UNIX Code Migration Guide

UNIX Application Migration Guide

Chapter 13: Creating the Live Environment

Larry Twork, Larry Mead, Bill Howison, JD Hicks, Lew Brodnax, Jim McMicking, Raju Sakthivel, David Holder, Jon Collins, Bill Loeffler
Microsoft Corporation

October 2002

Applies to:
Microsoft® Windows®
UNIX applications

The *patterns & practices* team has decided to archive this content to allow us to streamline our latest content offerings on our main site and keep it focused on the newest, most relevant content. However, we will continue to make this content available because it is still of interest to some of our users.
We offer this content as-is, without warranty that it is still technically accurate as some of the material is undoubtedly outdated. Note that the content may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.

Summary: Once a migrated application has been created and tested, it must be deployed. Chapter 13: Creating the Live Environment looks at the process and technologies that enable an application to be packaged for deployment ranging from the Windows Installer Service to software policies to Systems Management Server. (36 printed pages)

Introduction

This chapter shows you how to create and configure the live environment for your migrated application. There are many aspects to creating a live environment. If you do not already use Microsoft® Windows® operating systems, you will need to roll out new hardware and software to support your migrated application. A number of Microsoft resources can help you through this process, some of which are listed in Chapter 6, UNIX and Windows Interoperability.

The focus of this chapter is on the areas that are directly related to the implementing and managing your migrated application. The four areas that this chapter covers are:

- **Interoperability**

  This topic covers the UNIX and Windows interoperability issues that you need to address specifically for your live environment. Tools and techniques that you can use to address interoperability requirements are covered in Chapter 6, UNIX and Windows Interoperability.

- **Application deployment**

  For many organizations, deploying the new application is a major part of the migration project. This section reviews the tools and techniques that you can use to deploy the migrated application.

- **Network file systems and application servers**

  This section covers the problems in deploying Microsoft Win32® and Windows Interix applications on application servers.

- **Support and maintenance**

  The migrated application requires ongoing support and maintenance. In the UNIX environment are a range of tools that you can use for support and maintenance. This section looks at which of these tools you can continue to use in the Windows environment after migration, and it describes new tools that you can use for your support and maintenance tasks.

You will need to consider and plan for other issues beyond the scope of this guide, such as the following:

- **New hardware and software infrastructure**

  How to migrate an environment to one of the Microsoft Windows operating systems is documented elsewhere. Some references are in Chapter 6, UNIX and Windows Interoperability. For more information, see Migrating to Windows from UNIX and Linux.

- **User and support staff training**

  Staff will need training in the Windows operating systems, the new interfaces and the new tools. Training is available from Microsoft Certified Technical Education Centers (CTECs) in many cities around the world. For more information, see Microsoft Certified Training and Education Centers on the Microsoft Web site.

Operating a Mixed Environment

For many companies, a migrated application operates in a live environment containing a mixture of UNIX and Windows systems. Such companies need some interoperation between the two operating systems. To some extent, interoperation is also required where Interix and Windows systems coexist.

The main areas that you will need to address in mixed environments are:

- Windows to UNIX connectivity
- User authentication and authorization
- Resource and data sharing

The solutions available to you are discussed in Chapter 6, UNIX and Windows Interoperability.

Windows to UNIX Connectivity

In a mixed live environment in which users and support staff access applications running on Windows and UNIX, you will require tools to connect between the operating systems. Usually, you will have the client software on the Windows desktops provide character or graphical (X Windows) access to UNIX applications.

For more information about the options available to you, see Chapter 6, UNIX and Windows Interoperability.

User Authentication and Authorization

In a heterogeneous environment, users and administrators benefit by having a common authentication system and a single username and password pair across all operating systems. The options are covered in Chapter 6, UNIX and Windows Interoperability. The Microsoft Services for UNIX, Network Information System (NIS) Server (called *Server for NIS*) makes it possible to create a single user database. When this feature is coupled with the Microsoft Services for UNIX Password Synchronization tools, you can give users single usernames and passwords across UNIX and Windows systems.

Resource and Data Sharing

Chapter 6 also covered the various techniques and software available that enable you to share file systems and data between Windows and UNIX systems. In the live environment, you want to hide the differences between the two systems from the end users.

You can effectively achieve resource and data sharing by using Samba on UNIX systems and by using Network File System (NFS) on Windows systems. These options are fully covered in Chapter 6.

Deploying the Migrated Application

This explains how to deploy your migrated application into a Windows environment. It describes solutions for deploying Win32 and Interix applications.

The standard method of deploying applications in a Win32 environment is to use the Windows Installer Service. This section looks at the Windows Installer Service and some of the tools that you can use to package your application.

When you deploy applications in the Interix environment, you can use the standard UNIX tools available in Interix for file transfer, remote configuration and scripting. If you had managed the application before migration by using proprietary UNIX software management tools, it is unlikely that these tools will be available in Interix. If you use standard UNIX tools, such as the remote commands **tar**, **cpio** and shell scripts for application management, you will be able to migrate your deployment tools relatively easily.

With both Win32 and Interix applications, your application will reside either on the desktop or on application servers. In the latter case, you may have to rely on networked file-systems for access to your application executables. Because of differences in how UNIX and Windows network file systems operate, you will need to take specific action on some migrated applications. This is covered below.

Tools for Deploying Win32 Applications

After you migrate an application to Win32 from UNIX, you should deploy it by using standard Windows software management tools. This section describes some of the tools available and how you can use them in different deployment scenarios.

Windows installer service

The Windows Installer service uses the Windows Installer package, and this is now the way that application developers must deliver software for the Windows platform. By standardizing the format and a common set of actions, Microsoft leaves the tool vendors free to add value in creating, editing and distributing Windows Installer packages.

The .msi file format contains all of the instructions that a program needs to install itself: the locations of files, movements or deletions of existing files, creation of shortcut icons, a Start menu entry, registry settings, ACL changes, Windows service installation or changes and COM component registration. The program files may be contained in the .msi or in one or more .cab (compressed) files. The .msi uses database tables to describe the features and components of the product, the relations between the two and all of the actions that it requires to install, upgrade or uninstall the application.

The installer will copy the application's files to their correct locations from an embedded file or from the accompanying .cab file or files. Usually, the Windows Installer is invoked when a computer boots up and logs on to the domain (by using the Microsoft Active Directory® directory service computer account) or when a user logs on to the computer. The Group Policy feature of Active Directory is used to attach .msi installation packages to these events, though you can also invoke the Installer locally by using other means such as scripting or the scheduler service.

The installation process leaves a copy of the .msi instructions on the local computer. Each time an application launches, it checks the feature and component listings in the local .msi to see if everything is still intact. Missing or corrupt components can trigger an automatic repair known as *self-healing*. Self-healing can also be triggered during the operation of an application when a component is dynamically loaded. This feature is useful when .msi files are well architected by the software developer, but it is often turned off by administrators when the .msi has been repackaged. This is because many large applications are, by necessity, built as single feature packages and self-healing would force the administrator to reinstall the entire application.

Installation on demand

Install-on-Demand is one of the most useful features of the Windows Installer Service. Install-on-Demand is the feature that prompts for the location of the installation media after selecting an item that you have not used before, when using a product such as Microsoft Office. By using Install-on-Demand, you can leave components that you do not access frequently (grouped into feature sets) uninstalled and on the server until a user invokes them. You can apply this to whole applications in the case of Group Policy software advertisements, where only a desktop shortcut is deployed initially, or you can apply it to application features such as spell checkers, graphic libraries, charting and so on. To use this feature

properly, you must understand how to divide the application into independent chunks (or components), and you must then choose which components you must share across an entire installation and which you can hold back until they are needed. The local .msi database maintains a list of installation points where it can find the necessary files at the time of installation. You can edit this list by using subsequent patch (.msp) files to update network locations, such as Distributed File System (DFS) shares.

Installation rollback

As the installation occurs, the service backs up overwritten files and keeps track of any changes that are made to the system, such as registry entries or ACL changes. If the setup is not run to completion, an automatic rollback restores the original state of the system. You can run Windows Installer with verbose logging options to record these events; however, there are no built-in alert mechanisms in Group Policy. Most large enterprises using Active Directory as their only means of distributing software have developed scripts to query the .msi log files or interrogate the .msi local database by using Windows Management Instrumentation (WMI).

Deployment auditing using the Windows Installer Service database

This Visual Basic Scripting Edition (vbscript) interrogates the local .msi database and lists installed packages. You can use this, in conjunction with remote scripting, as the basis of a simple deployment audit.

```
Dim installer, product
Dim version
Dim productList, productString

productList = ""

Set installer = Wscript.CreateObject("WindowsInstaller.Installer")

For Each product In installer.Products
  version = CLng(installer.productInfo(product, "Version"))
  version = (version\65536\256) & "." & _
(version\65535 Mod 256) & "." & _
(version Mod 65536)

  productString =installer.productInfo(product, "ProductName")_
  & vbCrLf & " ID: " & product _
  & " Version: " & version & vbCrLf

  productList = productList  & productString & vbCrLf
Next

If productList <> "" Then
  productList = "Found " & installer.products.Count & _
  " applications" & vbCrLf & vbCrLf & productList
Else
  productList = "No .msi applications listed."
End If

WScript.Echo productList
```

The following is an example of output from this script:

```
E:\>cscript ListMSIDB.vbs
```

```
Microsoft (R) Windows Script Host Version 5.1 for Windows
Copyright (C) Microsoft Corporation 1996-1999. All rights reserved.

Found 7 applications

Windows 2000 Administration Tools
 ID: {B7298620-EAC6-11D1-8F87-0060082EA63E} Version: 5.0.0
Microsoft Windows Services for UNIX
 ID: {E8A81EF0-40DB-4B5B-ABE8-558D69CE2F09} Version: 7.0.1620
Hummingbird Exceed
 ID: {CFBD3858-2164-42B0-84A2-576C18C85082} Version: 7.1.0
Microsoft Office XP Professional with FrontPage
 ID: {90280409-6000-11D3-8CFE-0050048383C9} Version: 10.0.2627
WebFldrs
 ID: {6F716D8C-398F-11D3-85E1-005004838609} Version: 9.0.3501
Windows 2000 Support Tools
 ID: {242365CD-80F2-11D2-989A-00C04F7978A9} Version: 5.0.2072
Microsoft Windows 2000 Resource Kit
 ID: {4E1F3FCF-B205-427F-B52B-D13BDFB6526C} Version: 5.0.2092
```

Security rights and the Windows Installer Service

The Windows Installer service, when invoked by Active Directory Group Policy, runs a *managed installation*. This process runs under the Local System account, which has administrative rights. This allows applications to be installed on systems that are locked down (that is, systems on which the end users have limited rights and abilities).

Patch files

Another aspect of the Windows Installer service is the .msp or patch file. This is a specially formatted .msi that can identify and update existing installations of itself by unique product and version numbering. Many organizations will create build images by using the .msi format so that future patches and upgrades can be deployed by using Group Policy in Active Directory.

Window Installer Service transforms

If you have worked with Microsoft Office 2000 or Microsoft Office XP and are familiar with the Resource Kit Custom Installation Wizard, you have seen the Windows Installer's transform technology. Transforms allow you to amend the installation instructions in the .msi file on the fly. These are usually authored by using one of the commercial editing tools such as WinInstall or InstallShield.

A limited variety of user interface widgets are available to the Windows Installer to gather user input during the installation as well as the ability to lookup information in outside files.

Create new Windows Installer Service packages

There are many tools available for packaging, distributing, installing and managing applications in the Windows Installer format (that is .msi files). Microsoft Visual Studio .NET can create installation packages in the .msi format. In addition, there are third-party tools available that help you create and manage Windows Installer Packages. These products typically provide you with:

- An Integrated Development Environment (IDE) for developing installation packages
- Installation script editors

- Installation debuggers
- Options for Internet-based installations
- Support for password and digital signature security options on installation packages
- Support for the Windows Installer patch files (.msp files)
- Support for the Windows Installer transforms file (.mst files)
- Source control integration

The following are three third-party products that you can use:

- InstallShield Developer from InstallShield
- Wise for Windows Installer from Wise Solutions
- Veritas WinINSTALL from Veritas

The section "Using a Windows Installer Packaging Tool" illustrates how you would use the Wise for Windows Installer product to create and deploy a Windows Installer package.

Repackage applications

If a software installation process that has already been created does not support the Windows Installer (.msi) standard, you can use a repackaging application. A repackaging application, as a minimum will allow you to:

- Create fully featured Windows Installer setups by capturing installations that are not based on Windows Installer.
- Allow installations to be customized.
- Check and resolve any installation conflicts.

The following are two repackaging applications that you can use:

- InstallShield AdminStudio from InstallShield
- Wise Package Studio from Wise Solutions
- Deploy applications with Group Policy Objects

Microsoft Active Directory supports a technology known as *Group Policy*. You can assign Group Policy Objects (GPOs) to users or computers, and you can associate them with any of the hierarchical containers that make up the directory structure. This means, for instance, that you can apply a policy to all the computers in an engineering department at a particular site or even across the organization while the computers in an accounting department have their own policies. GPOs are filtered by the user groups in Active Directory so that you can keep precise control over users' applications.

GPOs can set and enforce hundreds of settings on desktop computers, including all of the security settings, but the setting applicable here is the software distribution policy setting. You use the software distribution policy to deploy Windows Installer files (.msi files). The Windows Installer service, **msiexec.exe**, can be set by Group Policy to run with elevated (administrator-level) privileges. Thus an installation program that needs access to resources that a typical user would not have access to (for example, directories and registry entries) can still operate without the user having power user or administrator privileges.

Deploy applications with Systems Management Server

Applications can be deployed using Active Directory's Group Policy software distribution feature. However, there are several limitations with using GPOs for software deployment. These are addressed by SMS. Here is a summary of the main reasons for using SMS rather than GPOs for application deployment:

- Active Directory Group Policy requires that applications be in Windows Installer (.msi) format, whereas SMS can deploy any executable package, including setup programs, scripting and batch files. Many large applications include legacy setup architectures that are difficult or impossible to replicate in a repackaged .msi installation.
- SMS has extensive Microsoft SQL™ Server-based reporting capabilities.
- Group Policy requires a user to log on or a computer to be rebooted to initiate software deployment policy.
- SMS includes an extensive hardware and software inventory.
- SMS allows you to query the client computer prior to installation to ensure adequate disk space, memory, operating system version and other software dependencies.
- SMS does not require Active Directory, though SMS can use Active Directory if it is available.
- Software installations can be advertised to users through desktop shortcuts, the SMS client icon and the Control Panel. Software installations can also be pushed to a client without user intervention.
- SMS allows you to define computer groups separately from Active Directory users and groups, based on inventory information.

Deploying Win32 Applications

This section describes different methods of application deployment and how you would go about using them.

Deploying Win32 applications by pushing them to the desktop

Active Directory Group Policy software deployment or other systems that rely on a user logging on or a computer being rebooted might need to have a second method of delivery that can be deployed without user intervention to client desktops.

SMS and other enterprise level systems can achieve this as part of their normal client-server interaction. Other methods of achieving this include remote scripting or maintaining a service (daemon) on each desktop that checks for updates periodically.

Two-phase deployment of Win32 desktop applications

When deploying locally installed applications, you might want to avoid the distribution of large installation images over the network over a short period of time. To do this you can use a two-stage approach, sometimes referred to as a *knife-edge* installation. In this scenario the two stages are as follows:

1. **Deploy the installation image**.

   An .msi or SMS package that is designed to do nothing more than copy a potentially large installation image to each user's local disk, possibly in a partition reserved for this purpose.

2. **Schedule the installation job**.

A second package or job is then scheduled with the actual installation instructions that operate against this local image. This can be an incremental deployment over several days or weeks.

In this way, large numbers of users can simultaneously install a new version of an application without affecting the network and without raising data compatibility issues.

Another benefit of this technique is that multiple versions of an application's installation image could be stored on the local drive for rapid rollback, piloting new versions and so on. Of course, we would need well-tested install and uninstall jobs, packages and processes to maintain this rolling cache of images.

If a deployment system such as SMS is in place with some additional Wake-On-LAN support, you can deliver the image deployment and installation packages outside of normal working hours.

Large package deployments can also utilize compression technologies such as WinZip to deploy the package without dependence on the .msi format.

Side-by-side deployment of Win32 applications

Although the Windows platform has been a successful development platform in part because of its built-in component sharing mechanisms, these same shared components have also caused administrative headaches. Components from Microsoft, provided as part of the base operating systems, option packs, service packs and various add-ons and numerous third-party sources save developers countless hours. However, true backward compatibility means that shared components must function exactly as they did in previous versions while providing new functionality. In the real world, this is difficult to achieve because all configurations in which the component may be used would need to be tested.

The practical functionality of a component is also not easily defined. Applications may become dependent on unintended side effects that are not considered part of the core function of the component. For example, an application may become dependent on an anomaly in the component, which when fixed causes the application to fail. The fact that Dynamic Link Libraries (DLLs) have been upgraded to newer internal versions while keeping the same names has also caused confusion.

This lack of backwards compatibility can result in the inability to deploy a new application without breaking applications already deployed or compromising the functionality of the new application. To provide for successful sharing while enhancing application stability, Microsoft introduced side-by-side sharing starting in Microsoft Windows 98 Second Edition and in Microsoft Windows 2000, creating a way to share components through isolation.

With side-by-side components, multiple versions of the same component can be installed and applications can use the one most suitable version.

Two different processes can load different versions of a Win32 or COM component at the same time and, independently, unload those components as required.

As outlined in the Windows 2000 logo certification guidelines, the best practice is to develop new applications and components with side-by-side use in mind, but there is also a way to selectively isolate the majority of existing components by using a Windows redirection mechanism. While redirection doesn't require changing any code, it does need to be thoroughly tested to ensure that the applications on the system continue to operate normally. Because these components may now be distributed into many application's directories, there is also an increase in the complexity of administering the components.

Creating new components for side-by-side is the best way to guarantee that applications can load them independently. In this case, the component must be developed with careful attention to where global data and state information are stored. Any other factors that could affect having multiple versions of components in memory simultaneously must also be addressed. For instance, instead of storing a particular setting using a registry key such as:

```
HKEY_CURRENT_USER\Software\Vendor\ComponentName\RegKeyName =
SomeValue
```

The component would be better isolated using a version specific key such as:

```
HKEY_CURRENT_USER\Software\Vendor\ComponentName\VersionNumber\RegKeyN
ame = SomeValue
```

Or even version and application specific:

```
HKEY_CURRENT_USER\Software\Vendor\ComponentName\VersionNumber\
ApplicationSpecificName\RegKeyName = SomeValue
```

Shared memory structures such as memory mapped files and named pipes also need to be taken into account and renamed or relocated on a per version basis. Better still, a component should be designed to be as stateless as possible and to let the client application handle state and user-specific data as much as possible. Where the component really does need to store its own state information; it should use a method or property of the client software instead of modifying memory structures or registry settings directly.

Windows 2000 allows administrators to take advantage of side-by-side loading with existing components as well using a feature called *DLL redirection*. The operating system changes its default method of locating components if it finds a special file in the directory with the application that is loading the component. The file itself is empty but is specifically named to match the application executable name and has a .local suffix. For instance, **myapp.exe** would have an empty file next to it called **myapp.exe.local**. When Windows 2000 encounters this file, it looks for a requested component in the directory where the calling application is located or in the subdirectories below it. It will use the version of the component it finds there no matter what path the system has registered for that component. If it can't find a version in the application's directory structure, the system will revert to using the path that's registered. Applications without a .local file continue to use the system registered path.

This method works for most components, but it needs to be tested to ensure that applications that use different versions actually can coexist. Components that store global state in registry keys that are not tied to a particular application or application version or shared memory structures may not operate correctly side-by-side. Sometimes this can be overcome simply by not running two applications simultaneously that load different versions, but each administrator must decide if that is acceptable to their user base. Other components may use relative paths to access system resources or other components—assuming they are located in a particular directory. Some things can be safely moved or copied to satisfy this, but Windows system components, especially those protected by Windows file protection, should never be moved.

Tools for Deploying Interix Applications

When migrating from UNIX to Interix, one of the main benefits is the similarity of the two platforms.

This makes migrating an application to Interix relatively easy. The similarity between the two platforms also extends to the tools available. If you previously created your own deployment scripts using these tools, it should be straightforward to migrate your deployment tools to Interix.

If, however, you have used deployment tools specific to your UNIX distribution, you have to either port these or write scripts using the tools available.

Berkeley remote shell commands (r commands)

Distributing your application into the Interix environment is likely to require the transfer of files over the network. Typically, you would use the **rsh**, **rcp** or **ftp** command to achieve this. These are available as standard in Interix. More details can be found in Chapter 6, UNIX and Windows Interoperability.

Scripts

You have a wide range of scripting languages and tools available to you in Interix for the creation of deployment tools. These have been described under the "Shells and Scripting" section in Chapter 2, UNIX and Windows Compared. Interix includes C and Korn Shell Scripts and UNIX Practical Extraction and Report Language (Perl) version 5.6.

Deploying Interix Applications

To deploy your applications to Interix, you will need to develop your own deployment scripts. Typically, you would use the approach described in the next section.

Deploy Interix applications by pushing them to the desktop

The push application delivery mechanism in UNIX environments is typically implemented by using scripted **ftp** or **rcp** command that copy the application binaries to a computer running UNIX and then activate the application by pointing a symbolic link at the new application binary.

From a management standpoint, remote managed computers contain a selection of management scripts (Perl, C or Korn Shell) that can be invoked remotely to initiate an application image deployment, enumerate some performance metrics or audit for security or asset management.

The section, "Deploying Interix Applications Using the Berkeley r Commands," illustrates remote deployment using the **rcp** and **rsh** commands.

Use the r commands for remote management in Interix

You can remotely manage Interix systems by using the Berkeley r commands. These commands, such as **rsh** and **rcp**, have been described in Chapter 6, UNIX and Windows Interoperability. Before you can use these commands for remote administration, you will need to configure the Interix daemon **rshd** to permit remote access between computers. This is because **rcp** uses a remote shell at the remote machine (source or destination of the copy) to start a shell to launch the **rcp** process with the appropriate arguments.

For example, when Interix is the source of the directory copy:

```
rcp -r <Interix hostname>:ProgDir <destination hostname>:ProgDir
```

Interix has two processes associated with the **rcp**, as follows:

```
user1   9281   1025 20:41:48 -    0:00.03 sh -c rcp -r -f ProgDir
user1   9345   9281 20:41:48 -    0:00.26 rcp -r -f ProgDir
```

When Interix is the destination (sink) of the directory copy:

```
rcp -r <source hostname>:ProgDir <Interix hostname>:ProgDir
```

Interix has two processes associated with the **rcp**, as follows:

```
user1   9729   1025 20:41:48 -    0:00.03 sh -c rcp -r -t ProgDir
user1   9793   9729 20:41:48 -    0:00.26 rcp -r -t ProgDir
```

Notice that the remote **rcp** is passed a different argument that depends on whether it is the source (**-f**) or the destination (**-t**) of the copy.

See the main page for more information on **rcp** command-line options.

To automate configuring the host equivalent files

1. Create necessary configuration files and set appropriate permissions:

```
/etc/hosts.equiv.
$HOME/.password
$HOME/.rhosts
```

1. Set appropriate permissions:

```
#!/bin/csh
#
# Does the hosts.equiv file exist?
# If not, create an empty one.
#
if ( ! -f "/etc/hosts.equiv" ) then
    #
    # create /etc/hosts.equiv
    #
    touch /etc/hosts.equiv

    #
    # Set the permissions on /etc/hosts.equiv
    #
    chmod 755 /etc/hosts.equiv

    #
    # Set the owner of /etc/hosts.equiv
    #
    chown +Administrators /etc/hosts.equiv
endif

#
# If environment variable HOME is defined then see if the users
  password file exists.
# If not, create it.
#
```

```
    if ( $?HOME) then
        if ( ! -f "$HOME/.password" ) then
            touch $HOME/.password
            chmod 600 $HOME/.password
        endif

        #
        # See whether the users .rhosts file exists.
        # If not, then create it.
        #
        if ( ! -f "$HOME/.rhosts" ) then
            touch $HOME/.rhosts
            chmod 600 $HOME/.rhosts
        endif

    endif
```

> **Note** You should be aware that the use of the r commands may be restricted for security reasons. If this is the case you will need to modify the above script to reflect your security policies.

Install MSI packages remotely by using Interix rsh

To launch Win32 application's from an Interix shell remotely, it is important to keep in mind that there are no stdin/stdout/stderr file handles on which to read and write. This is because the remote process is connected to pseudo terminals that do not have any corresponding Win32 object. So Win32 applications must be wrapped so that they are provided with stdin/stdout/stderr file handles. The following code shows the remote execution of the **ipconfig.exe** command:]

```
$ rsh remotesystem "/dev/fs/C/WINNT/system32/ipconfig.exe < /dev/null
  2>&1 | cat"
```

You can also execute Win32 command shell (**cmd.exe**) commands. Execution of a remote "**dir**" command is:

```
$ rsh remotesystem "/dev/fs/C/WINNT/system32/CMD.EXE /c dir <
  /dev/null 2>&1 | cat "
```

When passing Win32 paths by using Interix shells, the *\'s* must be escaped with a backslash mark (\). For example, to execute the MSI package installer from an Interix Korn shell, double backslash marks (\\) must be passed in the paths to **msiexec.exe**. The following example shows this for the installation of the Windows 2000 support tools (for example, *<drive>:\SUPPORT\TOOLS\2000 RKST.MSI*):

```
$ /dev/fs/C/WINNT/system32/msiexec.exe /I
  <drive>:\\SUPPORT\\TOOLS\\2000RKST.MSI /Lv C:\\Temp\\msi.log /qn
```

Installing this same MSI package remotely is interesting because **rsh** also requires that Win32 path backslash marks be escaped, and the command path being passed over the remote shell must also have its \'s escaped. So the following **rsh** command will install the Windows 2000 Support Tools remotely by using an **rsh** command:

```
$ rsh remotesystem "/dev/fs/C/WINNT/system32/msiexec.exe /I
  <drive>:\\\\SUPPORT\\\\TOOLS\\\\2000RKST.MSI /Lv
  C:\\\\Temp\\\\msi.log /qn < /dev/null 2>&1 | cat"
```

where <*drive*> is a drive letter available on the remote system.

Networked File Systems and Application Servers

UNIX applications are often stored on an application server and its executable files are accessed from the client using NFS mounts. This greatly simplifies the deployment of applications, as they need to be deployed to only one server (the application server).

If your migrated application continues to use application servers, there are issues that affect both Win32 and Interix implementations. The main issues arise due to differences in the ways the UNIX and Windows networked file systems operate. The next two sections describe how you should address these issues for Win32 and Interix applications.

Migration to the Windows platform gives you an opportunity to reevaluate whether to continue using application servers. Your evaluation should take into account network bandwidth requirements and the proximity of network shares.

You should deploy any Win32 application by using the Windows Installer service, whether or not it is to be located on an application server. This may entail as little as placing a shortcut on the user's desktop, although most applications require at least some skeletal installation to accommodate COM or Distributed Component Object Model (DCOM) component registration, user preference settings and building local configuration files. Creating .msi installation packages enables an enterprise to maintain a consistent technology, use Group Policy-based deployment and enjoy wide support by other tool vendors. Also, consider that most applications will enjoy increased performance and reliability when executed from local drives.

Server-Based Win32 Applications

For applications that were executed from a UNIX file server by using a symbolic link on the client, there are a couple of difficulties resulting from the migration to Win32:

- **Local desktop references in Win32 applications**

  Windows applications often use local references to DLLs, configuration files, the Windows registry and COM components. This is solved by locally installing the required DLLs, configuration files, registry changes and COM components on each desktop.

- **Differences between NFS and CIFS**

  The Common Internet File System (CIFS) file access protocol behaves differently in network or server failure scenarios than does the NFS protocol that is commonly used in UNIX systems. The solution to this problem is described in the rest of this section.

This section will discuss the use of processes and technologies, including Windows Client Side Caching, Windows Installer service and others to mitigate the issues in coordinating the deployment of very large applications across the enterprise. This is not meant to be an exhaustive discussion of these technologies but rather to concentrate on the case where the application is very large (both in size on disk as well as the number of files), and a large user base must be able to use a newly deployed or upgraded version quickly.

One of the challenges faced in migrating from UNIX to Win32 is that CIFS is a stateful protocol maintaining a session between the client computer and the file server. NFS, on the other hand, is stateless. This means that NFS is unaffected by a NFS server becoming unavailable on the network. In this situation, an NFS client will wait and retry the service periodically until the NFS server reappears.

The operating system and many UNIX applications are unaware of the loss of an NFS server and will continue to wait (even in the middle of a function call to a distant library file) until the connection returns or the user terminates the client process. CIFS, on the other hand, will raise an error if a server or a network anomaly makes a file share unavailable. Many applications that have been ported to the Windows platform have not allowed for this fact and do not perform adequate error trapping. You can use Microsoft's Client-Side-Caching (CSC) in Automatic Caching for Programs mode to overcome this issue.

Client side caching and server-based Win32 applications

Using caching to overcome network and server failure requires the use of Windows 2000 Server or Advanced Server network shares that support the Automatic Caching for Programs mode. This capability is a property of the Windows Server-side share. In some situations, the applications will reside on Windows 2000 Servers (even domain controllers), while shared user data may still be stored on UNIX file servers. This is fine because the advantages of caching that assists in serving application binaries may actually be undesirable for shared data files due to concurrency issues.

When a Windows 2000 server share has its **Caching Settings** property set to **Automatic Caching for Programs**, the Windows 2000 client will be directed to cache every file designated for caching that is opened for execution. After a file has been cached, the client will check on subsequent file open requests to see if either the local or the server version of the file has changed. If neither has changed, the client will open the local copy without further contact with the server. This methodology hides CIFS errors or warnings from the application. There is an assumption that the application performs standard error checking routines around any call to load a file (such as a .dll file) prior to invoking any of its members. If that file has not been called upon previously in the current session or has not been precached, the chance of an unchecked error exists.

Client side caching and administrative pinning

An adjunct technology to Automatic Caching is a client-side policy setting called *Administrative Pinning.* This policy, implemented through Active Directory Group Policy, ensures that files that must be available to the client machine are *pinned* in the client's file cache.

The term *pinned* refers to a capability known as "Automatic Caching for Programs." This provides offline access to shared folders containing files that are not to be changed. This caching option is used for files that are run, accessed for read-only. This greatly reduces network traffic and network time-outs because these *pinned* files are opened directly, without accessing their network version.

In this way, the executables and DLLs that are critical to the application can be prepopulated and are never removed from the cache to make room in the allotted cache space for more recently accessed files.

Once a file is pinned, no recaching is necessary for files that do not change (for example, binary executables). If a file changes on the server, CSC will automatically get the new copy and keep it pinned. Thus pinning is a one-time action.

To implement Administrative Pinning

1. Start the Group Policy Editor. On the Start menu click **Run**, then in the **Open** dialog box, type ***gpedit.msc***, and then click **OK**.

   You will see the Group Policy Editor window as shown in Figure 1.

2. In the Explorer window, click Computer Configuration, click Administrative Templates, click Network, and then click Offline Files.

   This is illustrated in Figure 1.

3. In the Offline Files dialog box, double-click Administratively assigned offline files.
4. In the Administratively assigned offline files dialog box, on the Setting tab, select Enabled. Then click the Show button under Files and Folders.
5. Then click **Add** in the **Show Contents** dialog box.
6. Under **Enter the name of the item to be added** in the **Add Item** dialog box, enter a name for this file or folder. Under **Enter the value of the item to be added**, enter the fully qualified Universal Naming Convention (UNC) of your file or folder, and then click **OK**.

   Embedded environment strings are allowed as part of the UNC file path.

7. In the explorer window, click **User Configuration**, click **Administrative Templates**, click **Network**, and then click **Offline Files**.
8. In the Offline Files dialog box, double-click Administratively assigned offline files.
9. In the Administratively assigned offline files dialog box, on the Setting tab, select Enabled. Then click the Show button under Files and Folders.
10. Then click **Add** in the **Show Contents** dialog box
11. Under **Enter the name of the item to be added** in the **Add Item** dialog box, enter a name for this file or folder. Under **Enter the value of the item to be added**, enter the fully qualified Universal Naming Convention (UNC) of your file or folder, and then click **OK**.

    Embedded environment strings are allowed as part of the UNC file path. See Figure 1.

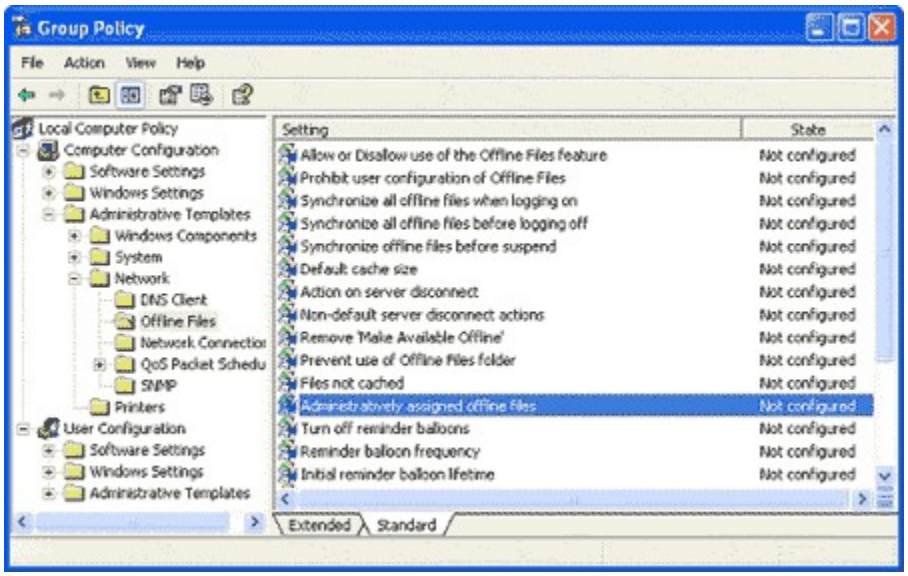

Figure 1. The Group Policy Editor and administratively assigned offline files

The Computer and User policies are merged at runtime to pin the union of the files specified in both settings. Whenever this policy is applied, the client will pin every file listed. If an entry is a folder, the folder's contents will be pinned recursively. This processing is performed in the explorer.exe process on a background thread at IDLE priority level. It's best that a subsequent synchronization occurs to ensure the pinned files are complete in the local cache. The typical synchronization at logoff is sufficient for this. In Microsoft Windows XP, this has changed so that the files are synchronized to the local cache at the time they are pinned. Some thought and testing needs to be put into which and how many of the application's files should be administratively pinned.

The system inspects several parameters when determining whether a file needs to be synchronized. *Modified date* is the most obvious parameter used. In addition, if a file has been modified on the client while offline, the CSC database sets a bit on that file indicating that it has been changed, and this bit is inspected as well during synchronization. There is no byte-by-byte comparison involved.

Performance considerations with client side caching

To synchronize files, the files must be placed in the local cache. As the number of files in the cache increases, synchronization slows. Large numbers of cached files will also affect the lookup speed for any given file.

If the user has enabled the **Synchronize all offline files before logging off** option, any directory being synchronized is scanned on the network so that any new files that have been created on the server since the last synchronization are pinned. If the user has many directories pinned, and those directories contain many files, this can take some time.

CSC attempts to dynamically reconnect after a network or server failure if no cached files have been modified while offline and the client has no files open from that share.

Server-based Interix Applications

For applications that were executed from a UNIX NFS file server, often using automount and NIS maps on the client, the transition to the Windows platform and Interix result in a couple of difficulties:

- Interix does not support dynamic mounts or automount NIS maps.
- As with Win32 applications discussed above, the CIFS file access protocol behaves differently in network or server failure scenarios than does the NFS protocol that is common in UNIX systems.

Generally it is not recommended to serve Interix application binaries from a file server, but when this is required, the use of static (nonautomounted) NFS file shares is the recommended approach. If static file shares are unacceptable, then use of the Microsoft Distributed File System (DFS) should be considered. This is discussed in the next section.

Replace automount with Microsoft Distributed File System

If Interix ported applications are to be served from the same set of NFS file servers as the original UNIX application, you can meet the need for automounting in Interix by using the Microsoft Distributed File System (DFS).

> **Note** Microsoft DFS is not the same as the Open Systems Foundation's Distributed File System (DFS).

DFS allows system administrators to make it easier for users and applications to access and manage files that are physically distributed across a network. With DFS, directories and files distributed across multiple servers appear as if they reside in one place on the network. It acts as a network-based file system with a single root directory. End users have no need to know or to specify the actual physical location of files to access them.

DFS includes both server-based and client-based DFS components. For more information about DFS components, see "Understanding the Distributed File System (DFS") and "Using the Distributed File System (DFS)" in Windows 2000 Server online help.

To set up the Interix user environment to look like the UNIX automounted environment

1. Create DFS links to NFS shares as defined in UNIX NIS domain automount maps. (See procedure below.)
2. Use Interix to create symbolic links to the top level DFS links (for example, apps, data, home, scratch and so on).
3. Change the local shell startup scripts (C and Korn Shell) to use the links.

This solution has the advantage of being a single point of administration for both the NIS automount maps and the DFS import script. The only portion of this procedure that needs to be performed on a Windows 2000 DFS Server is the use of DFSUTIL command line application to import the DFS configuration information to generate DFS links.

To create the DFS links from NIS automount maps

1. First, run a **ypcat** on each NIS automount map to generate text output of the NIS map.
2. UNIX text manipulation utilities are run to create a text file in the format needed for the import function of the DFSUTIL utility.
3. The import text file is copied to the Windows 2000 Server.
4. DFSUTIL is run to create the DFS links.

This process is shown in Figure 2.

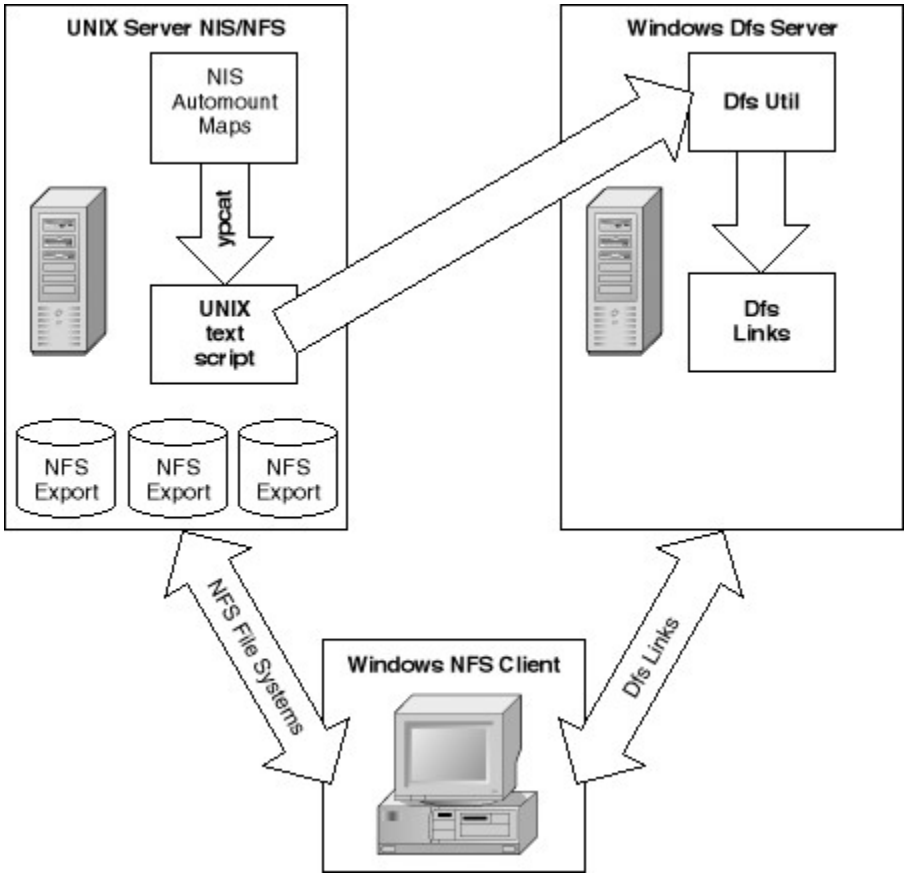

Figure 2. How to convert automounts to Windows DFS

The Windows workstation uses the information in the DFS links to mount the appropriate application image environment for each user with NFS shares on UNIX. This process can take place using a script run when each user logs on. You will find a script that will simulate automounting in the section "Using DFS to Automount in Interix."

Support and Maintenance Systems

Your system and application administrators will usually be looking to use tools and scripts to manage your migrated application. With migrations to Interix and Win32, you have the option to migrate any management tools that were used previously in your UNIX environment. In a Win32 migration, you can also use some of the Windows-based management tools, such as Microsoft Systems Management Server (SMS).

This section discusses the options available for managing and supporting your migrated application. Take an inventory of the UNIX based management tools that you have, and for each, decide the most appropriate solution on the Windows platform.

It is likely that you will look to porting your UNIX management tools to Interix. Consider porting the tools if your application has been migrated to Win32 and continues to use the same configuration files and scripting tools as it did on UNIX. In this case, Microsoft's Services for UNIX (SFU) 3.0 provides the tools necessary to port Perl and UNIX shell scripts to Win32, including those employing standard UNIX utilities such as **awk**, **sed** and **grep**.

Remote Management and Control

UNIX, Interix and Windows include basic remote management and control tools. These include **telnet**, **ftp** and the Berkeley r commands that most UNIX administrators will be familiar with. These tools are ideal for mixed Windows and UNIX environments and also for UNIX support staff who are experienced in using these tools.

In mixed UNIX and Windows environments and where Interix is deployed, X Windows–based management may well be employed. There is a range of X Windows Servers available for Windows, which are discussed in Chapter 6, UNIX and Windows Interoperability. This means that for UNIX support staff, it is possible to continue to use familiar tools in the Win32 and Interix environments. The Win32 Windows environment has additional remote tools which significantly enhance remote management and control, however.

Microsoft SMS is one such tool. It includes as standard a graphical remote control tool. In addition to SMS, many third-party tools provide remote control, including:

- Norton PC Anywhere
- Compaq's Carbon Copy

In Windows XP Professional, the remote assistance agent was introduced as part of the base operating system. This allows administrators to take control of remote desktops and solve or help solve problems.

Using scripts for remote management in Win32

Windows Scripting Host is the resident runtime environment for ActiveX scripting engines and their associated programming languages. Microsoft provides Visual Basic Script and Jscript (an implementation of ECMAScript Edition 3) with Windows 2000. There are third-party implementations of Perl, REXX, Python and other language engines. ActiveX scripting code can also be hosted by Active Server Pages, Internet Explorer and Active Directory network logon scripts. Together with Windows Management Instrumentation (WMI) and Active Directory Scripting Interface (ADSI) components, virtually anything in the Windows environment can be scripted.

Scripts run by the Windows Scripting Host service can access the local file system and mapped network drives; desktop shortcuts; the Windows registry; special folders, such as My Documents; Active Directory objects; User Profiles; Environment variables; the event log; performance counters; and the Windows Installer service. Scripts can leverage installed applications that provide ActiveX interfaces such as Microsoft Office, and they can be invoked remotely.

Other tools available to the Systems Administrator for automating jobs are the network logon script and the Task Scheduler and **AT** command line interfaces to the Windows Scheduler service. Many third-party tools are also available with additional or different ways of accomplishing automated management tasks. The Windows administrator should also be familiar with the tools provided on the Windows 2000 Professional and Server CDs and the Windows 2000 Resource Kits, which contain dozens of utilities, including many that are designed for remote automation.

Enabling Windows Script Host in Win32

You can enable remote scripting in Win32-based applications as described in the following procedure.

To enable remote scripting

1. Install Windows Script Host 5.6 on both systems.
2. Change the following registry key on the remote server, where the script will be executed:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Script Host\Settings
remote = 1
```

3. Ensure that you have administrative rights on the target computer.

Using Windows Script Host in Win32

After you have enabled remote scripting, pass a script to the WshController object's **CreateScript** method, provided it doesn't use any user interface elements such as **WScript**.**Echo**, **STDIO**, **WshShell::Popup**, **MsgBox**, **InputBox** and other user interface generating methods and objects.

Following are script templates that you can use to create remote administration scripts:

- Admin.vbs is a template for a standard administrator's tool for copying and remotely executing the client script.
- Client.vbs carries out the work and reports back events that are raised on the remote computer. In this case, the control script uses ADSI to retrieve the computer names from Active Directory.

```
'***********
'Admin.vbs
'***********
domainName = "YOUR_DOMAIN"
scriptName = "c:\client.vbs"

Dim ADContainer, ADComputer
Dim WSHCtrl

Set ADContainer = GetObject("WinNT://" & domainName)
ADContainer.Filter = Array("Computer")

For Each ADComputer In ADContainer
  Set wshCtrl = CreateObject("WshController")
Set wshRemote = _
  wshCtrl.CreateScript(scriptName, ADComputer.name)

WScript.ConnectObject wshRemote, "WSH_"
wshRemote.Execute

Do While wshRemote.Status <> 2
    WScript.Sleep 1000
Loop
Next

'Event handling functions…
Function WSH_End()
  Wscript.Echo ADComputer.name & " End Event fired"
End Function

Function WSH_Error()
  WScript.Echo "Error Number: 0x" _
    & CStr(Hex(WSHClient.Error.Number))
```

```
  WScript.Echo "Description:  " & WSHClient.Error.Description
  WScript.Echo "Line Number:  " & WSHClient.Error.Line
  WSH.Client.Terminate()
End Function

Function WSH_Start()
  MsgBox ADComputer.name & " Start Event fired"
End Function

Set ADComputer = Nothing
Set ADContainer = Nothing
Set WSHCtrl = Nothing
```

Next follows the code for the Client.vbs script:

```
'*************
'Client.vbs
'*************
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.CreateTextFile("C:\scripttest.txt", True)
WScript.Sleep 5000
f.WriteLine "Script executed at " & Now
WScript.Sleep 5000
f.Close
```

Windows Script Host (WSH) includes three methods to manipulate the Windows registry. Although many registry settings are set by using Group Policy, often many small changes need attention. Embedding these statements in WSH compliant .vbs or .js files, which are then called from Group Policy, reduces the use of the command-line equivalents in logon scripts.

```
'*************
'DeleteAuto.vbs
'*************

Dim WSHShell, regAuto
Set WSHShell = WScript.CreateObject("WScript.Shell")

regAuto = WSHShell.RegRead("HKLM\Software\Microsoft\Windows
  NT\CurrentVersion\Winlogon\AutoAdminLogon")

if regAuto="1" then
WSHShell.Popup "Turning off auto logon…"

WSHShell.RegWrite "HKLM\Software\Microsoft\Windows
  NT\CurrentVersion\Winlogon\AutoAdminLogon", 0
end if
```

Remote management and Windows Management Interface

Windows Management Instrumentation is Microsoft's implementation of Web-based Enterprise Management (WBEM), a set of industry standards for accessing, collecting and manipulating management information over an enterprise network.

WMI includes a Common Interface Model (CIM)–compliant object repository. The CIM Object Manager (CIMOM) handles manipulation of objects in the repository and gathers both dynamic information as well as semistatic data points from the WMI providers. WMI providers are intermediate

agents between components of the operating system, applications and drivers. The registry provider draws information from the registry; the Simple Network Management Protocol (SNMP) provider provides data and events from SNMP devices and from other sources.

WMI has its own remoting capabilities separate from Windows Script Host. Following is a VBScript example of attaching to the WMI service on a remote computer and instantiating a new process. You can extract the return value from the remote process.

In the following code, *remote_desktop* should be replaced with the actual name of the server on which the application should be run.

```
server = "remote_desktop"
Set myProc = GetObject("winmgmts:\\" & server &
  "\root\cimv2:win32_process")
Set InParams =
  myProc.Methods_("create").InParameters.SpawnInstance_()
InParams.CommandLine = "Calc.exe"
Set OutParams = myProc.ExecMethod_("Create",InParams)
wscript.echo OutParams.ReturnValue, OutParams.ProcessID
```

WMI validates each user before the user is allowed to connect to WMI on either the local computer or a remote computer. This security is another layer on top of the security that already exists for the operating system. WMI does not override or circumvent any existing security provided by the operating system.

By default, all members of the Administrators group have full control of the WMI services on the computer that is being managed. All others have read, write and execute permissions on their local computers only.

Using perl/tcl scripts with WMI

On Windows, ActiveState provides ActiveX scripting compatible versions of Perl and Python, which allows the language to be used from within Windows Scripting Host, Internet Explorer and Active Server Pages.

An example of a Win32 Perl script that will retrieve all the running processes on the local computer by using WMI follows:

```
#!perl -w
# Copyright (c) 1997-2002 Microsoft Corporation
# WMI Sample Script - List services (Perl Script)
# How to retrieve a list of running services from instances of
  Win32_Service.

use strict;
use Win32::OLE;

my ( $ServiceSet, $Service );

eval { $ServiceSet = Win32::OLE-
>GetObject("winmgmts:{impersonationLevel=impersonate}!\\\\.\\root\\ci
mv2")->
  ExecQuery("SELECT * FROM Win32_Service WHERE State=\"Running\"");
};
```

```
unless ($@)
{
  foreach $Service (in $ServiceSet)
  {
    print $Service->{Description}, "\n";
    print " Process ID: ", $Service->{ProcessId}, "\n";
    print " Start Mode: ", $Service->{StartMode}, "\n";
    print "\n";
  }
}
```

Using a Windows Installer Packaging Tool

This section shows you how to create a Windows Installer package using one of the third-party tools described in this chapter. Before using the tool, you should carry out some preparatory steps:

To create a Windows Installer package with a third-party tool

1. Set up a computer that is configured in the same way as the target machines in your live environment.
2. Turn off any virus detection and intrusion detection before you run the packaging applications snapshot function.
3. Restore the model machine to this state each time you repackage an application.
4. If you will be using Active Directory Group Policy to deploy the Windows Installer package file (.msi) , you should log on to the computer with an administrative account because managed applications (those assigned, not advertised, by GPO) install using elevated privileges.
5. If the .msi is distributed by some other method that may not operate under these privileges, you must take this into account as you create the .msi.

This example uses the evaluation edition of Wise for Windows Installer version 4.20 from Wise Solutions, which you should now install and use.

To create a Windows Installer package with Wise for Windows

1. Install Wise for Windows Installer to the default directory, C:\Program Files\Wise for Windows Installer Evaluation. It is an .exe file and will launch Windows Installer service to install itself.
2. On the Start menu, click All Programs, click Wise Solutions Evaluations, click Wise for Windows Installer—Professional Edition.
3. In the **New Installation File** dialog box, select **SetupCapture** as the project type, select the **Create standard .MSI or .MSM file** button, and then click **OK**.

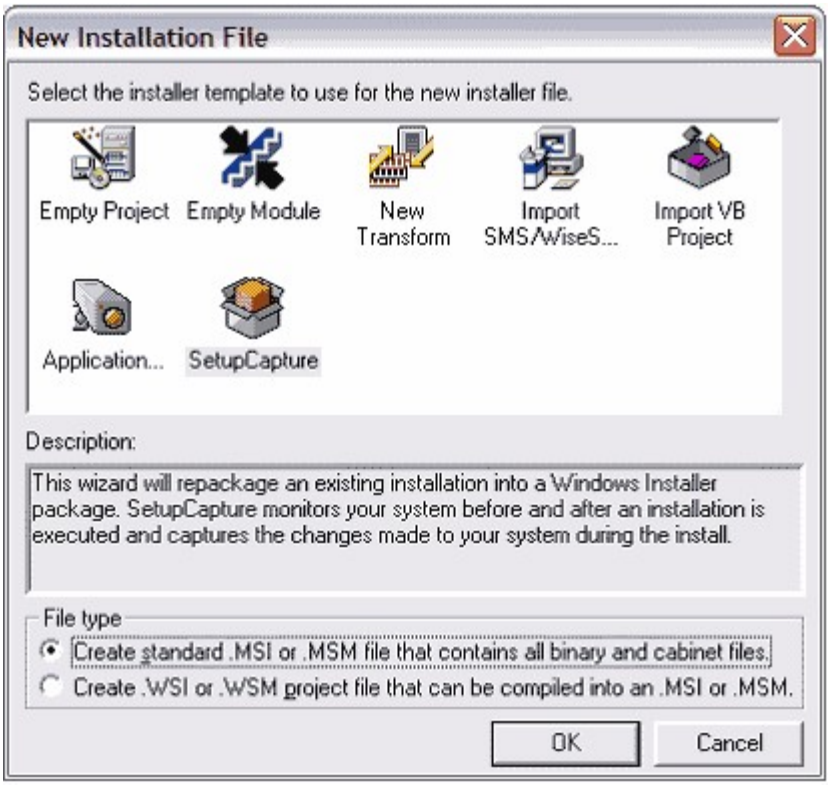

Figure 3. New Installation dialog box

4. Click the **Settings** button in the **SetupCapture**—**Welcome** dialog box.
5. In the **SetupCapture Configuration** dialog box, do the following:
   - On the **General Settings** tab, accept the defaults.
   - On the **Directories to Watch** tab, double-click the C:\Entry and clear the check box at the bottom of the dialog box to **Include Subdirectories**. Click the **Add** button to include C:\Windows (or your current Windows system directory) and :\Program Files and include the subdirectories, and then click **OK**.
   - On the **File and Folder Exclusions** tab, the current user's temp directory is listed. You can add other directories that might be modified during the course of the application's installation but do not affect the actual install process.
   - On the **Registry Exclusions** tab, list the registry keys that might change during the install but which you do not want captured as part of the process. (This tab is not normally used.)

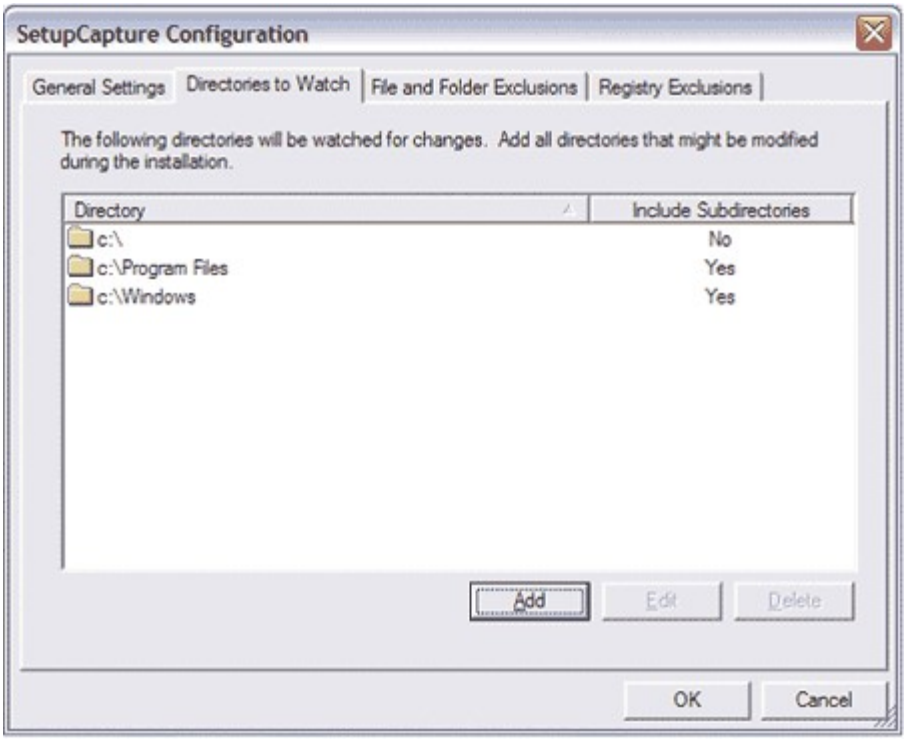

Figure 4. SetupCapture dialog box

6. Click **OK** to return to the **Welcome** dialog box, and then click **Next**.
7. Heed the warning about exiting all programs except Wise for Windows Installer, then click **Next**.
8. In the **Execute Installation** dialog box, click **Browse**. After you navigate to your **setup.exe** (or other legacy installation file) and return to the **Execute Installation** dialog box, click **Next**.
9. When the dialog box tells you to, reboot the machine even if the legacy installation does not require you to reboot.

   During this phase, do not close the Wise for Windows Installer dialog box. It will reappear after you have rebooted the machine. When it does, click **Next**. For now, accept the default feature name of **complete** and click **Finish**.

10. Fill in information in the **Product Details** dialog box, such as a name for the package or the Manufacturer's name, and accept the default or generate a new Product Code (GUID).

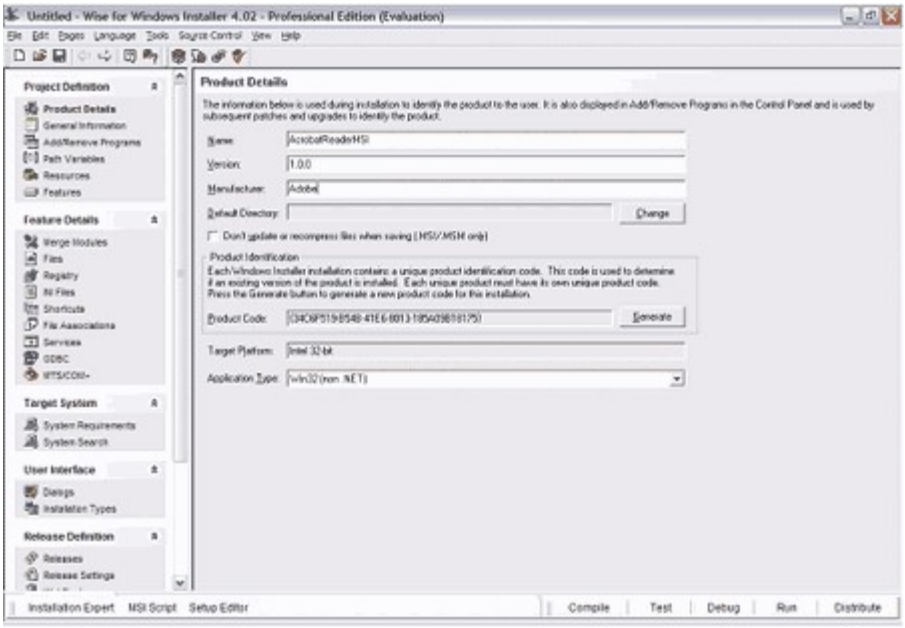

Figure 5. Product details

11. Click the menu choices on the left. In particular, click on the **Features** icon to view the package by feature. Notice that the "*Complete*" name is used for the feature as we accepted in the previous step. Right-click the feature and select **rename…** from the **context** menu. You may choose to rename the feature to something else and you may add and edit the contents of feature groups here.
12. Explore the **Feature Details** section to view and edit the files, registry settings, shortcuts, etc. that the capture has decided are part of the package. You can add and delete settings to the final package.
13. Finally, select **Save** from the **File** menu and name and save the .msi file. Exit Wise for Windows Installer.

You should now restore the test computer to its original state and test the .msi file that you have created. Remember when testing that you need to be logged on as an account with sufficient privileges to do whatever the package needs, or you should have set-up an Active Directory GPO environment to test deployment of the .msi as a managed application. Wise also provides a tutorial in the Help files provided with the product that you can run through to further familiarize yourself with the product. Wise also has a product that integrates directly into Visual Studio .NET that allows you to design .msi packages as projects in that development environment. You may want to investigate this if you have chosen to develop your application in Visual Studio .NET.

By creating a new blank project in Wise for Windows Installer, you can begin to construct your own .msi from scratch to install your applications components. You can decide which files get copied to which locations on the destination computer. Similarly, registry entries, shortcuts and so on can all be configured easily in the various menu screens.

To further explore the power and flexibility of this technology, click the **MSI Script** or **Setup Editor** tab at the bottom left of the screen to get into the more advanced options for building packages.

Deploying Interix Applications Using the Berkeley "r" Commands

The script copies an application's updated image remotely to a Windows Interix computer and modifies the application's startup symbolic link to point to the new application.

This installation script is copied (using **rcp**) to the target computer and then executed at the target computer using an **rsh** to copy a specified application directory to the target computer (where this script is executed) using **rcp**. It first checks:

- Whether the platform is Interix
- What the current application version is, if any
- Whether there is enough disk space at the target machine

The arguments to this script include:

- The directory or file where application will be installed on the target computer
- Disk space (in KB) needed for the file or directory where application will be installed
- The directory or file to be removed, which should not be the same as the second argument
- The source machine where the application's installation directory exists
- The path of application on the source machine

If you specify **-f**, the application directory or file will be removed if it already exists, and then installation will be performed. If this option is not specified and the application directory or file exists, it will not be removed and this script would result in the following error:

```
The product (same or different level) is already installed.
```

Script for remote deployment of applications in Interix

```
#!/bin/ksh
#
# ------------------------- Start of Functions-----------------------
CHECK_DISK_SPACE ()
{
  TARGET_DIR="$1"
  SPACE_REQ="$2"
  AVAILFILE1="$3"
  AVAILFILE2="$4"

  if [ ! -d "$TARGET_DIR" ]; then
    return 2
  fi

  cd "$TARGET_DIR"
  let AVAILSPACE=`df -k . | tail -1 | awk '{print $4}'`
  TARGET_DIRFILESYSTEM=`df -k . | tail -1 | awk '{print $1}'`

  if [ -a "$AVAILFILE1" ]; then
    cd `dirname "$AVAILFILE1"`
    AVAILFILE1FILESYSTEM=`df -k . | tail -1 | awk '{print $1}'`
    if [ "$AVAILFILE1FILESYSTEM" = "$TARGET_DIRFILESYSTEM" ]; then
      let AVAILSPACE="$AVAILSPACE"+`du -skx "$AVAILFILE1" 2>/dev/null
        | awk '{print $1}'`
    fi
  fi

  if [ -a "$AVAILFILE2" ]; then
```

```
    cd `dirname "$AVAILFILE2"`
    AVAILFILE2FILESYSTEM=`df -k . | tail -1 | awk '{print $1}'`
    if [ "$AVAILFILE2FILESYSTEM" = "$TARGET_DIRFILESYSTEM" ]; then
      let AVAILSPACE="$AVAILSPACE"+`du -skx "$AVAILFILE2" 2>/dev/null
        | awk '{print $1}'`
    fi
  fi

  if [ "$AVAILSPACE" -lt "$SPACE_REQ" ]; then
    return 9
  fi

  return 0
}
# ------------------------- End of Functions -----------------------
if [ `uname` != "Interix" ]; then
  echo "The operating system is not Interix."
  exit 1
fi

FORCE="n"
if [ "$6" = "-f" ]; then
  FORCE="y"
fi

#dir/file where application will be installed
INSTALL_LOC="$1"

#disk space (in KB) needed in `dirname $INSTALL_LOC`
SPACE_REQ="$2"

#dir/file to be removed (should not be same as $INSTALL_LOC)
REMOVE_LOC="$3"

if [ -a "$INSTALL_LOC" -a "$FORCE" = "n" ]; then
  echo "The product (same or different level) is already installed."
  exit 2
fi

TARGET_DIR=`dirname "$INSTALL_LOC"`
CHECK_DISK_SPACE "$TARGET_DIR" "$SPACE_REQ" "$INSTALL_LOC"
  "$REMOVE_LOC"
FLAG="$?"

if [ "$FLAG" = "9" ]; then
  echo "Not enough disk space is available."
  exit 3
elif [ "$FLAG" != "0" ]; then
  echo "The available disk space cannot be ascertained."
  exit 4
fi

rm -rf "$INSTALL_LOC"
rm -rf "$REMOVE_LOC"

#Source machine dns name where the application's installation
  directory exists
SOURCE_HOST="$4"

#Directory path of application on source machine
```

```
SOURCE_LOC="$5"
rcp -r "$SOURCE_HOST":"$SOURCE_LOC" "$INSTALL_LOC"
```

Using the remote deployment script

Using the following instructions, you will be able to deploy your application to a remote machine. In the following instructions, that remote machine is called *targetmachine*.

To deploy your application

1. Copy the script **install.ksh** to target machine using **rcp**.
2. Then execute the install script as follows:

```
$ rsh targetmachine /dev/fs/C/tmp/install.ksh /dev/fs/C/tmp/test1
200 /dev/fs/C/appdir/app1 sourcehost /dev/fs/E/SFU/app1
```

Where the arguments are:

- **/dev/fs/C/tmp/test1**—the directory or file where application will be installed on the target computer
- **200**—the disk space in KB needed for the file or directory where the application will be installed
- **/dev/fs/C/appdir/app1**—the directory or file to be removed (not be the same as the second argument)
- ***sourcehost***—the source machine where the application's installation directory exists
- **/dev/fs/E/SFU/app1**—the directory path of application on source machine

Note: If you attempted to run the script twice you would see the following:

```
$ rsh targetmachine /dev/fs/C/tmp/install.ksh /dev/fs/C/tmp/test1
200 /dev/fs/C/appdir/app1 sourcehost /dev/fs/E/SFU/app1
$ "The product (same or different level) is already installed."
```

This is because the application already exists on the target machine. If you had used the **-f** option, you would see:

```
$ rsh targetmachine /dev/fs/C/tmp/install.ksh /dev/fs/C/tmp/test1
200 /dev/fs/C/appdir/app1 sourcehost /dev/fs/E/SFU/app1-f
```

The installation would have installed twice.

Using DFS to Automount in Interix

The script below opens a connection to a domain and then queries Active Directory to get the user information. By using that information, the logon script then maps a drive.

Note This application will require modification before it will work on your system. You will need to change the sections indicated by "**CHANGE**".

```
<Job id="Map Drives">
<SCRIPT LANGUAGE="VBScript">
sub Process
On Error Resume Next
```

```
  dim szDrive

  szDrive = "H:"        'Drive letter to map, change this as
appropriate.

  'Build the LDAP string. If you want simply replace the code below
  'with the LDAP string directly. This was done only to make it
easier
  'to understand for someone filing in the strings.

  Dim szLdapPath      'Variable used to build the string.
         'This is not the most efficient way to
         'do this, it is just easier to explain. Use
         'ADSVW and open a user and look at the
         'Distinguished Name to get the string to
         'use with the path. The breakdown of the
         'path shown below is as follows:

  szLdapPath = "LDAP://"  'LDAP name space.     ***DO NOT CHANGE***
  szLdapPath = szLdapPath & "Domain" 'Domain where you will attach
***CHANGE***
           'to get the information.     ***CHANGE***
  szLdapPath = szLdapPath & "/CN=Users," 'Container where the user is
**CHANGE**
           'located in the directory.   ***CHANGE***
  szLdapPath = szLdapPath & "DC=Domain,DC=com"
         'Full DNS path expressed in   ***CHANGE***
         'LDAP syntax.                ***CHANGE***

  WScript.Echo "Active Directory Path=" & szLdapPath

  'Using the network object for the WSH get the name of the logged on
    user.

  Dim szUserName               'Name of the logged on user.
Dim szNetPath    'String holding the network path to return.
  Dim szDomain                    'String holding the name of the
domain.

  Set objNet = WScript.CreateObject ("WScript.Network")
  if Err <> 0 then
    WScript.Echo "1:Error opening WScript.Network - " &
Err.Description & _
                 " Error Code=" & hex(Err.Number)
    Exit Sub
  end if

  'Get the user name that was used to log on.

  szUserName = objNet.UserName
if Err <> 0 then
      WScript.Echo "2:Error getting user name - " & Err.Description &
" Error Code=" & _
                hex(Err.Number)
      Exit Sub
  end if
WScript.Echo "User Name=" & szUserName

  'Create the ADO connection object and the command object that will
```

```
be used to
  'execute the query.

Dim objConn    'Object used to connect to the Active Directory.
Dim objCmd    'Object used to issue the command to the database
          'connection.
Dim objRecord    'Object used to return the data.

  set objConn = CreateObject ("ADODB.Connection")
  if Err <> 0 then
      WScript.Echo "3:Error Creating ADO connection - " &
Err.Description & _
                  " Error Code=" & hex(Err.Number)
      Exit Sub
  end if

  set objCmd = CreateObject ("ADODB.Command")
  if Err <> 0 then
      WScript.Echo "4:Error creating command object - " &
Err.Description & _
                  " Error Code=" & hex(Err.Number)
      Exit Sub
  end if

  'Set the connection to use the Active Directory as a source of the
data.

  objConn.Provider = "ADsDSOObject"
  objConn.Open "Active Directory Provider"
  if Err <> 0 then
      WScript.Echo "5:Error opening ADO connection. - " &
Err.Description & _
                  " Error Code=" & hex(Err.Number)
      Exit Sub
  end if

  'Assign the connection to be used with the command object. This
tells the
  'command object to execute using the connection to Active
Directory.

  Set objCmd.ActiveConnection = objConn

  'Build the SQL string that will be used. This is selecting the name
and comment
  'fields, if a field other than comment is used to return the data,
then change the
  'string in the first part of the select statement. The search is
done on the user
  'object and the SAM account name, which is the name used to log in
as. This
  'Should return only one object.

  objCmd.CommandText = "select name, comment from '" & szLdapPath & _
                       "' WHERE objectClass='user' AND
sAMAccountName='" & _
                       szUserName & "'"
                         WScript.Echo objCmd.CommandText

  'Execute the command setting the record set to hold the results. In
```

```
this case there
  'will be only one record and no reason to loop.

  Set objRecord = objCmd.Execute
  if Err <> 0 then
      WScript.Echo "6:Error executing command - " & Err.Description &
_
                 " Error Code=" & hex(Err.Number)
      Exit Sub
  end if

  'The network path to connect to is in the comment field. If this
isn't the full
  'string used for a connection, then parsing needs to be done here.
This assumes that a
  'network share will be provided that can be used to do the mount.

  szNetPath = objRecord.Fields ("comment")
  WScript.Echo "Name=" & objRecord.Fields ("Name") & " MountPath=" &
szNetPath

  'If the path string is empty, then a value was not found, in which
    case simply return
  'the error and exit. If the path is not empty, then do the mount.
Change
  'the drive connection letter to the appropriate value.

  if szNetPath=Empty then
      WScript.Echo "No path"
  else
      objNet.MapNetworkDrive szDrive, szNetPath, FALSE
      if Err <> 0 then
    WScript.Echo "7:Error mapping the network drive - " &
Err.Description & _
                  " Error Code=" & hex(Err.Number)
    Exit Sub
      end if

          WScript.Echo "Mount successful"
end if

end sub

Process
</script>
</Job>
```

Send feedback to Microsoft

© Microsoft Corporation. All rights reserved.