



realtimepublishers.com<sup>™</sup>

# *The Definitive Guide™ To*

# Windows Software Deployment



Making Complex  
Technology **SIMPLE**  
Since 1985

*Chris Long*

Chapter 3: Repackaging .....	44
Repackaging Background .....	45
Repackaging Process.....	45
The Test Computer.....	46
Creating a Clean State .....	47
Creating a Good Image .....	48
Creating a Baseline State.....	51
Ask Questions .....	52
Document .....	55
Preparation .....	55
Research the Existing Application Media.....	55
Script Solutions .....	57
Creating a Working State.....	57
Snapshot Software Options .....	59
Strictly Snapshot Tools .....	59
Snapshot-to-Project Tools.....	60
Snapshot.....	61
Package Evaluation .....	62
UI.....	64
System Files .....	65
Internal Logic .....	65
Conflict Resolution .....	66
Getting a Profile .....	66
Test.....	69
Repackaging Is Easy, Right?.....	70
A Repackaging Example: Macromedia ColdFusion 5.....	70
Clean Requirements .....	70
Baseline Requirements.....	71
Preparing to Create the Package.....	71
Normal Install.....	75
Snapshot Process.....	79
Package Modifications .....	81
Comments on the Final Package.....	82

Summary ..... 82

## 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.

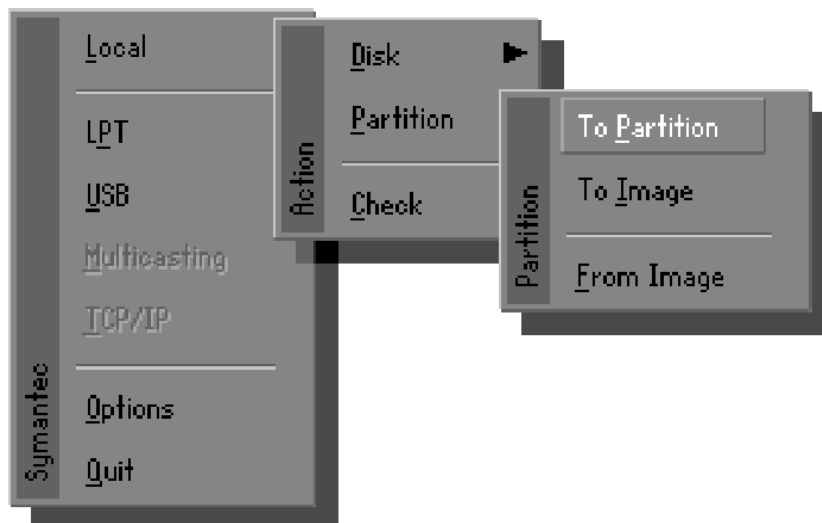


There are several imaging software applications available. A little bit of research on the Internet should get you up to speed quickly on the various features and how the different imaging applications compare. Two common imaging software products are Symantec's Ghost and PowerQuest's Drive Image. Both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

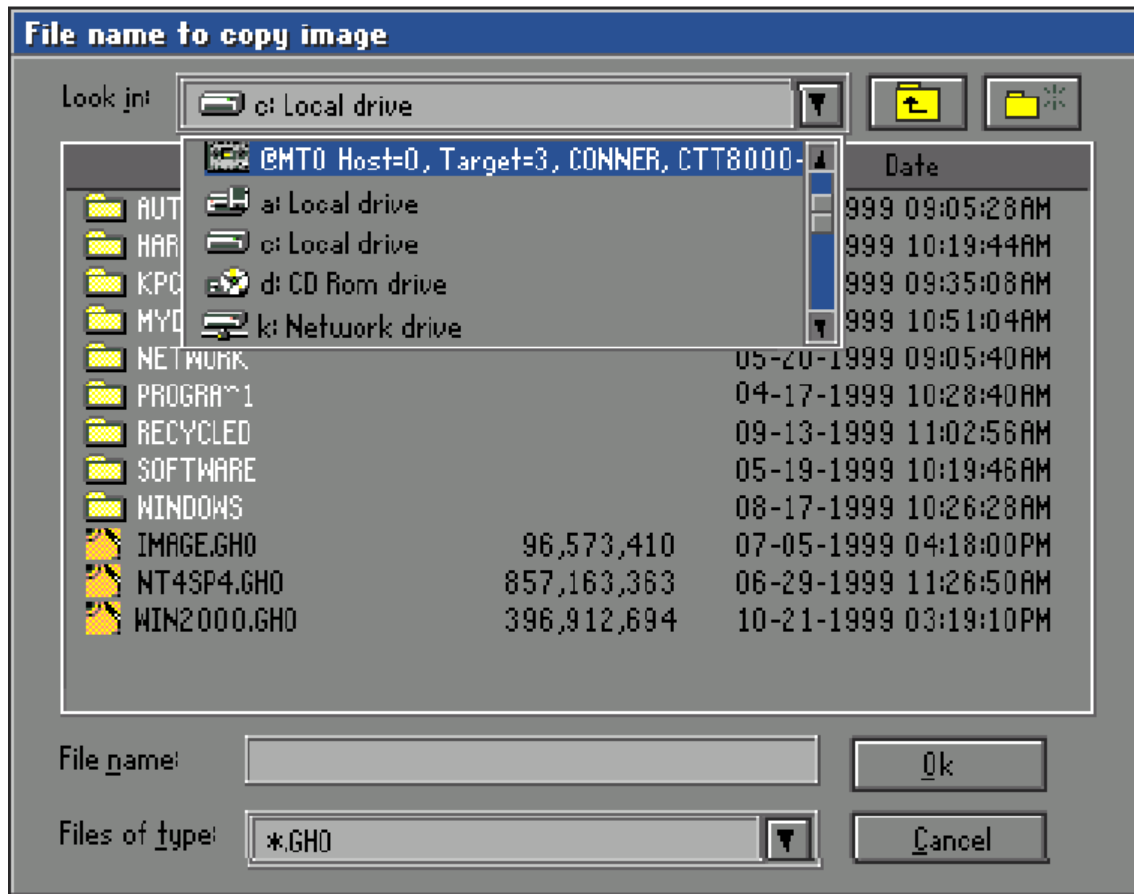
Regardless of the imaging software you choose, the software performs the same task: storing the current state of the computer so that you can restore it later. Create a bootable CD-ROM with this data file, and you can quickly reset a computer back to its original binary status. Simply amazing and a must-have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



**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](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 Lanovation's Prism (formerly PictureTaker) 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 repackaging 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 repackaging process. Being able to definitively state why a repackaging 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 repackaging.

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.

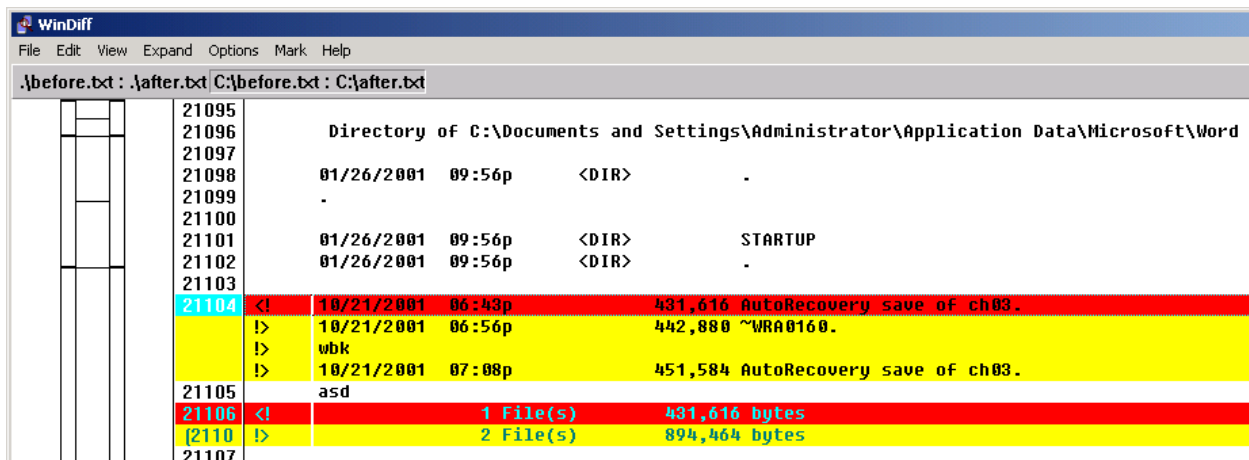
By using `regedit` (part of the Windows-included utilities), I could do the same thing in the system registry by selecting the root of the registry and selecting `Export Registry File` from the Registry menu. This action prompts me for a filename and a location to store the exported registry settings. 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.



Line	File	Size	Time	Attributes
21095	Directory of C:\Documents and Settings\Administrator\Application Data\Microsoft\Word			
21096				
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.
	!> wba			
	!> 10/21/2001 07:08p	451,584		AutoRecovery save of ch03.
21105				asd
21106	<	1 File(s)	431,616 bytes	
21107	!>	2 File(s)	894,464 bytes	
21108				

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 ASCII files. Attempting to compare Unicode files will continually generate a *Different in blanks only* message.

```

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Your Company Name]
@=""

[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 Lanovation

- 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)

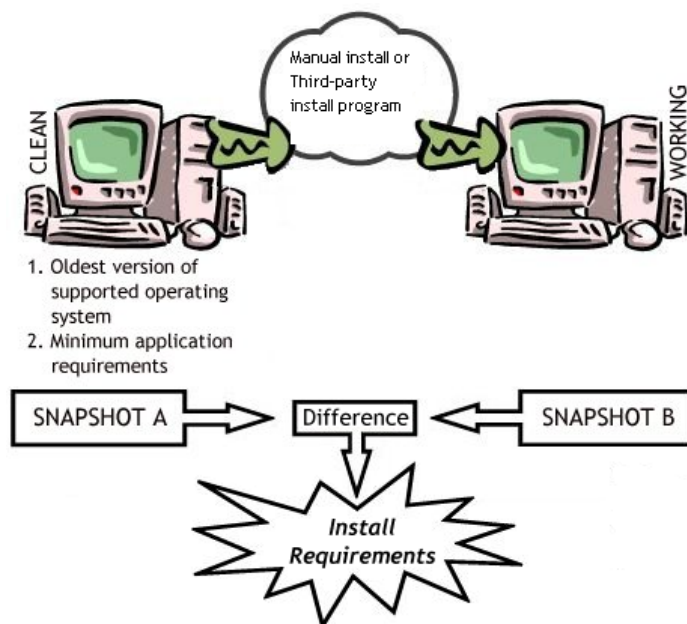
The questions to ask yourself as you're going over the feature lists are the basics: What is the final format of the deployable package? Does it have the option to create a Windows Installer database file? How can I deploy the package? Is additional software required on the end user's computer to support 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.




**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.

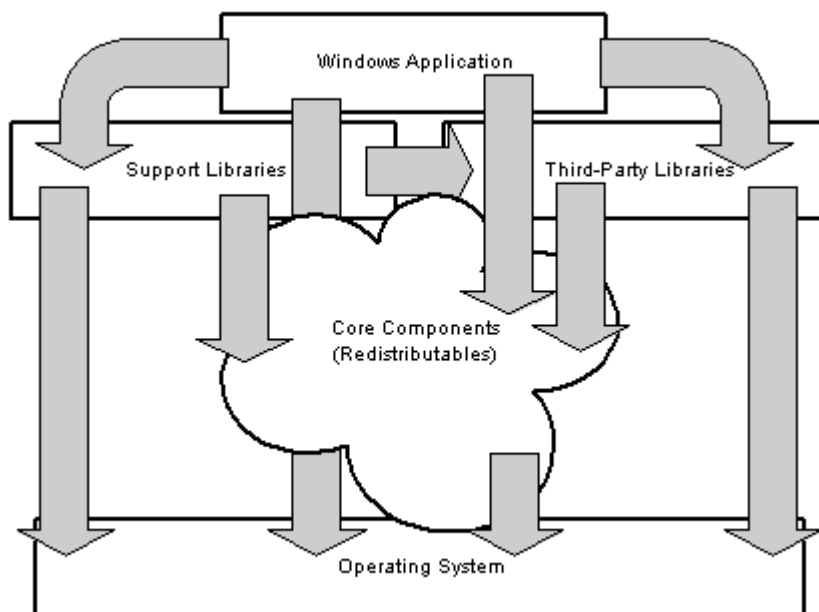
So once you know a conflicting core component must be shipped, spend some time tracking down the exact version required. You'll save yourself and the support team some headaches. You'll save your company some money. Several companies are looking into the conflict-resolution issue and are working to solve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



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, Lanovation's 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 Lanovation's 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)

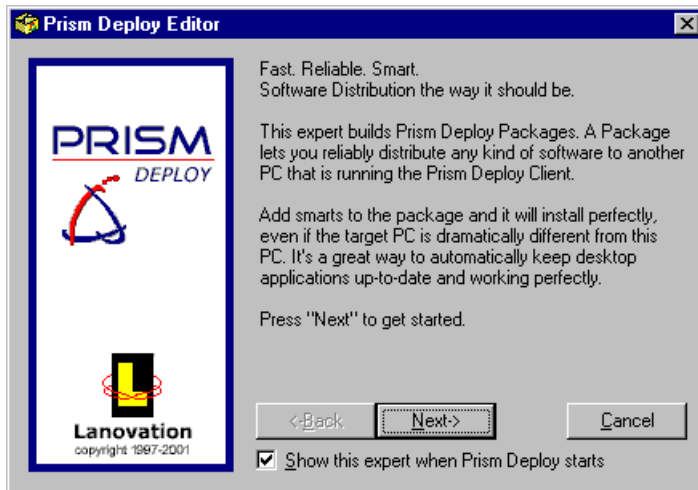
Registry keys as counters:

- HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Shared DLLs

Registry keys whose contents are merged:

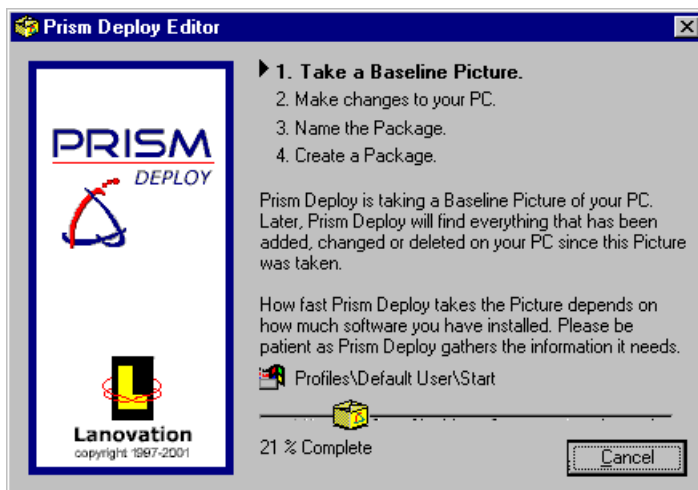
- HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\
  - Session Manager\Environment,Os2LibPath
  - Control\Session Manager\Environment,Path
  - Control\Session Manager\Environment,PATHEXT
- HKEY\_CURRENT\_USER\Environment,
  - Os2LibPath
  - Path
  - PATHEXT
- HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\
  - Services\Eventlog\Application,Sources
  - VirtualDeviceDrivers,VDD
  - Control\ServiceGroupOrder,List
- HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Net
  - workCards\\*\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.



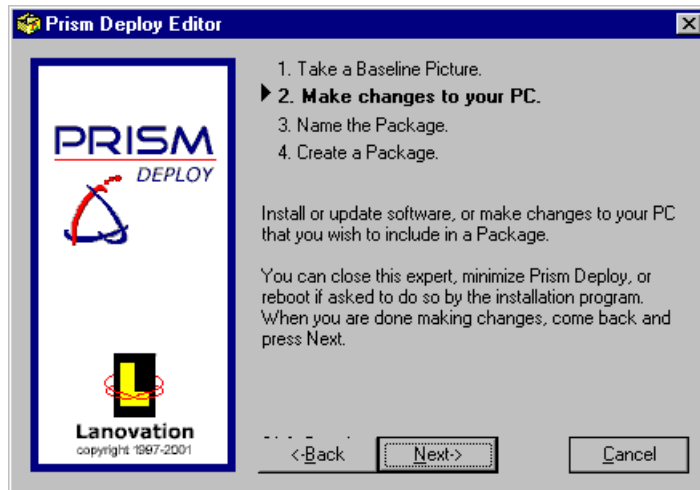
**Figure 3.7: Introduction to the Prism Deploy wizard.**

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.



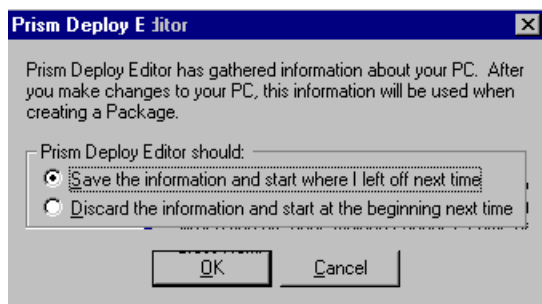
**Figure 3.8: The initial snapshot is performed while this dialog box is displayed.**

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.



**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.

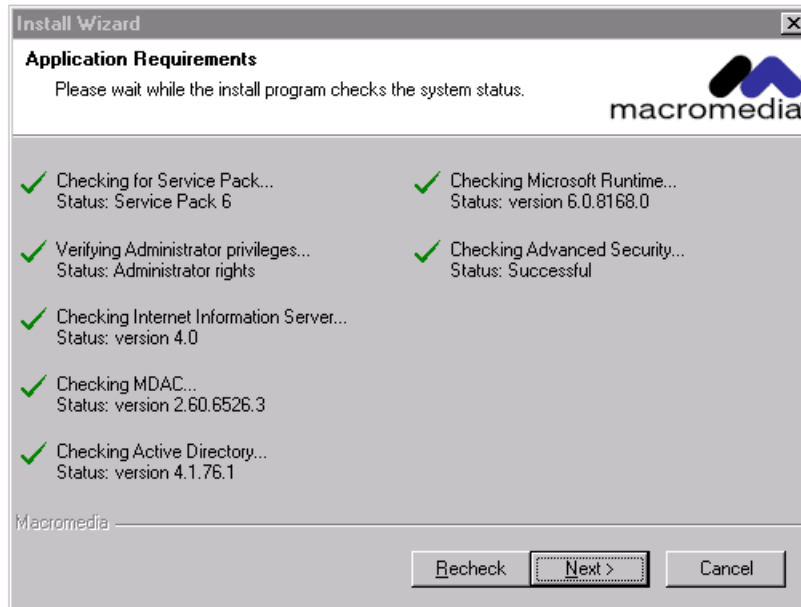


**Figure 3.10:** While exiting the wizard expert, we elect to have the detected information saved.

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

## Normal Install

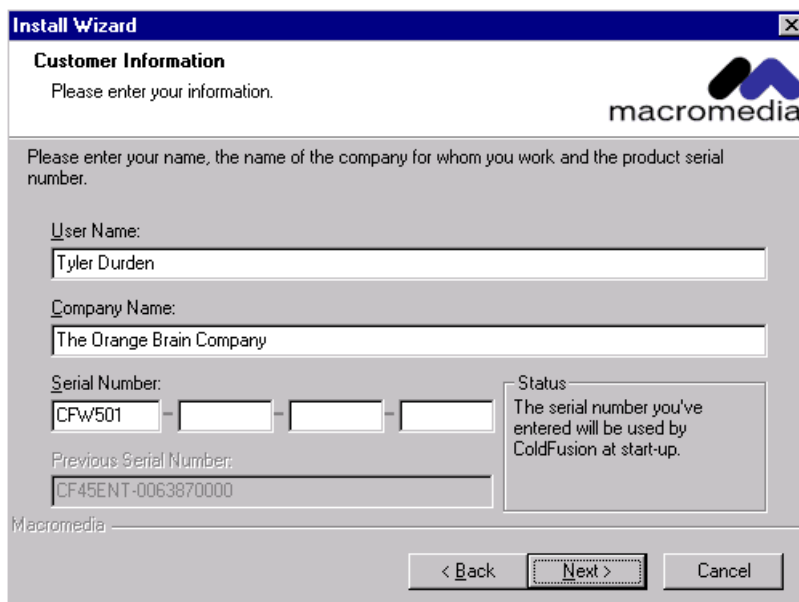
The ColdFusion 5 install is initiated from a CD-ROM. The first dialog box, which Figure 3.11 illustrates, displays the system status as a number of system prerequisites with checks. Although not shown in this figure, the ColdFusion 5 install program detected older versions of MDAC and the MFC runtimes.



**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.

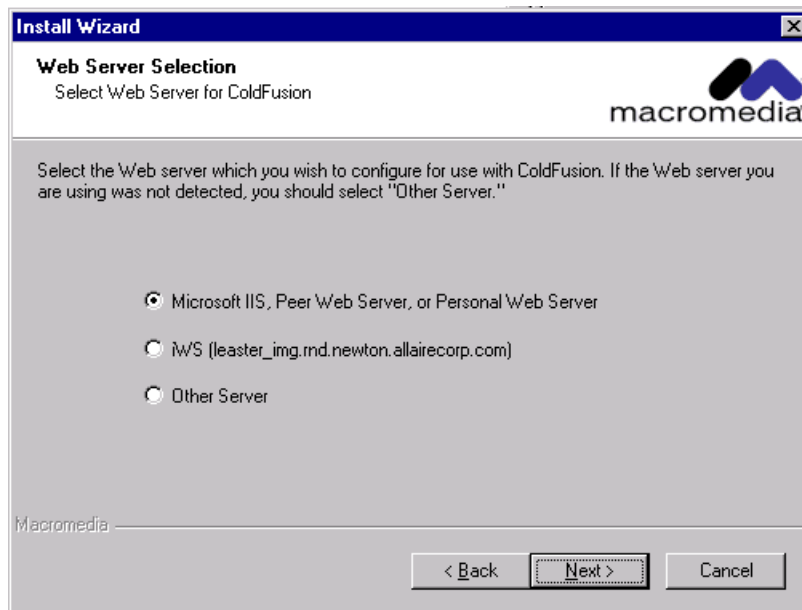


**Figure 3.12:** A customer information dialog box prompts for, among other things, a serial number.

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.

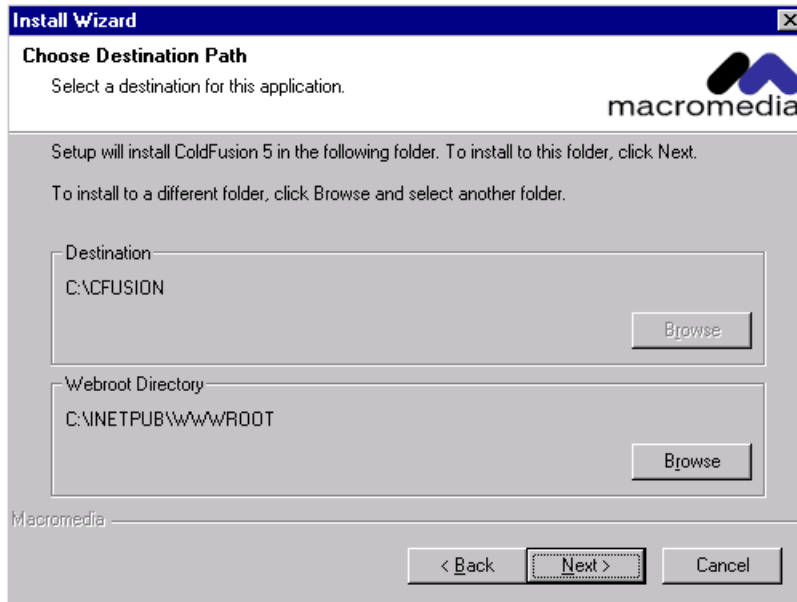
As Figure 3.13 shows, the ColdFusion 5 install program queries for existing Web servers.



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

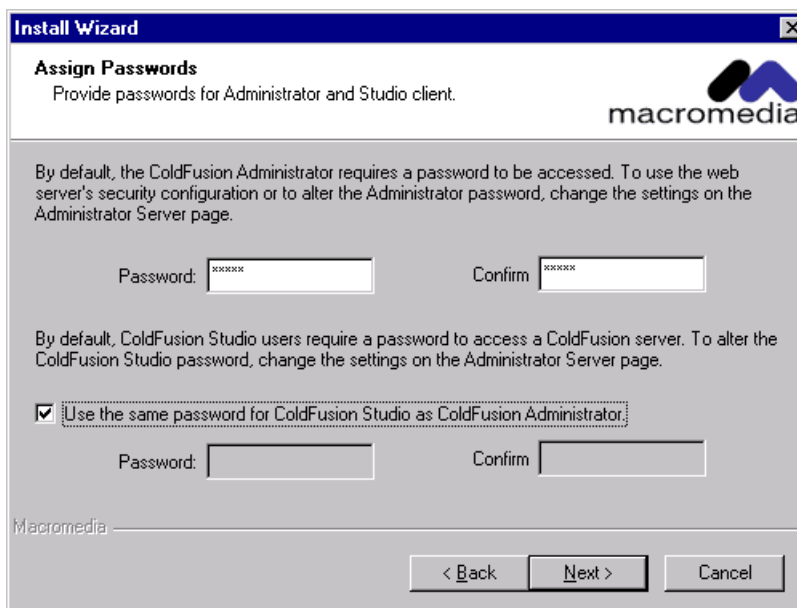
In this case, Microsoft IIS and Netscape iPlanet were detected. In our distribution model, we are more than likely expecting to find a single standard Web server. Making the appropriate selection is important to a consistent rollout.

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.



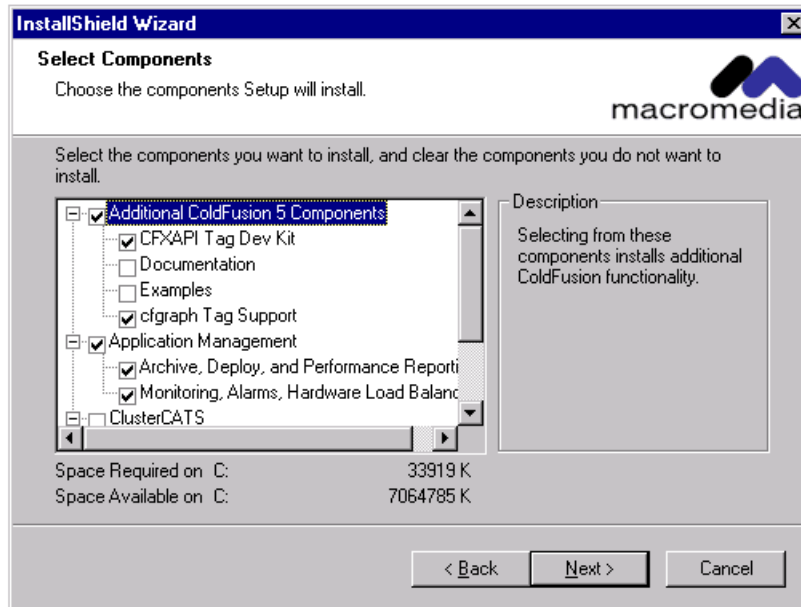
**Figure 3.14: Two path locations are requested.**

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.



**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.



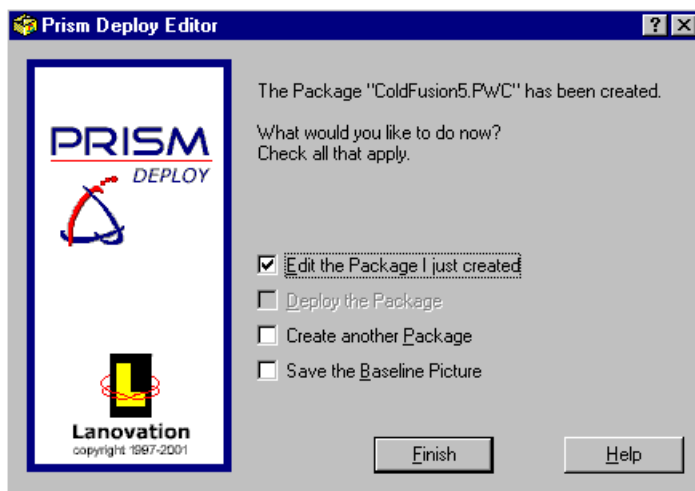
**Figure 3.16:** Component selection is a key dialog box that almost all install programs provide.

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

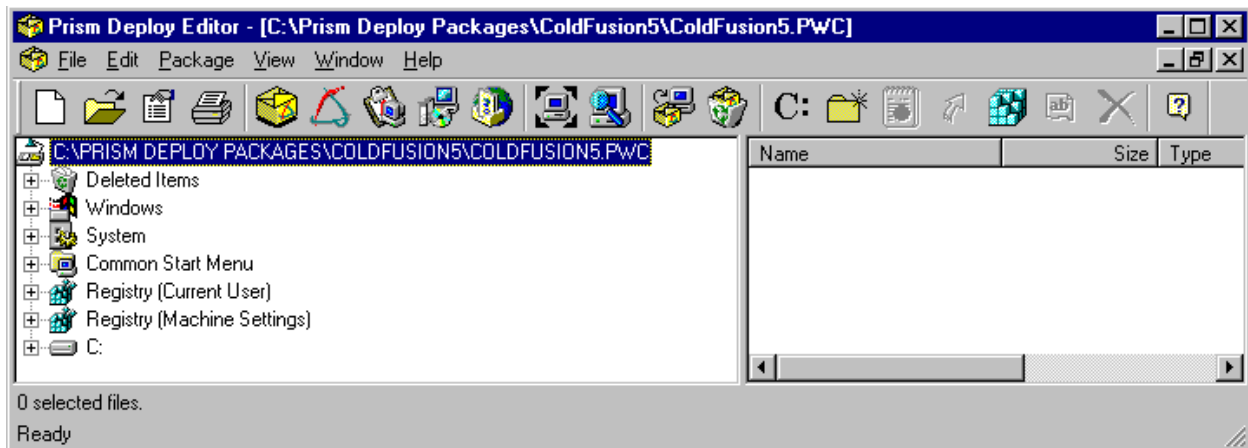
After the install is finished, I launched Prism Deploy again. It automatically detects that it's in the middle of a system snapshot, and we're prompted to continue. After selecting Next, the final snapshot is performed and a comparison is made. Afterward, the comparison is used to create the ColdFusion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



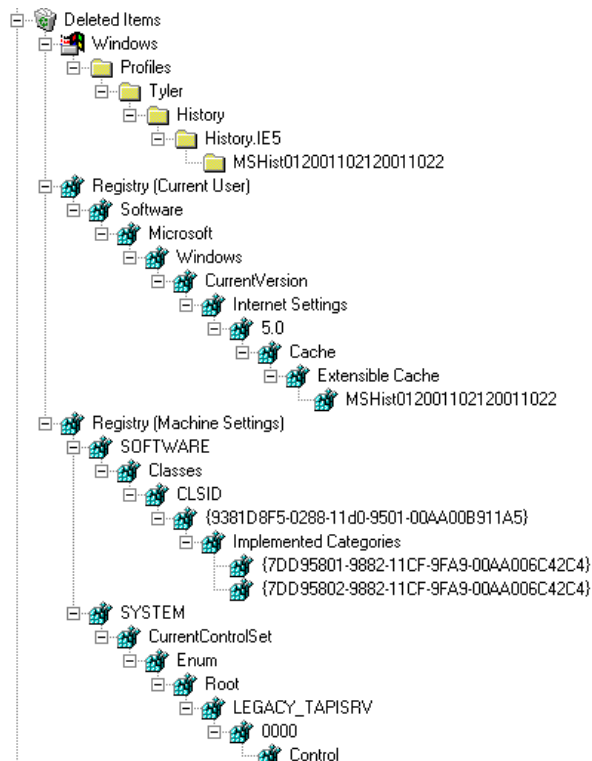
**Figure 3.17:** After the snapshot comparison is done, we're given the opportunity to modify the package.

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.



**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.

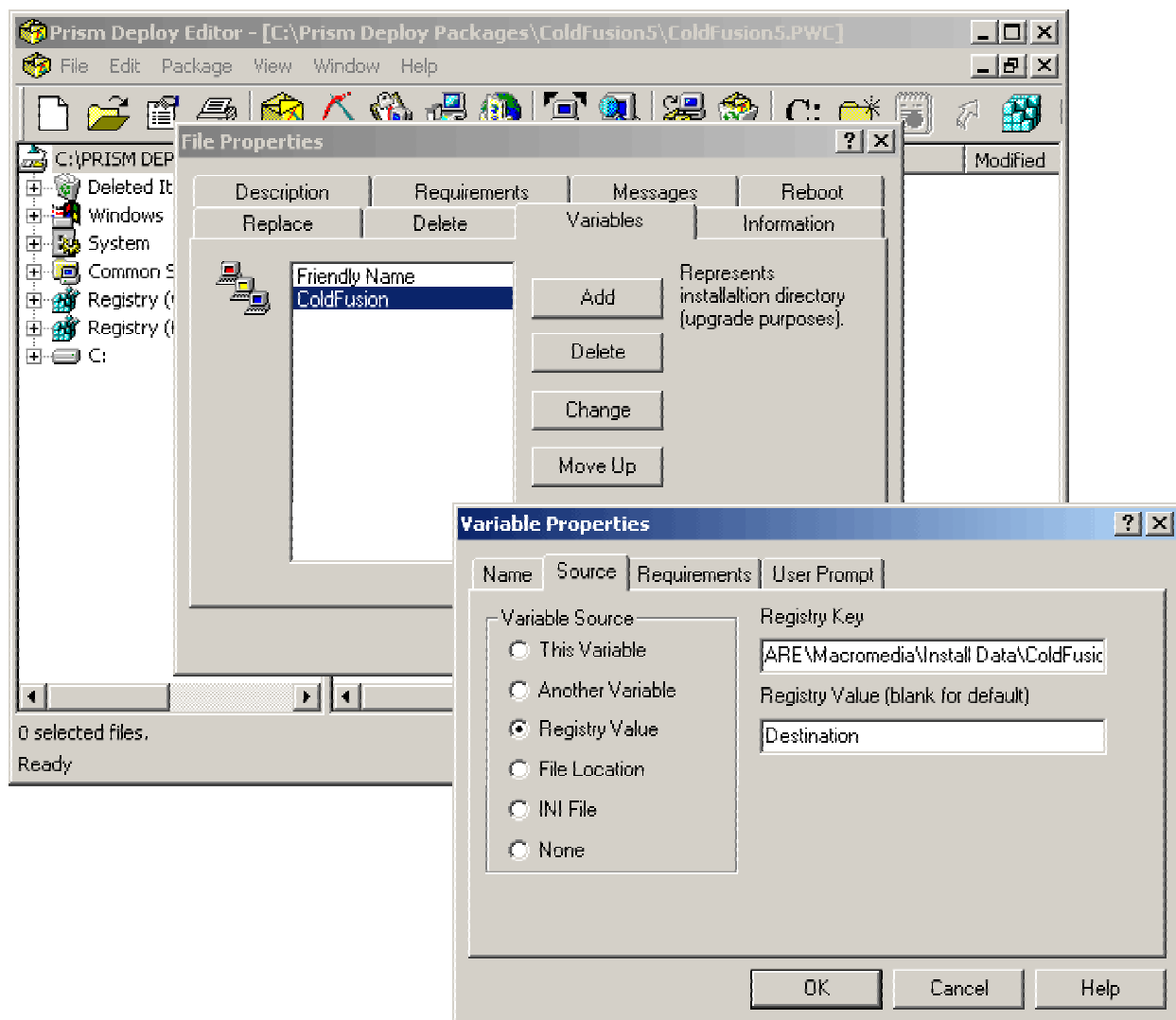


Figure 3.20: Turning an installation directory into a variable.

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.

### Copyright Statement

© 2001 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the “Materials”) and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at [info@realtimepublishers.com](mailto:info@realtimepublishers.com)