realtimepublishers.com™

The Administrator Shortcut Guide™ To

Software Packaging for Desktop Migrations

Chris Long

Introduction

By Sean Daily, Series Editor

Welcome to *The Administrator Shortcut Guide to Software Packaging for Desktop Migrations*!

The book you are about to read represents an entirely new modality of book publishing and a major first in the publishing industry. The founding concept behind Realtimepublishers.com is the idea of providing readers with high-quality books about today's most critical IT topics—at no cost to the reader. Although this may sound like a somewhat impossible feat to achieve, it is made possible through the vision and generosity of corporate sponsors such as InstallShield, who agree to bear the book's production expenses and host the book on its Web site for the benefit of its Web site visitors.

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

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

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

Thanks for reading, and enjoy!

Sean Daily

Series Editor

Copyright Statement

Chapter 1: Pre-Packaging Planning

Welcome to the world of application repackaging! Like most systems administrators, you have a need to enhance your knowledge of a product or technology, but the time you have to devote to that endeavor is in short supply. It would be great to spend all the time you need to fully understand a process before you become "the local expert." However, the reality is that you're likely only allowed a few minutes to fully understand all the intricacies of application packaging before you begin the desktop migration process. This Shortcut Guide is written to help you quickly get up to speed. I won't delve too deeply into the topic or spend large amounts of space explaining all the variances of each parameter. Instead, this guide will cover the topics at a level that will have you up and running quickly. Hopefully, you will find the format friendly enough to continue using the guide as a reference as you move forward in the packaging process.

The focus of this guide is on packaging applications for a migration of one desktop operating system (OS) to another. The migration may be Linux to Windows, Windows to Linux, or Windows to Windows (say, Windows NT to Windows XP, for example). Although the specifics for each OS are different and require a good knowledge of both systems, the general steps for getting applications migrated are the same.

This guide contains three chapters among which I've divided the content logically into pre-packaging, packaging, and post-packaging. For anyone familiar with general application packaging, you will note that most of the steps for desktop migration packaging are the same as those for regular packaging. Where particular attention is required for the migration steps, I'll expand and call your attention to it. Most of the examples in the book will use a Windows-to-Windows migration path. So let's get started by taking a high-level look at the migration packaging process.

Overview of the Migration Packaging Process

In the context of this guide, a repackaged migration is the act of taking a functioning application from one OS and repackaging it to automatically install and work on a different OS. To accomplish this feat, a number of conditions must be met. The list of conditions can include, but is not limited to

- User data
- Documents
- Macros
- Programs
- Browsers
- User settings

We'll focus primarily on the program condition; however, this focus isn't meant to diminish the importance of the other listed conditions. The general steps that we'll explore can be applied to the entire list.

If the scope of your migration is a couple of applications on four or five workstations, going to each workstation and manually performing the updates while you reload the software is probably the easiest method. You can save off data, tweak the system, manually install software, connect the printers, and move the user's Web links over. The time cost savings between the manual method and developing an automated process favors the manual method in such cases.

However, if your migration scope is in the hundreds (or thousands) of workstations, a manual approach will not enable you to complete the migration in a reasonable amount of time—your lifetime. The general steps taken to perform the application migration should be repeatable and flexible enough for the next application. The highest time expense will be in creating the initial process, and the time payback will appear with each subsequent use of the process.

The topic of desktop migration is complex. As the variables for each OS are added to the equation, the complexity grows. In addition to the issues of detailing the OS changes, there are considerations such as ensuring that users get their data moved, their pointers swung over, their macros are moved, and so on—a major part of the desktop migration process is the applications that reside on the machines.

As Figure 1.1 illustrates, desktop migration packaging follows the same basic steps as initially packaging an application:

- Planning
- Aggressive review of the plan
- Clear definition of the end result
- Standards review
- Detailed inventory of the applications
- Close integration with the migration project
- Testing
- Documentation

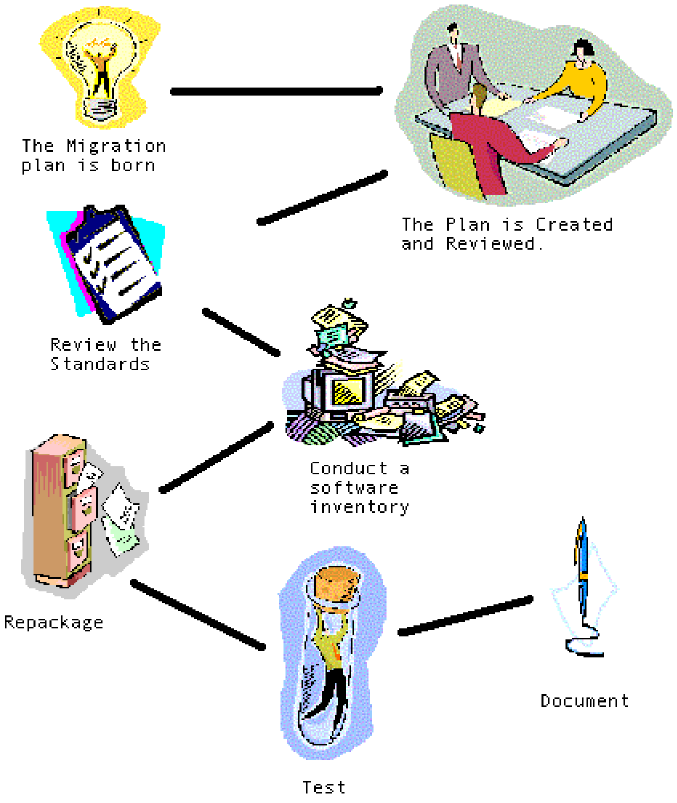
The Migration plan is born
The Plan is Created and Reviewed.
Review the Standards
Conduct a software inventory
Repackage
Test
Document

Figure 1.1: Overview of the desktop migration repackaging steps.

One aspect of migration packaging that is often overlooked is the deployment method. Will the same deployment software and deployment software version be used on the desktops? Chances are that the deployment process will change—if not to a different set of software, at least to a different version of the current one. For example, you may opt to use Microsoft Systems Management Server (SMS), but will your old SMS 1.2 version that worked for NT work as well in the new Active Directory (AD) and Windows XP world? Will you need to move to SMS 2.0, Marimba, or another solution? How will the new deployment method impact your packaging? We'll delve deeper into the discussion of deployment method as it impacts the repackaging process later. For now, mark deployment method as something to review during your planning process.

There is a lot of preliminary work required before you begin the packaging process. Most of the questions and issues are easily solved. Like most items within the IT world, the first effort is comparatively longer than any subsequent ones. Be thorough and detailed for the first repackage. The extra time and effort you spend will pay for itself quickly the next time you do a repackage.

Although there are many commonalities, repackaging for desktop migrations is not exactly the same as repackaging a version change in an application. Aside from the application differences, the scope of the package needs to address a multitude of added issues. OS variances, security changes, possibly drastic changes in file systems, and user account differences are just some of the items to be considered. Take heart, it is not that bad—even though the undertaking might sound daunting when you look at each piece, when you address one item, others are normally solved (or partially solved).

Planning the Migration

Now it is time to focus on just the application packaging effort. Remember this effort is only one small part of the overall migration, and you must consider the entire process. With that said, let's get into the reason you are reading this guide.

The decision to migrate to a new desktop has been made. The directive has been handed to you: make our applications work on the new desktop. You quickly take a mental inventory of the hundreds of applications in your environment, look at the time schedule the migration team has handed you, calculate the hours in a day, contain the sudden surge of panic that wells up in your chest, and sit down to begin the process.

Whether the scope of the task is one application, a couple dozen, or hundreds, you should ask the same questions:

- What issues exist from the current package?
- What package format will be used?
- What packaging utility will be used?
- What are the needs of the application, the software, the hardware, and any peripheral devices?
- Will there be multiple versions of the package (to support both the current OS and the new OS)?
- What is the application inventory—how many versions of the application exist, are there other applications that interact with the repackage target?
- What were the successes in the current packaging process?
- What do the users constantly ask for—will the packaging standards change as a result of any requests?
- What new functionality does the new OS present that needs to be accounted for?
- Does the application need to be migrated or will a new version be required instead?
- Does the Accounting group have a different set of applications than the Warehouse group? What applications are used in common with both groups? An inventory of all applications needs to be conducted. Be sure to account for applications by any logical groupings.
- Will the desktop migration require concurrently supporting two OSs or is the migration a cutover?
- What are the dependencies for applications? Does one application require data or DLL files from another package?
- Will different file versions conflict between applications (having a database or written record of the file versions for each application can help prevent problems later on)?
- How will testing be conducted? How extensive? Will the application be tested by itself or will it be tested in conjunction with all the other applications?

Legacy Applications

Legacy applications may not be perfect, but they function in your current environment. OK, stop snickering! We all have a list of production applications that should never have made it to the desktop. Even those problem children can help in your efforts. Start answering some of the previously listed questions by looking at the legacy applications.

> ✎ I'll use the term *legacy application* to refer to the older applications that will be migrated to the newer format.

In the rush to migrate everything, the most obvious question might be unintentionally bypassed: Does the legacy application really need to be migrated at all? The entire application may have become useless. Applications tend to take on a life of their own and may remain on systems long after their reason for existence has expired. Now is a good time to review the actual function of the legacy applications in your environment. Dropping the application if the function is performed better by another program is better than migrating the old dog just because it was there. However, be aware that if an application has been around a long time and is still used (even lightly), it may be difficult to drop it. Users may not like the change.

> ☞ Remember to evaluate the actual need for application(s). Don't bother repackaging an application that has outlived its life span.

If all we had to do was drop the old applications and install brand new ones, this guide would be extremely short, so we'll assume that all the applications need to be migrated.

Legacy applications have a strong advantage over newer applications because legacy programs have a history. Use that history to make your life easier. What to look for in a current legacy application:

- What are the historical issues or problems of the current version? Have the problems been solved? Will those issues need to be addressed again? If a patch was applied to the old printer drivers to get the reports out correctly, the reports should be checked carefully when the application is installed on the new desktop.
- Does the legacy application require an upgrade to work properly on the newer desktop environment? If the current legacy application is being used on a Windows 3.11 desktop (remember that one?), chances are that it will require a new version to function properly within an AD and Windows XP environment. Installing this new version might speed your packaging process.
- Performing a manual install of the application before the repackage effort begins will ensure that there are no problems with the application on the new desktop. Troubleshooting problems are easier if the focus is directed correctly, either at the application itself or the repackaging.
- What changes in standards have come about since the application was initially released? Have you moved from a server-based to a client-based application world?
- What about the DLL, EXE, COM objects, and so on that comprise the application? Are there any API calls the application uses that are changed or different on the new OS? If you have a conflict resolution database or document, review it. If the application is an in-house creation, the developers should be consulted.

- What issues have the users brought up? Is the application using the old company logo? Are there terms in the legacy application that are considered outdated? If these types of user concerns are addressable during the repackaging process (though changes such as modifying the contents of an INI file), now is the time to apply the changes. Otherwise, the issue needs to be directed back to the developers.
- Does the legacy application call for data files from a source that is scheduled to be removed or changed? Should now be the time for the change?

☞ Legacy applications have a history in your company. Review that history for any trouble spots that might appear on the new desktop.

Working with User Requests

When a migration becomes known to the user population, be prepared for the flood of Requests for Enhancement (RFEs) that will be disguised as problems. Determine whether the trouble report is a real problem or an RFE. Even if the issue is a true problem, evaluate the issue to decide whether you should apply a fix now or later. Problems may be postponed if they cause too much interruption to the repackaging process. By the same token, it may make perfect sense to include an RFE at this time.

For example, suppose you receive an RFE to move the default Save As settings to a network home directory instead of the local hard drive. You could make such a change through an adjustment to the repackaging process. However, the RFE should most likely go to the developers if it requires an adjustment to the computational logic on a monetary formula, for example.

Consider the Migration Package Format

When planning for the migration, decide on the format to package it in. Will you use Windows Installer (MSI) technology, an in-house technology, or a third-party format? Obviously, ensuring that the package is created in the standard format will impact the migration. Additionally, you need to review the current format. Can the chosen repackage utility perform the conversion for you or will it become a manual step?

✎ Deciding on the new format for packaging will obviously impact your work. Conversions from one package type to another (for example, from SMS Installer to Windows Installer) can complicate the entire process. Choosing the proper repackaging utility is critical.

📖 I will discuss how to choose the proper repackaging utility in more detail in the next chapter.

You may find issues with the migration if the application has OS dependencies that are not available on the new OS. Windows File Protection (WFP) exists to prevent the replacement of critical system files. If the application tries to replace any file marked as protected, the installation may fail.

> 🕮 For more information about WFP, see the Microsoft article “Description of the Windows File Protection Feature” at http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B222193.

Standards

Whether they are company-specific or a general acceptance of industry practices, standards are essentially the documented rules everyone adheres to for packaging and application creation. It is important to remember that these rules have some fluidity—they change over time as conditions change. As we are migrating legacy applications to a new desktop environment, chances are that the standards used in the initial creation are not the ones that the company wants to see enacted. That is assuming, of course, that the legacy application was actually built around a standard of some sort.

Standards are not strictly a technical issue. They can be extremely political depending on the culture of your company. As the IT person doing the packaging, you should be aware of the standards and how they will impact your packaging effort. For example, if the desktop for the general user population is locked down to a low security level and the new software deployment method defaults to installing the applications in the security context of the user, the install may fail with the deadly Access Denied message. The packaging will require some method of installing with elevated rights. Such a requirement highlights the need for a group agreement during the entire migration process (of which packaging is one piece). If one side of the migration team decides on a locked down desktop without communicating that to others, well, you can see the problems. Does that piece of communication failure actually happen? Everyday.

Like most aspects of systems administration, one right answer for standards does not exist nor is there a definitive list of standard policies to use. The concept of standards is complex, but we are concerned with the following three common categories of standards:

- Platform—The platform of the desktop determines the standard, which can be based on hardware, OS, or a combination of the two. The rules and policies for software are determined by the platform on which the application will be running—desktop, network, server, or environment related.
- Application—This category of standards can be a standalone set of standards or can be a subset of any other category. Normally, these standards determine how applications run and which regulations applications must adhere to in your environment.
- User—This standards category determines user groups, associations, data access, and whether the user is locked down or the desktop is left open.

Security is a standard that falls into every category. It is important to understand the security aspects in your packaging. We will get into the details of security in the next chapter. For the planning phase, be sure to consider security as a factor and note any special needs when the standards are reviewed.

If your company does not have a packaging standard already set up, now is the time to create one. Some items that might be included:

- Any company data—Coloring, logos, placement of disclaimers, addresses, phone numbers, and inventory location are a few items that fall into this category. This category needs to be carefully reviewed. If the changes are easily performed without having to delve into the application internals, now is the time to address it. The standard may dictate a new logo, but unless a prepared file is easily replaced in the existing package, the adherence to the standard will not be forthcoming from the repackaging crew (it falls into the hands of the developers).
- IT department registry requirements—This category covers any registry locations for software versioning, inventory data, custom registry keys, any limitations on what registry keys can be updated, and any tracking data for IT use.
- Default directories—Is user data saved on the workstation or on servers? Can the root of the system directory be written to? Are applications placed in the C:\Program Files directory or written to a D:\User Program directory? Is there common company data in use by multiple applications?
- Security settings—Do various levels of users have different access levels? Is the system directory off-limits to anyone but an administrator? Can applications write to certain directories but not others? How often is the systems administrator password changed?
- Advertising setup—If your deployment method uses advertising, how is it used? When is the advertising done: logon or scheduled or immediately? When should an application be published versus advertised?
- Antivirus settings—Which antivirus solution is in use for your company? Are scans performed at set times or when a user logs on? Will antivirus updates impact delivery methods? Are certain package types excluded from scanning?
- Group Policy—Which policies are global in nature? Which are local? How do the policies impact applications during installs or repairs?
- Clean up—What are the clean up requirements for applications? If a package fails to install as a result of space issues, are there defined file types that can be automatically scanned and removed before the application tries again? How much of the installation package can remain on the client system? Are there any security-sensitive files that need to be hidden or removed?
- Installation parameters—Is there a central location for installations or does the application install from the local system? Are there any requirements for space checking prior to installing? Do in-house programs (macros, VBScripts, and so on) require some form of stamping to confirm that they are OK to run? Are applications required to be self-healing aware? Do they perform verification at certain times? Can the user trigger repairs or verification? Are repair functions performed from a remote location?

☞ It is easy to get extensive and detailed in writing standards. If your company's set of standards are overly detailed, they become useless in a short time span as technology continues to evolve. If the standards are overly vague to prevent them from becoming dated, they are equally useless. Try to envision how the proposal will be implemented in the future—say 2 or 3years from now.

> ✎ Standards creation and review is a group process. Each proposal should receive wide-scale review and acceptance (or at least a majority consensus); otherwise, the standard will be ignored and won't be enforceable.

Standards need to be enforceable. You need to determine the answers to the following questions to avoid conflict later: Who will be the cop that determines a standard has been broken? Who is the judge? For every rule, there is an exception; who will approve the exception requests? How will the exceptions be processed? As sure as Microsoft and Sun Microsystems will continue to battle each other, you can bet some application will need an exception to a standard to produce a report that the CEO of the company is personally waiting for.

Make the exceptions and enforcement clear and simple. Otherwise, the process will be ignored. A clear statement indicating any application that does not follow the standards will not be placed into production is effective. The follow-up statement will explain how exceptions will be processed.

Supporting Multiple Application Versions

The next time the planning meetings get boring, toss out a simple question that is sure to stir things up a little. How will the applications be cut over? If a large desktop migration is being undertaken, chances are that everyone that uses XYZ application will not be migrated at the same time. Thus, the need for supporting the old desktop and the new one comes into play. How long the legacy application is supported and how much of the new data is allowed to be used by the application is an area of debate. There is no pat answer to this question; it all depends on the company, the type of data, the critical nature of the application, and the scope of the migration.

The migration packaging process needs to be aware of the schedule not only for a particular application but for the entire user community. The need to support multiple versions of an application is one aspect of the problem. The data source or output for an application must also be considered. If the legacy application requires a different version to function properly on the new desktop and it reads/writes to a central database, can the legacy and new version coexist on the same database? If the new version requires an upgrade on the database, is it backward compatible with the older version? What schema changes are required on the new version? Again, lots of questions without a cookie-cutter answer. The solution becomes a mix of all the variables in your company at the current time.

To add more fuel to the fire, what happens if the new application is resolving user IDs from AD? If the legacy application is based on a workgroup membership and both versions of the application use the same database, there may be all sorts of problems.

Supporting two versions of the same application on two platforms obviously presents a lot of issues and a few potential nightmares. Let's face it, the best method is to have a single version of the application on a single desktop environment. Sadly, such is rarely the case. When you are faced with this issue, plan on a little extra time in the testing phase.

As I stated earlier, there is no single answer to all the possible scenarios. However, I find there are a few common solutions to the multiple-version issue that have proven effective:

- Freeze the old version from future upgrades or changes
- Carefully plan the migration rollout to include entire groups or departments at the same time instead of doing bits and pieces
- If a common database is needed but is not compatible with both versions of the application, it may be possible to keep two versions of the database active for a period of time—a nightly (or hourly) data conversion between the two databases can keep them in sync; whether this method will work for your environment depends on the critical nature of the data, the size of the user base, and the volatility of the data.
- Another approach to the common database conflict is to get buy off from the user base that the old data will be in read-only mode and used for reference only
- Depending on the nature of the common data and the critical nature of that data, an added process (or application) may be needed to get the data updated and converted to a format useable by each version of the application

When the desktop migration schedule is laid out, the previously conducted software inventory must be consulted. Look for any instances in which multiple versions need to be supported. Not only will the packaging process be faster with one version, the administrators on the support end will be appreciative when it comes to troubleshooting.

> ☞ Supporting multiple application versions or multiple desktop OSs does not stop at the application. Investigate the data integrity issue and how the multiple software versions interact with data from a different version.

File Contention (DLL Hell)

Packaging for migrations is not just packaging the individual application; you must also consider other applications and plan which groups in the user population will get which applications. Additionally, you will need to decide how applications will be packaged in order to support multiple applications.

Why? It used to be called DLL Hell—the problem and some solutions go by other names, such as conflict resolution, application isolation, and version conflict management. They all boil down to the same issue: multiple applications using the same file (EXE, DLL, COM object, VXD, and so on) but requiring different capabilities from incompatible versions.

Backward compatibility is an often-recited mantra during programming sessions. Try as software writers will, they eventually have to draw the line at some point. Do you stay backward compliant forever or only for the past few releases? For the OS files, the issue is even larger. It might be impossible to stay backward compliant for new OSs, yet a DLL may reside in both OS versions.

One process to resolve the issue is to isolate all the files of an application to their own directory and never rely on shared access. Doing so makes installs, repairs, removals, and upgrades a lot easier as the only application to use the file is in the directory. However, there might be issues with isolation for system files. Depending on the OS, having multiple versions of the same DLL loaded may not be possible. Running multiple instances of the Java Runtime Environment (JRE) is normally possible. Attempting to run multiple instances of RUNDLL32.EXE could cause issues.

There have been great strides made to solving this issue in recent times. Central databases (either on a workstation or a server or in the packaging process) that track the use of files in all applications are becoming more common. This functionality helps in tracking which application is using a file. The idea is to catch potential problems ahead of time and work out problems before outages occur.

We will get into this issue in more detail in the next chapter. For the planning process, it is important to be aware of the fact that DLL Hell is a possibility that you might have to deal with.

Identify the Resources Needed

A resource is anything that an application needs to be repackaged and become functional. A resource encompasses files, hardware, and people. Yes, people. For example, the packaging administrator is a resource to the migration team. So what resources are needed for this effort?

- Information about the application
- Information about the deployment method
- Information about the migration schedule
- Knowledge of which applications will be used along side the current package
- Who will perform the packaging
- A test environment
- People to test the application in the packaging lab
- Users familiar with the application to test the package

We’ve already covered the first four bullet items. Once you understand the legacy application and what is needed to create the package, the application can begin the migration process. Let’s start this section with a brief discussion about the packagers (people) and what is need to have the right people perform the repackaging function. The repackaging staff should have:

- A good knowledge of the new OS as well as the old desktop OS. At a minimum, they need to understand how the new OS works and how to troubleshoot install issues.
- Familiarity with the repackaging tool of choice
- Access to the migration plan and deployment method
- Access to a test environment
- Good troubleshooting skills

The newer repackaging utilities such as InstallShield's AdminStudio and Wise's PackageStudio ease the complexity of a repackaging effort. However, if the person doing the work is not familiar with the concept of a Windows registry or what a COM object is, the effort is doomed to fail. There is always some tweaking required to get a package to perform as expected. To reach the desired goals, the administrator doing the work needs to understand the basics of the target OS.

> 📖 We're skipping over the repackaging specifics in this chapter, as it is focused on the pre-packaging steps. The next chapter will go into depth on the packing needs.

Testing Lab Resources—The Images

Once a legacy application is packaged, it needs to be tested correctly in the proper environment. No matter what pressures are applied to meet deadlines, this step must not be taken lightly. When the packaging is performed, it is best to perform package testing on a representative system of the target desktop. The packaging machine should be void of other applications (unless that is a packaging requirement) and should be clean without extraneous user entries. Newer packaging utilities can monitor the installation process and capture only the changes. This functionality reduces the requirement of a completely clean system. It is considered a best practice method to start with the clean system.

The test environment is one of controlled chaos. The desktop OS must be stable and of a known level. It is an environment of constant change within that known level. Let's spend a few minutes going over the testing environment. Make sure that the test computer is just that—a test computer:

- It is NOT your workstation on which you email, prepare documents, and play FreeCell
- It is NOT a development platform
- It is often reimaged to provide a clean environment
- It is representative of the real-world systems
- It is NOT the oldest piece of hardware in the company

The test environment is fluid. It must be rebuilt repeatedly to provide a known clean state for deployments. Ideally, the test computer is not used for anything but testing as it will constantly be wiped and reloaded. Remember that the test computer is the fundamental verification step for the repackaging process.

When setting up the test area, be careful about the quality of machines with which you populate your test area. Make sure that the test computer is compatible with the existing machines in your user population. Testing Windows Server 2003-based systems on an IBM PS2 with 16MB of RAM and a 286MHz CPU is a recipe for trouble! OK, that's an exaggeration, but you get the point.

The goal of migration repackaging testing is to ensure that the package installs and the application runs on the systems. The test computer doesn't necessarily have to be fast or new; it must, however, be within the hardware requirement constraints for the deploying package and be representative of your installed hardware base.

The testing of your repackaged application should go through three types of computer setups: clean, baseline, and working. Each stage builds on the preceding stage. The clean system starts the process, the baseline system is built from a clean platform, and the working phase is built on the baseline stage. Figure 1.2 illustrates the testing progression.

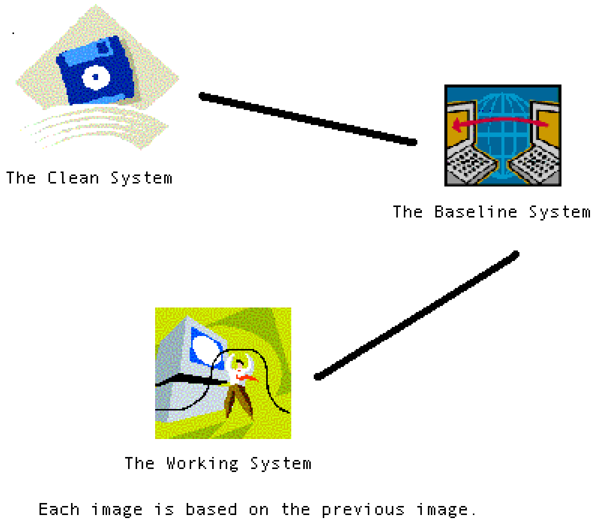

Figure 1.2: Each test image is built on the image before it.

Thus, do you need at least three computers? Not really. A single system would work through the process, but would be slower as each phase needs to be built before testing can begin. If the funds allow, having at least one system dedicated to each phase would speed the testing process. As the repacking testing progresses from one system to the other, the prior system can be rebuilt and prepared for the next application in line. If issues appear with the repackage, you could easily move backwards to a known level (if needed). However if your budget and resources are limited, you may need to work with only one machine. (For information about an alternative to this three-system process, see the sidebar "Virtual Machines.")

Virtual Machines

The virtual machine concept is gathering acceptance in the PC and Windows world. The IBM mainframe environment used the concept more than 20 years ago. Essentially, an OS is brought up on a PC as normal. The initial OS provides a virtual machine for subsequent OSs so that multiple images can be run on the same machine, usually at the same time. In contrast to a dual-boot environment in which only one OS is active at a time, a virtual machine runs multiple OSs at the same time. The virtual machine OS handles all the hardware calls and shares out the physical resources to the virtual OSs. This functionality allows expanded use of limited resources. In effect, two or three (or more) computers can be run on a single physical machine. For the test environment, this provides an option to have multiple images running on the same system.

The next chapter will cover virtual machines in more detail. Virtual machine vendors include Connectix and VMWare.

A *clean* test computer contains the core OS as well as any drivers needed to create a functioning *and generic* desktop.

Starting with a *clean* test computer gives you a lowest-common denominator to work with and build from. Everything builds from this ground-floor platform. Depending on your goals, this baseline may be a starting point for testing or additional image builds.

Note the absence of service packs and security patches in the clean system. This method of creating a clean system is a small area of debate among technicians. I like to create the clean system as a very basic one in order to fully test any and all packages. A lot of my work involves system-level aspects. In the course of my work, I may run through various service packs, numerous mixes of patches, or any combination of the preceding. Testing with this method provides the ability to test service pack installations and OS patches from a system that is completely void of any customization. It allows the ultimate flexibility. If my testing is based on applications only, the clean system will have the service packs and patches loaded to put the system to a common level.

Remember the fun and games with NT 4.0 Service Pack 6 (SP6—or SP4 or SP5)? SP6 encountered problems and finally settled down on SP6a. If the clean system had been built with the first SP6, simply installing version SP6a would not have provided a 100 percent clean install. To stay pure, the clean system would require a wipe out of the image and would be rebuilt with SP6a. The point: By having a clean system that is bare bones simple, you can avoid complications such as those that SP6 presented.

In its clean state, the test computer is pristine and untouched by any extensions other than those provided by the OS install program and needed by the hardware to run. This system is the one with the oldest version of the OS in use in your company (or target scope). If you're supporting multiple OSs, the clean computer will have to be reset for each one.

By oldest OS, I am talking about the oldest version for the level of OS in use. Don't package or test against a Windows 3.11 system if your environment is Windows XP.

The purpose of repackaging is to replicate the same final goal as the original install program—create a working application. By targeting the oldest version of the OS, we are forcing the packaging effort to ensure that the proper drivers and files are loaded. If the packaging program sees a file version that is part of a newer OS, it may not mark that file for inclusion in the package. When the package is put on older systems, the expected file version may not appear.

Of course, a clean system must be built from scratch. It must not have any pollution from other packages or system modifications. Once you have the system built, use a drive imaging product such as Ghost or PowerQuest DeployCenter to snap the drive and save the image off. This way, you can rebuild the clean system with little effort and faster.

A *baseline* test system is built on the clean system and should be representative of your environment. On this system is where the application testing begins.

A *baseline* system is built from the clean system. On this system, you install the service packs, security patches, and updated drivers. It should also include user accounts, Group Policies, and any other *existing* applications that are common across the target user population. This system is the lowest-common denominator found across the entire enterprise (or company) and has all the customization the company standards require. Consider this the platform to begin your application testing.

The baseline system may also contain any common applications needed for the target group. Note the word *common*. The baseline is still a generic platform and should function for a large group of users, not a small target audience. Unique variations of software are applied in the next phase.

> 🕮 For more information about establishing and deploying a working baseline, see Chapter 2 of Bob Kelly's *The Definitive Guide to Windows Desktop Administration* (Realtimepublishers). This book is available for free from a link at http://www.realtimepublishers.com.

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 IT group. It is common to have different baseline images, but don't get carried away with too many baseline images. Remember that this system is a common image across the company.

> ✎ A *working* test computer has the software application installed and functioning properly.

The working test computer is the one on which all the applications are applied for the target audience and the application being tested is applied. This platform is actually a state, not just an image. It starts with loading all the applications needed for the target audience and ends with the successful installation of the repackaged application. When you have a good working system, there are no conflicts or problems. It is the closest you can come to what the users will see without actually loading the new application to their systems.

Once the test environment is established, the next project is easier. The first time you create the test systems and the repeatable testing stages may require an investment of time that will see payoffs immediately. A best practice is to do initial tests against a system that you have no regrets about wiping out.

Testing requires a repeatable process. Document that repeatable process now. Write down where the images are, how to load them, what is contained in each image, when each image was created, and against what hardware platform. Six months down the road, when you can't remember which image has version 6.5.22a of a driver, the documentation will come in handy.

> ☞ Testing must be based on a repeatable process. Make that process simple and easy to perform.

Testing Lab Resources—The Staff

Once the test environment is established and prepared, you need to have someone test the application in the lab. Depending on the size of the company, this individual may be the same person who created the package or it could be an entirely different department with a separate staff. In either case, the testing staff needs to be

- Methodical
- Detail orientated
- Patient
- Technically knowledgeable
- Good communicators

A properly created package can install itself without problems or user interaction (or minimal interaction). Thus the testing of the install does not need to be done by a highly technical person. That is assuming, of course, the package installs properly and there are zero problems. Testing, by its very definition is confirming the functionality of a package. Unless the person doing the packaging is also the person doing the testing (which is not always a good thing), you want someone doing the testing who can assist the process, not hinder it.

Once the package has been manually run, be sure to deploy it to the test machines using your deployment method. Doing so will ensure that there are no interaction issues. As most deployment software focuses on simplicity of repetitive use, the same person that did the initial testing should perform the deployment. By doing both the manual test and the software deployment test, you will be able to isolate any issues to either the application or the interaction with the deployment method.

If a package has difficulty installing, the tester needs to have enough knowledge to troubleshoot the basics. Is the problem with the OS, the network, a failed hardware component, a bad password, or a missing file? Testing results should be relevant to the package, not a question of the environment. With a technically proficient testing staff, the results will not be a question of the environment.

The testing person should prepare a written document detailing the results, what was tested, what image was used, and whether the test was successful. Each company and repackaging effort is different, so the report format and content will be unique for each company. Plan ahead about what “testing” means and what the parameters of “success” or “failure” are. The testing staff should be fully aware of those definitions and work within them.

Testing Lab Resources—Testing the Packaged Application

Installing the application in the test lab is one aspect of testing. At this stage, the application is on a system and it looks good. We all know looks can be deceiving. Just because a package is installed does not mean it is functional. Approach people familiar with the application to test it. Complete testing will check every single option and component in the application against every application that will be running along side it. Although this level of testing won’t happen all the time, it is a good goal.

The bare minimum testing should check out the primary functionality of the application. Does it print properly, are the reports created correctly, is the user interface presented as expected, can the process trap errors? The goal at this level is not to confirm the programming aspect of the application, it is to confirm the repackaging was complete and did not drop registry entries or slip a file out of the package. Having someone familiar with the application can speed that effort as they will know what it should or should not do.

Testing Against the Real World

While we are on the subject of testing, we need to address real-world testing. Plan for it, schedule it, and do it. Repackaging legacy applications for a new desktop assumes that the application exists in the user environment already. Thus, finding users to conduct the lab testing should not be a problem. The next step is to ask those same people (if possible) use the product in the real world. This step might not always be possible, but it does aide in reaching the zero-defect goal for which all packaging efforts strive.

Assuming it is possible to load the new desktop into a pilot situation, install the repackaged application on the desktop of the volunteers and have them use the application in their production duties. Aside from testing the program, this setup will also provide added reassurance that the packaging was complete and nothing was missed.

Performing real-world tests also helps to keep the negative interaction with other applications out of the mix. Depending on the time and resources available for testing, doing a full check against other applications is ideal. Of course, if the political pressures dictate that testing is only performed during the lunch hour of a single user, the breadth of testing will suffer. It is best to be thorough and check all applications that might touch the repackaged application.

The length of time for testing is different for each project. An hour may suffice for some; a month may be needed for others. The end result is ideally to have 100 percent confidence that the legacy application was repackaged correctly and that it meets all the standards and criteria of the project.

Documenting the Package

Documentation is a unique item for each company and each package. Some prefer to have extensive details for everything; other companies only require simple statements about a package. There are also some places that don't have any requirements for documentation; those are generally the small shops.

Documentation is an area that few administrators enjoy, but when problems occur, the documentation is one of the first places administrators look for answers. Thus, although creating it is a pain, it is the source of complaints if it does not exist. While planning the repacking effort, be sure to set aside time for documentation. If the desktop migration project is large, now might be a good time to set up or revamp standards for documentation. The goal of the documentation should be to provide enough data for problem solving. Ensure that the detail level is deep enough so that the entire package is laid out in black and white or at a minimum, that there are enough details to provide directions on where to go for the answers.

For off-the-shelf software, the documentation could consist of a copy of the software manual. If the software is packaged in MSI format, it will have a list of the codes and options for the MSI setup. Most documentation covers three general topics: cover sheet, trouble shooting data, and package details. Again, the level of detail is up to your company's standards. I suggest that application cover sheets contain:

- An overview—A paragraph or two about what the application (package) does and who the users are; this overview is written at a high level.
- Contacts
    - Vendors—List which vendor(s) created the software, their contact numbers, and any support numbers that are needed to access support.
    - Company—Who in the company is responsible for the program? List the department and a name or two; if the application is critical, off-hour contact numbers should be included here

- Package prerequisites—What is needed for the package to run on the desktop? Are other applications required to be installed? Is there a memory minimum? What about CPU speed? Anything that the application must have to function properly should be listed here.
- Installation directions—Short, simple, concise steps about how to get the application installed. The content level here is aimed at a knowledgeable administrator. It should contain step-by-step directions to get the application to install and run. Screen shots to help present the information are a good idea, just don't go overboard.
- Post installation—What tasks are required after the install of the application? Is there any required clean up?
- Troubleshooting—This section should list certain areas to check for details when problems are being researched. Where are the logs located? This area of documentation is an overview for troubleshooting assistance.
- Application flowchart—If the application is an off-the-shelf application, the documentation will provide this information. If the application is an in-house creation, the programmers should provide a flowchart of how the application works. The flowchart can take many forms: a programmer's detailed chart, a chart of functions, or a list of menus.
- Known issues—This area of documentation is often overlooked. During the creation of the package, certain issues are likely to appear; detail them here if there is a possibility that errors may appear in the production world.
- General comments—Always leave an area open for added notes, diagrams, print outs of scripts, error messages, or links for added data.

The next section of documentation should cover any troubleshooting data that could be beneficial in the future. During the repackaging process, the information is readily available and easily located. However, the goal of the documentation is to provide information to help the administrator 18 months from now at 2AM on the Sunday of a 3-day weekend. Whatever information is available to meet this level of need should be included:

- Registry keys and values
- Copies of any scripts
- A list of the files and their locations
- A list of all files listing the version data for each file
- Any listing of conflicts with OS files (or other applications); this list comes in handy down the road when a new service pack is installed by the user without going through the normal application packaging process (yes, it happens a lot!)

> ☞ Basic documentation should be created with two goals in mind: There is enough data in the documents to begin troubleshooting issues under the most stressful of situations and that the creation of documentation does not take longer than the repackaging process.

Clean Up

Any install of an application will leave residual "junk" on the workstations. As hard drive sizes increased on the workstation, the need to be diligent about clean ups receded. Although space shortage is less of an issue than it used to be, cleaning up after installs is still necessary. The following questions will help you develop cleanup standards for your company:

- How does the install package perform the install?
- Is it obtaining the data from a network share and running administrative installs from there?
- Does the delivery mechanism copy the files to the local system and run it from there?
- What happens to the package directory when the install is completed?

Leaving the install package on a workstation presents several pros and cons. The pros include:

- Repair source files are local, which allows for faster repair functions.
- Reinstalls are quickly performed from the local cache.
- Laptops do not need a network connection to perform repair or verify functions.

The cons include:

- If the install package contains passwords, there is a security risk.
- Delivery consistency may be impacted. It might be possible to copy the install package to another system that should not receive the package. In addition, if a user ID has access to the install directory, desk sharing users could install applications for which they're not authorized.

If a desktop (workstation) will be accessed by the general public, it is not a good idea to leave your installation packages on the local system. Even if the desktop is locked down, there are always backdoors that can be accessed. Remove one source of potential problems from the system by removing the installation package.

Cleaning up does not exclusively entail removing the source package. The registry, temporary files and directories, logs, and system variables (such as the path statement) need to be reviewed and purged as necessary. Performing housekeeping duties as part of an installation will increase the success rate for all applications. Taking it to the extreme, if installs are not cleaned up after they are applied, there will eventually be a space shortage on the system. Constant additions to the path statement for one-time install runs will reach the statement size limit and cause problems. Unless careful system variable monitoring is used, applications could fail when they use the same variable name with different data content.

Temporary directories need to be cleaned on a regular basis. Space, file integrity, and security are the top three reasons to keep any common temporary areas clean. Often, installs will leave pointers or markers to indicate the progress of the install. Should a restart be needed or another application use the same pointer naming coding, the restart could fail or start in the wrong location.

The registry is a favorite location to record the status of installs. Failing to properly clean up the registry may impact installations or the activation of the application. When an application is removed, it must clean up the registry correctly or future installs may fail when the installation assumes that the application is currently installed based on registry entries.

> ☞ After an install, you should leave behind only information that is necessary to run an application. Any additional leavings may present problems at a later time.

Summary

In this chapter, we focused on the pre-packaging aspects of the application migration and how the overall desktop migration process impacts the repackaging process. Some of the items that we explored:

- Planning steps of the repackaging
- The importance taking an inventory of the existing applications
- The need to decide whether an application was to be repackaged or not
- The test lab and the testing resources, both machines and people
- Some of the issues with supporting multiple OSs
- How multiple versions of the same application impact both the input and the output of the programs
- Documentation
- Standards and the importance of reviewing any existing policies
- Why clean up is needed and is often overlooked

In the next chapter, I'll cover the repackaging process in depth. We'll explore file contention management, repackaging utility selections, various ways of getting the package together, and some pitfalls to watch out for.