



realtimepublishers.com[™]

The Definitive Guide™ To

Windows Software Deployment



Making Complex
Technology **SIMPLE**
Since 1985

Chris Long

Chapter 4: Methods for Deploying Software	84
History	84
The Parts of a Deployment Methodology	85
Choosing a Deployment Method.....	85
Sneakernet	86
Sneakernet Recommendation.....	87
Simple Scripting Methods.....	87
Scripts Recommendation.....	91
Email	91
An Example Email Deployment	93
Email recommendation	94
Web	95
Web Recommendation	97
Image Inclusion.....	97
Image Inclusion Recommendation.....	99
Third-Party Solutions	99
Push.....	100
Pull	100
Combination	101
Standalone	101
Suite.....	101
Prism Deploy.....	103
Marimba Desktop Management.....	107
Third-Party Tools Recommendation.....	108
Microsoft.....	109
SMS.....	109
IntelliMirror.....	111
Microsoft Recommendation.....	112
Rollouts	112
Pilot Users	114
Errors Are Found—Do You Continue to Roll Out?	115
Ensuring the Deployment Worked.....	115
Some Tools to Confirm the Deployment Went Out.....	116

The Human Error Factor 116
Summary 117

Chapter 4: Methods for Deploying Software

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

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

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

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

History

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

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

Network administrators began creating scripts and repackaging the applications into various methods they were familiar with. This method was met with varying degrees of success,

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

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

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

The Parts of a Deployment Methodology

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

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

Choosing a Deployment Method

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

Sneakernet

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

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

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

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

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

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

However, this interaction comes at the cost of removing users from their workstations while you're performing a Sneakernet install. While the install is going on, users are not able to do their work. So to help reduce that impact, the installer tries to schedule a convenient time that does not impact the user as much. Maybe the install is done during lunch, scheduled breaks, or

outside the normal work hours. This translates to the installer having to work longer hours or adjusted hours, and eliminates the interaction benefit I previously mentioned.

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

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

Table 4.1: Pros and cons of Sneakernet.

Sneakernet Recommendation

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

Simple Scripting Methods

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

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

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

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

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

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

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

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

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

- The user might be delayed in signing on until the install completes.

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

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

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

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

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

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

The scripting abilities inherent within the Windows OS can provide you with a simple deployment utility. Using the Browser data available on your network, a few IF statements and the resource kit utility RCMD.EXE, you can send out a simple spray script that will run a command on a target workstation in the security context of your user ID. I can already hear all the comments

about how other utilities or methods are better. These comments might be true—this instance is shown as an example about how to script a simple deployment method.

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

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

Listing 4.1: StartIt.bat

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

```
DeployIt.bat - Notepad
File Edit Format View Help
@echo on
echo ***** checking connection to server *****
if exist \\%1\c$\users\default\*. * goto fix
md \\%1\c$\users
md \\%1\c$\users\default

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

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

:end

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

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

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

Pros of Scripts	Cons of Scripts
Highly customizable method of deployment.	Not easily transportable to other application deliveries.
Depending on the skill of the script creator, may be quick to create.	Which programming language is used is determined by the script creator and might not be supported by others.
Provides one method to reach workstations that do not have other deployment methods installed on them.	Maintenance of the script might be difficult depending on creator's documentation of the process and the complexity of the script.

Can utilize other utilities to perform functions.	Might depend on other utilities that might not be on the workstation.
The logon process can trigger actions for you.	If used at logon, the startup time could be expanded and might require a reboot (or two) before the user can start to work.

Table 4.2: Pros and Cons of Scripting.

Scripts Recommendation

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

Email

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

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

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

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

Sounds simple, right? Everyone is on email, so why not use it all the time? The deployment mechanism is already on the targeted machines. All that is needed is for the user to click on a

UNC and run the install. So what is the big problem? The primary point of disruption in the deployment are the users. They are required to do something. Thus, the deployment success is at the whim of the users. If they have the time, understand the instructions, do not experience difficulty, and have the security context to run the install, they might do the install when they get the email.

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

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

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

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

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

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

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

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

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

An Example Email Deployment

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

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

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

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

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

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

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

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

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

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

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

Email recommendation

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



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

Web

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

- Using the browser and Web security features, you can provide added security to the application install. This security can ensure that only approved people install the applications.
- Applications can be displayed in groups. Arranging the displays according to business unit or functionality might make it easier for the user to locate the desired application if you have a large list to choose from.
- Installation auditing can be enhanced. You can track who is obtaining the applications, when they did so, and which computer and IP address they were sitting at.
- Additional logic can be applied to the installation process. For example, when the user clicks on the Web link, instead of just firing off the installation process, a Web screen could ask for the cost center to charge the licensing fee to and send an email to the manager of the cost center for approval. With the proper back-end programming, the Web-based process could even insert the accounting data into your accounting program automatically.
- Some level of network bandwidth control is provided by the browser and Web server. A direct file copy (or UNC trigger) does not allow for retries on poor connections. Most browsers will retry bad connections for missed packages and don't have the sending timeout problems that a direct UNC pull would experience.
- You can hide the true location of the files—not only to keep the knowledge secret (which you may want to do), but also to provide an easy method for moving the source files to another location if you need to. Changing the pointer on a Web link is easier than having to send out emails and notification to everyone explaining that the source location has moved.
- With proper security applied, the Web page can be made available outside of the corporate firewall so that users can grab the data from the Internet without having to become authenticated to the corporate infrastructure. If your user community cannot get to your corporate LAN, they could most likely get to the Internet and reach a Web address. This setup is obviously dangerous, so be sure about your security before you turn out the page to the outside world.
- Added text can be displayed. UNC locations are only directory and file names. A Web page can explain any company procedures associated with the install and provide a description of the application. If users need to do other configuration tasks, those can be shown at the same location. All this information can be provided in a nicely displayed page instead of relying on a README file the user may not see or open.

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

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

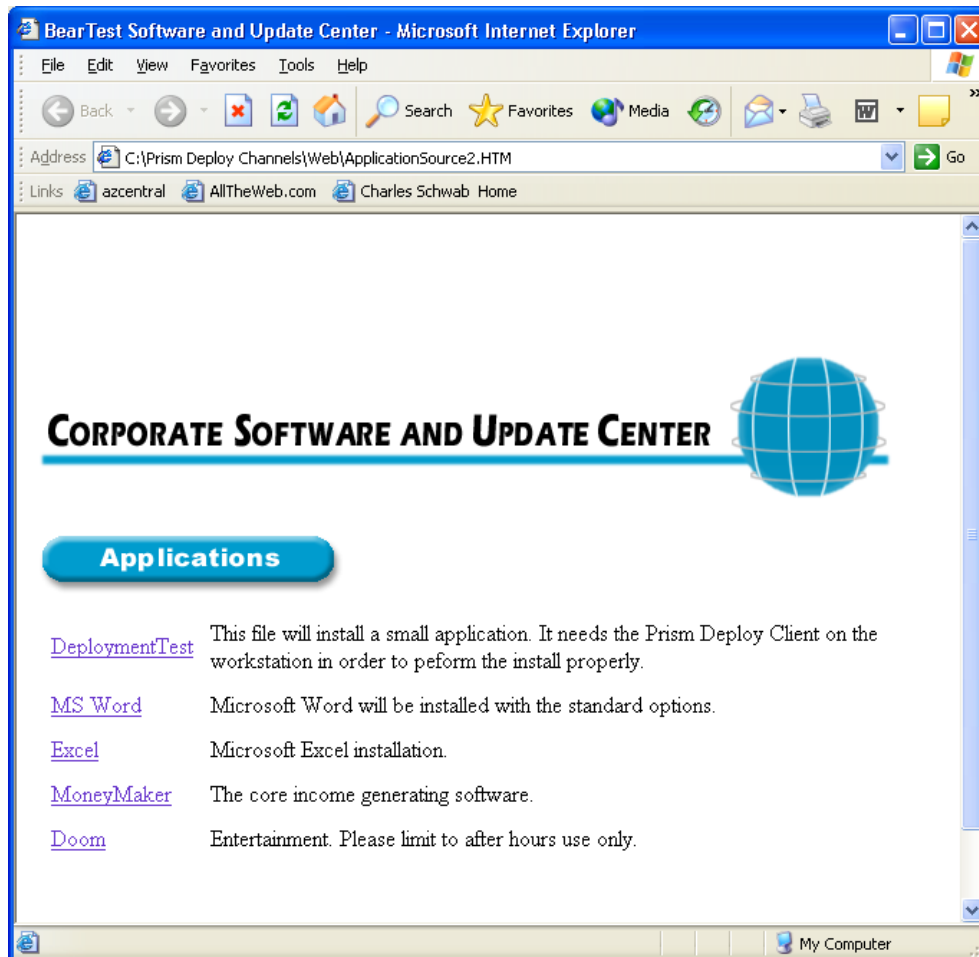


Figure 4.2: A sample Web page with links to application installs.

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

Pros of Web	Cons of Web
A central location can be established that becomes part of the company “common knowledge” for where to grab software.	The installation normally requires some initial action by the end user. Thus, it is user dependent to complete the installation.
Software can be segmented by any categories you want: departments, software category, project needs, and so on.	Web programming knowledge is required.
Front-end scripts can be applied to the Web page to assist in tracking, license issues, and cost.	The success of the deployment depends not only on the skills of the application packager, but also on the skills of the Web page designer.
The install location could be opened to users outside of the firewall.	Anything exposed to the Internet is a potential security risk and must be monitored.

Many commercial deployment software solutions are beginning to provide a Web-based method.	
The Web browser or server can provide an added layer of security.	
Moving the install source location can be done behind the scenes without impacting users.	

Table 4.4: Pros and cons of using Web deployments.

Web Recommendation

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

Image Inclusion

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

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

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

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

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

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

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

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

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

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

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

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

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

Table 4.5: Pros and cons of image inclusion.

Image Inclusion Recommendation

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

Third-Party Solutions

OK, now we get into the meaty deployment methods. Remember that deployments consist of getting the package created, getting it out to both online and offline users, and ensuring that the package properly got to your targets. There are tools out there that give you more and more control on all aspects, giving you one-tool-for-all type functionality.

Except for the image-inclusion method, using any other deployment methodology that we've discussed so far requires some or all of the following:

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

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

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

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

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

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

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

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

Push

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

Pull

The client instigates the deployment process and the server acts as a storage depot in the pull method. If your target machines are not always online, this method works as it allows users to gather the needed applications when they are online. If the client is not online when the server sends out the application, the client might not get it. If your clients move around a lot and need

their applications to be installed at the new workstation, a pull method allows the client piece to obtain the software on demand and not require a manual trigger action from a server. This setup translates to easier support for you.

Combination

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

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

Standalone

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

Suite

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

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

Deployment software takes care of the basics that you had to deal with using any of the previously mentioned methods: Network bandwidth considerations, scheduling a deployment time, back outs (or roll backs), and insuring the application arrived are all dealt with by the product. Suddenly, your job just got easier!

Every vendor has different thoughts about and designs for how to get your applications out. Most are GUI-based and have some sort of command-line options so that you can script the process. With a command-line trigger, you can use the deployment software with a Web-link trigger. By using the central Web page method we discussed earlier, a user clicks the application link and fire up a pull trigger that starts the deployment software running. Such a combination can give you the benefits of both methodologies: a Web front end to do any added functions and the standard deployment process so that you don't have to redesign what is all ready out there.

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

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

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

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

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

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

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

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

Prism Deploy

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

Prism Deploy includes utilities for packaging as well as the deployment process. The packaging process was detailed in Chapter 3. With the Prism Deploy Console, you can effectively ship out your packaged applications to your targets. The true power of the product is the integration between the packaging, conflict checking, and deployment features.

Before a client can participate in a channel, the client program needs to be installed. Prism Deploy Console can directly attach to any NT, Win2K, or Windows XP client that you have administrative access to and install the bits. For Win9x clients or machines that are not on the network, the install can be packaged into an executable file that will run on the local machine. That executable can be delivered through methods such as AD group membership, logon scripts, email, Web links, or Sneakernet. Prism's client installs the PrismXL Service, which has the ability to elevate install privileges so that you can automate deployments in locked down environments.

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

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

Just how easy is the process? I was very skeptical to say the least. As a test, I asked an entry-level systems administrator to install the software and send out a simple batch file to my test computer. The batch file did a *net send* to the computer it was on that said "Hi." Nothing fancy or complicated in the package as the test was focused on the deployment software functionality. Less than 2 hours later, with only one request for a reboot on the targeted machine, a pop up message arrived. I was impressed. Using only the manuals and CD-ROM that comes with the application, the junior systems administrator was able to set up the infrastructure on his test system, create a Prism Deploy install file, create a task, add my test computer, remotely install the client on my test system, and distribute the batch file in less than 2 hours. Of course, that overall time would be extended depending on the complexity of the package to be installed and the customization that would be needed. As evidenced by the example of my junior

administrator, the issues of costly integration and consulting that I talked about earlier, will most likely not be a consideration with Prism Deploy.

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

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

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

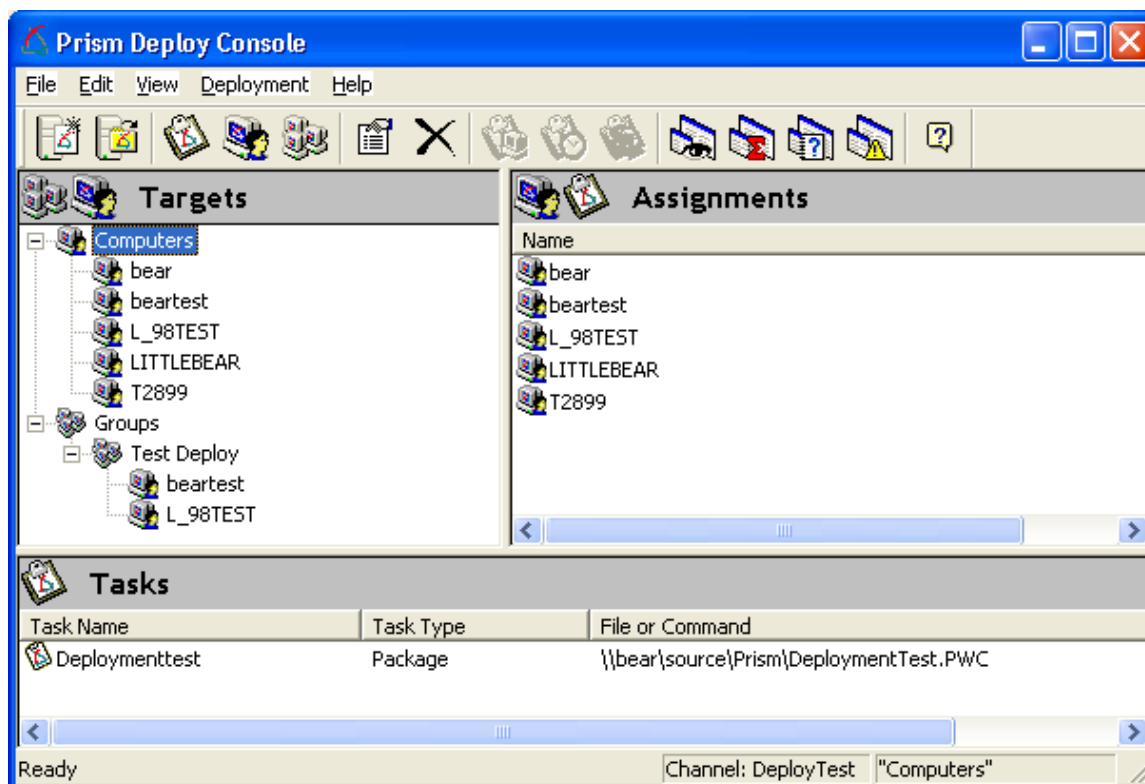


Figure 4.3: A screenshot of the Prism Deploy Console showing populated Targets and Tasks.

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

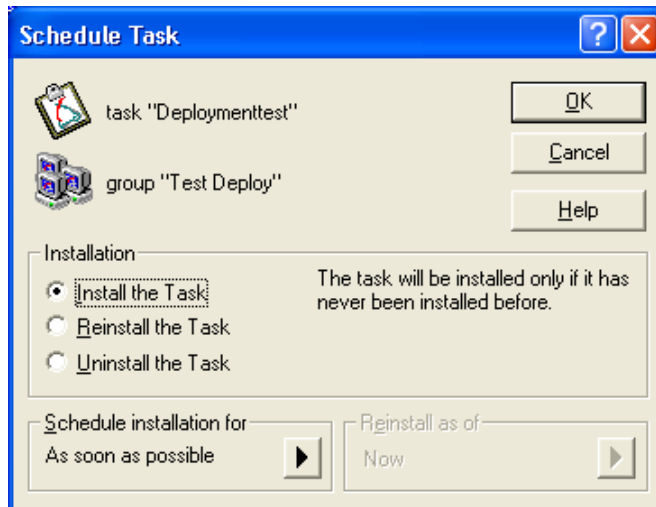


Figure 4.4: Setting the properties for the Task.

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

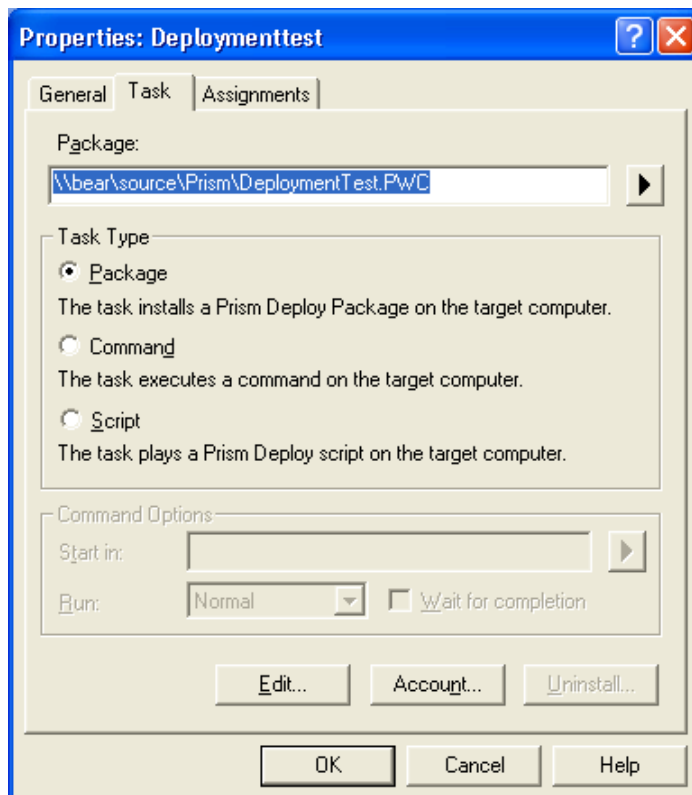


Figure 4.5: The properties of a scheduled Task.

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

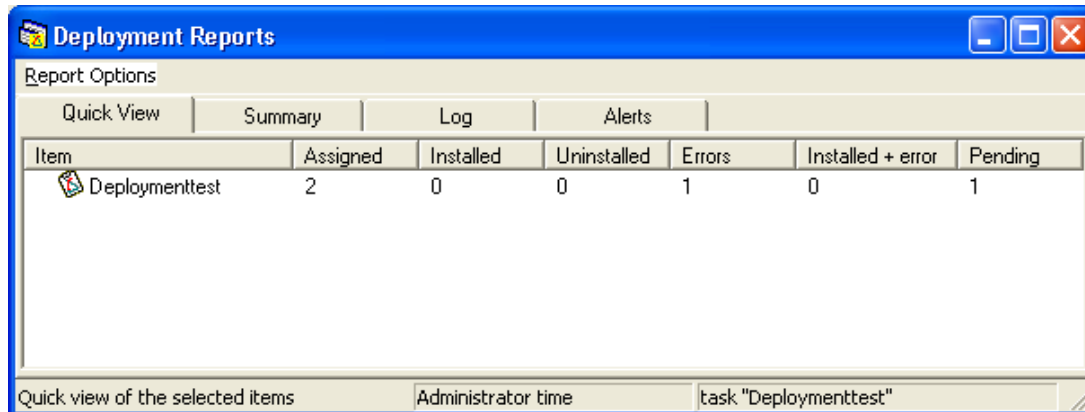


Figure 4.6: The Deployment Reports window.

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

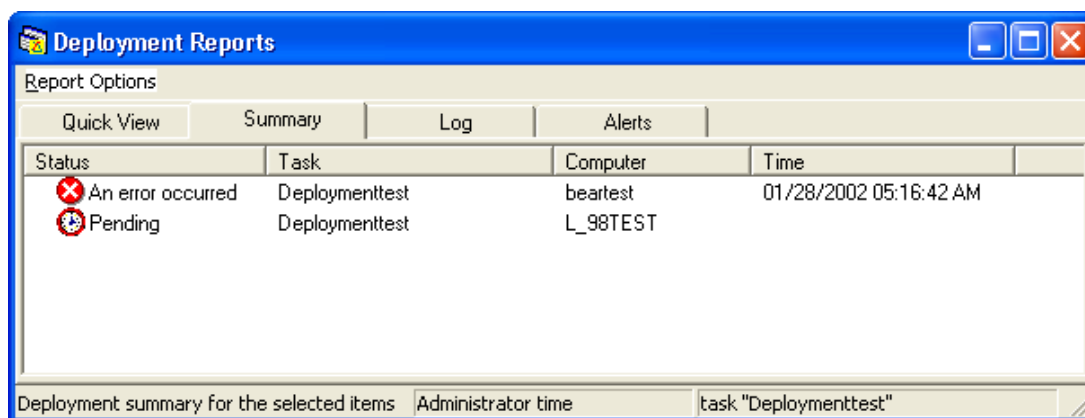


Figure 4.7: The Summary tab displays a little more detail about the deployment status.

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

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

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

Marimba Desktop Management

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

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

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

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

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

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

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




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

Marimba Desktop/Mobile Management uses checksum verification for file transfers. When you package a channel, you always package a full channel as if it had never been sent out. The clients that will only be getting an update to the channel get only the checksum differences not the entire file. So a fresh install might get 10MB and an update might get only 1MB. With the checkpoint restart, if a file download fails at the 8.99MB mark of a 10MB file, when a Tuner reconnects, it



picks up at the 8.99MB point and continues instead of starting all over again. This feature is helpful for users that are only online for a short time each day on a slow link. It might take several days, but they eventually would receive the entire channel. Only after the channel is fully downloaded and the checksum verified is it installed.

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

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

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

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

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

Third-Party Tools Recommendation

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

Microsoft

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



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

SMS

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

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

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

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

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

All the SMS data resides on SQL Server 6.5 Service Pack 4 (SP4—or later), either on the same server or on another server. There are always discussions about which method is better: Move SQL Server to a separate server or keep it resident on the SMS primary site server. Depending on



your environment, budget, and the size of the server (and memory) you could go either way. Real world experience shows that several thousand clients and secondary site servers can be part of an SMS domain in which the primary site server houses a combined SMS and SQL environment.

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

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

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

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

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

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

☞ There are a number of Web sites and news groups that talk about SMS. As a starting point, try <http://www.microsoft.com/smsmgmt/default.asp>
<http://appdeploy.com/tools/sms2>
<http://www.myitforum.com>
 On your newsgroup server, check for microsoft.public.sms.* for several different groups.

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

Table 4.7 compares the pros and cons of SMS.

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

Table 4.7: Pros and cons of SMS.

IntelliMirror

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

- User data management
- Software deployment
- User settings management

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

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

An *assigned* application is available to the user in the Start menu, but is not installed. It is merely an icon for the application. Once the icon is clicked, it activates a background service called Windows Installer that processes the request and performs the MSI installation.

Although the overall process sounds simple, there is some background work that needs to be accomplished to enable the features properly. After the initial design work is done, setting up future deployments through IntelliMirror is easier.


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

Table 4.8 compares the pros and cons of using IntelliMirror for deployments.

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

Table 4.8: Pros and cons of IntelliMirror deployments.

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

Microsoft Recommendation

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

Rollouts

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

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

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

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

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

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

A phased approach requires that you take several steps to roll out the application. How many steps are needed depend on

- The complexity of the package.

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

For most rollouts, using a three-phased approach works effectively. The first phase is sometimes called the *pilot* group because it is small, maybe 5 percent of the total target distribution. The second phase is a little larger, maybe 10 to 25 percent of the total. And the third phase would be the remainder of the rollout targets, assuming you are comfortable and have not found problems.

The number of phases is adjustable. For the majority of your rollouts, using three phases will work. For complex deployments such as rolling out Office XP, increasing the phases to have a smaller number of users in each phase would be best. Adjust the number of phases as you see fit for the situation.

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

Pilot Users

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

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

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

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

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

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

Errors Are Found—Do You Continue to Roll Out?

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

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

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

Ensuring the Deployment Worked

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

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

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

Review the rollout status in your deployment software. Investigate any targets that don't show a good completion status. Remember that depending on the software, you may be seeing the delivery status, not the status of the installation. If the installation failed to update the registry properly, the delivery status might show as a success because the install process completed and returned control back to the deployment software. The user sees the application as not functional, but your delivery mechanism shows it as good. You may find that some machines are constantly coming up with failures. These systems are good candidates for some proactive investigation to determine why they are having problems.

Prism Deploy deployment reports include both deployment status *and* errors such as *incorrect OS*, *user cancelled the operation*, and *error writing to the registry*. These reports indicate that an installation had a problem.

If your deployment software has an inventory option, use it to confirm the deployment. Trigger it to gather data on selected files that are critical to the application. Inventory the primary executable, the existence of a directory, a particular registry entry, or a flag file that you set.

You could also make use of the temp directory on the workstations. Create a C:\TEMP\Logs directory and set the applications to echo their install logs to the location. Have your installation process create a flag file that is only present if the installation went to the end. The flag file can be empty or have a short message or the date. It does not matter what is in the flag file, it needs to exist only so that your inventory modules can note the existence of it. The same function could be done by creating a registry entry.

Once your inventory module has gathered the data you are looking for, run a report to see which machines have the reference points you set. If you know what the targets were, a simple compare should show any that are missing.

Some Tools to Confirm the Deployment Went Out

Aside from using your deployment software and the tools that come with it, you can do some simple scripting with free utilities to determine whether the deployment reached its targets. Here are a few recommendations:

- Nearly any scripting language will provide some means for checking files and getting some data from the registry. Languages such as Perl, C++, and VB6 provide various methods. Choose the language you are most familiar with and use the hooks available.
- The Microsoft resource kits have some nifty utilities such as REGDMP, SCANREG, REGFIND, ROBOCOPY, and WINDIFF. Use these utilities to compare file dates, registry entries, or the existence of files. Check the Help screen that comes with each one to see the format.
- The Windows OSs contain different built-in utilities such as REG, DIR, IF statements (if exist [\\machinename\c\\$\ntdetect.com](#) set test=true), and the like. If you are not familiar with the choices, from a CMD prompt, enter HELP.

The Human Error Factor

Many deployment issues can usually be attributed to some kind of human error, but the problem is reported back as “the deployment software caused my system to blue screen.” Issues with the package creation, damaged workstations, improper user interactions, machines turned off, or

interaction with unknown software on the workstation are very common. Deployment software cannot resolve all those issues, no matter how good the software is. The deployment application depends on the intelligence of the administrator and the packager to resolve issues.

As an example, one of my favorite end-user complaints is the user that claims the deployment software rebooted his or her workstation in the middle of the day. When you research it, you find that a major application was rolled out that required a reboot. For example, a service pack update was sent out on Friday night with the intention that it would install over the weekend when no one was around. You get the complaint the next Thursday and find the user had the reboot on Monday. Checking into the logs you discover that the user had turned the machine off on Friday even after repeated emails telling everyone to leave their computers on. So, yes, the deployment software did install a piece of software that caused the reboot on Monday. Of course the issue of the user turning the machine off on Friday doesn't surface in the complaints. The error is the fault of the deployment software and it starts to get a bad name. The first statements that upper management folks hear are the original complaints.

It is also common for an error, or perceived error, to give the deployment team a bad name causing them to be skeptical of every system problem that follows. Taking care early on and avoiding this kind of reputation is well worth the extra time. True, if the application was packaged to check for delivery times and only run during certain hours, the previous scenario could be avoided. The point is that the problem is not the fault of the delivery mechanism.

Software deployment is not 100 percent about the technical aspects. You need to talk to your managers and make sure that they know what is happening and that the word gets sent up the ladder.



Remember that any deployment software is just that—a way to deploy software. Its primary job is to get the packaged application to a destination and trigger the installation. It is NOT a system that can correct faults or problems with applications, installs, or human errors in deployment targets. Never accept at face value the excuse that problems on a target machine were caused by the deployment software.

Summary

There are as many methods for deploying software as there are target environments. Once you find a solution that fits your budget, client needs, infrastructure, and corporate culture, you should set it as the standard. Any applications coming in should be packaged with the idea that your standard method will be the mechanism for distribution. But stay flexible. Blindly following a process without looking at the desired results can be burdensome. Maybe adding a central Web page for optional installs that tie in to your standard process is the best method for your organization. Keep an open mind and look for the opportunity to improve your methods; it will make your life easier. In Chapter 5, the issue of deploying to the remote user will be discussed.



Copyright Statement

© 2001 Realtimerepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimerepublishers.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 Realtimerepublishers.com, Inc or its web site sponsors. In no event shall Realtimerepublishers.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 Realtimerepublishers.com, please contact us via e-mail at info@realtimerepublishers.com