your triggering processes will either be keyed from the client side or the server (or corporate) side. In either case, an event will occur to indicate a connection has been established and the deployment process can begin.

Other alternatives include having a polling process run at a regular interval or including a service or executable that always runs and monitors various API calls or processes within the OS. If you don't use commercially available deployment software or your deployment software does not have a trigger method, I've listed a couple of simple methods that can perform the functions for you without much of a problem.

Because users will be calling in via a modem, having a script for them to use is the easiest method. Include an after-dial script to run a process calling your custom distribution client. The client should do a check for connectivity to a known point within the corporate network. A ping against a static IP, a server name, or a flag file on a server are all easy items to check for in a loop process. After the connection is made and the known end point is confirmed as reachable, the client piece can proceed with any downloads you have. The advantage to this method in a manually activated dial-up situation is the automated check and response the deployment process has. It does not depend on a polling interval that might miss a short connection window, and it does not rely on the end user to activate a second process. It is basically all-inclusive and functional. One downfall is the ease of editing scripts. Scripts are subject to modification by anyone who can do modem script edits, and your deployment client might be edited out.

Although I mentioned that the dial-in script could call another batch file, it could just as easily start a deployment service. If your client piece is configured as an executable and installed as a service on the target machine, it can be activated by the dial script. By leaving the service stopped until the dial up is completed, the service does not try to connect when there is no open channel. In short, it keeps the computer from burning cycles on a process that will never succeed in contacting the server. There are a number of methods to convert an executable into a service; the NT Server 3.51 and 4.0 and Win2K Server resource kits all contain two utilities, SRVANY.EXE and SC.EXE, that you can use.

SRVANY.EXE and SC.EXE are available in the resource kits. Use the documentation in the resource kits for the syntax. Check out the following Microsoft Web sites for more help http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/winxppro/proddocs/sc.asp and http://support.microsoft.com/directory/article.asp?ID=KB;EN-US;Q137890&.

Client pieces that rely solely on timed polling events need to be carefully adjusted and monitored. In a timed polling process, the system periodically checks for access to a known location. Either a ping or a query to an HTTP location is used. Obviously, if the machine is not online, the polling process will fail. If the machine is online but not connected to the company network, the process still fails. Setting the polling time too short could place a burden on the client system. Setting the polling time too long might miss the connection window. When the polling process is a service that activates on boot up, the initial queries will fail if the system needs to dial in first.

Take the case of a remotely located laptop user with a polling service set to poll at system startup and then every 30 minutes afterward. The first poll will meet with failure as the laptop has not connected yet to the company network. The user dials in, gets authenticated to the corporate network, does the normal email transfer, and disconnects from the network. 10 minutes later, the polling service queries again and meets failure. It just missed an opportunity to connect. The solution to this problem, assuming the polling service does not create burdensome overhead, is to set the service to poll every 1 to 2 minutes. Leaving it to poll at startup is a good idea as the machine may be hardwired to the LAN at a company building once in a while.

If the polling service could be activated by a dial-in script, the short polling period issues would not be a concern. Connecting the service to the startup of the email application is another way to catch when users are online.

On the corporate side, we will assume the users are calling into a central modem pool or a monitored location. I'm naturally paranoid about unauthorized access to the business network and have a problem with the general user population being given the ability to dial in to private modems. If they dial in to a central modem pool, monitoring is easier and you can prevent the common snooper from gaining access to your systems. (For a brief reminder about security, see the sidebar "Modem Security.")

Modem Security

One side step from the connectivity subject: modem security. If a user is calling in to your network, make sure it is the right user! Have some sort of security—don't leave things wide open. Password access (with a password that changes regularly), call back to a specific number, or token access are all easy to implement and not really intrusive to the user. *Be sure to monitor the logs daily.* Daily monitoring of who called in to your network is a requirement so that you know when someone has hit your network.

OK, back to the subject. I'll take the point of view that all your users call into a central modem pool. Or maybe several geographically based modem pools. Either way, access is limited to getting in via a monitored and controlled method.

- If the dial-in security is call-back controlled, the deployment trigger script is activated when the call back is completed. After-dial scripts are common, so this part should be documented with the modem software you are using. After the connection is made, your deployment method can investigate the client machine and perform your selected actions.
- If the modem pool relies on password or token registration, once the access port is open to the client, a polling script can be used to query the status of any mandatory software updates. You would want to query against the opened port. Scanning the logs for successful activations can trigger the call against the user ID or the machine name. Details will depend on the security method and program you are using. Check the documentation for methods to use.

The question of infrequent connections has been answered: You need to have a trigger activated depending on your deployment standards and connection method. This will get the package download started.

What about the short connection time? Now the fun starts. You need to really know your users to decide on the proper method. How long is the average connection time? What about the average connection speed? If all your users are coming in via cable modem or DSL and the application is only 500Kb, chances are you won't have to worry about short connection times. The download will happen quickly, and the user won't even know it. Problems start popping up as the package size increases and the connection time slows. If you use commercially available deployment software, the issue of dropping a connection during a download is addressed.

Deployment software offers two common methods of transmission support: Prism Deploy has a unique delivery scheme that is able to download files across a 56Kbps line in a short time. According to New Boundary Technologies, the schema is successful enough that the company doesn't bother with check point restarts or parsing out files into multiple segments. The file either gets downloaded completely or it does not. (I discussed the specs from New Boundary Technologies in Chapter 4.) For most application installs, this issue won't be a problem for the users that are online for 10 or 15 minutes at a time. Marimba uses a WinZip type of compression and relies on checkpoint restart with checksum verification for resolving interrupted transmissions. This method means a user can sign on for a few minutes at a time and will get a little more of the application each time until they get the complete download.

If you are doing your own downloads, you have to deal with the problems of interrupted downloads yourself. Some suggestions to aid in the process:

- Break up the large files into smaller chunks and assemble them on the target system after they all arrive.
- Use a compression method that squeezes the data into the smallest foot print possible. The WinZip and PKZIP utilities are commonly used.
- Send a wrapper process down first to monitor the download of the individual segments. The wrapper would be activated by the deployment trigger and would be the one to call the individual sections. On each connection, the wrapper would check to see what has been downloaded, then call the next section in line. After the wrapper detects that all the pieces have arrived, it can decompress everything and activate the installer.
- Your package could be designed to transmit set files. When the package starts up, it checks for the existence and size of the defined files before it begins the install process.

Stale or out-dated data is one big problem with infrequently connecting machines. The machine may not connect long enough to ever fully download the package or checks in so infrequently the data share point has changed content. If you split your packages into segments, be careful not to overwrite older segment names with new data, otherwise a system could end up with segments from two or three (or more) versions. Then you have the issue of remote support for a highly confused application.

Again, the issue of stale data due to partial downloads is usually addressed by commercially available deployment software. The processes from the vendors either reconfirm the partial downloads or restart the download from the beginning. Either way, the vendors remove the headache of data validity for you.

For machines that dial in but remain online for long periods of time, life is easier than for those in the previous scenarios. You still have the concerns of the trigger piece, using commercially available deployment software or custom created methods, and activation of the package on the machine. The entire scenario takes on the tone of the hardwired remote user. The long connect clients are available and can be accessed; they just take a little longer to get the data due to the smaller connection pipe.

Small remote satellite offices or work-at-home users fall into this category. They generally contact the corporate network often and remain online for longer periods of time (hours or days). Their desktop machines don’t travel much (hopefully), and the administration is simpler. The primary concern has to deal with the slower line.

In all cases, ensure your deployment method has a reporting structure to report when the machine completes the install. Because the machines don’t stay online all the time, they need to report immediately to a collection location. Trying to gather installation statistics when a machine is not online will add extra overhead to your already busy day.

Security

Security for software deployments is an area not to be over looked. Deployment security is concerned with network and Windows security. If you work in an organization in which the IT responsibilities are split to different departments, staying friendly with your network staff will pay off. If they make changes to the firewall access list and accidentally block a needed Windows port (say the server side TCP 135 RPC port) or block a port range above port 5000 after you have set your remote clients to use RPC calls above 5000, your remote users may suddenly have difficulties ranging from not connecting at all to package delivery failures.

You could easily spend days investigating the distribution failures. When only one router access list is changed, it will complicate the issue even further, causing you to experience gaps in your hair growth as you slowly pull out the individual strands. Stay in the loop on changes to anything in the network that could impact your data flows. Ask the network folks to include you on any network change notifications they send out. Simply review any changes for potential impacts and save yourself wasted life cycles in troubleshooting preventable problems.

The other primary concern is Windows security. File permissions, domain trusts, transient trusts, and a host of other Windows security features can present unique problems to your distribution efforts. For stationary or static machines, your only concern is the user ID permissions: can the user access the source files or can the commercially available deployment software client contact the proper server? For roaming machines such as laptops, your concerns multiply.

How complex is the domain structure of your environment? If your company works with one domain, domain security aspects are not as much of a concern. When you add another domain and populate it with users and machines, you have complicated your delivery process. The juggling act between accessibility and install speed are two more bowling balls you get to keep in the air when sending out distributions. As if you needed any more to deal with.

As Figure 5.1 illustrates, in the simplest form, if Laptop is part of Domain A, and the applications source is in Domain B on a machine called Server, you need to ensure Laptop has access to Server or the trusts between Domain A and Domain B allow machines to talk to each other. Otherwise, the application install will fail due to Access Denied errors.

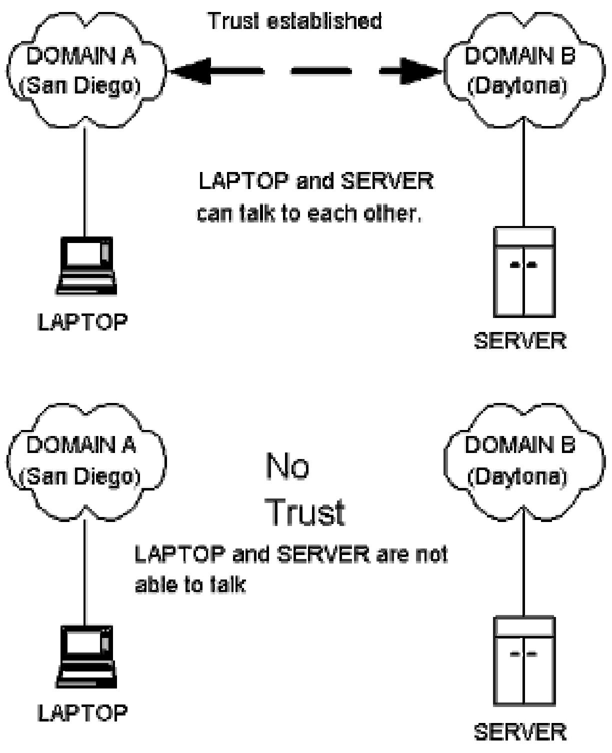


Figure 5.1: How domain trust affects deployments.

Another solution to the same problem is to place the files on a standalone server to provide access to the group Everyone from both domains. HTTP is becoming a popular method for distributing files. Many commercially available deployment software products are moving toward that direction. Either the entire deployment software process is HTTP based or it utilizes the ability to pull the source files from a Web server. This process can provide an easier method for complex security issues as the Web server connection can be configured independently of the Windows domain/file security structure.

To expand a little more on the example, assume the following scenario, which Figure 5.2 shows: Laptop is still a member of Domain A, which resides in San Diego. The distribution method points Laptop to Server A in Domain A to obtain all the applications. Laptop travels to the Daytona office for a long-term visit. The Daytona office is part of Domain B. All the machines in Domain B get their applications from Server B. To make it a little easier, assume there is a trust setup between the two domains and there is a slow link connecting the two domains (128Kpbs or so). Are you still with me? Do you let Laptop pull its applications from Server A in San Diego over a slow link or do you move Laptop into Domain B in Daytona to obtain the applications faster without impacting the slow network link? Leaving Laptop in the Domain A would be best if the Daytona visit is a short one. If Laptop will remain in Daytona for several weeks, moving it into Domain B would be the best method.

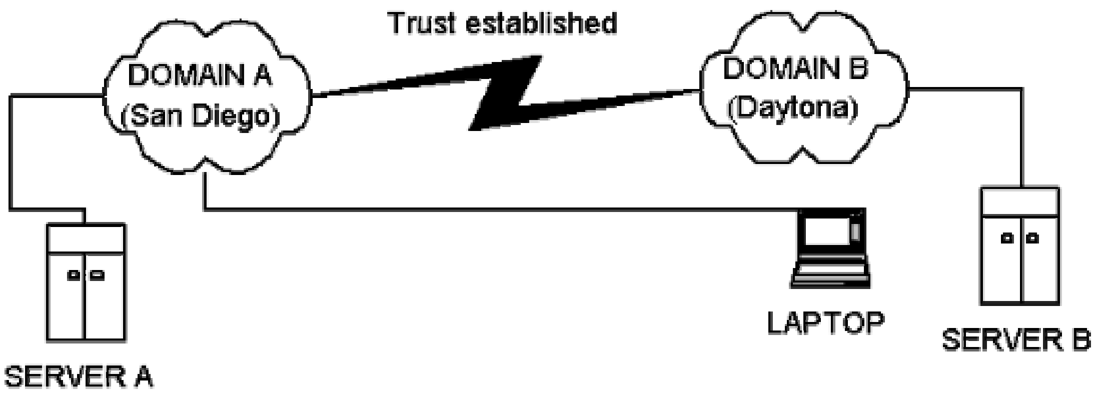


Figure 5.2: Showing the distance Laptop must travel to get data from Domain A Server A.

As you can see, the remote user presents you with added security concerns. If your applications are pulled from a Web server, do you require all users to connect to the corporate network to reach the server? Or do you place the server in a DMZ and grant restricted access from the Internet? Having the users authenticate to your network will provide a level of security for the Web server and is highly recommended.

A word of caution about opening a distribution server to the Internet for application installs. Be very certain your security is tight. You don't want the entire Internet using your server to download licensed applications. The legal ramifications can be painful.

If you opt to send applications to remote users on a CD-ROM for them to load, you now have another facet of the security issue. Aside from the obvious issue of the CD-ROM getting lost at the airport and someone else installing Office XP using your corporate license number, you need to be concerned about any corporate data present on the CD-ROM. Weigh the potential of a lost or stolen media against the importance of having sensitive corporate data available on the easily transferred medium. Users tend to realize rather quickly they have lost their laptop with all the data and connection ability at the airport. You can take steps to prevent the laptop from accessing your network and begin changing any passwords that may be compromised. A lost or misplaced CD-ROM can go unnoticed for a long time and present an open window to your company without you knowing of it.

If you send out media with sensitive data on it, encrypt the data and keep all the passwords off the media. Send the activation codes in an email or letter or phone call. An encrypted CD-ROM without the activation code is useless to 99 percent of the world that might find it. Hard-core hackers can still break into it if they find it and want to put in the effort. The issue then becomes theft and not just a lost media situation.

The credit card industry is a great example to follow. You get a new card in the mail but it is useless until you call in and activate it. The credit card company requires you to call in from your home phone so the caller ID can be verified. Or they talk to you and ask for specific data to prove you are you. The PIN is sent in a separate mailing, usually in a plain envelope with no account data associated with it. If it works for them and the millions of users they deal with, it can also work for you when shipping an encrypted CD-ROM of your company payroll information to remote pay clerks.

To repeat a concept from Chapter 3, when a package is installing, it needs to have the proper authority level. In a locked-down environment, users generally don't have the authority context to adjust the registry. The package would need to run under a higher authority level of some method. Security is a large issue that often comes in conflict with distribution methods. Locking down a user ID to prevent the user or unknown users from getting access to your system(s) can also prevent the distributions of approved packages. Windows Installer from Microsoft provides the ability to install the packages under an administrative context. Test your installs under a user's context to ensure it works. When you are doing remote distribution to clients who will be contacting a source location from various end points, test all the possible versions you can to ensure the security platform is not a hindrance to the install.

User Impact

The user will be impacted no matter what you do. However, you can limit the impact as much as possible. Your packages should take into consideration the normal distribution issues to be dealt with as we discussed in past chapters. Remote users place an emphasis on delivery and timing.

In the section about scheduling, the issue of timing was discussed. Dark windows, cut off times, and system critical use times were reviewed. Those are issues of user impact that can cause problems. You don't want the package to be installed and force a reboot during the business day. Remember if the target machine is not being used to make money for the company, your budget can't buy all the neat toys you want.

If your distributions have any chance of causing an impact to the user, make sure the user is aware of what is happening. Pop up message boxes, email notifications, or phone calls can be used to ensure that the user is aware of distributions. How much you use these methods (and others) are up to you and how much you want the user to know of the distributions.

Let’s talk about pop-up message boxes. These are wonderful items if used properly. In principle, when a package is being installed, a pop-up box would appear indicating the status of the install, a progress bar of the download and of the install, and any informational data the user might need. The user then knows what is happening on his or her system and can adjust accordingly. I’m bringing this up in the chapter about remote users to emphasize the point of communicating with the targets so that they help you out.

If a package has the potential of negatively impacting the user (causing a reboot or a termination of a program), decision pop-up boxes would be appreciated by the users. Having the option to defer a package keeps the users happy as you show consideration for their needs. I would strongly suggest you limit the deferrals to once or twice and clearly state it in the pop-up box. Otherwise, the user might never allow the application to install or reboot the system.

For large downloads that will take a while on a slow line, pop-up boxes with a download status bar help the user. Assuming downloads are silent, the user may not know when the download is complete, and either terminate the connection too soon or keep the connection up longer than needed.

One option is to present a deferral option to the user with a pre-defined delay time. If they click Later, a second box appears stating the package will activate at a specific time (such as 4:00 or 5:00 in the afternoon) or within an hour. The second box states very clearly there will not be a second chance for a deferral. This option gives the user plenty of time to contact you if there are issues or to plan for the impacting action. Either way, the user is generally appreciative of having an option to install the mandatory application at a time a little more convenient to them.

If you do use deferral boxes for a package that does reboots, place a little logic into the package. Assuming the package has done the install and only needs to reboot and the user defers the reboot to later, your countdown timer needs to identify if a reboot was done earlier than the program scheduled it for. The RunOnce registry key can be used to clear out the countdown timer if a reboot happens. Otherwise, your attempt at good will is negated by the user doing a reboot and you forcing another one later.

Pop-up boxes, while great, can be annoying if they interfere with actions. You’ve experienced the annoyance. You are in the middle of doing some complicated action and a message box appears, taking the focus away from your active window and interfering with your work. A minor annoyance, but one you need to be aware of. Use of pop-up boxes is good if done within reason.

Keep in mind how you handle pop-up boxes on your own systems. When a message comes up, do you always read the lengthy ones or just click OK or hit ENTER and take the default? The users you are dealing with are even less likely to read and react to messages than you are. Make the messages short and clear. If a default action could cause a problem, have a confirmation pop-up box verify the action.

Remote Hardwired

Let's address the easy remote machine first. This machine is

- Connected to your LAN/WAN
- Physically located a distance from your location

This type of machine is in the corporate environment but located in a different location from where you are. These machines generally stay online most of the time, have a good fast network connection, and don't move around much. They classify as a remote machine due to the geographically different location. The major consideration is getting the package to the machine. Of all the remote machines you need to deal with, this machine is the easiest. Why? Your standard deployment process can get the application to the machine (we covered this in the last chapter) without any unusual considerations.

The common issues I discussed earlier apply to this class of machine. Take particular care about the security aspects. Normal Windows security will apply if the target is in a different domain from you or your package source location. Your testing should have covered the delivery process and the application install, proving there were no problems.

Evaluate the packages. If there are reboots or major interruptions to the system that could cause issues with the end user, be sure to take into consideration any difficulties that might be encountered. Your delivery mechanism or the package can also deal with the common problem of leaving a floppy disk in the drive (by turning off the floppy drive search or setting the search order differently) if your application performs reboots. With regard to the floppy search order, you will need to check your hardware specs, some vendors provide methods to turn it off.

Remote Stationary

This type of machine:

- Does not move around much. It stays within one geographical location.
- Is not part of the hardwire LAN/WAN. It depends on a connection method that is not always on. Dial-up, cable modem, and DSL are examples of the connections used.

The main difference between the remote hardwired and the remote stationary machine is the connection. Unlike a laptop, which is meant to move easily, the remote stationary machine is a desktop unit meant to stay in one location most of the time. Picture the desktop units that are taken home for remote users to work out of their homes. Or a small office with desktop units that only connect via a modem. The issue of variable connection speeds is not a major concern because the machines will use the same method each time. What makes this class of machine difficult to administer is the fluctuating online status.

More of the challenges we discussed in the Common Issues section start coming into play with the remote stationary machine:

- Multiple user IDs
- Questionable domain or workgroup membership
- Controls on local security are not as strong or may have been comprised
- Uncertain levels of software due to users having a higher level of access on their IDs

Compared with the roaming machine, this machine group is the easier to work with. Most machines are in a known state when they leave your imaging process, and you know how the machine will connect to the network.

The concerns will center more on when the user connects. On each side of the extreme scale, there is the system used at home on one end and a business unit system that stays connected via a phone line on the other. While both pose unique issues with packaging and implementation, they are not any different than what was covered in earlier chapters. The concern is how to get the application data to the machine.

I'm going to stereotype the two extremes for a minute. The business system is going to remain online for long periods of time and have a good quality network connection. You will be able to reach the machine during known connection times, say during normal business hours. Package deliveries can be scheduled around the known connection times and easily worked with.

The home office type of machines is more of an unknown. The network line is a lower transmission level, probably depending on dial up via a 56Kbps modem. Connection times will not be as consistent as the business unit system, although the durations will be long enough to support distributions. The home office side of the scale will be connecting for more than 10 or 15 minutes a day.

This class of machine begins to loosen up the controls on the system (or the controls are forced open by the user), and your packages may begin to experience difficulties with conflicting and unknown applications. Given security on the machine is not as tight as a hardwired machine located in your corporate world, your packages will need to clearly define connections and security concerns.

The transmission issues outlined in the Common Issues section will play a large role in your distributions. Transmission interruptions will begin to occur more often than in the hardwired machines. Calls to a central repository that has been locked down may begin to fail if your packages don't explicitly define the access right for the connection. Pay closer attention to the packaging and access rights needed to install the applications. Your transmission integrity will begin to be tested but won't be as challenged as it will by the next class of machine.

Roaming Users and Machines

Roaming users and machines create their own unique issues. I'll discuss users that roam from machine to machine in the next chapter. For this chapter, the focus will be on roaming machines. In other words, laptops. Laptops are difficult to administer. When administrators are talking about laptop administration, colorful adjectives often preface the description of the task. Laptops present many challenges:

- Connecting to the corporate network may rarely occur. If ever.
- Laptop users may have a higher authority level than a non-laptop user. Given the users need for flexibility in installing applications, the laptop user is given added access rights. Maybe not full Admin level, but Power User group membership (which is nearly Administrator) for the machine.

- Passing a laptop from one user to another is common. A laptop pool to share the expensive asset solves many equipment purchase budget issues although it increases the level of administrative overhead for you. Some organizations re-image a laptop every time it comes back into the pool to reduce the influence of the previous user (and remove the various programs that might have been installed). If the shared laptop is not re-imaged each time it changes hands, the system integrity tends to diminish with each favorite game install or customization applied.
- Domain or workgroup residency may change.
- The logged on user may only authenticate to the laptop and not to an NT domain or AD structure. If the user logs on to the disconnected laptop with the domain or AD account, the credentials are cached. This behavior can cause issues if you rely on the logon process to apply dynamic changes to the machine. If the user logs on first, then dials in, the logon process won't be restarted. Yes, you can force the user to log on via the dial-up process, but be aware that most users quickly figure out how to get around this limitation.
- Connection speeds vary all over the map depending on the needs and line speed at the moment. Thus, relying on a set network timing parameter for a laptop delivery can be difficult.
- In the ideal world, the machine name would identify the laptop from a desktop unit. In the real world, it is often difficult to identify a roaming laptop from a stationary desktop machine. The delivery mechanism needs to have a means for identifying the machines. If the commercially available deployment software has the machines grouped accordingly, any checks you put in are for those non-standard imaged machines. For applications targeted to a laptop rather than to the desktop machines, you need a fool proof method to determine the difference. A corporate standard stating all laptops are from Compaq and all desktops are from Dell (for an exaggerated example) would make your job easier. Any inventory you get will identify the differences immediately. Your packages/delivery methods could also check for vendor-specific drivers for video or network cards. For systems created with your standard image process, a common practice is to create custom company keys in the registry or add environment variables to identify the system as a laptop.

☞ To help positively identify laptops, check for the existence of unique laptop registry keys such as HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\Root\COMPOSITE_BATTERY.

The roaming machine will stress your delivery process. Connection times are uncertain, the quality of the connection will be unknown, the duration of the connection will vary, and the process of calling source locations can be confusing. Remember the example of Laptop in Domain A when it visited Daytona in Domain B? The machine was roaming between the two domains but was generally static in each domain for a period of time. Keeping the laptop updated with data and applications was a matter of sending the laptop to the proper domain location after evaluating speed and security. What happens if Laptop travels between Daytona and San Diego a couple of times a week? And if a large chunk of your workforce are roaming machines?

The emphasis now moves to the infrastructure instead of just the machine. Your data source locations will need to be configured for the best access from various domains and connection styles. Centralizing all the data in San Diego, for example, may be fine for the folks on the West

Coast, but the employees on the other side of the country will not have the same speed. Running the fastest dedicated network line you can afford between the Daytona and San Diego offices will help the speed issues, as Figure 5.3 illustrates. Have the roaming machines dial in to the closest office and connect via the infrastructure that way. Your central data store can stay central and the users will have a good connection.

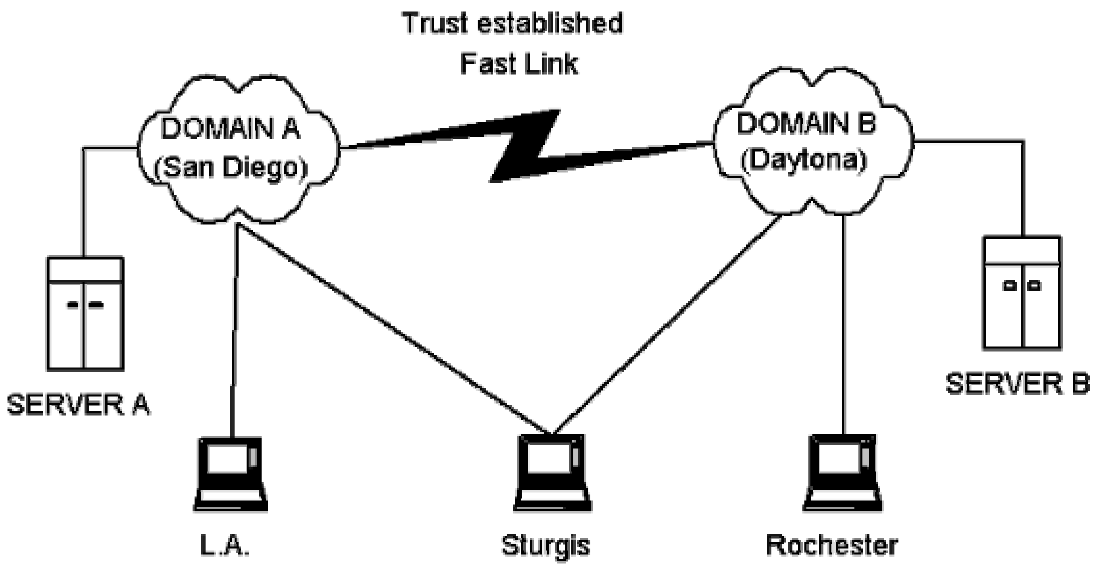


Figure 5.3: Remote users call into the nearest access area. Distributions can be based upon which access point they call into so that the closest server is used.

For the roaming machines that have data sources statically defined and that go from company location to company location, you still must face the trade off between speed and ease of administration. If your company uses DHCP, one solution is to compare the IP address of the current connection to the previous connection. By creating a master list of the IP subnets for each location, you can redirect the source locations for each machine on a dynamic basis. If the IP is the same, the machine can grab its data from the previous defined sources. If the IP address changes, a script would locate the new source server and direct the machine there. The subnet change process could be a part of the commercially available deployment software or your custom solution. Including it in the logon process has been done. Keep the subnet source list central so that you can update it without replication worries.

Yes, it takes a bit of time to set up. Large companies with numerous subnets will take a while to configure the initial setup process. You wouldn't have to check each octet of the address, only to the level your network is subnetted. If all your West Coast machines are part of at 10.x.x.x network and your East Coast machines are part of 192.x.x.x, you only need to check the first octet. Some commercially available deployment software products do this check for you in their structure. For dial-up situations in which the calling machine uses the ISP defined IP, you could either ignore the changes, which would force the machine to the last defined source location, or you could define a default location. The speed issue would become moot at this point due to the possibility of numerous masking schemes you would need to detect.

Another variation to the IP checking scheme is to have all the source locations hidden behind a URL pointer. The calling application installer or wrapper would contact the initial source location and offer the IP address. The initial source location wrapper would determine the closest source to use based on the IP, and send the information back to the calling program. The calling program then contacts the defined source system and begins the install.

Sorry, no easy answers for this one. The call is yours to make, depending on your environment needs and the needs of the users. Commercially available deployment software can provide the answers for you in a defined process, or you can create the added details if you are creating your own distribution method. Personally, if I were involved with a company that had the issues of speed over location, the costs of obtaining a deployment software compared with the administrative factor of the custom solution would help justify the deployment software purchase.

Rarely Connected

These machines will provide all the encouragement you need to make your vocabulary more colorful. This category includes both desktops and laptops, in your corporate buildings and at home sites, and connecting via the hardwire LAN or the worst dial-up connection possible. The commonality is the lack of connection. These machines rarely connect to your network. When they do, it is with the intention of grabbing one or two pieces of data only.

Rarely connected machines may connect once a week or once a year. With few exceptions, the machines are not directly involved in generating revenue but deal with functions such as word processing. My experience has mainly been with personal home machines being used to dial in and get email or a document for a user who is sick at home. I will not even begin to discuss the security risks and the connection issues for these users. Those are pretty obvious.

For software distribution, you can assume these machines don't have recent versions of any applications used within the company. The client software for your distribution method may not exist on the machine. The domain residency is also in doubt.

In many cases, it is easier to manually find the needed data for the machine and send it down after it connects. For those production-related machines that need to be updated with your distribution method, your challenges are many. The following list provides my recommendations to support this class of machine:

- Use status boxes for downloads. Providing data to the user will aid them in determining when they can disconnect from the network and will help prevent early termination errors.
- Ensure your packages do validity checking of the download. Otherwise, you may have several parts of distributions spreading over several versions. Trying to install the entire version spread will cause errors and generate support calls.
- Use a pull-type technology for the software distribution client. When the client connects and contacts the distribution points, the client piece will help sort out which packages need to be delivered. Relying solely on a push technology may cause problems, as the advertisements on the push server can age and terminate before the machine contacts the server. Although this problem can happen with pull processes, it tends not to happen as often.

- Be sure your packages are all-inclusive and not incremental in nature. Let the client determine how much to install each time. Incremental packages work best for clients that contact the network on a regular basis. Rare connection machines could miss several increments, causing the package to be missing critical files. If the data being sent is application data in nature and some increments are missed, there could be aged data deleted in some of the missed versions. The result is unreliable data.

Many companies have set policies on vacant connection durations. If a machine does not contact the core network in 1 or 2 months, it is not allowed on the network. With the advent of portable token access methods (thumb prints, synchronized RNGs, and so on) and Internet front ends, it is becoming more and more common for a user to be on a borrowed machine when they contact your network. User may never be on a machine again, but they expect the connection to function as if they were on their machines. We'll cover the roaming user in the next chapter, stay tuned for that one. In the mean time, be aware users can call in for assistance on machines that do not properly belong on your network and are not configured for your methods.

Your machine access policy needs to consider how to handle the rare connection. Do you require a known registry key or file as part of the validation process? If so, try to make your software distribution method the key to be used. If your deployment methods are not on the machine, the machine should not be allowed into the network. If machines are allowed on the network, do they need to load your commercially available deployment software? The answer will be in the expectations of the user to obtain packages and distribution data.

Setting up a common Web page that contains the packages for such users will ease the issue. The user can then obtain needed packages in one-off situations by going to the Web page and activating the packages manually. You can also include the distribution client on the page (or have it as the only item there) so that the users can install the client themselves, then let the automation take over for the package delivery. The main issue with the rare connection machine is its unknown status. It may be a part of the domain at one time and associated with the right groups to activate automatic deployment of any distribution clients you have. GPO and logon enforcement methods will ensure the machine obtains the client. If the machine is removed from the domain or AD due to lack of use and is allowed back on the network at a later time, it may not be properly added back to the domain or AD structure. The machine may still show residency in Domain A, but the SID is not in the accounts database, so the machine is not getting the security tokens passed properly. Unless an administrator adds the machine back in properly. See the problems? If you rely on a human process to get a machine included in all the right groups to ensure the mandatory deployments are done (such as anti-virus software updates), humans will find a method to side-step the process because they "do not have time to wait."

The best you can do is to provide a number of methods to resolve this situation. Set company policy properly, communicate to everyone on a regular basis how to ensure the rare connect machine can be included in the software-deployment process, check your logs, and set up options for others to easily activate the process for you (Web pages, email archives, or known source locations).

A Real World Example

Dealing with remote users is only as difficult as you want to make it. By insuring your application package has distribution intelligence and accounts for your worst case remote scenario, your distributions will work well. The basic distribution system needs a client piece and a server location. Whether you are using commercially available deployment software or a process you create, the distribution system needs to know when the connection is available.

In the fairly closed environment of the business world, your targets and network are of a known quality. Thus, your distributions are going to a known quality. As an extreme case of distribution flexibility, imagine sending your distributions out to a world-wide audience. The number of machines are unknown, the type of systems are generically known but not the specifics, the contents of the target systems are unknown, and the connection methods include every known method available. Your package must be available to do client data updates on a 7×24 basis as the clients call for it. The reporting process is as-of-report-date and updates a live database with the latest information at all times. Dreaming, you say?

Check out the SETI project. This project was the first world-wide computing effort that strives to take advantage of spare CPU processing cycles on Internet-connected computers to search for extraterrestrial intelligence (hence, the acronym). In short, a radio telescope records data signals from outer space, and sends the recorded bits to the University of Berkeley. Anyone wishing to participate in the effort downloads the client piece for their particular type of computer and OS. The client piece contacts the servers at Berkeley and downloads a chunk of the recorded data. Processing is done on the local computer. After the chunk of data is analyzed on the local computer, the client contacts the Berkeley servers, uploads the results, reports on the status of the processing, downloads another work unit, and continues the processing.

If the computer contacts the Internet via modem, the client piece has options to dial out. Control is given to dial at certain times and to disconnect after downloading the work units. For systems that use an always-on method, the client contacts the Berkeley server as needed. Proxy capabilities are included in the client so that firewall situations are resolvable.

This process is an example of all the discussed extremes. The program handles the reporting, the distribution to multiple clients, the automated download when a connection is made, and the updates to the central database with the latest data. Over 3.5 million users are registered in the database as of this typing. The total user count is not the actual number of computers running the client. A single registered user can run the client on multiple computers simultaneously. That makes SETI one of the largest software distribution projects ever!

📖 If you haven't come in contact with the SETI@home project, check it out at http://setiathome.ssl.berkeley.edu/.

Summary

Deploying to remote users is much the same as deploying using methods that we have discussed in previous chapters. About the only deployment method that does not work for remote users is Sneakernet. CD-ROM, email, Web, commercially available deployment software, and custom methods all work. The main concerns are the connection methods and the length of the connection. Make sure that your deployment strategy has a method to deal with interrupted transmissions.

The next chapter will focus on deploying software in the enterprise. The topics will include roaming users, metering, and management in both small and large enterprises.

Chapter 6: Deploying in the Enterprise

I've talked so far about creating a package, how to get the package out, and how to handle deployment to remote users and machines. The focus so far has been from the application point of view. It's all been about the mechanics of the process and how to make the deployment do what you want. I've talked about testing methodologies and about some of the different technologies for delivery and packaging. Now along comes this chapter about deploying in the enterprise, and you're probably wondering "Hasn't this topic been discussed?" Well, sort of.

The discussion up to this point has been centered on the technical aspects of getting the first package out. The goal of the early stages of the book was to provide enough data to get your packages up and out that first time. Although there have been casual mentions of doing deployments in repetitive forms or multiple deployments, the main theme has been on just one package at a time. Software deployment is not a one time function nor does it entail only the technical details. Windows software deployment is an entire process that must be repeatable in both the technical and procedural arenas, and the process must show a reduction in effort (and basic expense) as the numbers of deployments continue.

In this chapter, I'm going to delve into the area that all technologists dream about: Process, procedures, and documentation. To make it sound a little more technical, I'll refer to this topic as PP&D.

PP&D is just as critical to software deployments as the actual creation of the package. Remember that the deployment system you use within your enterprise must be repeatable by others, not just by you. The steps to create a package, the standards to use, the SLAs, and the methods to use must all be known to the technical staff responsible for the work. Management must be aware of the process as well as informed about the work you've already done. Because you most likely enjoy taking a day off here and there and dream of an actual vacation that is not interrupted by work calls, documenting everything to the level that others can follow is critical. It's all about making your life just a little easier.

Before I get too deep into the chapter, let's define a few terms that I'll use frequently:

- Enterprise—This term has come back into vogue in recent years. For this chapter, it will be interchangeable with business unit, company, division, department section, or any other term you like that describes a group focused on making the business run. It's a good generic term, as it can cover the large company as well as the mom-and-pop shop.
- Shops—The IT department. It can be one person that does everything or several thousand specialists. Adjust the term to fit your situation.

Does Size Really Matter?

Does size really matter? Sure it does. When you are talking about motorcycles, bench pressing, monitor viewing sizes, CPU speed, and RAM. When you are dealing with software distributions, the logic and methods applied to a distribution is the same whether you apply it to 20,000 machines or 200. True, the time factor, distribution points, and network issues are increased as the number of end points increase, but the basic structure and process remain the same.

As a recap, remember that in smaller shops in which distributions are to a few machines (say a couple dozen), getting things done is generally faster. Notice I didn't say better—just faster. You have one person that does the packaging and testing and distribution. If something happens to the distribution, everyone in the enterprise knows where to go for resolution. With only a few machines, the packaging may be skipped in favor of the old faithful shrink-wrapped Sneakernet distribution with manual adjustments at each point.

In larger shops, the same distribution that takes only a day or two in a smaller shop may take longer. The length of time depends on the processes, procedures, number of people involved, and the company's political environment. If something happens to the distribution, the solution may take a little longer as different groups will be involved in finding a solution. However, the entire process becomes cost and labor effective when there are multiple remote client end points.

As technicians, chances are you have a stronger interest in the bits and bytes of the distribution than the political side. Left to your own devices, the distribution would roll out without issues and be done rather quickly. The larger the enterprise, the more others need to be involved in the distribution—it is in these situations that things start getting sticky. Change management, version control, distribution notifications, corporate communication, business unit negotiations, Help desk staff, and support technicians all want a piece of the distribution knowledge. Rightly so, because they will all be impacted in some form. Of course, each of those departments will want documentation in a specific form about what is going to happen and how to support it.

Documentation

No matter the size of the enterprise, your distributions should involve some form of documentation. I can hear the common statements being muttered already: "My shop supports 15 machines and I'm the only one in IT, so documentation is a waste of my time. Besides, real techs don't need documents or manuals."

Sorry to burst your bubble, but RTFM (Read the Flipping Manual) does come in handy at times. Small shops share the same problem as large shops—turnover. You may opt at a later date to move on to other opportunities with other companies, but someone has to support what you are doing today. As large as the computer field is, the community itself is small. You will run into the same people time and again at some point. You may not have met the tech that was in your position before you, but you know who it was. Thus, after you leave your current position, the next person will know about you. All those complaints and problems you have because the distributions and packages were not documented can be corrected by doing some simple documentation as you perform the rollout.

Even if you don't leave your company, a wise old techie once told me that if no one else can do your job, you will stagnate and never move from that job. I had a great time doing DOS/VSE (mainframe stuff) in the early 80s, and at the time, could not imagine doing anything else. It's a little hard finding those kinds of jobs now. Make sure someone can cover you so that you can move on to other technologies as they come out. PP&D can help. OK, enough of Career 101.

Documentation is the backbone of computers. However, not many people like to write it—the developers don't have the time, nobody likes the way someone else wrote it, and the documents never have the data you need to solve your problem. Look at how many books and manuals and magazines and Internet articles are available for any major OS. Everyone has a better method of creating the documentation and solving a problem. So how do you create a document about a rollout so it appeases everyone? You don't. Don't even try.

Not exactly the response you were expecting, is it? Documentation is important. It has to be done. The larger the enterprise and the more groups that are involved, the more important the document becomes. In the first part of this book, some samples were discussed that showed the importance of documenting everything about a rollout. If you follow those guidelines, chances are that the necessary answers will be available when someone needs them. However, if you are a small shop, you may not have the time to devote to a detailed documentation endeavor. So where do you draw the line? How much documentation is needed and how much is too much?

Look at documentation as way of preventing interruptions in your non-work life. Vacations are only vacations if you don't really need to check in everyday to solve problems. If your company has standards already, you only need to fill out the necessary forms. If you are creating the standards, create documentation based on what you would like to see about a deployment you had nothing to do with. If you satisfy your thirst for deployment details in the document, the document is correct. Just don't assume that all facts are known to everyone. There are two general types of documentation you will need to provide: package and distribution.

Package Documentation

There is a point in documenting a package at which more time is spent than what is needed. You can judge a rollout by the size of the supporting documentation—the thicker it is, the hairier the rollout. Of course, you need to decide what you want in your application documentation. I suggest a cover sheet, conflict data, and content details.

The application cover sheet should cover these basic categories:

- Overview—The overview is a short one or two paragraph description of the application, where it goes, and what it does. Aim this description at the non-technical level so that everyone understands what the application does.
- Contact—This information explains who to call for problems or questions about the application and package. If the application is a custom piece supplied by a vendor, you should provide the vendor contact data, including any contract items, costs, or phone numbers. Be sure to specify if an internal escalation path should be used first; otherwise, you might find your allocated incidents being used to answer questions on how to change the desktop wallpaper.
- Prerequisites—What is needed on the target systems to get the application? Are there any customizations to be done to the application or package prior to distribution?
- Detailed Steps—Detail in short and concise sentences what needs to be done and when. No fluff, no explanation, simple step by step instructions. If the application is set to follow standards and is silent, say so. Screen shots are a good idea, within reason. Don't provide shots of the Welcome screen, only the pertinent shots of decision points or where data needs to be included. Remember that the document will be used in the heat of battle, so clear and concise information is ideal.
- Post Tasks—Identify anything that needs to be done after the application is installed. Use the step-by-step method again.

- Troubleshooting—In this section, list where to go to find solutions. If your package creates logs, list where they are located and what each log will do for the troubleshooter. If you know of any other places to look for help, write them down. Anything to help the troubleshooter. You notes during testing should help write this section. It's okay if this section is blank because you don't have any information.
- Process Overview—If you created a home-grown process, detail it here. What does each step of the script or .exe file do?
- Known Issues and Errors—Sometimes an application is rolled out with known bugs. List them here.
- Optional Items—Detail a checklist of steps to follow and provide script listings.

Figure 6.1 shows a sample cover sheet for a deployment of Adobe Reader 5.0. As you can see, the document is short and simple and provides basic information. It assumes that the reader knows the standard distribution process. This document could be expanded by listing the various scripts in detail or providing any additional requirements needed for the rollout. Because Adobe Reader 5.0 is free from Adobe, the vendor data was not listed. The point is that the overview document is a good spot to start with and provides the cover data for the application rollout.

> Corporate use of Adobe Reader may require licensing or other legal considerations. Be sure to contact Adobe (http://www.adobe.com) for details. This application was used as an example only.

Adobe Reader 5.0 Install

Overview
```
The Adobe Reader version 5.0 is used to read PDF files available in
various applications. The install on the workstation will ensure the
documents are readable without user intervention or the need to
manually activate a separate program.
```

Contacts
```
Help Desk   888-123-4567
Chris Long  123-456-7890 (Work) 602-555-1234 (Home) 602-555-9876 (cell)
```

Prerequisites
```
1) Ensure target system is online and corporate operating system image
is v.1.342 or greater.

2) Run the pre-scan against the target machine for the Problem
Definition Forms application. If the PDF application is resident on the
workstation, ensure the workstation name is appended to the Association
Fix script. The AF script will be called after the Adobe application is
installed to confirm the AF fix has been applied.
```

Detailed Steps
```
1) The application has been packaged for use with the standard
corporate distribution method.

2) Deployment targets will be identified by the Project Manager.

3) Schedule the distribution for after hours deployment
```

4) The installation is silent; no added functionality required by the user or deployment staff.

Post Tasks

1) Confirm the pre-scan collection of workstation with the Problem Definition Forms application have had the Association Fix applied.

2) Check the Central Reporting Server under \\CRS\Reports\Adobe\AF to confirm.

Troubleshooting

1) Installation logs are located at C:\Temp\Install\Adobe50
Shows installation steps and time of execution. Return codes are included.

2) Application logs have been modified to reside in C:\Temp\Apps\Adobe50
Any application created messages not available in the Event Viewer should reside here. Application specific.

3) http://www.adobe.com/support/readguide.html
Knowledge base for Adobe Reader

4) www.microsoft.com
Review the knowledge base for operating system generated errors.

Process Overview

The installation will create the Adobe Reader application and insert it into the Default User hive. Installation is silent

Known Issues

1) If the association to the *.PDF file extension is disabled, the application will not be called up. Recreate the association for the impacted user.

2) Conflicts between standard Adobe PDF extension and the custom created Problem Definition Forms application may cause conflicts to some users with both applications on their system. Manual startup of the Adobe files is recommended.

Figure 6.1: A sample deployment documentation cover sheet.

Automated methods of creating the supporting application documents are another good reason for a commercial application distribution system. Such a system can provide some of the data for you. Save the conflict-resolution details in a DOC folder that goes with the application data. Any logs that are generated during the package creation should be saved and stored with the application source files. This information gives you a reference to visually review when troubleshooting problems months later—or to hand off to someone else.

Listing 6.1 shows a copy of the file generated by Conflict Checker Professional (included in New Boundary Technologies' Prism Deploy product). Any conflicts listed would be investigated during testing to ensure no problems occur on the workstation images. Including this file in the application package provides added data for troubleshooting. The generation of the conflict file took less than a minute to create on the test system. Add another couple of minutes to save the file off to the application directory, and inside of 5 minutes (for this example) you have a record that will be with the package for the life of the application in your organization. Five minutes invested during the package creation that would take much longer to recreate months down the road.

```
System File Conflicts

---------------------
C:\deploy\AdobeFiveO_1.PWC
 AceLite.dll
  397312,04/16/2001 04:39:02 PM,1.2.0.1,"C:\Program Files\Adobe\Acrobat
5.0\Reader","C:\deploy\AdobeFiveO_1.PWC"
  397312,04/16/2001 04:39:02 PM,1.2.0.1,"System\Adobe\SVG
Viewer","C:\deploy\AdobeFiveO_1.PWC"
 Agm.dll
  1138688,09/05/2001 02:10:34 PM,4.4.26.1,"C:\Program
Files\Adobe\Acrobat 5.0\Reader","C:\deploy\AdobeFiveO_1.PWC"
  1138688,09/05/2001 02:10:34 PM,4.4.26.1,"System\Adobe\SVG
Viewer","C:\deploy\AdobeFiveO_1.PWC"
 Bib.dll
  147456,04/16/2001 04:39:02 PM,1.0.20.1,"C:\Program
Files\Adobe\Acrobat 5.0\Reader","C:\deploy\AdobeFiveO_1.PWC"
  147456,04/16/2001 04:39:02 PM,1.0.20.1,"System\Adobe\SVG
Viewer","C:\deploy\AdobeFiveO_1.PWC"
 CoolType.dll
  1441792,07/24/2001 08:02:54 AM,4.4.26.1,"C:\Program
Files\Adobe\Acrobat 5.0\Reader","C:\deploy\AdobeFiveO_1.PWC"
  1441792,07/24/2001 08:02:54 AM,4.4.26.1,"System\Adobe\SVG
Viewer","C:\deploy\AdobeFiveO_1.PWC"
 nppdf32.dll
  103344,09/10/2001 03:47:38 AM,5.0.5.452,"C:\Program
Files\Adobe\Acrobat 5.0\Reader\Browser","C:\deploy\AdobeFiveO_1.PWC"
  103344,09/10/2001 03:47:38 AM,5.0.5.452,"C:\Program Files\Internet
Explorer\PLUGINS","C:\deploy\AdobeFiveO_1.PWC"


Program File Conflicts
----------------------
C:\deploy\AdobeFiveO_1.PWC
 No conflicts found


Registry Setting Conflicts
--------------------------
C:\deploy\AdobeFiveO_1.PWC
 No conflicts found
```

Listing 6.1: An example conflict report file that you should include in documentation for later troubleshooting.

In addition to the conflict report file, having a detailed listing of what the application does to the workstation is invaluable further down the road. Again, a commercial application distribution system can provide this data for you. Listing 6.2 shows an edited file from Prism Deploy. (I edited the sample to save space, but it shows the different sections of information from the original file.)

You can see the type of file and registry modifications captured in the package. By including the data now when the package is being created, you have a viewable file to reference if you are caught in a troubleshooting session. If you had to recreate the file later, it would take time from your troubleshooting and delay the resolution of the issue.

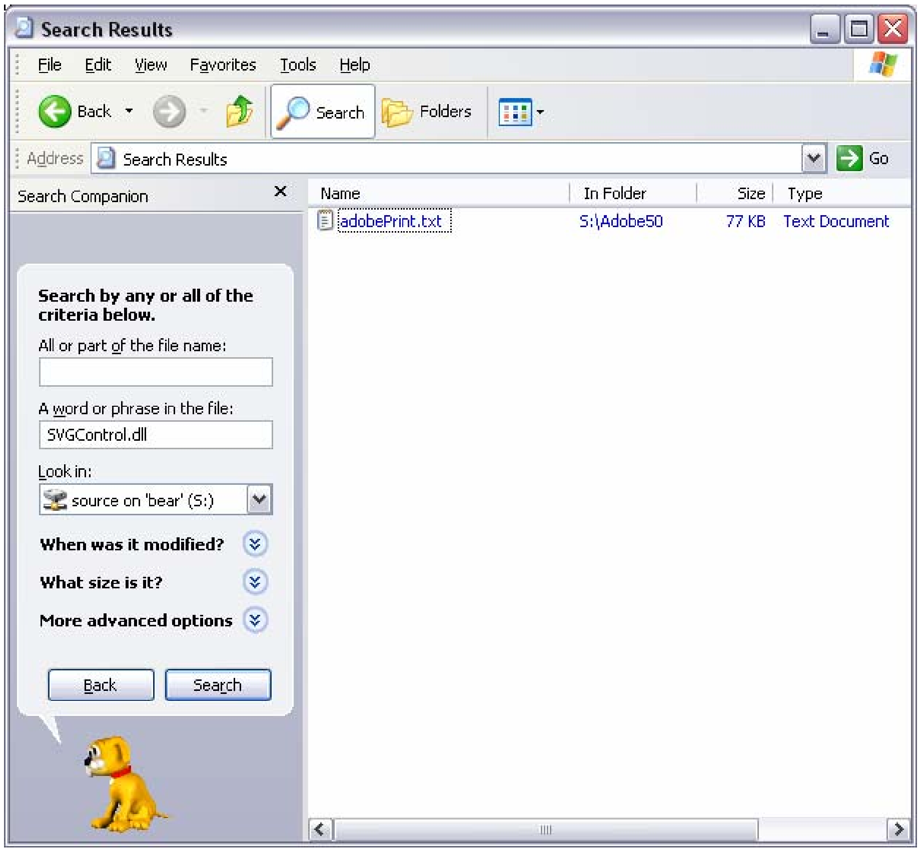
```
                                   CONTENTS

                           C:\deploy\AdobeFiveO_1.PWC

  C:
   Documents and Settings
    Admin
     Application Data
      Adobe
       Acrobat
        Whapi
         CreatePDFWinColor.ico           2KB   Icon
04/16/2001
         CreatePDFWinGray.ico            2KB   Icon
04/16/2001
         SearchPDFWinColor.ico           2KB   Icon
04/16/2001
         SearchPDFWinGray.ico            2KB   Icon
04/16/2001
         WHAppList.xml                  16KB   XML Document
08/17/2001
    All Users
     Desktop
      Acrobat Reader 5.0.lnk             1KB   Shortcut
03/23/2002
      |
      |
      |
      |

   Program Files
    Adobe
     Acrobat 5.0
      Help
       ENU
        ACROBAT.PDF                     28KB   Adobe Acroba...
09/10/2001
        DocBox.pdf                      82KB   Adobe Acroba...
08/01/2001
        MiniReader.pdf                  76KB   Adobe Acroba...
09/10/2001
      Reader
       AceLite.dll                     388KB   Application ...
04/16/2001
```

```
       ACROFX32.DLL                     52KB  Application ...
04/16/2001



      |
      |
      |
      |
  HKEY_LOCAL_MACHINE
   SOFTWARE
    Adobe
     Acrobat Reader
      5.0
       AdobeViewer
        EULA = 01 00 00 00
        TrustedMode = 00 00 00 00
       InstallPath
        (Default) = "C:\Program Files\Adobe\Acrobat 5.0\Reader"
       Language
        current
         (Default) = "acrord32.exe"
        next
         (Default) = "acrord32.exe"
     Adobe SVG Viewer
      2.0
       dir = "%WINSYSDIR%\Adobe\SVG Viewer\"
       path = "%WINSYSDIR%\Adobe\SVG Viewer\SVGControl.dll"
```

Listing 6.2: A sample of a detailed listing of what the application does to the workstation.

I keep stressing to have a copy of the data in a text-readable format in a central location for a reason. Speed to resolution in problem solving. For instance, several months and many applications after deploying Adobe Reader 5.0, you discover a custom application is having a problem with a DLL called SVGControl.dll. All you see in the pop-up screen is the DLL name, but no path or reference to where the DLL is being created. As Figure 6.2 shows, on your Windows XP system, you pull up the Search wizard, tell it to search the application source location and look for any entries that contain the string SVGControl.dll. The search pulls up the detailed file listing for Adobe Reader 5.0. You are now well along the path to resolving your problem.

A strong suggestion: If possible, keep the document readable via Notepad or WordPad. Complex documents needing Word look nice but are difficult to read on systems that don't have Word on them. You would need to move to another machine to read the document, a machine that may not be resident to your work location. You need to balance the level of detail with the ease of reading.


Search Results
File Edit View Favorites Tools Help
Back Search Folders
Address Search Results Go
Search Companion
Back Search

Search by any or all of the criteria below.

All or part of the file name:

A word or phrase in the file: SVGControl.dll

Look in: source on 'bear' (S:)

When was it modified?

What size is it?

More advanced options

Name	In Folder	Size	Type
adobePrint.txt	S:\Adobe50	77 KB	Text Document

Figure 6.2: The results of your search for troubleshooting information.

Distribution Documentation

After the package documentation is complete, you need to detail how the distribution is going to be done. This step is the form filling that helps get the package sent through the proper channels. You've already done the work, so you simply need to compile it into clear and short reports.

As Figure 6.3 shows, I've included a sample distribution document form for you to review. As you can see, the form takes up only one page, includes common answers that only have to be checked if needed, and provides basic data to get started. The document shows the form completed for a deployment of Adobe Reader 5.0. All this data is saved off to the application source server in the application directory. Of course, the source server is backed up so that all the data can be recovered when the server fails. (We all know that hardware fails, it is never a matter of if, only when.)

Filling out the simple form takes only a few minutes, and it provides the basic points from which to start a deployment. As new versions of the application are rolled out, updates are done to the document. You can create entirely new sheets, create a copy and update the new data, or scratch over the old data. Personally, I like creating a copy of the previous version, updating the changed information, versioning the document, and keeping all the versions in the same place. This way you can track changes by version.

Distribution Sheet

Item		Value	Notes
Name of Deployment		Adobe Reader	
Version		5.0	
Initial Deployment Date		January 19, 2002	
Base Package:		Commercial Vendor	Web download for source; free Reader version
Source location:		\\bear\source\adobe50	
Programmer (if custom):		N/A	
Manuals / documents		\\bear\source\adobe50\docs	
Dependencies			
	Applications	N/A	
	%PATH%	N/A	
	Specific Server	No special server needed	
	Variables	%USER%, %DPRTMNT%	
	OS	Win2K, Windows XP	
	Corp. Base OS Image	Version 1.342 or later	
	Reboot Needed?	No	
Deployment Method:		Standard	Check document for Problem Definition Form application issues.
Target Group(s)		All	Check with Project Manager for target list.
Deployment restrictions		Must be done after normal business hours.	Normal considerations; no special issues
Rebuild Instructions		Trigger standard repair first. If not successful, remove/install	

Figure 6.3: A sample distribution document.

So far we've created the following documentation for the application deployment:

- The overview document
- Conflict listing
- File / registry data
- Distribution sheet

Will these four documents solve any and all issues encountered with the application? Obviously, they won't. Will the documents enable a deployment-knowledgeable person to come along at a later time and deploy the application properly? They should—assuming the person is familiar with the deployment methods in use within the company. The documentation is not a primer on how to perform the basics of the job. If problems arise in the future, can the documents provide help in solving them? They should be able to provide a starting point for file or registry references along with listing any known issues.

If you are part of an organization that has been doing deployments for some time, your documentation requirements might be more extensive to fit the unique situations within your enterprise. If you are starting off, use the provided samples as a starting point and adjust as your needs grow. Now, let's move into a big part of the overall enterprise strategy—coordinating the multitude of changes.

Change Management

Any enterprise, no matter what the size, needs a form of change management. It is critical for the integrity of the applications and the environment. If possible, the change-management people should not be the same people that do the distributions. Otherwise it becomes a case of the fox guarding the chicken coop, and the entire process becomes a joke. Change management has a simple directive: Review all production changes that impact the production environment and ensure the changes don't cause outages.

That large umbrella directive is a daunting task. The larger the enterprise, the harder the task becomes. Let's start with a mom-and-pop shop. One person does it all. Because only one person is doing all the work, that person generally doesn't try to deploy applications while converting the entire shop from Token Ring to Ethernet. That one person understands that the Web server is not really down, it's just being upgraded with new hard drives and memory. Anything that happens in the small shop is generally known throughout the organization.

Now look at the large enterprise that employs 500 staff members in the IT shop. You sent out a deployment last night and are getting reports of problems this morning. Not from everyone, just from some end points. You start digging and after an hour (or more), you find only the clients that were homed to a specific server failed to get the dark window deployment. A little more digging, calling around, and casually talking to the other members of the department, and you find the server was down for maintenance last night during the dark window. Now you have to reschedule the deployment and perform the normal reviews again in the morning. Let's see: an hour for basic troubleshooting, 30 minutes to confirm the server is functional for your needs, 10 minutes for the reschedule job, another hour to recover the end clients that got only part of the deployment before the server went down, and 30 minutes to review the rescheduled deployment. Three hours and 10 minutes wasted. And don't forget the reporting time to management on the reason for the failure. Four hours gone from your life. Change management is a process (remember, PP&D) that works to prevent those wasted 4 hours. By coordinating with all areas that impact the production world, needless outages are eliminated. Ideally, I'll admit.

Because change management is a process it does slow things down. Having to go through another procedure and another person prevents the casual production changes that get tossed in on a whim. Because of the process, it also tends to stop some installs. A simple tweak may not get installed because someone decides the benefit of the tweak is not worth the effort of going through the process. This result could be a good thing.

One benefit of change management that is overlooked is that it helps prevent duplication of effort. If a change goes through the process, someone may realize it is identical to another project and tie them together. Otherwise, both projects could proceed and end up stepping on each other. Effective change-management enforcement will pay for itself quickly in terms of preventive costs. If you figure your current environment experiences three outages a week totaling 3 hours a week, you need to figure all the costs associated with finding resolutions:

- The number of full-time employees required for troubleshooting an issue.
- The number of full-time employees required to provide the fix (or fixes).
- On-call phone support of your staff (plus any off-hour call-ins).
- Lost income (real or potential) of the impacted application or system.
- Costs of vendor support, including vendor support calls; vendor call outs; and software creation, testing, and deployment.
- If the application or system is customer interfacing, figure in the cost of the lost goodwill from the customer. If the application directly impacts the customers, they will start becoming wary of using your systems if they can't be sure your systems will remain functional.

Do a spreadsheet on your own for the last outage you went through. Add up everything involved and try to associate a cost with it. Be sure to include the pizza and soda for the late night emergency work you did. If you haven't totaled up an outage before, you might be surprised how much it actually costs. By controlling the change process, enforcing effective peer reviews, ensuring all the back out or rollback plans are in place and workable, change management can reduce or eliminate preventable outages. That is the goal.

One comment about the peer review process. Make it valid. If your shop has only one or two IT people, the review is simple. "Hey Tim, check out the package and tell me if I missed something, would you?" is all it would take to get a package reviewed. If your shop is large, the review process may involve a separate department that could require an entire process itself (which might add days or weeks to the scheduling date). Remember I said that size does not matter? In this case, the practical application of the peer review is impacted by size, but the concept is not. The idea is that someone with acknowledged skills will review your entire application. This review is not a certification step in which the application is tested against all images and applications, but a review of the package. The peer review checks for syntax errors, logic flow, completeness of the documentation, effectiveness of the package (is the delivery method going to work?), and compliance with company (and/or legal) policies and standards. Even having a buddy check your work is a good thing to run through. We so often get swallowed up in one logic flow that we miss alternatives and overlook the simple mistakes. The situation is analogous to spell check. When spell check runs against a document, does it go *threw* it *two* check for proper word use? The previous sentence made it through the spell check, but is obviously using the wrong spelling. Another set of eyes will catch a lot of those simple mistakes.

☞ To really make sure the entire package is correct, send it to someone that you don't really get along with. There is nothing like having a known antagonist review your work to ensure it's correct!

A danger of the change-management process is having too much interference. When the process becomes a hindrance, a review of the process is required. Make sure all the steps that have been added bring a value to the process and aren't just included to make someone to feel important.

How does a change-management process work? In keeping with the simple directive, anything that will impact the production environment needs to be reviewed prior to implementation. Thus, changes and updates are scheduled long before they need to be installed. This time lets everyone review the change request. It also forces people to plan ahead. OK, OK, OK, I know. Forcing changes to be planned in advance is Nirvana. A strong change-management process will balance the needs of a true emergency with the need of a Project Manager to meet a looming deadline. As the change-management process gets better, the need for true emergency updates will decline.

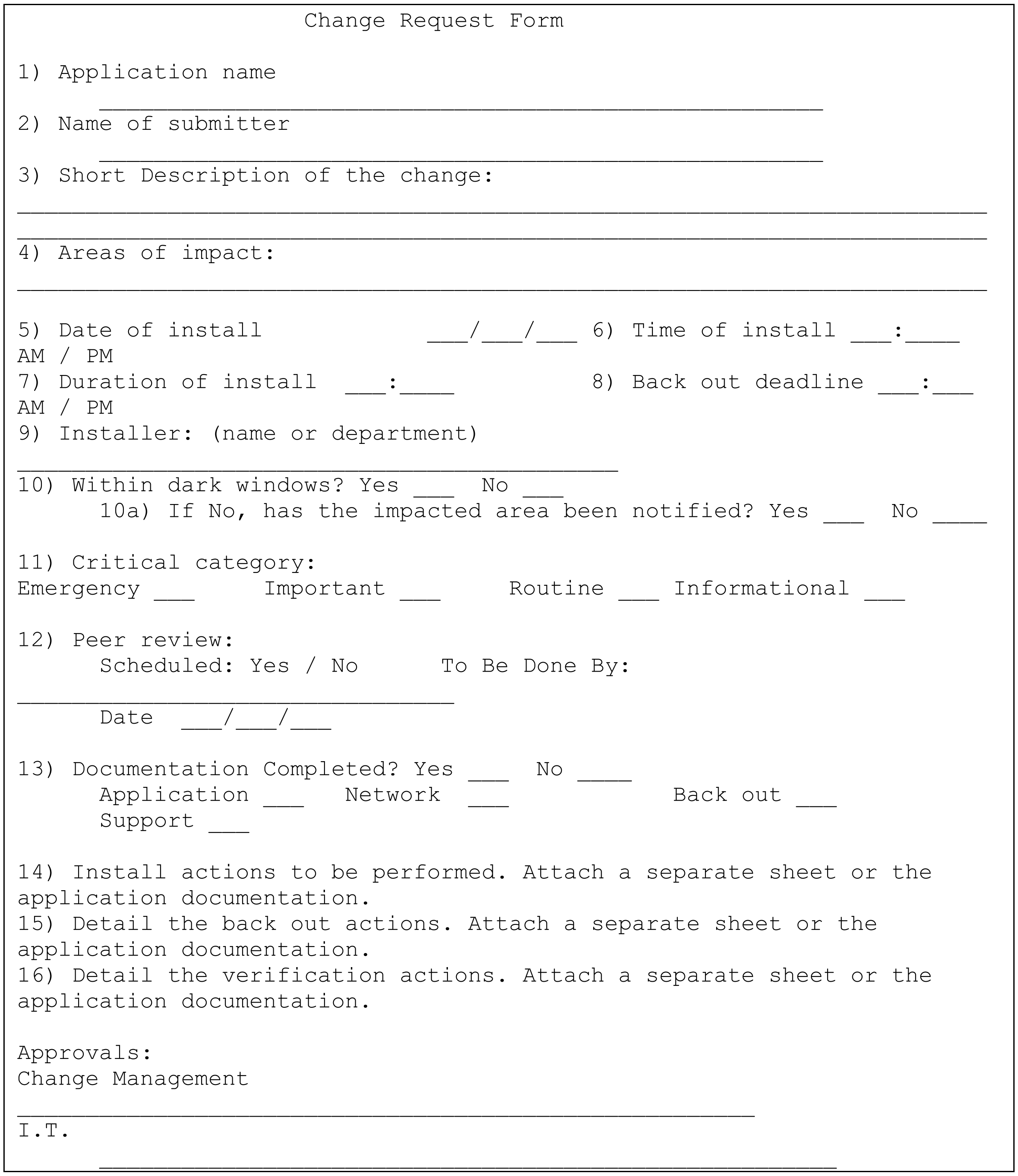
To staff a change-management process, you need to have at least two technical (underline that part) people in the shop. One peer reviews the other. Thus, the person creating the install does not do the review and schedule the change. In a two person shop, trading off the change-management review hat would be expected. Larger shops would have a dedicated person in the role. The change-management process reviews all documentation, thus forcing it to be completed on time. And the process ensures the documentation is useable, not just slapped together.

Change-Management Request Forms

Let's go over the change-management process in a general manner. Prior to applying any changes to the production environment, the change is submitted to the process in the form of a request. The change request consists of

- Any documentation (as discussed earlier).
- A description of the change.
- The scope of the change (how many clients, servers, users, and so on).
- Who exactly will be impacted? Which business units?
- Who is doing the change? (In a small shop, it will usually be the same person each time.)
- What steps are needed to implement the change?
- How is a rollback or back out performed? Is there any backup that needs to be done first?
- What are the dark windows? Is the change going to be within the accepted dark window or is it being done at some other time?
- How is a rollback decided on? Does the installer make the decision? Is there a drop-dead time needed to ensure there is enough time to perform the rollback?
- What steps are needed to verify the change and who does the change?
- Are any signatures needed for the change?

- Figure 6.4 shows an example form that you can use as a starting point, if you don't already have a change request form template.



```
                         Change Request Form

1) Application name
      _____________________________________________________
2) Name of submitter
      _____________________________________________________
3) Short Description of the change:
_______________________________________________________________________
_______________________________________________________________________
4) Areas of impact:
_______________________________________________________________________

5) Date of install            ___/___/___  6) Time of install ___:____
AM / PM
7) Duration of install  ___:____           8) Back out deadline ___:___
AM / PM
9) Installer: (name or department)
____________________________________________
10) Within dark windows? Yes ___   No ___
      10a) If No, has the impacted area been notified? Yes ___   No ____

11) Critical category:
Emergency ___     Important ___     Routine ___  Informational ___

12) Peer review:
      Scheduled: Yes / No      To Be Done By:
________________________________
      Date   ___/___/___

13) Documentation Completed? Yes ___   No ____
      Application ___   Network  ___              Back out ___
      Support ___

14) Install actions to be performed. Attach a separate sheet or the
application documentation.
15) Detail the back out actions. Attach a separate sheet or the
application documentation.
16) Detail the verification actions. Attach a separate sheet or the
application documentation.

Approvals:
Change Management
______________________________________________________
I.T.
      ______________________________________________________
```

Figure 6.4: An example change request form template.

The Change Management Coordinator

To properly implement a change-management policy, there must be a Change Management Coordinator. Don't rotate the duties; keep one person as the Change Management Coordinator.

This provides the consistency needed to make the policy work. If you must rotate the duties, keep the tour of duty 6 months or so.

The main function of the Change Management Coordinator is to do a cursory review of all the requests and ensure everything has been addressed. The Change Management Coordinator is a process coordinator with a good understanding of technology. The job's main duties include:

- Reviewing all the required documentation for completeness.
- Keeping the change calendar accurate and available for everyone.
- Ensuring that the proper people review a request for conflicts or problems.
- Distributing the requests prior to the formal scheduling meetings.
- Calling emergency reviews as needed.

A Change Management Coordinator must understand the technology enough to know whether the data they collect is accurate and to get the right people involved as needed. Could the Change Management Coordinator be an administrative assistant? Sure. Many of the duties involved compliment each other nicely. I've seen this double duty work in many shops; it all depends on the person doing the functions.

Flexibility is important to a good Change Management Coordinator. Not all change requests warrant a full process. Other changes may require added information. That is where the technical knowledge of the Change Management Coordinator comes into play.

If you are rolling out a registry change that adds text to the Winlogon\LegalNoticeText, the process is pretty simple. The change is not a major change to the production environment. A peer review would ensure the script was tested and does not accidentally apply the change to a different registry key. The back out is easy, so the overall issue is pretty simple. The Change Management Coordinator would confirm the basic documentation was in place, the peer review was done, and that the method of delivery is a proven method. Time of distribution is not a problem.

Now take the same change request and apply it to a service pack upgrade. In addition to the service pack install, you are going to apply added registry tweaks and modifications to workstations and servers. Now you have a complicated process that will take more care in reviewing and ensuring all the testing is done for the install and roll back. The Change Management Coordinator is going to carefully review the request and ensure the business part of the enterprise is aware of the request and expected install date. A little more work will be applied to the request.

Obviously, the two examples are on either side of the extreme scale. It is important to realize the change-management process needs that type of flexibility. If a rigid system was in place that followed a detailed process from step 1 to step 700, making simple changes would never happen. The balancing act is being able to adjust as needed without going too far to either side.

A good Change Management Coordinator will review all the steps, follow the procedures, and know when to call a halt to a process or just delay it. If a critical deployment is needed and the only thing amiss in the procedures is a missed field on a document, the deployment would not be held up. However, if the review process shows that problems are evident, the Change Management Coordinator should put a stop to the change until all the issues are addressed.

✎ The Change Management Coordinator does not need to be the sole decision maker. A committee can be used and usually is. The members of the committee can rotate depending on the technical knowledge needed of the request. In these cases, the Change Management Coordinator is acting as the chairperson of the committee, not the sole decision maker.

An effective change-management process becomes the de facto enforcement arm of the IT shop. Everyone will know what kind of fallout will happen by not following the process and that will prevent many last-minute adjustments or rollouts. And by forcing everyone to schedule the changes, those implementations that are not planned out will be reduced or eliminated. The key to success is having the proper management buy-in of the process. When someone circumvents the process, some form of retribution needs to be uniformly applied. And if the process circumvention causes an outage, the enforcement should be swift and mildly public—a town square flogging is probably a bad idea, but making it public knowledge that "someone on the project team" is in trouble might a good idea. Otherwise the change-management process is a waste of time for everyone. Either everyone follows the process or the process should be removed.

If a change is a true emergency, the change-management process should have a clearly defined method of streamlining the process to facilitate a time-critical change. Review that emergency process often to keep it accurate and not too burdensome or easy.

An Example Change-Management Process

It's time for an example of the process. Let's take the Microsoft security rollup package for Win2K Pro. You need to get the application out to a static group of users via your commercial application distribution system. The package has been adjusted and tested, and you have done all the due diligence required to ensure the security rollup package will deploy properly without issue. Your defined change-management process requires that you

1. Submit a request to the change-management group that includes an overview of the application. The overview is high level only and will be used for general reference during the change-management process.
2. Schedule the request one week in advance.
3. Attend a scheduling meeting to discuss the changes. The frequency of the scheduling meeting is dependent on the number of changes the enterprise experiences. Weekly is common for most shops. Larger shops in a dynamic environment often have daily meetings.
4. The Change Management Coordinator will review the required documentation before the scheduling meeting and distribute the information accordingly. Doing so gives the attendees time to review the documents and process before the meeting.

5. Get signatures from specific groups or people before the meeting. If another department will be doing the distribution, you need to have their buy-in that the proposed date and process is OK for their schedule. Because the security rollup package will require a reboot, you need to confirm that the timeframe is within an agreed upon dark window and that there are no changes to the window during your proposed time. The scheduling meeting also prevents conflicts between groups; your goal for your application is to ensure that the targeted machines are available. Scheduling a reboot on March 14th at 4:30 for the accounting department most likely will not win many friends for you (tax crunch time for the number crunchers).

You've submitted your proposal, have checked with major parties involved, and have your documentation in line. You've identified the rollback options and have had a peer review done. You've done the change proposal request and have all your ducks in a row. The date for the scheduling meeting arrives. Now it is time to sweat a little.

A good scheduling meeting should make you nervous. You know you will be facing a tough audience that will be checking whether you're prepared and you've done everything. You will also be placed in to a negotiation stance if there are conflicts with other changes.

Another strong payback to the change-management process is that you can schedule changes to prevent conflicts. If you are sending out changes and the network suddenly goes down, wouldn't it have been nice to know the routers were going to be undergoing firmware updates before you got part of your deployment rolling? The scheduling meeting will not only review the applications for completeness, but will also be looking for conflicts and overlaps in other changes. For that reason, it is critical to have a single scheduling meeting instead of offshoots for each discipline. One meeting for NT, one for UNIX, one for network, or one for mainframe most likely will not work as overlaps will occur.

The scheduling meeting begins and you are in the hot seat. The Change Management Coordinator brings up your request (gives a little evil laugh) and begins the review process. The group reviews the basic request, confirms that all the i's are dotted and t's are crossed, and confirms the request conforms to the standards and policies. Not technically, just in the procedural arena first. Because you are an efficient professional, nothing is amiss. Now the Change Management Coordinator begins the technical review. Because everyone in the meeting has reviewed the request before the meeting (we can only wish for Nirvana), you give only an overview of the request. You identify the groups that will receive the security rollup package, explain that it will be sent out Friday after business hours, note that it requires a reboot which might impact anyone working late, and tell the group that you will be in on Sunday afternoon to review the deployment for any issues that might cause problems Monday morning. The Change Management Coordinator opens the floor for general comments and the questions come in. After fielding a barrage of questions, the group admits that you have thought of all the possible problems and how to resolve them. Your request is approved for scheduling.

The Change Management Coordinator then checks the schedule and finds that the building you are going to be sending the application to will be experiencing a power outage on Sunday for normal maintenance. Your request is adjusted a little in that you will have to confirm the package deployment on Saturday afternoon or evening instead of Sunday afternoon. Although this change shortens the time of the deployment by a day, it also keeps you from wasting a trip into the office on Sunday. And your part of the meeting only lasted 10 minutes. In summary, the change-management process

- Acts as the enforcement arm for change control.
- Prevents conflicts in scheduling.
- Tracks changes.
- Forces compliance with company and legal standards and policies.
- Ensures documentation is complete.
- Ensures back out processes are defined.
- Keeps the flaky installs to a minimum.

Where is the Source?

Here is a difference between deployments in large and small enterprises: Where do you keep the source files for an application and how do you get it distributed? When I ask systems administrators this question, they quickly reply "Oh, it is stored on server X. Everyone knows that". Excellent. The next question: Is that source SECURED? The usual response is "Huh?"

Here is where the size difference comes in to play. Depending on the commercial application distribution software you use, you will deal with three general types of source locations for software deployment:

1. Application source—This source is a CD-ROM or floppy for shrink-wrap packages or a source location for in-house created software. This source is generally the unedited and unpackaged source data that is has not been modified for your distributions.
2. Deployment source—This source contains the edited and deployment-ready packages. It is sometimes called the Golden server.
3. Distribution endpoints—Your commercial application distribution software generally likes to have a place to find the software-modified source or your custom deployment solution has locations from which the files can be installed.

Dealing with the application source is easy. The files are generally read-only on the CD-ROM (or floppy or downloaded file). Smaller shops generally have a desk drawer in a systems administrator's desk designated as the place to put all the CD-ROMs. The larger the shop, the more sophisticated the storage location becomes. It moves from a desk drawer to a file cabinet. Not much difference, just a larger storage location. As the shop gets larger, the source files move to a server location for easier access between buildings. The source files are still resident on the CD-ROM, so if the server location gets wiped out or the files are modified, the original source files can be replaced. No problem so far.

Now we deal with source location number two—deployment-ready source. You have worked hard to create the package and have it ready to go. No matter which distribution method you use, that deployment-ready package needs to reside someplace prior to being inserted into the commercial application distribution software or your custom method. And the prepared package is usually the only copy around that includes all the final tweaks and modifications.

If anything were to happen to the package, your life would become very difficult. Worse, if something happens to only one file, you could spend days figuring out what happened. Not just a file deletion or version update that breaks things, but maybe a registry INI file is modified and a value is changed by one digit, and the change only happens after 75 percent of the deployment is done. Now you have a troubleshooting nightmare. The solution is easy. Ensure the package-prepared source files are read-only and limited as to who can modify them. If that means you burn a CD-ROM of each application and store it with your other CD-ROM source files or you lock down the central location server, the end result should be to protect those files from any modifications after the package is ready to go. If you rely on a server lockdown, check your backup strategy on a regular basis and confirm that the data is valid on the backup media. A couple of minutes each week to confirm that the backups are working will save you from trying to rebuild a package that was created last year.

I recommend a combination of the CD-ROM and server methods. After the package is setup and ready to distribute, I like to burn a CD-ROM copy of the application and store it in the central corporate store. If it is a package that I've created or worked on, another CD-ROM is created and stored in my desk drawer for my own archives. In large companies in which duties change and re-organizations occur on occasion, the source CD-ROM location can sometimes get lost in the shuffle. Plus, I'm paranoid and believe you can't have too many backups.

For the source server, you want to do two things:

1. Ensure everyone that needs access can get to the data when they need it.
2. Prevent unauthorized users from changing the data, even if by accident.

To accomplish these two goals, the source server needs some minor adjustments. The source server does not need to be a dedicated box (unless your volume is that large and warrants the cost). Because it only needs to provide file storage, nothing fancy is needed in the processor department. Sharing the load with other functions is a good cost-saving mode, which will endear you to the accounting folks.

Read-Only User ID

For the read-only user ID, create a userid that is used solely for read access to the source location directory. Set the password to something easy and don't change it. Ensure that the user ID is a simple one, either a domain user or machine resident, depending on the role of the server. For example, a user ID called SourceRead or AppView is generic enough to be remembered easily by the technical staff. You can set the password to be the same as the user ID to make it even easier to remember them both. I've seen implementations that have created department- or enterprise-level read-only IDs. That allows further segmentation of the source directory and who gets into it.

Why a user ID with a simple password or the same password as the user ID? It is easy to remember, won't be locked out as much, and is for general use by all parties. It keeps the unwanted out by virtue of the user ID and password, but still provides read-only access. You only keep out 50 percent of the computer population.

Set the directory permissions to read-only for that user ID. Make sure you leave an administrator user ID with full access (more about this requirement in a couple of paragraphs). Remove the Everyone group. Yes, you could also set the Everyone group to read-only. By requiring a separate user ID to access the data, you prevent the temporary mail clerk from downloading all your software when he accidentally stumbles across the source location. Using a specific group to do the same thing might work, assuming you restrict the user IDs in the group. You can use this method to track which departments or groups are using the directories.

Create a share name for the read-only user ID to connect to. I suggest that this is the exception to normal share creations. Limit the access to the read-only user ID—you want to make the access available to those that know of it, not to the entire company. (Unless that is your goal, then leave it wide open). Make sure your technical staff is aware of the read-only ID and password.

Of course, make sure your security staff is aware of the need for and the use of the read-only user ID. They should be aware of the process and the simplification of the password, as it might be contrary to the security policy for the company.

You need to ensure that other groups or user IDs have access to the directory. SYSTEM needs full access for backups and other basic OS needs. Be sure to remove Domain Admin or Administrators from the directory access. Why? You want to control who has change access to the directory. Create a separate group with full access. Call it CM or Release Mgmt or whatever you like. Place only specific users into that full access group and limit the number of those users. If the group is opened too wide, you defeat the purpose of edit control.

You have now accomplished both goals. Anyone that needs access to the source application files can get it with the read-only user ID while only a small group is available to configure changes to the directory and contents. You are one step closer to a Golden server concept in which the files resident on the server are a known quality and don't get accidentally modified (or deleted).

Be sure to turn on auditing for the server and for the directory and subfolders. If there is any discrepancy with the source content, you can use the Event Viewer to determine who was updating that particular location. It also helps to keep folks honest if they know they are being audited, so be sure to publish that piece of information.

Along the same lines of the packaged source location and including change-management functions, you may want to add a few procedures to protect this location. It depends on your environment. The larger it is and the larger the access to the source files, the more you need to track who is doing what. Some sort of paper trail is a good thing. Possibly include in the scheduling meeting when the files are to be moved to the application source server from the certification group. Nothing should be on the source location without being approved by the certification group. By having the responsible people agree when a package is ready to go, you add greater validity to the Golden server concept. The entire enterprise then knows the files located in the source server are Golden and are the source files for distribution.

If you place the applications from the certification group onto the Golden server immediately after certification, there is a chance that the application may be on the source too early. The application may not be scheduled for deployment until the following month and could be an update to an existing deployment. See the problem? The new data could be overwriting the old too soon or there may be dual applications on the Golden server. Which one does the field systems administrator use? The concept of the Golden server is to be the source location for all *production* applications. Therefore, don't move the files too soon.

Small or large, your enterprise should have the first two source setups: The original files and the Golden server distribution files. If your user base is widely dispersed (across the country or across the ocean), a geographically close copy of the source server would benefit everyone. It would reduce the network usage and provide faster local access to large files. Trying to download a ghosted image from Denver for every workstation you are installing in London would take a long time. Downloading the same image to London one time when the network usage is low (at night or on the weekend), then having the image creations calling the local server in London suddenly has increased the image speed, reduced the network bandwidth use, and reduced network conflicts with other business needs. Evaluate your connections, file sizes, and use to determine the best fit.

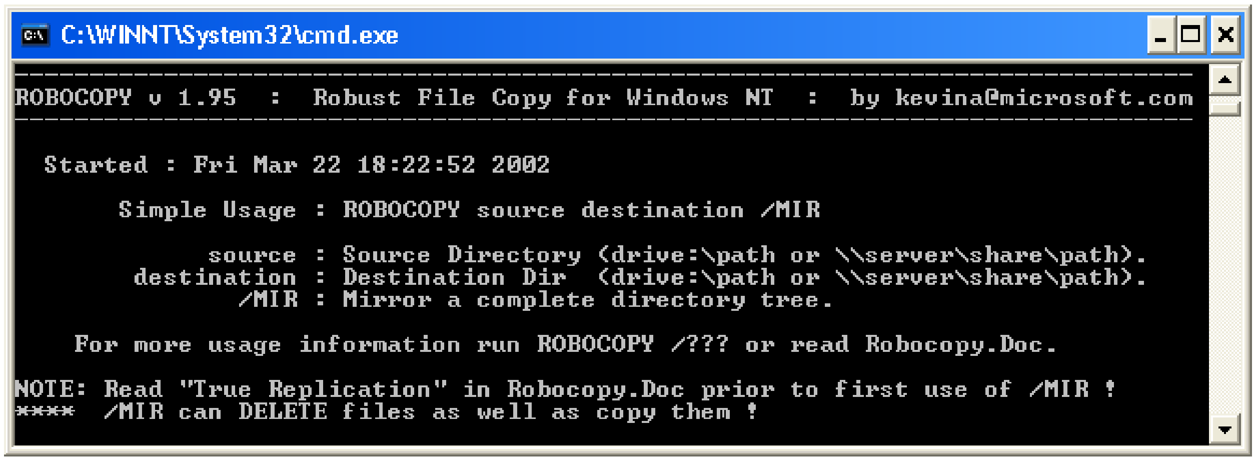
Mirroring

One issue with the duplicate source server is mirroring. Each source server must stay identical to the master Golden server at all times. Without fail and without question, the duplicate must be verified against the Golden server master on a regular basis. Lock down the duplicate servers just as tightly as the Golden server. Possibly even tighter if you can, so the only way that new files can be placed on the duplicates is from the master.

There are several methods available to provide the mirroring needed for the slave servers. The commercial applications have some enhanced features that allow for real-time mirroring. Mirroring means that any changes (additions, deletions, modifications) to the source location are duplicated to the slave locations. This functionality requires some added capital investment. Your ROI evaluation will determine whether mirroring is worth the cost for your situation. If you perform large numbers of modifications to the Golden server, having such a product is worth it. If you only make one or two (or less) modifications a week, the added cost may not be worth it.

One commercial replication solution is provided by Legato Systems. For information about RepliStor, check out http://portal2.legato.com/products/replistor/.

As Figure 6.5 shows, you can use the NT resource kit ROBOCOPY utility for replication. Use a release version of 1.95 or later, as it supports the /MIR (mirror) and /SEC (security) options. By setting up an AT command on the Golden master, you can have the master server replicate to the slaves on a fixed time schedule. Be aware that the /MIR option will delete files as well as copy, so make sure you have the source and destination parameters set up properly. It would be disastrous to mix up the two servers and have the Golden server overwritten by the duplicate!



```
-------------------------------------------------------------------------------
ROBOCOPY v 1.95  :  Robust File Copy for Windows NT  :  by kevina@microsoft.com
-------------------------------------------------------------------------------

  Started : Fri Mar 22 18:22:52 2002

        Simple Usage : ROBOCOPY source destination /MIR

              source : Source Directory (drive:\path or \\server\share\path).
         destination : Destination Dir  (drive:\path or \\server\share\path).
                /MIR : Mirror a complete directory tree.

    For more usage information run ROBOCOPY /??? or read Robocopy.Doc.

NOTE: Read "True Replication" in Robocopy.Doc prior to first use of /MIR !
****  /MIR can DELETE files as well as copy them !
```

Figure 6.5: You can use ROBOCOPY for replication.

Whichever method you use, be sure to copy the security settings on the files and directories so that you don't have problems. Otherwise, plan on adjusting the security settings after the copy process is done each time.

When replicating to slave servers, whether via a commercial product or ROBOCOPY or other means, don't rely on only replicating just the changes. Do a full replication of the master server to the slaves on a regular basis. That way, the consistency of the files is always confirmed. In a Golden server and slave server environment that experiences heavy source modifications, verify the consistency once a week.

Verification

If the commercial application distribution software you use can reliably replicate data without problems, verification is not needed. In fact, it may be impossible to force a replication or to update the data outside of the commercial application distribution software's method. You will need to carefully review the process and watch it, especially when you first install the software.

Verification is an issue. How to do it quickly is always a concern. WINDIFF, that infamous file-verification utility is a good way to check things, but its speed is not the best for large directories or over long distances. Check the Web for freeware utilities. ROBOCOPY works, as it only replicates files that it believes are different, though it tends to only check the filename, size, and date. Servers in different time zones can cause false readings. If you use the /XO (eXclude Older) switch, it will prevent some time zone issues. Test it for your situation.

The commercial application distribution software you use may have slave servers that it uses to speed up the distribution to the clients. Often the structure of the software's servers is not directly editable by anyone, so the contents are safe from accidental modification. If the commercial application distribution software places the files in standard directories, you always run the danger of having a file or setting modified by accident. Check into the software's documentation for information about having the files verified on the distribution endpoints to keep those accidents from happening.

Be careful about locking down a commercial application distribution software distribution point. Review your documentation carefully and check what access rights the software is looking for. If you lock down the distribution point to only your read-only user ID and a few select user IDs, the software (or GPO, if you use it) may not have access to directories. Make sure you don't lock the systems out. If your software relies on service user IDs, keep those IDs functional in the directories. Keeping SYSTEM at full access generally keeps the OS considerations functional. Test it before you roll it out.

If you don't use commercial application distribution software for deployment but still opt to have remote distribution points, you can use the same Golden server that I discussed earlier. Keep the access restricted and things should be fine. The main points about the source location are:

- Keep the source files locked from unauthorized modifications. Accidents happen. Don't assume the Domain Administrators will never adjust the files. Assume a mistake will happen and prepare for it.
- Keep the original source separate from the packaged deployment source.
- For replicated servers, verify the files and replicate changes on a regular basis.
- Track who makes changes to the files and keep that access to a very small group.
- Keep the Golden and slave (duplicate) servers consistent at all times. Don't let their reputation ever become tainted or it will be difficult to trust the contents in the future.

Hardware Thoughts

In previous chapters, I brought up hardware: I discussed the need to know your targets, ensure that you test against a good representation of what you have in the target list, and have a good understanding of the hardware drivers. The importance of this topic warrants a short recap of the subject in this chapter.

The question often pops up whether it is better to have a mass-marketed computer or to create a scratch-built unit. The discussion usually centers on the initial cost of the unit. The common argument states it is cheaper to go to the local PC store and put together a system out of parts on sale this week than spend the extra money to purchase a mass-marketed system. The technical folks usually argue that they can build a better system cheaper with more features than can be purchased in a pre-made configuration. To those statements, I always agree. By carefully shopping around on a regular basis, you can save a few hundred dollars in piecing a system together.

However, the issue is consistency. For the mom-and-pop shops that have maybe a dozen machines, piecing together a system each time you need one and saving money is the prime concern. Because in this type of environment most distributions are probably done via Sneakernet, saving the money is more important.

For any organization that uses an automatic deployment method and has several machines to deploy against, knowing what you have out there is more important than the initial cost savings. Having a consistent and known hardware environment is important as it saves the back-end costs of troubleshooting and having to adjust packages for one or two machines.

Suppose that your environment has 100 machines that were all built from spare parts. You don’t really know how many different video cards are out there, which driver versions there are, or even how many different manufactures of the cards exist. For example, if you roll out a service pack update, you can’t test the video driver effectively.

Now take that same 100 machines and figure that they are all from a major maker (insert your favorite hardware company). If it took 2 years to build up the 100 machines, you have a good understanding of what is out there. Major makers don’t bounce around much in the major components. You will see different versions of video cards but you can be pretty certain that the cards are from the same maker, thus the number of variances are smaller. And most companies work to keep the drivers backward compatible within a generation or two. Instead of having to locate the video driver for each card variance on the video card Web site, the PC maker may incorporate the variations in a driver download.

A side note about vendor hardware: watch the model lines. Buying nothing but Dell or Compaq or IBM machines does not mean the systems will be consistent between the various lines or within the model line. Different model lines within the vendor may have different end-of-life schedules for components. As a general rule, higher-end models aimed at the business world tend to stay consistent and compatible longer than the entry-level lines aimed at the consumer market. Do your research.

The point is that you can save money upfront to the tune of a couple hundred dollars (maybe) per unit and pay higher back-end maintenance costs in the form of distribution rollout troubleshooting. Or you can go for the longer term savings of lower maintenance costs in the rollouts with a slightly higher up-front cost.

The same goes for the servers. Don't build Frankenstein-type models and expect them to be the same from region to region or offer identical functionality. Keeping a known environment will solve many problems and you can adjust the packages and distribution methods accordingly beforehand because you know what issues to expect. Consistency is the mantra to follow.

Repeatable Processes

Big or small, an enterprise gets the best cost savings (money and labor) from deployment methods and procedures when the entire process is repeatable. The methods talked about in the past few chapters describe how to set up a distribution method and testing environment and how to get the applications rolled out. After you've done the first deployment, the subsequent methods are easier to use as the setup work is done.

The change-management methods discussed in this chapter—the process and forms—are time consuming for the initial setup. After they are developed, the same steps are followed each time. What may have taken a day to set up initially is now an executable step that takes only a few minutes.

When you are doing the design work, keep in mind that everything needs to be done again and again by others. Review the process with an eye toward a new person being able to read the documents and continue the process without problems.

Software Metering

Software vendors have an annoying tendency to nag you for payments. They like getting paid for all the copies of their software. Payment may be a flat fee, per user, per concurrent use, or by some other rather creative method. Every contract is different, and the same software license can even vary from company to company, depending on how sharp your purchasing department is. Compliance with the terms of the license relies on you. Paying for a one-workstation copy and sending it to 3000 workstations is not compliance. The software vendor and their lawyers would have issues with such a manner of deployment.

In recent years, software piracy has been raised from the classic stealing and selling of the software on the black market to the soft lifting of businesses not paying for all the versions they have. Unless the software license specifically allows it, a copy at the office is not permitted for home installation. Making a copy for archival purposes is allowed per the Copyright Act, Title 17, Section 117 of the US Code. If you aren't sure about the rules for your software, check the license very carefully.

Staying compliant with the software license is not just the venue of the budget department. With the publicity that the Software & Information Industry Association (SIIA) has been doing about reporting software piracy at work and other areas, doing so has become a revenge tactic for disgruntled employees. If the Software Publishers Association (SPA—a division of SIIA) comes into your shop, your company needs to show how compliance is being done with the license agreements. Suddenly, you are in a legal hot seat.

Check out http://www.spa.org/piracy/default.asp for details about SPA's report software piracy program.

You can use software metering to help avoid any conflicts. Software metering isn't just doing an inventory of what is on the workstations in your control, but actively controlling use in compliance with the various agreements. Life would be easier if all the license agreements were complimentary.

The big problem that faces you is wading through the myriad of agreement terms. For the licenses that require payment for each copy resident on a computer, all you need to do is gather your inventory and count the instances of residence. Most commercial application distribution software offers an inventory method, or you can do a simple check for critical files when a user signs on. However, if you are being charged by concurrent use or you can make a copy for the office and the home, the inventory solution won't work. Trying to figure if a copy is duplicated at a paired home and office system or how many copies are being used at the same time at any point can drive you crazy. How does your license agreement handle terminal server sessions? The software is on one system but is being accessed by multiple users. Does the one-copy per-machine clause work in this case or do you need a separate copy for each session? The issues keep getting bigger and bigger.

Software metering is one solution. It not only covers the inventory aspect but can also control who gets a copy of the program. Because your license may be based on active use, not just workstation residence, knowing which copies are actively being used is critical. Metering has been around for a long time. The idea is to monitor actual use, not just inventory residency. With most metering, a high-water limit is set and enforcement is either activated or set to a warning notice. For example, suppose application XYZ can have 3000 active users. When user 3001 tries to activate the program, the user is locked out from use until a license opens up or a warning is generated indicating the high-water level has been raised.

When you first install metering, it is a good idea to set it in a warning mode for awhile. You may have licenses for 3000 users but find active use is 4000 or even 2000. By setting the metering software to monitor and not lock out, you can find the true active use without upsetting users by suddenly and perhaps needlessly locking them out.

Monitoring software and not enforcing lockouts or license limitations is termed *passive metering.* Enforcing license limits is termed *active metering*.

When metering is first installed in a company, there will be problems and complaints when enforcement of limits is activated. Over advertise the need and reason for the software to ease the calls to your Help desk and managers.

Metering software requires an agent to be loaded on the client piece. If it is included in a suite such as SMS 2.0, the installation is easy. Otherwise, it is another service application to roll out. The client piece will contact a central controlling interface and relay the use on the local client. Even if applications are only resident on the workstation, the client metering piece contacts the central location for approval. Some metering software lets the application start while the confirmation is being negotiated. Should use exceed the limit, the metering client may cancel the application (annoying as it is). If the central location can't be reached, the application continues to run. This behavior allows offline use on laptops and the like.

☞ Each metering software vendor has its own methods for handling offline users. Check the vendor carefully when you are shopping for a product.

An important software-metering feature to be looked for is a VIP override switch. You don't want to prevent your CEO or CIO from opening Excel because of a software license limitation. Especially if it happens to be near the time for your raise. You do want them to get access but to have the use reported to you if it exceeds the license limit. Don't allow exceptions to the license limit. Instead monitor the excessive use and true up the license as needed.

The software-metering product should also report to you the number of users that are waiting for a free license to become available. To keep licenses available, a timeout parameter should be set so that a user who opens a copy of Excel on Monday morning shuts it down before Friday night. If the user actively uses the copy for the entire time, the user has a legitimate right to it. However, if the user simply uses the copy occasionally, a timeout parameter would catch that and remove the license. Check to see how your metering software handles inactive users.

Getting the money to purchase metering software can be hard. It's one of those essential items that does not bring a visible benefit to the organization. Instead, it constantly brings up the need for more licenses, is considered an invasion of privacy, and is a software cop that prevents people from freely using the software as they want. Try this selling point: It can save money by pointing out a reduction in licenses is needed: When you purchased the XYZ application, the price was based on concurrent use. Unless you knew exactly how many users would use it on a high-water mark, the license number was probably guessed at and guessed high. Installing a metering package and monitoring the active use, the high-water mark shows only 589 concurrent users out of 3000 licenses for 10,000 users. Add a little for growth, then return 2400 licenses. At $25 a copy, a savings of $60,000 is realized. Not a bad savings.

For a partial list of metering software, check out http://www.spa.org/priacy/asset/default.asp and http://www.appdeploy.com.

Summary

Deploying software in the enterprise is a process and procedure that needs to be documented. As techies, PP&D are areas we tend to avoid as much as possible. However, to have a deployment system that is workable without our direct involvement in all stages, the steps need to be known by others. The best way to get that known and repeatable process is to clearly document the policies of the deployment use, the procedures to create and implement the applications, and the entire process. If PP&D is effectively implemented, vacations can actually be vacations and not a series of check-ins.

Chapter 7: Installation Tool Alternatives

Being the expert software deployment technician that you obviously are, you keep abreast of all the new technology. You have a list of favorite Web sites to review, you keep up on Microsoft changes, and you know your workstations. Making new changes are not that big of a deal. So if a new process comes along that is not compatible with your existing environment, you have a bag of tricks available that will help you solve the problem, right?

No matter what type of deployment method you have in your environment, there will come a time when you will need to adjust or modify the standards. To that end, you need to stay up on the various methods and tools available to you. The biggest tools out in the Windows world right now are Windows Installer and the resource kit utilities.

Windows Installer

If you have a Win2K or later OS on your system, you already have access to some pretty good tools. You've heard of Microsoft Installer (MSI—it also goes by the name of Windows Installer and MSIEXEC, which is the name of the executable). In this chapter, I'll use the names Windows Installer and MSI interchangeably to reference the same technology.

For more information about MSI, you can do a search on the Microsoft Web site. Doing so gives you lots of hits with several key pages to check for further details. Do the same for Windows Installer on http://www.zdnet.com and you will get a large number of hits.

Microsoft has placed a major emphasis on the technology—most of their applications are written to support MSI. Other companies are following suit when they create their installer programs.

Check out http://www.appdeploy.com, which has several pages devoted to the MSI world, including a good cross reference that shows which packaging technologies are compliant (or use) MSI.

So what the heck is this thing everyone is using? In short, it is a tool that works to ensure the consistency and viability of an application. Application installation is controlled with a standard set of procedures. Elevating user privileges can be done within the process (if you are on the proper OS). DLL Hell becomes a quaint term from past nightmares. (Well, not 100 percent, but pretty close). By monitoring files, registry entries, and source locations, MSI strives to keep applications functional.

Windows Installer 2.0 will run on Win95, Win98, Windows ME, NT 4.0, Win2K, and Windows XP. Though native with Win2K and Windows XP, a version of the installer service is available for the earlier OSs. Functionality is a little different for the older OSs and not all options (such as advertising) are available for all versions; you may need to do some minor upgrading to get all the goodies. Using a combination of an internal database, files, and comparisons, Windows Installer can ensure that an application has all the components needed to run the application as you packaged it. Applications can be repaired, updated, and upgraded using internal logic to the technology without having to go through massive design hurdles to design the logic flow.

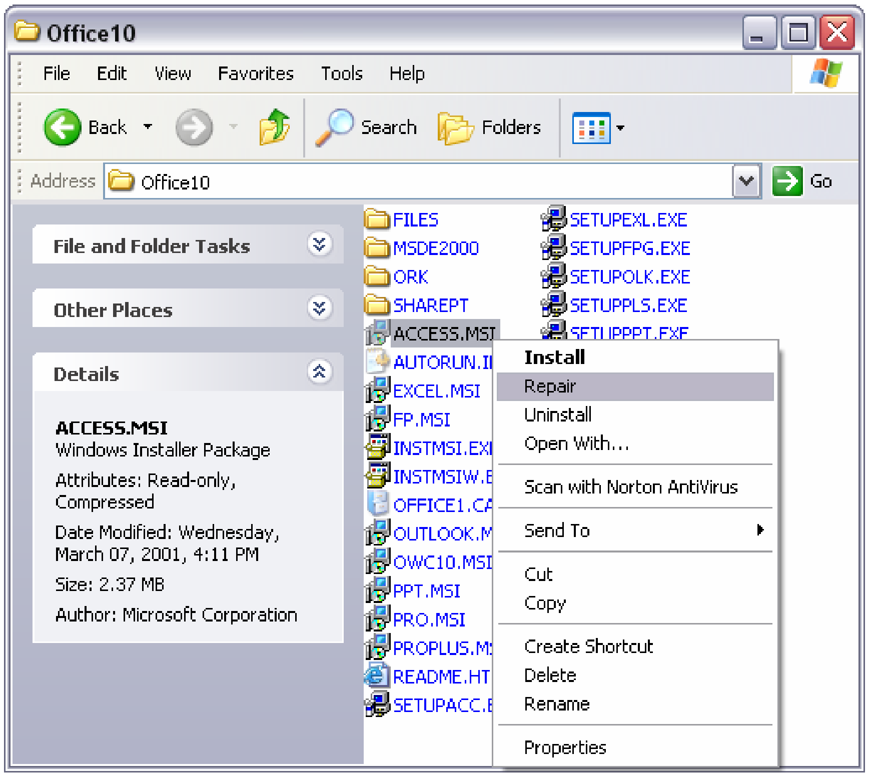
Like all technology, Windows Installer is not 100 percent foolproof. If an application is not designed for MSI use, even Microsoft states that the repackaged application will only be installed via Windows Installer and not truly managed by installer rules. I'll explain this caveat in more detail later.

Overview

Before I get too far into a discussion of Windows Installer, an overview of this technology is needed. Windows Installer 2.0 is the current release version and runs as a 32-bit or 64-bit service, depending on the OS and the hardware platform you are using. For 64-bit use, the Intel IA64 hardware platform is required along with either Windows XP Pro or Windows .NET Server. 64-bit applications are not available for installation on a 32-bit OS. Because Windows Installer 1.1 is 32-bit based, most of the install packages available today are 32-bit. Any application that is engineered for 64-bit use will be clearly marked as such.

Windows Installer uses a series of files and databases to perform its functions. The primary package is contained in a file with an .MSI extension. This file contains the database and the instructions and data needed by the Installer service to properly manipulate the application. An .MSI file contains details about how to install, upgrade, or remove the application plus how to change features in the application. If you look at an MSI-compliant application suite (such as Office XP), you will see several *.MSI files listed in the setup directory. If you right-click one of the files, you will see additional options in the pop-up menu: install, repair, and uninstall, as Figure 7.1 illustrates. This pop-up menu gives fast access to the canned functionality of the package. Notice that I said canned functionality.

From the users' point of view, an MSI file gives greater flexibility for installs than past mass technologies. But only to the limit the install developer is willing to offer the user. The true power of Windows Installer lies in its nuts and bolts. All those setup files make adjustments to the way the application is installed. The various parts of the application are called *elements* in the world of MSI.



ACCESS.MSI
Windows Installer Package
Attributes: Read-only, Compressed
Date Modified: Wednesday, March 07, 2001, 4:11 PM
Size: 2.37 MB
Author: Microsoft Corporation

Figure 7.1: The pop-up menu options for the MSI install file are different from the standard pop-up menu options.

To understand the technology, I need to dig a little deeper into the terminology and logic of Windows Installer. Within an MSI file, details about how to deal with the various elements are included. An element is one of three general containers: components, features, or products.

- Component—A collection of the files, registry keys, shortcuts, or databases that are to be worked on, components are considered required for install or removal and are acted on together, not separately. Components are the smallest of the elements and are the base level. Picture them as small modules with only a couple of files, registry entries, or OLE registrations (or other actions) inside.
- Features—Working up the size ladder, a group of components is called features. During the install (or uninstall) process, a user chooses a feature to act on. The feature would call the proper components to perform the action. Opting to install Word Perfect Help when you install MS Word is an example of selecting a feature.
- Product—This element is just as you expect—a collection of features create a product. MS Word is a product comprised of features.

A suite is a collection of products. The Microsoft Office suite is a collection of products. One variance to the logic; if a native Windows Installer package is used to control the setup of the products, the entire setup is considered a product, not a suite. So if you install just Word via Windows Installer, it is a product. If you install Office and include the applications Word, Excel, PowerPoint, Access (and everything else), then the entire Office install is considered a product and the individual applications are the features. If you install each application separately, the grouping is a suite. The key is the Installer package; any major grouping within the Installer package becomes a feature of the Installer package. As Figure 7.2 shows, each element builds on the previous piece.

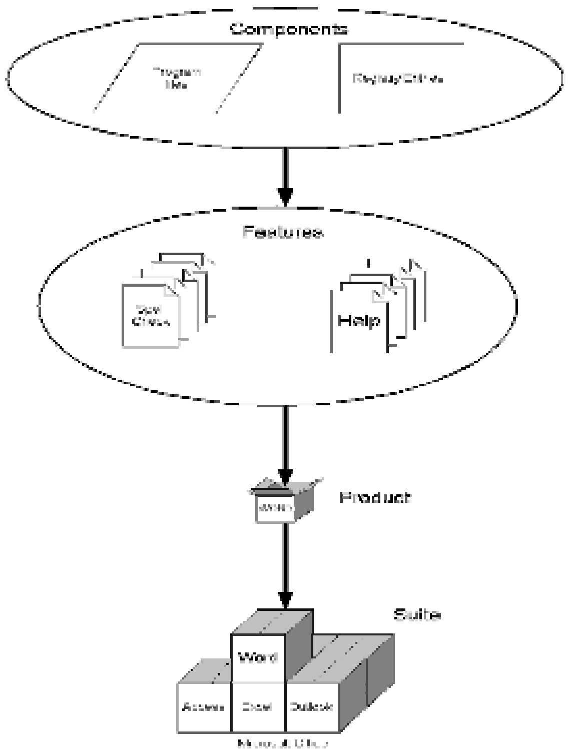


Figure 7.2: Windows Installer element hierarchy.

The component level is the basic building block. One of the functions for which MSI uses the component is verification of package consistency. In other words, when a Windows Installer–based application starts up, Windows Installer can do a verification to ensure that nothing has been removed or overwritten. For the verification function, Windows Installer does the check at the component level. In the component, one of the files or registry entries is designated as the *keypath*. When the application does a call to the component, the keypath is the part that is actually called rather than each file. For this reason, the components are usually small.

If a file that is defined within a component is deleted, the call to the component identifies that the file is missing; the component also defines where to go for the proper source to restore the component pieces to operational level. That proper source location is called the *installation point*. This is the location from where the application was originally installed.

Can you have a single file as a member of multiple components? Sure, if the file reference is created within the component properly. And that is the catch—the component must be defined properly during the creation process.

From your point of view as the administrator, knowing that a file is referenced by multiple components is good troubleshooting basics. When a package is removed, knowing why a file is removed (or not) can help determine why another application is suddenly breaking.

If a file is part of both Component A and Component B, and you removed Component B, the system would know that the references to Component A still exist. For files that are shared between products, the applications include the same component. Microsoft says that the components are small pieces and only include resources that are directly related to the function of the component. The MSI internal database records that two applications are using the file. When one of the applications is removed or changed, the database tracks the change and strives to keep the modification from impacting the other application. In other words, a series of building blocks are tied together by the Windows Installer service, and the Installer service acts as the traffic cop to keep everyone moving properly.

As a software deployment expert, you are familiar with the term GUID. Windows uses these identifiers extensively to keep track of the numerous aspects of the operation of the computer. To keep track of which application is using which component, a GUID is assigned to each component and product. The technical terms are *component code* and *product code*, respectively. I won't get too deep into a discussion about the use of GUIDs; just remember that a Windows Installer database tracks which component code (such as a file) is assigned to which product code (such as an application).

Windows Installer removes files from a system only with the other elements of the component and when the magic reference counter (*refcount*) shows 0 for that file. Thus, removing an application that is controlled by Windows Installer does not accidentally remove a file from another Windows Installer application. In Chapter 3, I discussed the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs registry key? That key does a reference count on how many applications are registered to use a DLL. The Windows Installer database does much the same, only instead of just listing the number of associations, it can actually tell you which application is using the file. This tracking extends past the DLL tracking and includes any component installed via the Windows Installer process.

Features group the components together into a logical collection. Within the installer file, the user chooses which features to work with. Instead of the user (or you) having to figure out which file and registry element is needed to work with those features, Windows Installer does the work for the user (or you) by referencing the components. So selecting the WordPerfect Help feature in Microsoft Word will install the proper components, which means that the proper files, registry entries, and hooks into Word are installed. The grouping makes life easier for everyone.

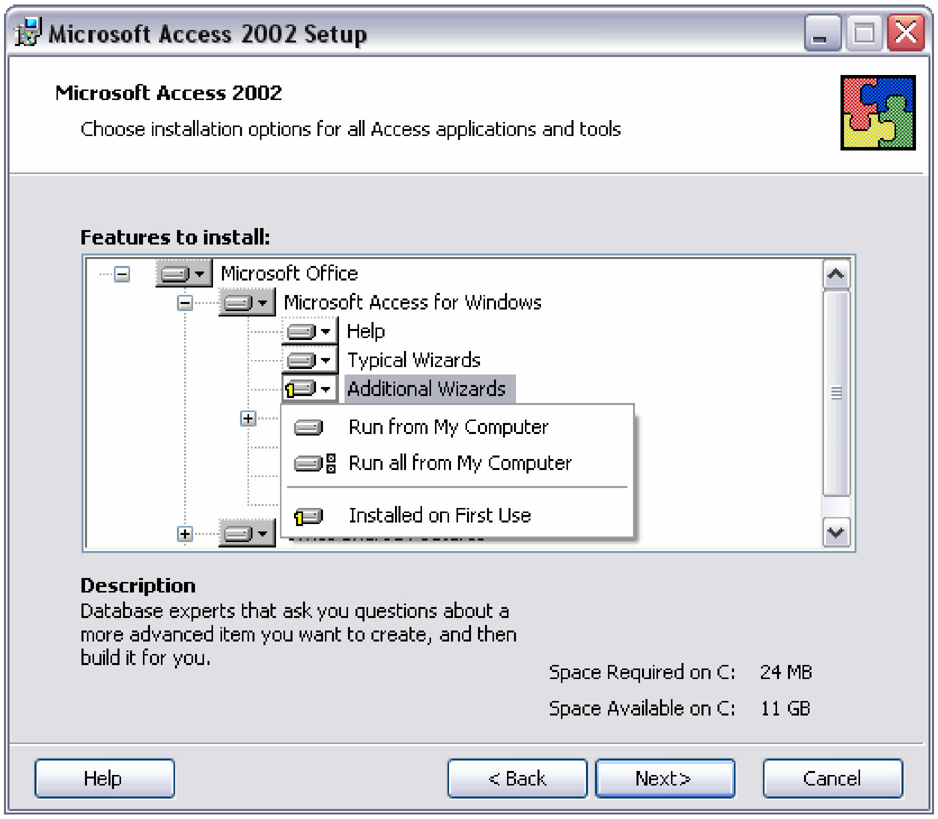
When an application is installed, Windows Installer will provide four main choices for each configurable feature:

- Installed on local hard disk
- Installed to run from source
- Advertised
- Not installed

The first two options are pretty clear. You can install the application to the local hard drive to run it locally or choose to install (or execute) the application from the defined source. The source location could be a server or a CD-ROM depending on how you have the application set up. If you need to save space on a workstation and don't have issues with network bandwidth, opting to install and run from the source might be the best option for you. Within a product, you could mix the install options depending on how you see the need. For example, you could install all the basic features locally to provide faster processing for the commonly used items. Additional fonts, wizards, templates, and other lesser-used features could be left to install to run from source (or install on first use). This setup provides a good mix for the end user—faster response for most items and access to additional features as the user needs.

Advertising applications is a heavy concept in the software distribution world. When you advertise an application in AD deployments, you essentially are telling the user that an application exists but the application is not yet installed. When the user clicks the file icon, an install program is activated to drop the program onto the machine or userid. Basically, consider MSI as repairing the entire application because none of the components are present on the system.

In Windows Installer, the same concept is available at the feature level. In Figure 7.3, which shows a custom install for Microsoft Access 2002, notice the disk icon with the number 1 in it. This graphic represents that the feature is to be installed on first use. Thus, the components are not yet installed on the system, but the keypath in the feature's component defines where the files are located for install. Shortcuts may be installed and OLE registration may be done, depending on the configuration needs.


Microsoft Access 2002 Setup

Microsoft Access 2002

Choose installation options for all Access applications and tools

Features to install:

- Microsoft Office
  - Microsoft Access for Windows
    - Help
    - Typical Wizards
    - Additional Wizards

Run from My Computer
Run all from My Computer
Installed on First Use

Description

Database experts that ask you questions about a more advanced item you want to create, and then build it for you.

Space Required on C: 24 MB

Space Available on C: 11 GB

Figure 7.3: Feature installation options.

In the highlighted graphic of the snapshot, the Additional Wizards feature is only a ghost entry on the computer; the added wizards don't exist yet. In the course of using Access, a call to a wizard that is not installed may be encountered. At this time, the Windows Installer kicks off, presents a pop-up window indicating that the system is going out to get the added files (components), and the feature is installed.

You can see how this functionality saves space on a system. Why load a bunch of features if you don't use them? Having the space taken up on the drive by unused features only wastes the space. If you think a feature might be used, loading it might be worth the tradeoff. Older technology and some other systems give you the option to install a feature set or not. Because you might have a need for a feature at some point, not loading a feature meant going through the setup process again if you discovered that you need the option. This process is time consuming—it's a hassle to find the data source and a major interruption to work. By having the advertise ability within Windows Installer, the application loads in the feature with only a few minutes delay. This discussion assumes, of course, that the source is located someplace convenient and the new feature is not a major tweak that requires reboots.

Windows Installer Transforms

All this data and more are included in the .MSI file. The .MSI package is the base file system for a Windows Installer application and will install an application in a manner and method that the installation creator chooses. If you want to adjust those install options, a *transform* file is needed to effect those adjustments without having to modify the vendor's MSI. The file extension of .MST identifies transform files.

A transform lets you make changes to the base .MSI files at install time. Instead of letting users adjust the installation features to their whims, you can create separate transforms for various features, depending on the need. Creating a transform for different departments is a common example. The Accounting Department does not need the same features of Word that the Warehouse Department needs. By applying your group distribution methods, each department could call a Windows Installer application that loads specific transforms to Word the first time that Word is loaded.

Transforms are applied when an application is first installed. Transforms created from .MST files are not used for making modifications to existing applications; those changes are done via .MSP (that is, patch) files.

When doing GPO distributions in a Win2K Server or .NET Server environment, transforms are the supported method for customizing the base application during the MSI-based installation process

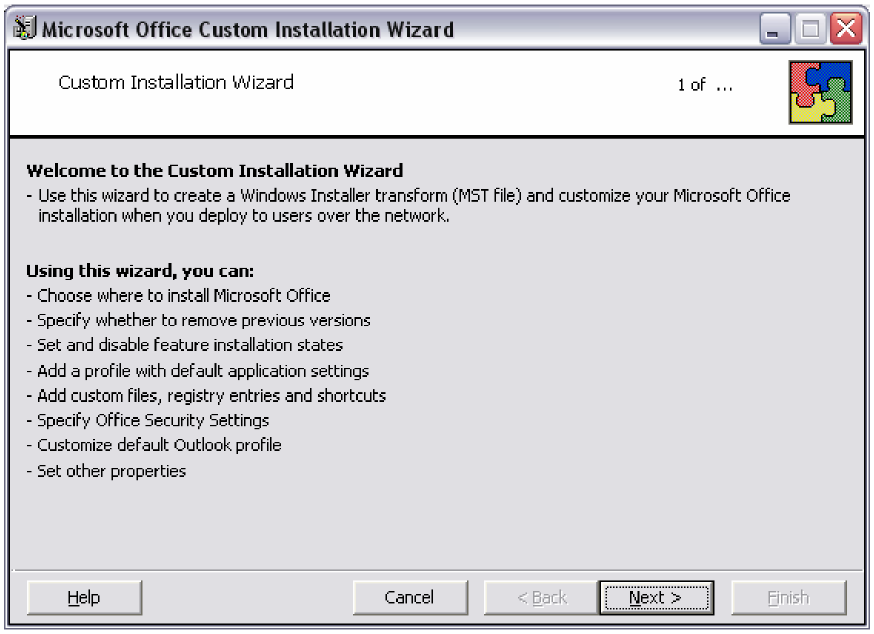
Transforms give you access to feature sets as if you were doing the install. After the transform modification is saved, you can create a different transform from the same base .MSI application that could have an entirely different feature set. Saving the transforms with a reference in the name to the reason for the modifications will help you manage the variances. This way, the Word .MSI application can have a unique install for both the Accounting Department and the Warehouse Department without having to create entirely separate packages for each department.

If you are deploying a commercially available application via Windows Installer, a method of creating transforms is sometimes included. Now you will understand why I've been using Microsoft Office products as the examples for this chapter. The transform tools are readily available for you to use while you get a good understanding of Windows Installer.

The Office XP resource kit and toolbox Web site (http://www.microsoft.com/office/ork/xp/appndx/appc00.htm) has the utilities to do customizations.

The Office resource kit tools are located about halfway down the page. The executable is called OrkTools.exe. (And yes, the executable is a Windows Installer setup.)

After the resource kit is installed, you can begin the customization by opening the Microsoft Office Custom Installation Wizard. The welcome splash screen, which Figure 7.4 shows, gives you the overview.


Microsoft Office Custom Installation Wizard

Custom Installation Wizard 1 of ...

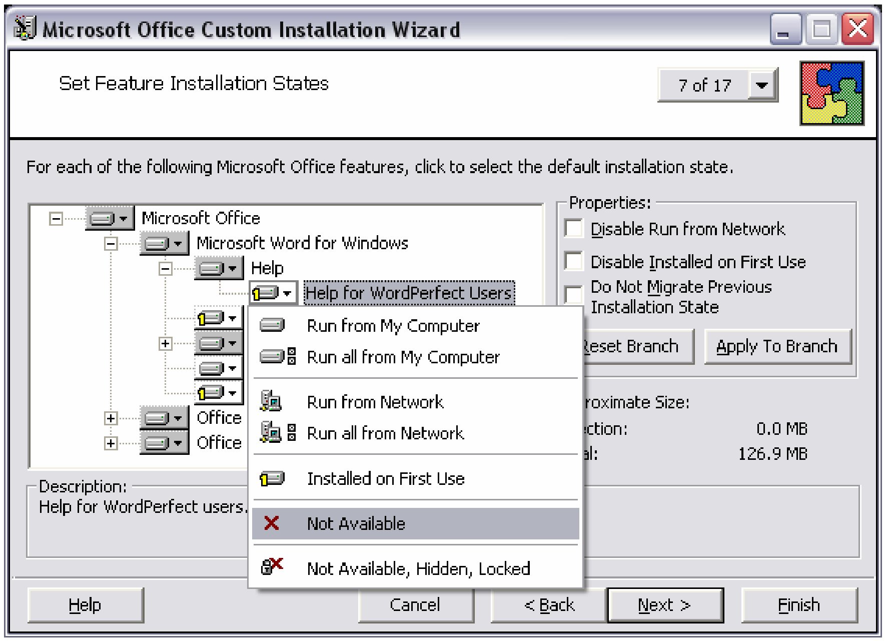
Welcome to the Custom Installation Wizard
- Use this wizard to create a Windows Installer transform (MST file) and customize your Microsoft Office installation when you deploy to users over the network.

Using this wizard, you can:
- Choose where to install Microsoft Office
- Specify whether to remove previous versions
- Set and disable feature installation states
- Add a profile with default application settings
- Add custom files, registry entries and shortcuts
- Specify Office Security Settings
- Customize default Outlook profile
- Set other properties

Figure 7.4: This screen gives you an overview of the functions that the utility will perform.

The next series of screens guides you through the process of selecting the base .MSI file, naming the MST file and location, and selecting various options. You don't need to save the .MST file to the same location as the .MSI file. In fact, if you are going to do many transforms for the base application, you might want to create a separate directory in which you can save the transforms.

I've selected the word.MSI file stored on my server for customizing and have instructed the program to save the transform file called WordTest.MST to the C:\RTP directory. For this example, I've changed the feature installation state for the Help for WordPerfect Users to Not Available, as Figure 7.5 shows. All other features remain as they are currently configured.


Microsoft Office Custom Installation Wizard

Set Feature Installation States 7 of 17

For each of the following Microsoft Office features, click to select the default installation state.

- Microsoft Office
  - Microsoft Word for Windows
    - Help
      - Help for WordPerfect Users
  - Office
  - Office

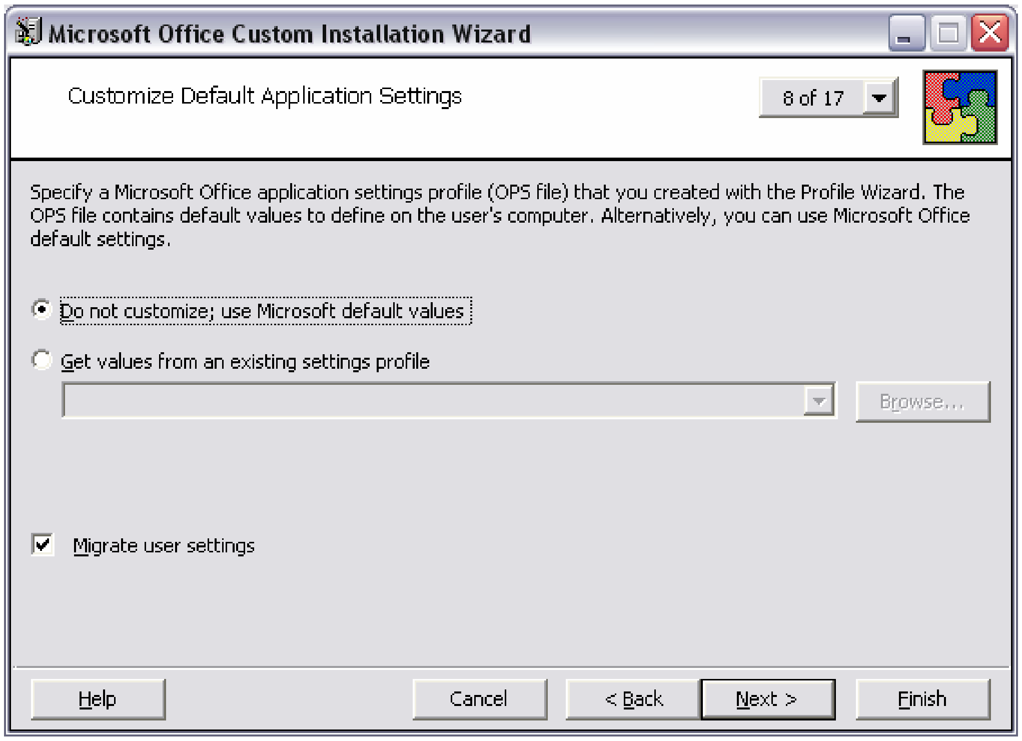
Properties:
- Disable Run from Network
- Disable Installed on First Use
- Do Not Migrate Previous Installation State

roximate Size:
ction: 0.0 MB
al: 126.9 MB

Description:
Help for WordPerfect users.

Figure 7.5: Disabling the feature Help for WordPerfect Users.

One type of Microsoft Office–specific file needs a quick mention at this point; the OPS file. This file contains default custom user settings for applications that are applied to a computer. These settings are mostly the configuration options in the Tools, Options menu and include such common items as toolbar use, menu options, and positioning of toolbars. You can create the file with the Profile Wizard from the resource kit, and import the settings into the transform. If you're not familiar with or don't want to use the Profile Wizard (an administrator-level function), you can get the same results from the Save My Settings Wizard (which is for the user level). What is configured is dependent on the INI file settings. (See the Help file in the Profile Wizard for more details.) In this case, I've opted to use the Microsoft default settings, as Figure 7.6 illustrates.


Microsoft Office Custom Installation Wizard
Customize Default Application Settings
8 of 17
Browse...
Help
Cancel
< Back
Next >
Finish

Specify a Microsoft Office application settings profile (OPS file) that you created with the Profile Wizard. The OPS file contains default values to define on the user's computer. Alternatively, you can use Microsoft Office default settings.

● Do not customize; use Microsoft default values

○ Get values from an existing settings profile

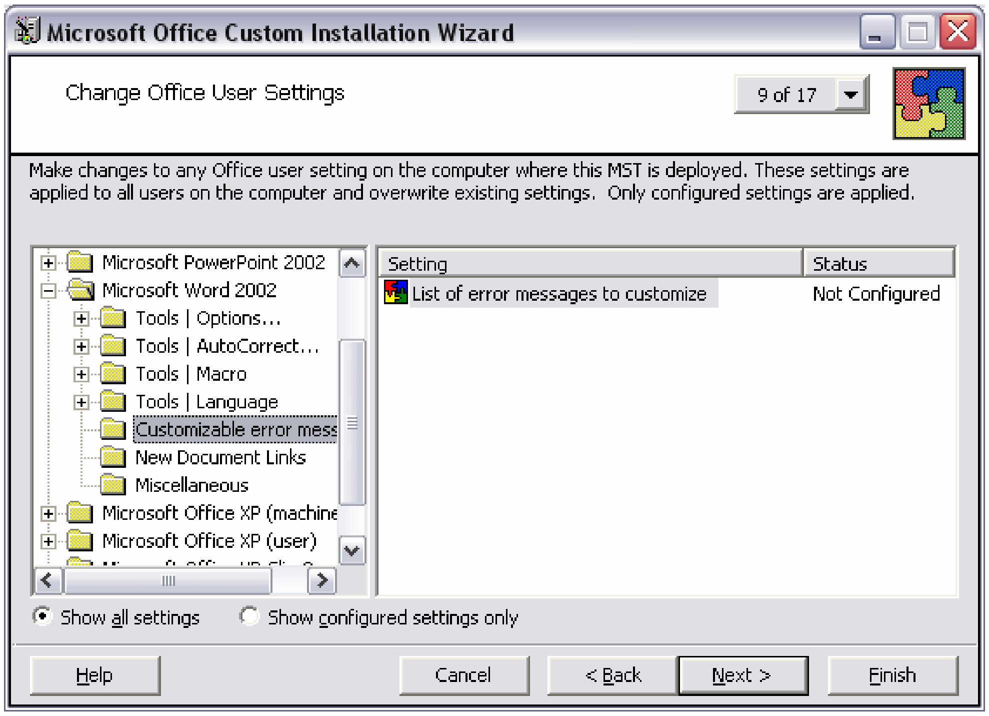
☑ Migrate user settings

Figure 7.6: Default settings page.

The next major screen, which Figure 7.7 shows, gives you the option to customize several user settings for the products that the transform supports. The screen indicates that *These settings are applied to all users on the computer and overwrite existing settings*. After the transform is applied, the settings become active.

Be careful when you are in this screen. If you change a setting and decide that the setting isn't correct, simply backing out or canceling the action does not really mean the change has not been written to the transform.

Be sure that what you have changed is what you really need to use. Click Help and review the information in the screen for more details.


Microsoft Office Custom Installation Wizard
Change Office User Settings
9 of 17

Make changes to any Office user setting on the computer where this MST is deployed. These settings are applied to all users on the computer and overwrite existing settings. Only configured settings are applied.

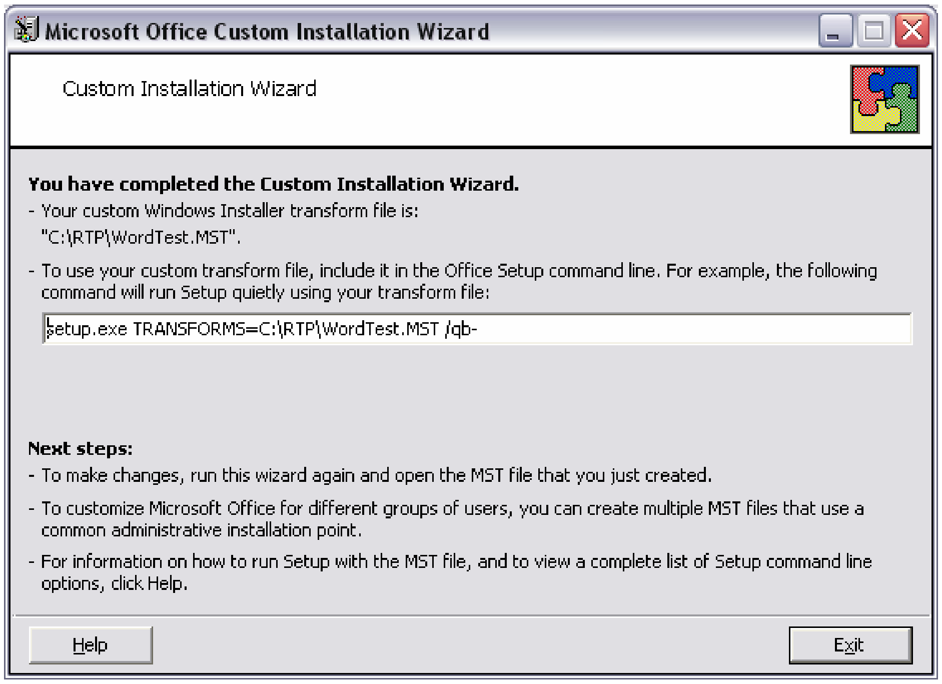
- Microsoft PowerPoint 2002
- Microsoft Word 2002
  - Tools | Options...
  - Tools | AutoCorrect...
  - Tools | Macro
  - Tools | Language
  - Customizable error mess
  - New Document Links
  - Miscellaneous
- Microsoft Office XP (machine
- Microsoft Office XP (user)

Setting	Status
List of error messages to customize	Not Configured

(●) Show all settings ( ) Show configured settings only

Figure 7.7: In this screen, most of the configuration options are changed in the Office XP transform editor.

Subsequent screens step you through adding or removing files, adding or removing registry settings, modifying security settings, specifying any server locations, including additional programs, and choosing any setup properties. You then reach the end, and by clicking Finish, save the .MST file. At the end of this transform-creation process, a sample command line is given to you about how to include the transform, as Figure 7.8 shows.


Microsoft Office Custom Installation Wizard
Custom Installation Wizard

You have completed the Custom Installation Wizard.

- Your custom Windows Installer transform file is: "C:\RTP\WordTest.MST".
- To use your custom transform file, include it in the Office Setup command line. For example, the following command will run Setup quietly using your transform file:

```
setup.exe TRANSFORMS=C:\RTP\WordTest.MST /qb-
```

Next steps:

- To make changes, run this wizard again and open the MST file that you just created.
- To customize Microsoft Office for different groups of users, you can create multiple MST files that use a common administrative installation point.
- For information on how to run Setup with the MST file, and to view a complete list of Setup command line options, click Help.

Figure 7.8: The summary screen provides a sample of the MSI command line needed to include the .MST file.

For a complete discussion about Windows Installer properties, download the Windows Installer SDK at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/wiport_6gf9.asp or http://www.microsoft.com/windows/reskits/webresources and select Windows Installer SDK).

If you needed to create other transforms that are similar but have minor adjustments, open the same .MST file, make your adjustments, and save the transform with a different name. This method lets you easily create multiple transforms that have different default save locations.

Transform creation GUIs are different between applications and vendors. The previous example was used to help show the process. Different vendors will create different transform utilities (if at all), and even within the same vendor, the various products could have their own GUIs that differ greatly from each other. Which features you are able to modify depends on what the developer is willing to create for you. Normally, if the option is part of the install process, the transform GUI will have the same option defined.

Can the MST file be viewed without the transform utility? Only if you enjoy looking at garbled data. The Office XP resource kit has a Windows Installer Transform Viewer that lets you view the data in a Notepad format. You need to specify the base package (.MSI file) and the associated transform (.MST file). The WordTest transform that I created (which had only a single change within it) produced Listing 7.1 from the Transform Viewer. All the settings listed in the file will be applied during installation of the Office.MSI file if the transform is specified in the setup command line. You can also specify the MSI file in the GPO customization tab when you define the package properties in the GPO. Take some time to review the listing. You can see the GUIDs of the product codes clearly listed along with the setup properties. The section in which the Help for WordPerfect Users change is listed is boldfaced in this listing. You can use listings such as this for troubleshooting problems.

```
Enforce Validation Flags: False
Base package: S:\Office10\WORD.MSI
      ProductCode: {901B0409-6000-11D3-8CFE-0050048383C9};
ProductVersion: 10.0.2627.01; UpgradeCode: {001B0000-6000-11D3-8CFE-
0050048383C9}

Transform: C:\RTP\WordTest.MST
Expected values - ProductCode: {901B0409-6000-11D3-8CFE-0050048383C9};
ProductVersion: 10.0.2627.01; UpgradeCode: {001B0000-6000-11D3-8CFE-
0050048383C9}
      Validate Major Version
      Validate Version is Equal
      Validate UpgradeCode
      Ignore Error of AddExistingRow
      Ignore Error of DeleteNonExistingRow
      Ignore Error of AddExistingTable
      Ignore Error of UpdateNonExistingRow


CREATED - OCW_Opt_Props DisplayName (s128*)      Value (S0)   ActualName
(s128)       Type (I2)
  +   ALLUSERS     2     ALLUSERS    8
  +   ARPCOMMENTS        ARPCOMMENTS 12
  +   ARPCONTACT         ARPCONTACT  12
  +   ARPHELPLINK http://www.microsoft.com/support    ARPHELPLINK 12
  +   ARPHELPTELEPHONE         ARPHELPTELEPHONE  12
  +   ARPNOMODIFY        ARPNOMODIFY 14
  +   ARPNOREMOVE        ARPNOREMOVE 15
  +   ARPNOREPAIR        ARPNOREPAIR 16
  +   COMPLETEINSTALLDESCRIPTION    <Default>
      COMPLETEINSTALLDESCRIPTION    11
  +   CUSTOMINSTALLDESCRIPTION      <Default>
      CUSTOMINSTALLDESCRIPTION      11
  +   DEFAULTREMOVECHOICEDESCRIPTION      <Default>
      DEFAULTREMOVECHOICEDESCRIPTION      11
  +   DISABLESCMIGRATION            DISABLESCMIGRATION      17
  +   NOFEATURESTATEMIGRATION       NOFEATURESTATEMIGRATION 18
  +   RUNFROMSOURCEINSTALLDESCRIPTION     <Default>
      RUNFROMSOURCEINSTALLDESCRIPTION     11
  +   RUNFROMSOURCETEXT <Default>   RUNFROMSOURCETEXT 11
  +   StrContactInfo          StrContactInfo    12
  +   TRANSFORMSSECURE        TRANSFORMSSECURE  19
  +   TYPICALINSTALLDESCRIPTION     <Default>
      TYPICALINSTALLDESCRIPTION     11
  +   TYPICALINSTALLTEXT      <Default>   TYPICALINSTALLTEXT      11
  +   TYPICALUPGRADEDESCRIPTION     <Default>
      TYPICALUPGRADEDESCRIPTION     11
  +   TYPICALUPGRADETEXT      <Default>   TYPICALUPGRADETEXT      11

CREATED - OCW_Strings   Name (s64*) Value (s128)
  +   Msi_ProductVersion      10.0.2627.01
  +   Msi_UpgradeCode   {001B0000-6000-11D3-8CFE-0050048383C9}

ALTERED - Feature Feature      Feature_Parent    Title Description
      Display     Level Directory_  Attributes  Lock (I2)
      OCW_DisableRFS (I2)     OCW_DisableJIT (I2)
 <>   WORDHelpForWPFiles                          100{1}
      4{20}

DATA CHANGE - _Validation     Table Column      Nullable    MinValue
      MaxValue    KeyTable     KeyColumn   Category     Set    Description
  +   Feature     Lock  Y      0      1                             Lock
  +   Feature     OCW_DisableJIT     Y      0      1
      OCW_DisableJIT
  +   Feature     OCW_DisableRFS     Y      0      1
      OCW_DisableRFS
  +   OCW_Opt_Props      ActualName  N
      Identifier        Actual Name of OCW_Opt_Props property-like
record
  +   OCW_Opt_Props      DisplayName N
      Identifier        Display Name of OCW_Opt_Props property-like
record
  +   OCW_Opt_Props      Type  N                              Integer
      Type of OCW_Opt_Props property-like record
  +   OCW_Opt_Props      Value Y                              Text
      Value of OCW_Opt_Props property-like record
  +   OCW_Strings Name  N                               Identifier
      Name of OCW_Strings property-like record
  +   OCW_Strings Value N                               Text        Value
of OCW_Strings property-like record

DATA CHANGE - ActionText      Action      Description Template
  +   OCW_Create_AppDataFolder      Removing temporary working files

DATA CHANGE - Binary    Name  Data
```

```
  +   OCW_CPYF.DLL      Binary.OCW_CPYF.DLL

DATA CHANGE - CustomAction    Action      Type  Source      Target
  +   OCW_Create_AppDataFolder      1     OCW_CPYF.DLL
      _OCW_Create_AppDataFolder@4
  +   OCW_INST_LOC_NEW  51    INSTALLLOCATION
      [ProgramFilesFolder]\Microsoft Office
  +   OCW_INSTALL_MP_ADDED_CONTENT  51    OCW_INSTALL_MP_ADDED_CONTENT
      yes
  +   OCW_INSTALL_MR_ADDED_CONTENT  51    OCW_INSTALL_MR_ADDED_CONTENT
      yes
  +   OCW_INSTALL_UP_ADDED_CONTENT  51    OCW_INSTALL_UP_ADDED_CONTENT
      yes
  +   OCW_INSTALL_UR_ADDED_CONTENT  51    OCW_INSTALL_UR_ADDED_CONTENT
      yes
  +   OCW_SET_COMPANY_NAME    51    COMPANYNAME RealTimePublishers

DATA CHANGE - Feature   Feature     Feature_Parent    Title Description
      Display     Level Directory_  Attributes
 <>   WORDHelpForWPFiles                              100{1}
      4{20}

DATA CHANGE - InstallExecuteSequence      Action      Condition
      Sequence
  +   OCW_Create_AppDataFolder      (NOT AppDataFolder) AND
(OCW_DestDirUsesAppData OR OCW_UserSettingsUsesAppData OR
OCW_AddFileUsesAppData OR OCW_ShortcutUsesAppData)    799
  +   OCW_INSTALL_MP_ADDED_CONTENT  $OCW_Added_Content_MP=3 AND (NOT
OCW_AC_REGKEYPATH_FOUND_MP)   1399
  +   OCW_INSTALL_MR_ADDED_CONTENT  $OCW_Added_Content_MR=3 AND (NOT
OCW_AC_REGKEYPATH_FOUND_MR)   1399
  +   OCW_INSTALL_UP_ADDED_CONTENT  $OCW_Added_Content_UP=3 AND (NOT
OCW_AC_REGKEYPATH_FOUND_UP)   1399
  +   OCW_INSTALL_UR_ADDED_CONTENT  $OCW_Added_Content_UR=3 AND (NOT
OCW_AC_REGKEYPATH_FOUND_UR)   1399
  +   OCW_SET_COMPANY_NAME    NOT DONTUSEOCIWORGNAME  799

DATA CHANGE - InstallUISequence     Action      Condition   Sequence
  +   OCW_Create_AppDataFolder      (NOT AppDataFolder) AND
(OCW_DestDirUsesAppData OR OCW_UserSettingsUsesAppData OR
OCW_AddFileUsesAppData OR OCW_ShortcutUsesAppData)    999
  +   OCW_SET_COMPANY_NAME    NOT DONTUSEOCIWORGNAME  999

DATA CHANGE - LaunchCondition Condition   Description
  +   ReleaseType <> 1  [CANNOTRUNCIWTRANSFORMS]

DATA CHANGE - Property  Property    Value
  +   CIWVersion  10.0.3228
  +   COMPANYNAME RealTimePublishers
  +   DISABLEFEATURECONDITIONS      Yes
  +   OCW_INST_LOC_NEW  [ProgramFilesFolder]\Microsoft Office
  +   REMOVELPK   0
  +   TransformType     Customization
 <>   SKIPREMOVEPREVIOUSDIALOG      1{0}
```

Listing 7.1: Transform viewer output.

Could you manually make modifications to the .MST file and execute it that way? You could. However, unless you are sharp on the Windows Installer variables and values for the application as well as the interrelationships of the entries, I suggest sticking with the GUI method.

Windows Installer Patches

Remember that the .MST file is used for initial setup tweaking. After you have an application out to your clients and you want to make adjustments, Microsoft says that you need to create an .MSP file. The patch file (.MSP) modifies the .MSI information. The advantage of the .MSP file is the ability to modify the .MSI file without requiring a fresh install of the application. The patch can be a simple data change or a complete upgrade. Whereas the .MSI file contains all the databases and data stream of the application, the .MSP contains a database transform that modifies the .MSI database. Patches are generally created by the initial developer of the application and deployed using your usual deployment methods.

Although you, as the deployment administrator, might never create a patch file, understanding how it works when you get one from your application vendor is needed for your troubleshooting skills. In shops in which multiple job functions are combined, the packaging side of your duties might require you to create a patch file.

Fixing Damaged Applications

In the past couple of chapters, one of the many minor points that was raised is how to repair an application remotely. If a user removes files accidentally or other applications overwrite files needed for another application, how do you deal with the resulting damage? If you write your own distribution method, this situation can reduce the number of hairs on your head very quickly. Commercially available delivery systems (CADSs) have numerous methods for handling repairs. Some work, some don't. Windows Installer addresses this issue.

Two main functions are available in Windows Installer files when working on damaged applications: Rollback and Repair (or Verify). Rollback is putting a system back to a known pre-install state. During application installations, if the install fails, Windows Installer controlled setup will attempt to remove all the changes back to a known state. Windows Installer keeps track of the changes as they occur and will create a rollback file for replaced files. After an install is complete, the rollback files are removed from the system. Rollbacks are generally performed automatically by the setup logic.

A rollback is not always possible. If changes to a schema are performed, the changes are in the system for good or bad. If an application is running with custom programs, the logic of the custom program needs to account for the rollback possibilities. You are at the mercy of the developer in those cases.

Repair is a part of the verify process. When a product is launched, it automatically does a component verification based on the keypath data of the component. If the keypath is not in a known state, Windows Installer attempts to replace the keypath data based on the source location in the component.

A little clarification on this point is needed. As always, the level of performance that the MSI process does is dependent on the level of intelligence that the developer applies to the package. If the developer of the MSI package does not specify full repair abilities (such as version and corruption checking), the repair option will check for existence of the keypath and apply a repair process only if the keypath is not there.

You can also trigger a repair function from the command line. By specifying the repair property within the parameters of your command, you can force the application to do a full repair. However, if the user has customized the application and you have not taken this fact into consideration, the user's tweaks could be removed and replaced with the data from the .MSI and .MST files. Of course, that may be the goal of the repair. Using this feature works great in training situations in which you want to return an application to a known state between classes. Be sure to test the repair function and fully understand what is being done before you trigger it globally.

Check out http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/wiport_6gf9.asp for details about repair functions. There are several considerations for using this feature properly, including the ability to prevent rollbacks.

Windows Installer Elevated Privileges

A major advantage of Windows Installer packages is the ability to elevate the security context of the install process. If your users are locked down, running installations at their security level could prevent the successful installation. With Windows Installer, you can adjust the security context for the duration of the package installation. Because users of Win95, Win98, and ME already have administrative rights on their systems, elevating privileges is not available to those folks. For the rest of the Windows users, this ability solves many problems for distributions.

Right now, to fully use the elevation principle, you need to be using Group Policy distributions within Win2K Server or .NET. Other deployment methods are beginning to support the elevation concept but implement it differently. Check the documentation on your CADS for the details.

To turn on the elevated rights option, adjust the local registry (the same key applies to the HKEY_LOCAL_MACHINE hive):

HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Installer

AlwaysInstall Elevated = REG_DWORD:00000001

Creating a Windows Installer File

If you have a package that is not a native Windows Installer–based application and would like to take advantage of the functionality of the Installer functions, there are many methods available to do the conversions. Several third-party applications are available and more are coming along all the time.

You can get a list of third-party applications that support .MSI files on the Web. Authoring tools are also included in the lists. To get you started, check out http://www.appdeploy.com, http://www.microsoft.com, and http://www.zdnet.com.

Many of the major software companies provide support for creating Windows Installer files in some form or another. If you decide to start creating your own Windows Installer setup applications, make sure you fully understand your needs and review the products available before you settle on a packages. A few of the companies that have Windows Installer creation capabilities include New Boundary Technologies, Wise Solutions, InstallShield, Microsoft (of course), and Veritas.

New Boundary Technologies Prism Deploy provides the ability to create Installer packages. After you create a Prism Deploy package and save it as a Portable Windows Change (PWC) file, you can create the Installer file. This method gives you some advantages over a standard Windows Installer package. You can use the snapshot abilities of Prism Deploy to get an accurate package. Using the Adobe Acrobat 5.0 package created in Prism Deploy from a previous chapter, I converted the PWC package into an .MSI package.

To do so, open the Prism Deploy Editor, and select the PWC file of the package, in this case, the AdobeFive0_1 package was selected. From the Package menu, select Create Windows Installer File, as Figure 7.9 shows.

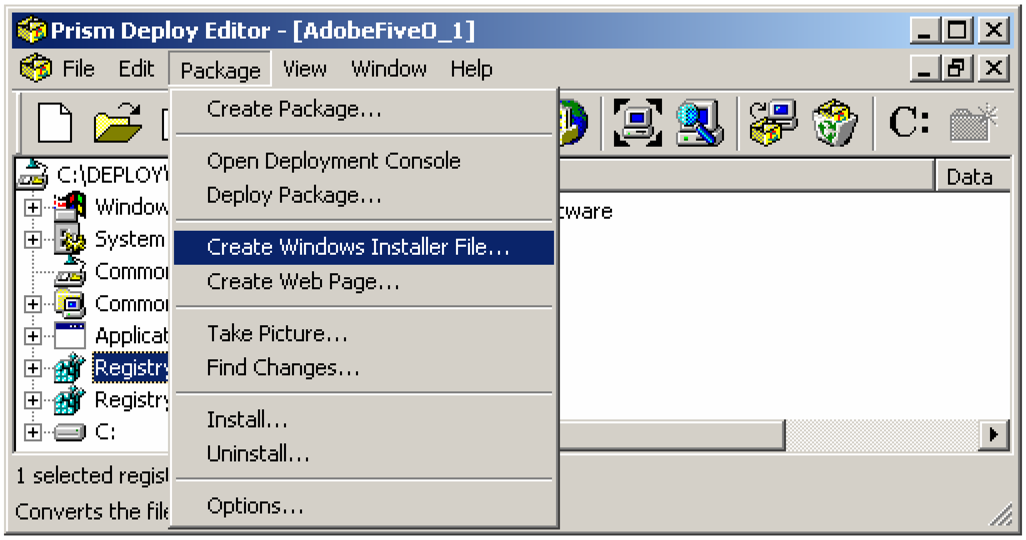
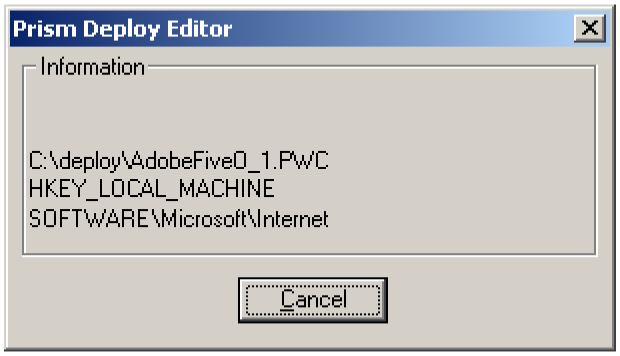
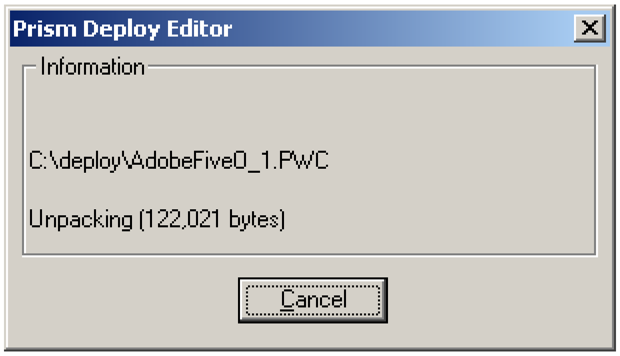


Figure 7.9: Starting the Windows Installer creation process within Prism Deploy.

The process will then ask for basic author data such as package name and the name of the manufacturer (company name). After entering this data and clicking Next, the editor begins the process of converting the PWC formatted file into the Installer format. Status messages similar to those that Figure 7.10 shows give you a progress report.



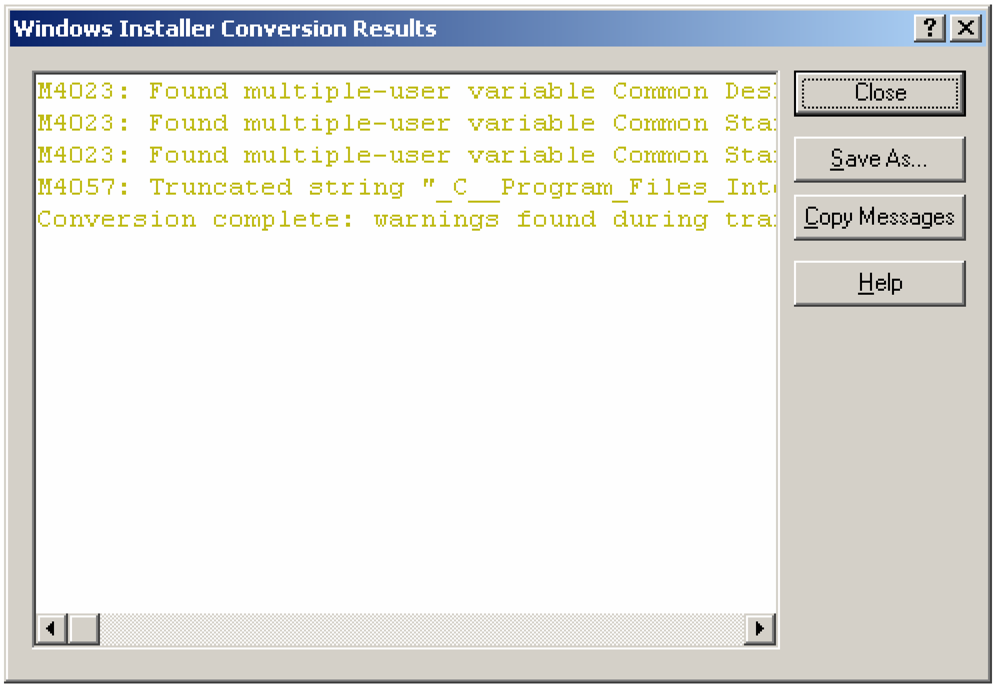
```
C:\deploy\AdobeFive0_1.PWC
HKEY_LOCAL_MACHINE
SOFTWARE\Microsoft\Internet
```



```
C:\deploy\AdobeFive0_1.PWC

Unpacking (122,021 bytes)
```

Figure 7.10: Status messages presented during the Prism Deploy Windows Installer conversion process.

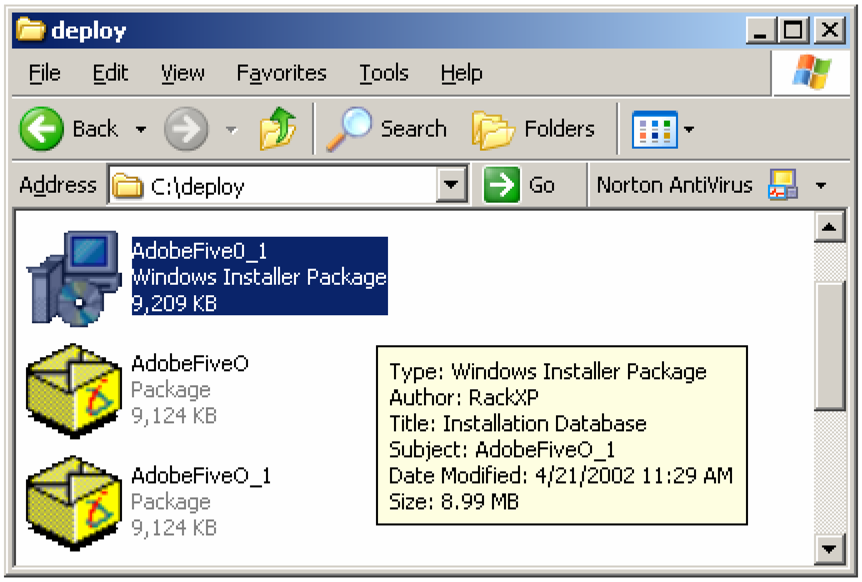
After a few minutes, the Windows Installer Conversion Results window appears with a status of the package process. Any errors listed should be investigated and resolved before you use the resultant .MSI file. In this case, the messages listed in Figure 7.11 are warning messages that are indicating differences between the Prism Deploy setup and the Windows Installer process.



```
M4023: Found multiple-user variable Common Des
M4023: Found multiple-user variable Common Sta
M4023: Found multiple-user variable Common Sta
M4057: Truncated string "_C__Program_Files_Int
Conversion complete: warnings found during tra
```

Figure 7.11: The results screen from the Prism Deploy Windows Installer conversion process.

Clicking Close closes the conversion window. When you complete the conversion process, you end up with an .MSI file in the chosen directory, as Figure 7.12 illustrates.


deploy
File Edit View Favorites Tools Help
Back Search Folders
Address C:\deploy Go Norton AntiVirus
AdobeFive0_1
Windows Installer Package
9,209 KB
AdobeFive0
Package
9,124 KB
AdobeFive0_1
Package
9,124 KB

Type: Windows Installer Package
Author: RackXP
Title: Installation Database
Subject: AdobeFive0_1
Date Modified: 4/21/2002 11:29 AM
Size: 8.99 MB

Figure 7.12: Snapshot showing the Prism PWC and Installer packages.

Running an Installer File

After you have the package created, you need to be able to run the application on the target system. The command-line format to do so is rather simple

```
msiexec /i \\server\share\filename options
```

To install the Adobe package I just created, the target machine would run

```
msiexec /i \\server\sharename\AdobeFive0_1.msi /qb
```

where the /qb option sets the user interface to the basic level. You can include logging options, an .MST file, and parameters for the properties in the command line or included files. For a complete list of the options and the requirements for each, review the Installer Web site for details (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/wiport_6gf9.asp). There is a large amount of data available about Windows Installer. The preceding information will hopefully give you enough of an understanding to start delving into the data pile.

Resource Kits

Whether you need to create a simple distribution package or just run a simple script to your workstations, you might need some utilities to create the package for you. Using the proper resource kit for the OS you are targeting is a time-saving method as many functions are already created for you. As the software distribution guru of your company, you should be aware of the various utilities in the resource kits.

Many of the utilities found in older resource kits work within new OSs. Some utilities written for NT 3.51 still work with Windows XP. Additionally, some of the older utilities that worked so well in the older technologies have been updated for each succeeding OS as needed. As with any utility you work with, test it to your satisfaction before you release it into the production world.

If you haven't played with the resource kits for your OS, you should take a few hours away from the demands of the day and play. Excuse me, I meant you should enhance your contribution to the departmental goals by proactively fine-tuning your skills and enhancing your knowledge of OS functionalities to meet your quarterly objectives in a timely manner. Everyone knows you're playing productively, it just looks better on the status reports using larger phrasings.

> Two good Microsoft sites to visit for resource kit information include http://www.microsoft.com/windows/reskits/default.asp and http://www.microsoft.com/windows/reskits/webresources/default.asp.

> Resource kits are a collection of utilities that have been written to perform various functions. Not all the tools are from Microsoft programmers, and support for the tools is not always available. Microsoft states throughout the resource kit documentation that support is not provided and the products are used as is. Microsoft is mainly doing distribution of the tools that have proven effective from the company's viewpoint. Although most are safe and don't cause problems, you need to prove that to yourself first. With that word of caution, we can proceed.

After you start using the tools, several will become staples in your growing arsenal of utilities. Some help with distributions, some with customizations, some with data gathering, and others with data manipulation. In just the Win2K Server resource kit, more than 300 tools are included. Many of the resource kits (and many of the individual tools) are available on the Microsoft Web site for free.

Some of the tools available in a resource kit are not functional tools but references to documentation or information for your use. Much of the information files can be found in other areas of the Web. Keep searching for those pearls of wisdom. Overall, the kits are essentially a requirement for anyone doing serious support work for a Windows environment. Because the tools in the resource kits are not fully supported, their structure and user interfaces can be unique. It was quite common when the resource kits first came out to notice typographical errors in the Help files and instructions. But the tools work well and are worth using. Many of the tools are command-line driven, some are GUI-based. Most can be used remotely to distant machines or combined with other resource kit tools to create custom utilities for your needs. As an example of the mix that you can do with resource kit tool versions, I've included a simple report-gathering script for fun.

> One point before we get into the script example: The process used as an example can be done via numerous other methods. There are other ways to obtain the same data. The point here is to highlight the ease of use the resource kit tools can provide to solve a problem. With a skeleton script in place, adding resource kit tools into the method can give you fast data to solve problems. When creating application packages to solve problems when your CADS does not provide the functionality you need for the task, a resource kit utility may be just what you need.

Here is the scenario for out example: You need to identify whether a certain hotfix has been applied to the NT workstations that are currently online. Additionally, you'll need to gather the information from the hotfix registry key. (An assumption is made that the userid being used to run the script has administrative authority on the target machines in the queried domain.)

As a modular quick-run script, the script is actually two scripts: Listing 2 shows SPRAY.BAT and Listing 3 shows HTFX.BAT.

```
REM ** SPRAY.BAT **
@echo off
REM ** Generic dynamic spray script.
REM ** Input variables.
REM ** %1 is the name of of the domain to check.
REM ** %2 is the name of the batch file to spawn.

REM ** Housekeeping. Create a report directory.
REM ** If an report dir exists, clean it out.
if not exist .\report md .\report
if exist .\report\*.txt del .\report\*.txt

REM ** Create the source to use.
REM ** Assumes the common name of workstations start
REM ** With \\NT??????? for a max of 10 characters.
net view /domain:%1 | findstr \\NT > %1.txt

REM ** The loop generator.
for /f "tokens=1" %%i in (%1.txt) do call %2.bat  %%i

REM ** Report time.
echo . > .\report\Report.txt
echo Total machines searched  >> .\report\Report.txt
find /c /i  "\\NT" %1.txt >> .\report\Report.txt

echo . >> .\report\Report.txt
echo Total entries found >> .\report\Report.txt
find /c /i "Report" .\report\good.txt >> .\report\Report.txt

echo . >> .\report\Report.txt
echo Total Bad connections >> .\report\Report.txt
find /c /i "failed" .\report\BadC.txt >> .\report\report.txt

REM ** Pop up the final report summary
start notepad .\report\report.txt

:End
```

Listing 7.2: The SPRAY.BAT file.

```
REM ** HTFX.BAT **
@echo off
REM ** Check for connection to the target machine
If NOT exist %1\c$\boot.ini goto :BadC

REM ** Remove the slashes from the NETVIEW generated name.
set MCHN=%1
echo %MCHN%
set MCHN=%MCHN:~2,10%

REM ** Do the work. Save the output to a file for each machine.
REM ** This allows research per machine if needed.
dumpel -l system -e 4359 -m NTServicePack -f  .\report\%MCHN%.txt -s %1


REM ** Find the trigger keyword for the report
find "Q300845" .\report\%MCHN%.txt >> .\report\Good.txt

REM ** The searched for string is checked, grab the entire HOTfix list
for reference.
regdmp -m %1 "\Registry\Machine\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Hotfix"
goto :end

:BadC
REM ** Report that a NETVIEW found machine could not be connected.
echo %MCHN% connection failed >> .\report\BadC.txt

:end
```

Listing 7.3: The HTFX.BAT file.

The process is started by calling SPRAY.BAT with two operands—the domain to review and the work script to feed the machine names into. SPRAY.BAT is used as a shell for other small scripts so that you have the same front end and only need to adjust the core work script.

The NET VIEW command is used by the script to gather the list of machines available in the domain browser list for the target domain. The browser list is not 100 percent complete, it shows only machines that have contacted the browser and announced their presence. As there can be different browser masters depending on the network, using NET VIEW will show only the machines that are online and talking to the same browser master that you are. In practice, the list shows between 95 percent and 99 percent of the online machines when you do a NET VIEW statement. It depends on your network and segmentation factors.

The NET VIEW statement pipes the results to a FIND command that searches only for machine names beginning with \\NT, assuming all the NT workstations are named NT*something*. The results of the FIND command are fed into a *domain*.txt file for later use. SPRAY.BAT then parses the *domain*.txt file one entry at a time, and feeds the machine name to the core work batch file called HTFX.BAT.

HTFX.BAT uses a simple test to see whether it can talk to the target machine. It checks for the existence of the C:\boot.ini file on the target machine. There are only four results possible from the check:

1. The target machine is online and reachable
2. The target machine is not online
3. The script does not have authority on the target machine
4. The target machine does not have a boot.ini file

The script only needs the first condition (online and reachable). Any other condition is not a concern of the script. Failed connections are recorded in a bad connection log file. After HTFX.BAT confirms that the target can be reached, the resource kit utility DUMPEL is run. This utility is useful and has been around since the NT 3.5 resource kit. The Win2K Server version (dated 12/99) is the one used for this example. This utility can obtain data from event logs and parse the information into different formats. This ability lets you place the event log information into spread sheets or other databases for massaging as you need. DUMPEL has options for changing the separator character, changing the output results, filtering, and selecting which log to use.

This script grabs any system events related to hotfix installs and sends the results to a text file that starts with the machine name. This information all goes into a directory called REPORT for ease of data collection. As the script is tailored to search for a particular hotfix number (Q300845), a quick search to the saved off file is done. After the FIND /C command completes the filtering, another resource kit utility is called to dump the registry for all hotfixes. The REGDMP utility (from Supplement 4 of the NT resource kit) is called in case the event logs happened to wrap and the installed hotfixes have rolled off. REGDMP is one of several REG* utilities that provide a simple means of gathering registry data or manipulating the registry contents.

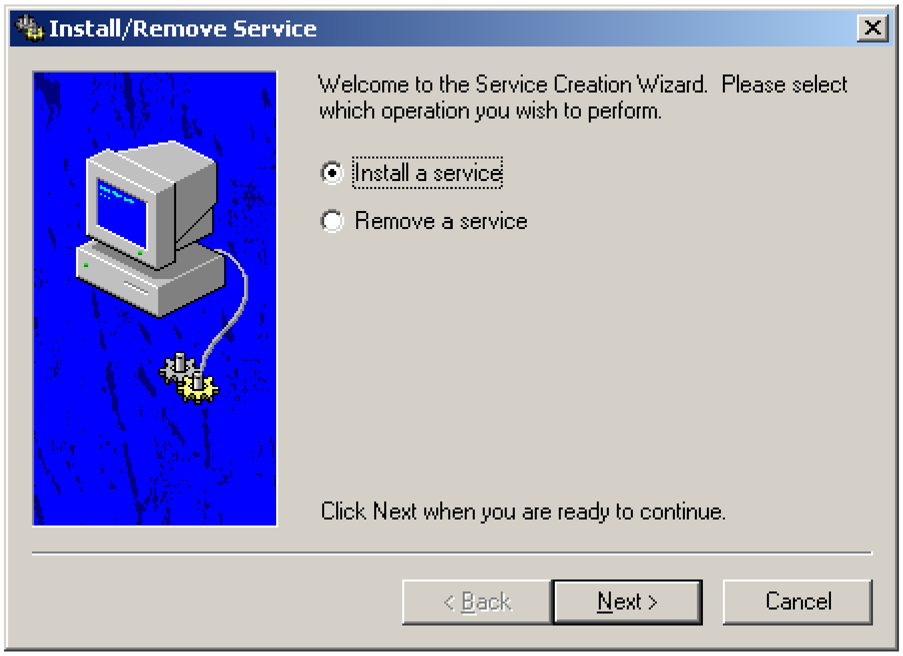
After the data is gathered for the target machine, HTFX.BAT ends and returns control back to SPRAY.BAT. After the entire *domain*.txt file has been individually processed, SPRAY.BAT does a fast count of the results and presents them in the form of a notepad report. If you have a CADS, the hotfix data may already be available in any inventory report the CADS generates. Additionally, most CADS will provide methods to gather the same data dynamically from inside their processing methods.

REGDMP and DUMPEL are both resource kit utilities that were originally designed for older NT-based OSs. For this example, I ran the script from a Windows XP Pro workstation to an NT 4.0 SP6 workstation, a Win2K Server, a Win2K workstation, and to another Windows XP system. All four systems returned the proper data as expected without problems. Thus, don't limit yourself to only the resource kit for your particular OS.

> ✎ You can remotely install services by using other resource kit tools: the command-line INSTSRV tool, which is great for scripting needs, or the GUI-based SRVINSTW tool, which guides you through the process.

If you need to remotely install a service, the SRVINSTW.EXE tool can do the trick. The steps are pretty easy:

1. Copy the source files for the service to the remote machine. Place them in the proper directories depending on the needs of the server.
2. Start SRVINSTW and follow the prompts. (Figure 7.17 shows the startup screen.)



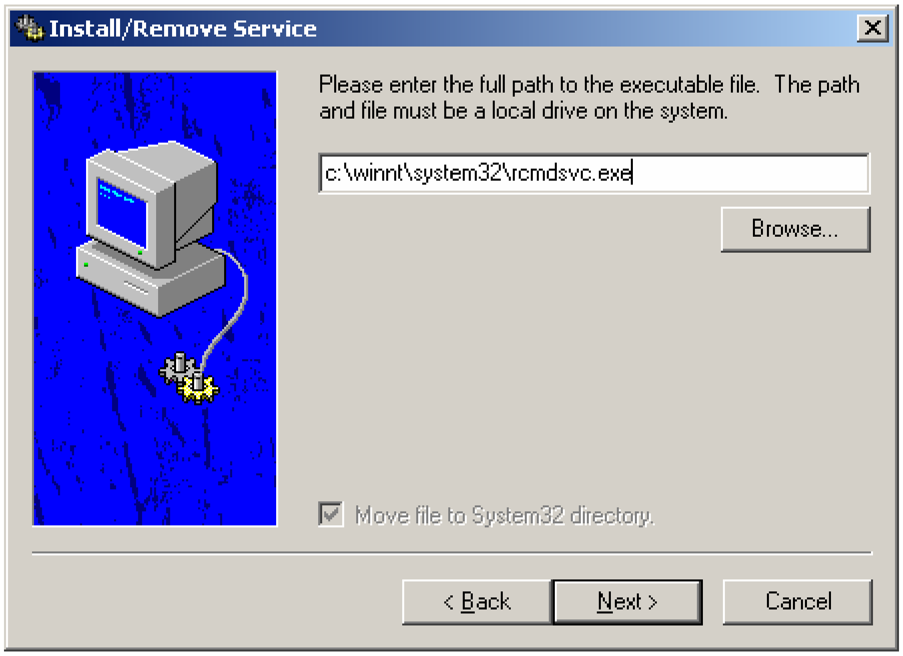
Welcome to the Service Creation Wizard. Please select which operation you wish to perform.

[● Install a service]
○ Remove a service

Click Next when you are ready to continue.

Figure 7.17: The startup screen for SRVINSTW.exe.

3. When the service asks for the source location, be sure to use the remote machine's point of view. Files placed on the remote machine's disk should be referenced by C:\, not a UNC or network connection, as Figure 7.18 shows.



Please enter the full path to the executable file. The path and file must be a local drive on the system.

c:\winnt\system32\rcmdsvc.exe

☑ Move file to System32 directory.

Figure 7.18: Be sure to specify the proper source location from the remote machine's point of view.

4. After the wizard completes, start the service via normal remote service install methods.

Another resource kit tool that you may have already used is the Remote Command Service. RCMD is a UNIX-like command-line tool that provides access to a client. It requires installation of a service (RCMDSVC.EXE) on the client, and the service must be running on the target machine before you can start the service.

RCMD needs the RCMDSVC.EXE file copied to the target machine first, then the RCMDSVC service installed on the target machine. You can use either of the previously mentioned tools to install the service on the client if it isn't already installed. As RCMD logs you onto the machine using the userid you are currently using, your userid must have the proper access level for the target machine.

RCMD is a great scripting and troubleshooting tool. It is limited to command-line aspects only. Any command you enter from the machine you're on will run as if you were physically at the target machine. A Windows XP machine can gain access to any other Windows machine via RCMD.

For software distributions, you can use RCMD for triggering scripts for silent installs, activating AT commands, or making modifications to directories. At times you might find that performing functions on a target machine works better than attempting to do the function remotely. One example is installing software remotely when the source is closer to the target than to you. Using RCMD to gain access to the target machine, map a drive via the NET USE command to a source server, then activating the silent install can speed the process. If your install package was not part of your software deployment method and you had no other way to activate the install, RCMD is a fast solution. The goal is to have different methods to perform the needed functions.

Be aware that RCMD is a little buggy and there are various versions out there. Commands entered via RCMD must run without user interaction on the target machine or the machine hangs. If a command calls up a GUI window, you will never see the window and the RCMD session is hung until the window is closed. That same hang condition happens if a pop-up window, such as an error message, happens. In scripting, you need to test the logic as RCMD might handle things a little differently than you expect.

The big consideration to remember with RCMD is to execute the commands locally with the local drive mapping logic. Because the RCMD session will not appear to the user, you can come in under the covers and execute PSTAT or TLIST or KILL commands for troubleshooting processes, but you can't see what the user is seeing. It is not a remote-control service, just a remote execution. The overhead for RCMD is lower than for remote-control services.

When scripting, you can copy a batch file to a known location on the target machine, then have your script call the batch file as if you were on the system. The command

```
rcmd \\machine c:\temp\run.bat
```

will connect to *machine*, execute the c:\temp\run.bat command and return focus back to your machine to continue on with your script. This process is a down-and-dirty method to make simple changes or activate processes on remote machines.

Resource Kit tools are not just focused on the distribution world. You can find utilities for managing AD, monitoring the network, monitoring API calls, verifying your cluster, manipulating the registry, manipulating services, and gathering hardware data. As I mentioned earlier, there are more than 300 utilities in just the Win2K Server resource kit. Instead of trying to cover all the utilities from all the resource kits, I've listed some of my favorites that have proven useful at one point or another within my software-distribution experiences (see Table 7.1).

Utility	Comments
APIMON	Monitor API calls; great for troubleshooting
DELSRV	Delete services installed on clients
DIRUSE	Obtain data on a directory structure; great for checking for size limits when space is an issue
DRIVERS	Locate data on installed drivers; great for pre-checking systems when preparing packages
DUREG	Check the size of registries—not just the space, but the actual data size
EXTRACT	Expand cab files
GETSID	Obtain the SID of two different userids
GPRESULT	Determine the impact that a GPO has on a userid
HEAPMON	Troubleshoot unknown issues or occasional hangs in processes or systems
IFMEMBER	Use in logon scripts for group membership branch logic
INUSE	Replace system reserved files great if you create your own installers
NOW	Determine what time is it NOW
OH	Determine what handles are open; great for troubleshooting
OLEview	Check out OLE and COM stuff
PSTAT	Determine how many processes are running on a system; this utility helps locate problems
RPCDUMP	Check the RPC side of the network
RCMD	Get console access to a remote machine; has some limitations but works well for most command-line needs
Robocopy	Mirror and filter very quickly; only copies what it needs to (if the file exists as you directed, it skips copying it over); great copy utility
SCOPY	Copy the security aspects of directories and files
SOON	Run a process inside the next 24 hours (like the AT command)
TIMETHIS	Determine how long it took to run a process
Whoami	Check a userid prior to running a script
XCACLS	Modify security parameters for directories; good for new machines or resetting older machines to standards

Table 7.1: Some useful resource kit utilities.

Summary

It is good to have a standard deployment method and to use it as much as possible. Being aware of other options and possibilities expands your scope and abilities to perform those one-off jobs that creep up once in awhile. Many of the tools needed to do simple distributions or data gathering are available free on the Web.

Although every deployment method has its own interface and rules for use, Windows Installer has an increasing presence in the deployment field. As a result, even if you don’t use it or plan to, you should still be familiar with it and how it works. Commercially available applications may only provide setups within the Installer framework, and understanding the technology will assist you in creating the application package and deployment design that best fits your environment.

Resource kits are a wealth of tools that are needed in every administrator’s bag of tricks. The documentation may not be the best for all the tools and not every tool will fit into your environment. However, the little gems that you find will quickly become a staple for your daily administrative functions.

Appendix

Chapter Summaries

I like having a short summary of chapters for reference books. Aside from the compiled index, the summary helps me in grabbing the overall thoughts of a chapter when I need to review a book for a single piece of information. This review is kept very simple, just the highlights in a note format to help me scan for data.

Chapter 1: The Foundation of Software Deployment

- Provided a historical review of software and deployments.
- Discussed linking files/libraries (both dynamic and static).
- Explored DLL hell and how it impacts installations.
- Provided a general overview of licensing, deployments, testing, and ROI.
- Discussed the deployment life cycle.

Chapter 2: Conformance Evaluation

- Began reviewing software deployment concepts. The three main items covered:
    - Acquisition—Incorporates the process of verifying whether the software is necessary and achieving a plan of deployment.
    - Distribution—Includes delivery, maintenance, and reporting of the software packages.
    - Support—Encompasses the responsibility of maintaining the existing software in a productive state for the end user.
- Introduced the test environment concept.
- Discussed the standardization of images and how to enforce it.
- Reviewed repackaging, retooling, and scratch package creation.
- Discussed image delivery (push, pull, advertise, and so on).
- Reviewed software management:
    - Inventory—Keeps a record of what is currently installed and by which user.
    - Delivery—Provides a means of getting the install package to users' computers.
    - Licensing—Keeps a record of in-use vs. available licenses.
    - Reporting—Creates a variety of status reports.
    - Warehousing—Physically provides storage space for distribution images.
    - Managing components—Performs a limited automated support role for the distributed software.

Chapter 3: Repackaging

- Reviewed the repackaging process.
- Discussed testing, which generally involves three levels of computers:
    - A clean test computer is one that has the core OS installed, all drivers are present, and any required general network settings done.
    - A baseline test computer is built on the clean system and has all the service packs, patches, user accounts, Group Policies, and other existing applications applied that make it look like a normal system in use in the target environment.
    - The working test computer is the end result after the new application or software has been applied and is functioning.
- Described that when you are performing your tests, being able to roll back to the baseline state (or clean, depending on the type of testing) is important. Scripting the levels of application installations can be done if needed.
- Discussed that when testing, keep backups of everything. Don't overwrite the sole source of the application.
- Explained that you should know your target environment, know what is out in the production environment, understand both the hardware and software levels to perform valid testing.
- Described the questions to ask during the beginning of the packaging process. Understand the expectations of the package sponsor.
- Explored how to document the install and package as you go. Provide enough data in the documentation to assist with troubleshooting 18 months down the line.
- Explained how snapshot software provides a comparison of your test system before and after installs. Use it to confirm what the install programs are doing to your systems. Review the results with a careful eye as some of the changes may not really be needed and may cause serious problems when committed to target workstations.
- Outlined the eight steps of packaging:
    - Create a base rollout specification.
    - Run the native install program and incorporate the results into the specification.
    - Interview team members and submit the first pass of the rollout specification.
    - Take a snapshot of the application on a clean machine with the required baseline.
    - Profile the application for linked libraries.
    - Process the snapshot and profiling results into the distribution package.
    - Update the rollout specification.
    - Test and iterate.

Chapter 4: Methods for Deploying Software.

- Outlined six general deployment methods:
    - Sneakernet—Manually installing software to each machine.
    - Simple scripting—Can be used to enhance more complex methods or used solo for special projects.
    - Email—Commonly used but hard to track for compliance and reliability. Either the package is in the email, a link is provided, or instructions are given for how to do the install.
    - Web—Used with other methods for delivering applications. A link is provided for access or another method calls the Web-based location to provide the application.
    - Image inclusion—The easiest to ensure that applications are installed and clean. The application is included in the image when it is laid down; however, the image is hard to update in a timely manner without causing issues to the end user.
    - Third-party solutions abound—Microsoft also has suggested solutions. Which option is best depends on your needs—check them out carefully.
- Discussed that when it is time to roll out the application, ensure the testing was done to the proper environment and roll it out in a controlled manner to prevent the unseen from appearing.
- Finally, discussed that you need to check the deployment. Have a reporting method of some type to confirm the applications rolled as you expected to the target audience.

Chapter 5: Deploying to Remote Users

- Defined remote as a definition, not always a condition. A remote user can be in the next building or the next state.
- Described that line speed plays a big part in your deployment methods. The size of the deployment package can impact the slow line speed connections.
- Explained how security in remote deployment is equally as important as line speed. Make sure the application goes to the intended machine.
- Explored how user impact is only an issue if the deployments are long or if bandwidth is impacted for other applications.
- Grouped remote users into four types:
    - Remotely hardwired machines—Connected to the LAN/WAN but in a location some distance from your deployment center.
    - Remote stationary—This machine is not part of the LAN/WAN and does not move around much (for example, home machines).
    - Roaming users and machines—Laptops and their users; you never really know when and where they are going to connect.

    - Rarely connected—These machines rarely connect to your network and often are only online for a short time when they do hook up.

Chapter 6: Deploying in the Enterprise

- Defined deploying software as not just the technical aspects. The Process, Procedures, and Documentation (PP&D) of a company play just as large a role.
- Warned you to document everything—the package, the process to roll it out, the method of deployment, how to fix it, and what the targets are. The data will come in handy at a later time during troubleshooting issues or when a package from last year needs to deployed again.
- Explained how to use your packaging software to help create the package documentation.
- Implemented a change management process. The change management group should be the overall controllers of change in the enterprise to help prevent conflicts and do scheduling among widely spread groups.
- Described how to control the source location—keep it golden; not only the primary source the package was created from, but also the source of the final deployment package. Only a few people should have access to make changes to the golden source packages.
- Advised keeping an eye on your hardware. Know what is out there and strive for consistency that you can account for in the deployments.
    - Keep the entire deployment process repeatable for any deployment. Minor adjustments are expected, but the overall process should stay known and consistent.
    - Software metering is becoming a larger issue. Investigate the need for license compliance in your environment.

Chapter 7: Installation Tool Alternatives

- Explored how Windows Installer (MSI) is a major player. It pays to become familiar with it even if your use is low.
- Used the Office MSI and transform processes as an example; however, the methods are similar to other applications. As MSI is flexible for the developers on the tools and processes, expect variances in implementation and support of MSI-compliant applications.
- Explained how resource kits provide many tools that can assist in improving deployment methods.

MSI Commands

Windows Installer installations are called via the msiexec.exe command. Use the information in Table A.1 for quick reference about the options and parameters of the command line.

Option	Parameter	Comments
/I	Package	Installs the product.
/f	[p|o|e|d|c|a|u|m|s|v] Package	Repairs the product. The default is 'pecms'.
		p—Reinstall if a file is missing.
		o—Reinstall if a file is missing or an older version of the file is installed.
		e—Reinstall if a file is missing or the currently installed file is an equal or older version.
		d—Reinstall if a file is missing or the installed version is different.
		c—Reinstall if file is missing or the stored checksum does not equal the computed checksum.
		a—Force a full file reinstall.
		u—Rewrite all required user registry entries.
		m—Rewrite all required computer-specific registry entries.
		s—Overwrite existing shortcuts.
		v—Run from the source point and re-cache the local package.
/a	Package	Admin install—Installs a product on the network.
/x	Package	Removes the product.
/j	[u|m]Package or [u|m]Package /t Transform List or [u|m]Package /g LanguageID	Advertises the product.
		u—Advertise to the current user.
		m—Advertise to all users of a machine.
		g—Language ID.
		t—Apply the transform to the advertised package
/L	[i|w|e|a|r|u|c|m|o|p|v|+|!]Logfile	Logging option—Specifies the path and data to log.
		i—Status messages.
		w—Non-fatal warnings.
		e—All error messages.

		a—Start up of actions.
		r—Action-specific records.
		u—User requests.
		c—Initial UI parameters.
		m—Out-of-memory or fatal exit information.
		o— Out-of-disk-space messages.
		p—Terminal properties.
		v—Verbose output.
		+—Append to existing file.
		!—Flush each line to the log.
		"*"—Wildcard—log all information except for the v option. To include the v option, specify "/l*v".
/m	filename	Generates an SMS status .mif file. Either -i, -x, -a, or –f needs to be specified.
/p	PatchPackage	Applies a patch.
/q	n|b|r|f	Sets the UI interaction.
		q, qn—No UI.
		qb—The basic interface.
		qr—Reduced interface.
		qf—The Full interface.
/y	module	Calls DllRegisterServer to self-register modules from the command line.
/z	module	Opposite of the /y command; un-registers the modules.

Table A.1: MSI command-line options and parameters.

MSI Error codes are lengthy in size. One good place to identify what the codes are is the Microsoft Web site (http://www.microsoft.com).

Deployment Tools

The list of tools that can do deployments is ever increasing. By now, you have some tools that are favorites; Table A.2 lists a few others to look into.

Tool	Description
http://www.kixscripts.com	An enhancement to KIX32 that comes in the resource kit; RWK Systems has added new functionality and smoothed some of the older processes. Check out the Web site for more information.
kix32	Part of the Windows NT 4.0 resource kit, this scripting process provides functionality for doing simple scripts. Though some folks claim KIX is a dying language, remember the same thing was said for COBOL. It is worth checking into.
Windows Script Host (WSH)	No surprise here, the Microsoft product is making headway into the Windows world. Providing a method to tie in Jscript, VBScript, API calls, and other hooks from other languages, WSH gives you a fast method for creating enhanced scripts beyond the old standby of simple batch scripting. Check out http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001169.
QChain	There will come a time when you need to install multiple hotfixes in a single run. With the increasing number of patches being released each month, chances are you have already run into the annoying requirement of rebooting between each hot fix install. Microsoft's QChain eliminates the need for most of the reboots. Many hotfixes need to update files that are in use by the system and can't be modified immediately. When the system is rebooted, the hotfix marked file replaces the previously-in-use system file. When you install multiple hotfixes, you run the danger of having the same file replaced with an incorrect version. QChain reviews the list of marked items and keeps them straight. In your package, execute QChain after all the hotfixes have run and before you reboot the system. For more information, check out the Microsoft article "Use QChain.exe to Install Multiple Hotfixes with Only One Reboot" (Q296861).

Table A.2: Deployment Tool Options

Web Resources

Although far from exhaustive, the information in Table A.3 will get you started in your quest for software deployment expertise.

Resource	Description
http://www.spa.org	The home site of the Software & Information Industry Association provides good references for metering and license compliance.
http://www.appdeploy.com	Good site for general searching of software deployment issues. The site stays current on new products, provides some hints and tips about deployments, and has a good collection of tools.
http://www.newboundaries.com/	This site provides good information and tools for packaging and deployment (makers of Prism Deploy).
http://www.marimba.com	This site provides good information about packaging and deployment.
http://www.wise.com	This site provides good information about packaging and deployment.
http://www.dependencywalker.com	Access to the freeware utility (currently version 2.1) that investigates modules (exe, com, dll, and so on) and presents a view of the dependent modules as well as other views. Good troubleshooting tool when you need to get into the nitty gritty.
http://www.systemtools.com	Makers of Hyena, a tool to ease management of your enterprise that works well with deployment methodologies.
http://desktopengineer.com	Administration site with many articles and tips.
http://www.swynk.com	Primarily an administration site for Microsoft products, this site has good information about deployment issues, although the main focus is Group Policy Object (GPO) and Systems Management Server (SMS) deployments.
http://www.symantec.com	This site provides drive replication software to reproduce disk images.
http://www.powerquest.com	This site provides drive replication software to reproduce disk images.
http://www.installshield.com	This site is the home of InstallShield.
http://www.winscriptingsolutions.com	This site provides information about Windows scripting.
http://cwashington.netreach.net	This site provides information about Win32 scripting.
http://www.userfriendly.org/	Classic computer humor—no operating system is immune!
http://www.zdnet.com/downloads	This site provides all kinds of downloads (freeware, trials, demos, and so on).
http://www.rational.com/products/testfoundation/index.jsp	This site provides information about the Test Foundation for Windows 2000 (provides tools for testing software applications).
News groups	Check your local news group server for some off-the-cuff help. However, the answers are not always from experts, so take the advice with a strong dose of caution. Some suggested search words include Microsoft, deployment, windows, windowsinstaller, wise, installshield, and installer.
http://www.microsoft.com/smserver/default.asp	This site provides information about SMS deployment.
http://www.microsoft.com/technet/tre	This site provides a good overview starting point for Microsoft

eview/default.asp?url=/TechNet/prod technol	product information.
http://www.microsoft.com/technet/default.asp	The Technet home page.
http://www.microsoft.com/windows/reskits/default.asp	The resource kit home site for Microsoft products.
http://www.microsoft.com/windows/reskits/webresources/default.asp	This site provides greater detail about the resource kits and the contents of each in a table format.
http://www.microsoft.com/windows2000/techinfo/reskit/en-us/default.asp	The Microsoft site for online Windows 2000 resource kit manuals. Server, professional, registry reference, performance counters, GPO, error messages, and a glossary are all included.

Table A.3: Software deployment resources.

Glossary

Acquisition

The process of verifying whether the software is necessary and achieving a plan of deployment.

Application Documentation

See Package Documentation

Application Source

The unedited and unpackaged data for applications—what the packaging process starts with.

Assigned Applications

An application that is available for users but not installed at user sign on. It is installed when a user clicks the application icon.

Baseline Test Computer

A test computer that is built on a clean computer and has had service packs and any required patches applied and user accounts, Group Policies, and any other *existing* applications installed that you are likely to see in the end-user environment.

Change Management

A process to prevent conflicts in deployments.

Change Management Coordinator

A central contact that provides coordination in the change management process and ensures that the processes are followed.

Clean Test Computer

A test computer that has the core OS installed, all drivers installed, and any required general network settings configured.

Conflict Resolution

The process to resolve problems identified during a review process.

Conformance Testing

Testing to verify that software applications function with existing software.

Deployment Method

How an application is distributed.

Deployment Source

Location that contains the edited and deployment-ready package.

Distribution

The delivery, maintenance, and reporting of software packages.

Distribution Documentation

A document to detail the deployment method and source location of the application.

Distribution Endpoints

Your commercial application distribution software generally likes to have a place to find the software-modified source or your custom deployment solution has locations from which the files can be installed.

DLL Hell

An annoying residual effect of installing multiple applications—a situation in which each application installs its own DLL files. Version conflicts, overwrites, and unforeseen modifications can result. Deployment software in recent years is focusing more and more on controlling this problem.

Elevated Privileges

When applications are installed on a client system, the installer may require privileges to the computer at a higher level than the user context (such as getting edit access to the registry). Windows Installer has a method for elevating the privileges of the application. If the deployment method relies on an installed service, the service can be installed with a userid (or system access) that has elevated privileges.

Email Deployment Method

Aside from the obvious use for sending information, email deployment can send links or executables (for example) to users.

Enterprise

A generic term that describes a business unit, company, division, or departmental division.

Explicit Linking

An executable will attempt to load any needed libraries when the application calls for the file.

Image

Generally, a snapshot of the computer operating system.

Image Inclusion Deployments

A deployment method that packages applications into an image snapshot. The application is deployed by the act of imaging a workstation.

Implicit Linking

The list of libraries an executable needs is located within the header of the executable.

Installation Point

Location where the application was originally installed.

Mass Deployments

The process of distributing applications to all end points in one deployment.

Package Documentation

A form that contains data needed by the deployment group to get an application out; it should be customized for your particular environment. Some suggested headings: Overview, Contact, Prerequisites, Detailed Steps, Post Steps, Troubleshooting, Process Overview, Known Issues and Optional Items. Its primary focus is the application, not the distribution.

Peer Review

A good practice to catch errors prior to a deployment. After the product or application or deployment method is completed, a technically knowledgeable person that was not involved in the process reviews all aspects, looking for errors or conflicts. Having another set of eyes (and a different viewpoint) can find overlooked areas.

Phased Rollouts

A process of sending applications out in small sections.

PP&D

Process, Procedure, and Documentation.

Published Application

An application that is available to the user but not installed at user sign on.

Pull Technology

A distribution technology that primarily relies on client end points triggering the deployment event.

Push Technology

A distribution technology that primarily relies on servers (distribution points) triggering the deployment event.

Redistributables

Libraries that are needed and not included in the original operating system—the libraries are contained within the software application.

Remote Users

Users not physically located in the same location as the deployment starting point.

Repackaging

The process of getting a software application prepared for deploying to your environment.

Resource Kit

A collection of utilities focused on a particular product.

Rollout

The overall process of getting an application distributed.

Rollout Team

The group of people involved in the creation of the deployment application.

Shared Resources

Basically anything an application will need to function, including (but is not limited to) file extensions, user document workspace, ActiveX controls, Open Database Connectivity (ODBC) databases, drivers, fonts, shortcuts, DLLs, and so on.

SLA

Service level agreement, which is a document that describes the relationship between the owner of the application or process and the support organization. Although the document's content is very flexible, the agreement normally details the acceptable outage durations, who to contact on both sides, when dark windows are allowed, critical times for the application to be online and any processes needed to resolve issues. Documenting the backup process and what it takes to restore the data is also a good inclusion item. By detailing the information ahead of time, conflicts are prevented or at least more easily resolved.

Snapshot Software

Software used to take a status image of an application or operating system.

Sneakernet

A deployment method that describes the manual method of deployment.

Software Metering

Process to control and monitor software licenses.

Source

The starting point; where the data is located.

Standardization (or Standards)

A set of criteria that all applications abide by to ensure compatibility and mutual functionality.

Target Systems

The distribution end-point computers.

Test Computer

Computer used for testing.

Web Deployment

A deployment method that provides a Web page with links. Users click the link and the application is installed.

Windows File Protection

WFP is a process in Windows 2000 and later that ensures the core operating system files aren't inadvertently overwritten.

Windows Installer Transforms

A method to modify Windows Installer files.

Working Test Computer

A test computer that has the software application installed and functioning properly. It is the final and desired state of the testing process.