



realtimepublishers.comtm

The Definitive Guide™ To

Windows Software Deployment



Making Complex
Technology **SIMPLE**
Since 1985

Chris Long

Chapter 7: Installation Tool Alternatives.....	168
Windows Installer	168
Overview	169
Windows Installer Transforms.....	174
Windows Installer Patches	182
Fixing Damaged Applications	182
Windows Installer Elevated Privileges	183
Creating a Windows Installer File	183
Running an Installer File.....	186
Resource Kits	186
Summary	194


Chapter 7: Installation Tool Alternatives

Being the expert software deployment technician that you obviously are, you keep abreast of all the new technology. You have a list of favorite Web sites to review, you keep up on Microsoft changes, and you know your workstations. Making new changes are not that big of a deal. So if a new process comes along that is not compatible with your existing environment, you have a bag of tricks available that will help you solve the problem, right?


No matter what type of deployment method you have in your environment, there will come a time when you will need to adjust or modify the standards. To that end, you need to stay up on the various methods and tools available to you. The biggest tools out in the Windows world right now are Windows Installer and the resource kit utilities.

Windows Installer

If you have a Win2K or later OS on your system, you already have access to some pretty good tools. You've heard of Microsoft Installer (MSI—it also goes by the name of Windows Installer and MSIEXEC, which is the name of the executable). In this chapter, I'll use the names Windows Installer and MSI interchangeably to reference the same technology.

 For more information about MSI, you can do a search on the Microsoft Web site. Doing so gives you lots of hits with several key pages to check for further details. Do the same for Windows Installer on <http://www.zdnet.com> and you will get a large number of hits.

Microsoft has placed a major emphasis on the technology—most of their applications are written to support MSI. Other companies are following suit when they create their installer programs.

 Check out <http://www.appdeploy.com>, which has several pages devoted to the MSI world, including a good cross reference that shows which packaging technologies are compliant (or use) MSI.

So what the heck is this thing everyone is using? In short, it is a tool that works to ensure the consistency and viability of an application. Application installation is controlled with a standard set of procedures. Elevating user privileges can be done within the process (if you are on the proper OS). DLL Hell becomes a quaint term from past nightmares. (Well, not 100 percent, but pretty close). By monitoring files, registry entries, and source locations, MSI strives to keep applications functional.

Windows Installer 2.0 will run on Win95, Win98, Windows ME, NT 4.0, Win2K, and Windows XP. Though native with Win2K and Windows XP, a version of the installer service is available for the earlier OSs. Functionality is a little different for the older OSs and not all options (such as advertising) are available for all versions; you may need to do some minor upgrading to get all the goodies.

Using a combination of an internal database, files, and comparisons, Windows Installer can ensure that an application has all the components needed to run the application as you packaged it. Applications can be repaired, updated, and upgraded using internal logic to the technology without having to go through massive design hurdles to design the logic flow.

Like all technology, Windows Installer is not 100 percent foolproof. If an application is not designed for MSI use, even Microsoft states that the repackaged application will only be installed via Windows Installer and not truly managed by installer rules. I'll explain this caveat in more detail later.

Overview

Before I get too far into a discussion of Windows Installer, an overview of this technology is needed. Windows Installer 2.0 is the current release version and runs as a 32-bit or 64-bit service, depending on the OS and the hardware platform you are using. For 64-bit use, the Intel IA64 hardware platform is required along with either Windows XP Pro or Windows .NET Server. 64-bit applications are not available for installation on a 32-bit OS. Because Windows Installer 1.1 is 32-bit based, most of the install packages available today are 32-bit. Any application that is engineered for 64-bit use will be clearly marked as such.

Windows Installer uses a series of files and databases to perform its functions. The primary package is contained in a file with an .MSI extension. This file contains the database and the instructions and data needed by the Installer service to properly manipulate the application. An .MSI file contains details about how to install, upgrade, or remove the application plus how to change features in the application. If you look at an MSI-compliant application suite (such as Office XP), you will see several *.MSI files listed in the setup directory. If you right-click one of the files, you will see additional options in the pop-up menu: install, repair, and uninstall, as Figure 7.1 illustrates. This pop-up menu gives fast access to the canned functionality of the package. Notice that I said canned functionality.

From the users' point of view, an MSI file gives greater flexibility for installs than past mass technologies. But only to the limit the install developer is willing to offer the user. The true power of Windows Installer lies in its nuts and bolts. All those setup files make adjustments to the way the application is installed. The various parts of the application are called *elements* in the world of MSI.

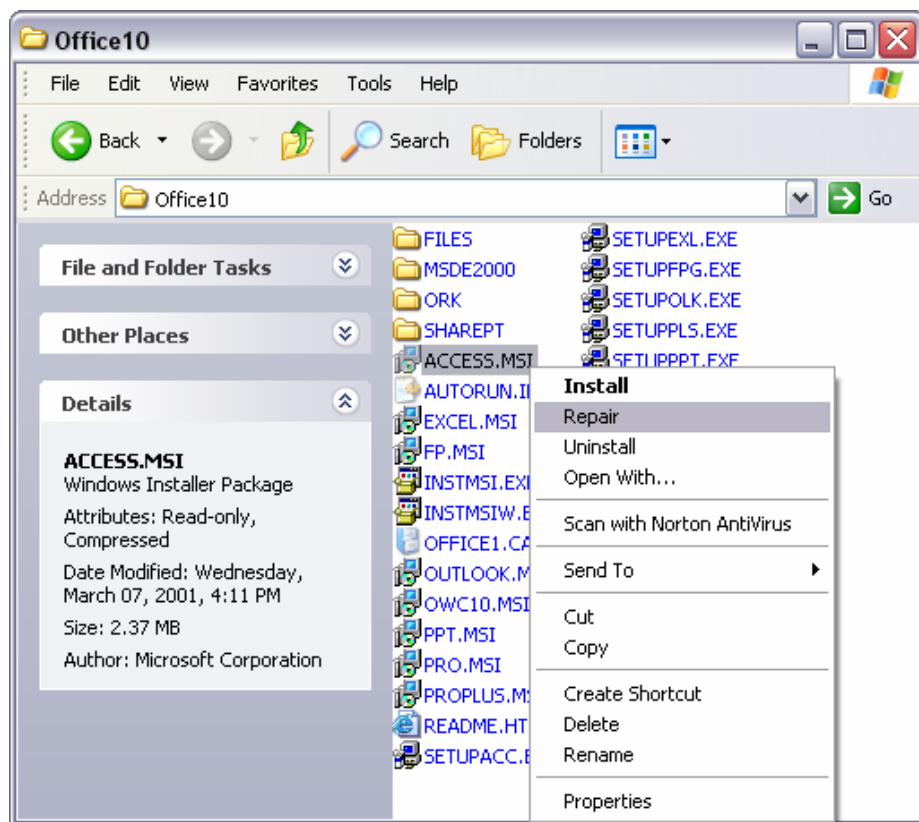


Figure 7.1: The pop-up menu options for the MSI install file are different from the standard pop-up menu options.

To understand the technology, I need to dig a little deeper into the terminology and logic of Windows Installer. Within an MSI file, details about how to deal with the various elements are included. An element is one of three general containers: components, features, or products.

- **Component**—A collection of the files, registry keys, shortcuts, or databases that are to be worked on, components are considered required for install or removal and are acted on together, not separately. Components are the smallest of the elements and are the base level. Picture them as small modules with only a couple of files, registry entries, or OLE registrations (or other actions) inside.
- **Features**—Working up the size ladder, a group of components is called features. During the install (or uninstall) process, a user chooses a feature to act on. The feature would call the proper components to perform the action. Opting to install Word Perfect Help when you install MS Word is an example of selecting a feature.
- **Product**—This element is just as you expect—a collection of features create a product. MS Word is a product comprised of features.

A suite is a collection of products. The Microsoft Office suite is a collection of products. One variance to the logic; if a native Windows Installer package is used to control the setup of the products, the entire setup is considered a product, not a suite. So if you install just Word via Windows Installer, it is a product. If you install Office and include the applications Word, Excel, PowerPoint, Access (and everything else), then the entire Office install is considered a product and the individual applications are the features. If you install each application separately, the grouping is a suite. The key is the Installer package; any major grouping within the Installer package becomes a feature of the Installer package. As Figure 7.2 shows, each element builds on the previous piece.

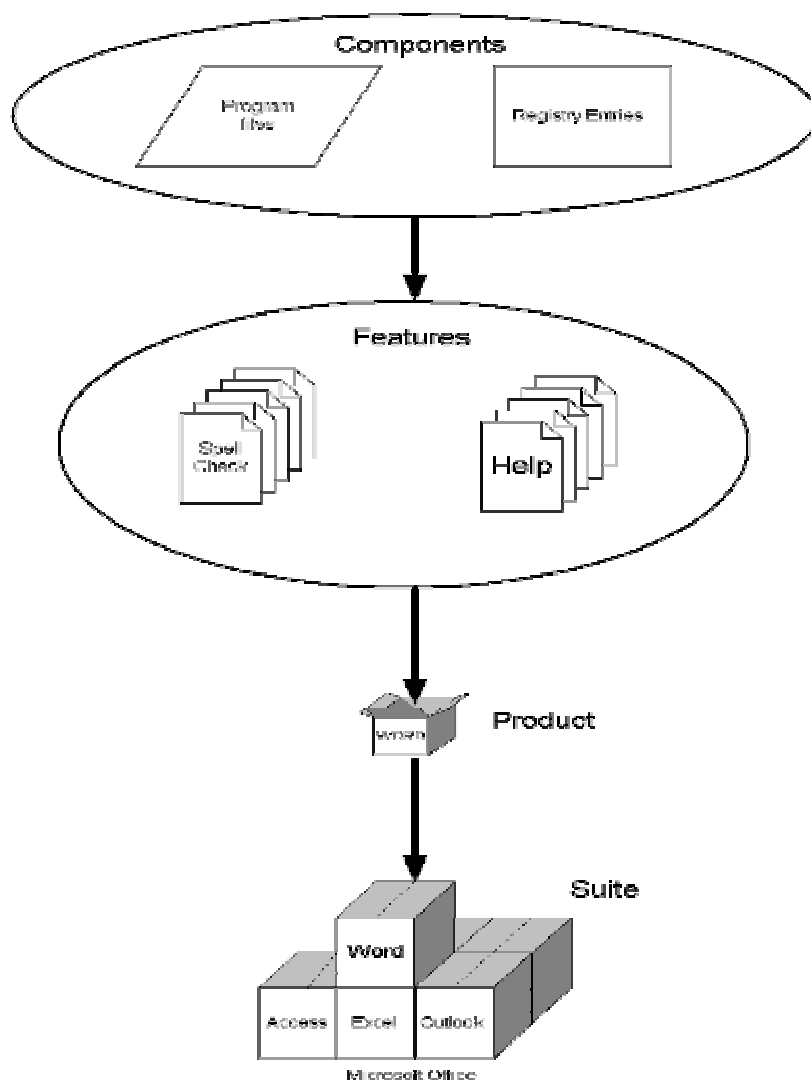


Figure 7.2: Windows Installer element hierarchy.

The component level is the basic building block. One of the functions for which MSI uses the component is verification of package consistency. In other words, when a Windows Installer-based application starts up, Windows Installer can do a verification to ensure that nothing has been removed or overwritten. For the verification function, Windows Installer does the check at the component level. In the component, one of the files or registry entries is designated as the

keypath. When the application does a call to the component, the keypath is the part that is actually called rather than each file. For this reason, the components are usually small. If a file that is defined within a component is deleted, the call to the component identifies that the file is missing; the component also defines where to go for the proper source to restore the component pieces to operational level. That proper source location is called the *installation point*. This is the location from where the application was originally installed.

Can you have a single file as a member of multiple components? Sure, if the file reference is created within the component properly. And that is the catch—the component must be defined properly during the creation process.

From your point of view as the administrator, knowing that a file is referenced by multiple components is good troubleshooting basics. When a package is removed, knowing why a file is removed (or not) can help determine why another application is suddenly breaking.

If a file is part of both Component A and Component B, and you removed Component B, the system would know that the references to Component A still exist. For files that are shared between products, the applications include the same component. Microsoft says that the components are small pieces and only include resources that are directly related to the function of the component. The MSI internal database records that two applications are using the file. When one of the applications is removed or changed, the database tracks the change and strives to keep the modification from impacting the other application. In other words, a series of building blocks are tied together by the Windows Installer service, and the Installer service acts as the traffic cop to keep everyone moving properly.

As a software deployment expert, you are familiar with the term GUID. Windows uses these identifiers extensively to keep track of the numerous aspects of the operation of the computer. To keep track of which application is using which component, a GUID is assigned to each component and product. The technical terms are *component code* and *product code*, respectively. I won't get too deep into a discussion about the use of GUIDs; just remember that a Windows Installer database tracks which component code (such as a file) is assigned to which product code (such as an application).

Windows Installer removes files from a system only with the other elements of the component and when the magic reference counter (*refcount*) shows 0 for that file. Thus, removing an application that is controlled by Windows Installer does not accidentally remove a file from another Windows Installer application. In Chapter 3, I discussed the `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs` registry key? That key does a reference count on how many applications are registered to use a DLL. The Windows Installer database does much the same, only instead of just listing the number of associations, it can actually tell you which application is using the file. This tracking extends past the DLL tracking and includes any component installed via the Windows Installer process.

Features group the components together into a logical collection. Within the installer file, the user chooses which features to work with. Instead of the user (or you) having to figure out which file and registry element is needed to work with those features, Windows Installer does the work for the user (or you) by referencing the components. So selecting the WordPerfect Help feature in Microsoft Word will install the proper components, which means that the proper files, registry entries, and hooks into Word are installed. The grouping makes life easier for everyone.

When an application is installed, Windows Installer will provide four main choices for each configurable feature:

- Installed on local hard disk
- Installed to run from source



- Advertised
- Not installed

The first two options are pretty clear. You can install the application to the local hard drive to run it locally or choose to install (or execute) the application from the defined source. The source location could be a server or a CD-ROM depending on how you have the application set up. If you need to save space on a workstation and don't have issues with network bandwidth, opting to install and run from the source might be the best option for you. Within a product, you could mix the install options depending on how you see the need. For example, you could install all the basic features locally to provide faster processing for the commonly used items. Additional fonts, wizards, templates, and other lesser-used features could be left to install to run from source (or install on first use). This setup provides a good mix for the end user—faster response for most items and access to additional features as the user needs.

Advertising applications is a heavy concept in the software distribution world. When you advertise an application in AD deployments, you essentially are telling the user that an application exists but the application is not yet installed. When the user clicks the file icon, an install program is activated to drop the program onto the machine or userid. Basically, consider MSI as repairing the entire application because none of the components are present on the system.

In Windows Installer, the same concept is available at the feature level. In Figure 7.3, which shows a custom install for Microsoft Access 2002, notice the disk icon with the number 1 in it. This graphic represents that the feature is to be installed on first use. Thus, the components are not yet installed on the system, but the keypath in the feature's component defines where the files are located for install. Shortcuts may be installed and OLE registration may be done, depending on the configuration needs.

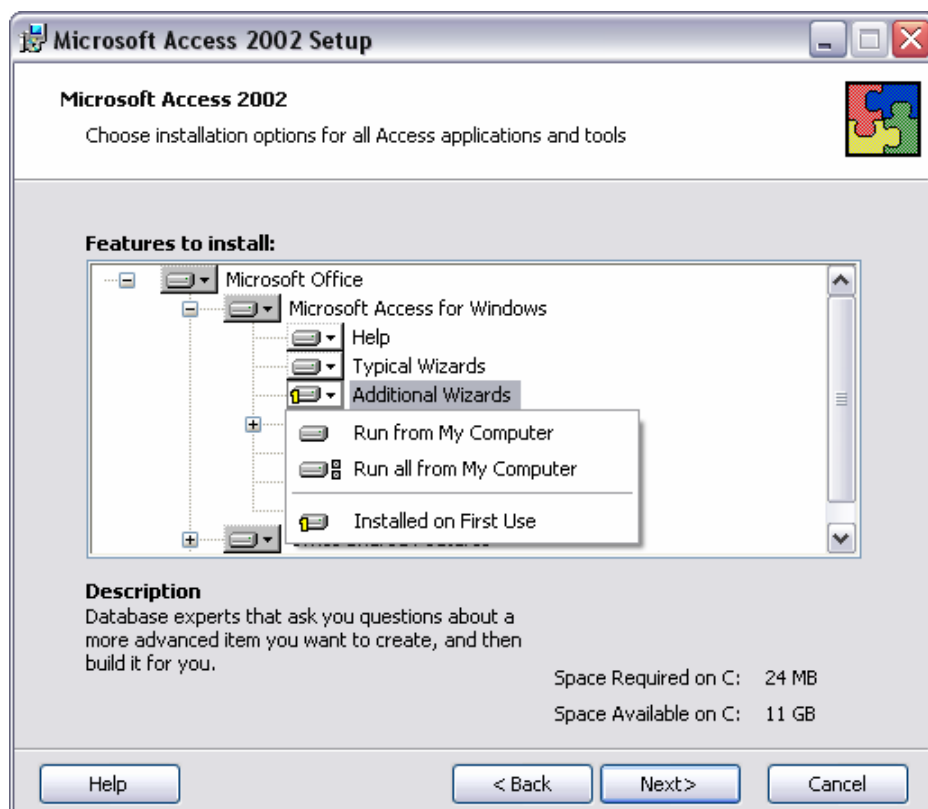


Figure 7.3: Feature installation options.


In the highlighted graphic of the snapshot, the Additional Wizards feature is only a ghost entry on the computer; the added wizards don't exist yet. In the course of using Access, a call to a wizard that is not installed may be encountered. At this time, the Windows Installer kicks off, presents a pop-up window indicating that the system is going out to get the added files (components), and the feature is installed.

You can see how this functionality saves space on a system. Why load a bunch of features if you don't use them? Having the space taken up on the drive by unused features only wastes the space. If you think a feature might be used, loading it might be worth the tradeoff. Older technology and some other systems give you the option to install a feature set or not. Because you might have a need for a feature at some point, not loading a feature meant going through the setup process again if you discovered that you need the option. This process is time consuming—it's a hassle to find the data source and a major interruption to work. By having the advertise ability within Windows Installer, the application loads in the feature with only a few minutes delay. This discussion assumes, of course, that the source is located someplace convenient and the new feature is not a major tweak that requires reboots.

Windows Installer Transforms

All this data and more are included in the .MSI file. The .MSI package is the base file system for a Windows Installer application and will install an application in a manner and method that the installation creator chooses. If you want to adjust those install options, a *transform* file is needed to effect those adjustments without having to modify the vendor's MSI. The file extension of .MST identifies transform files.


A transform lets you make changes to the base .MSI files at install time. Instead of letting users adjust the installation features to their whims, you can create separate transforms for various features, depending on the need. Creating a transform for different departments is a common example. The Accounting Department does not need the same features of Word that the Warehouse Department needs. By applying your group distribution methods, each department could call a Windows Installer application that loads specific transforms to Word the first time that Word is loaded.

 Transforms are applied when an application is first installed. Transforms created from .MST files are not used for making modifications to existing applications; those changes are done via .MSP (that is, patch) files.

When doing GPO distributions in a Win2K Server or .NET Server environment, transforms are the supported method for customizing the base application during the MSI-based installation process

Transforms give you access to feature sets as if you were doing the install. After the transform modification is saved, you can create a different transform from the same base .MSI application that could have an entirely different feature set. Saving the transforms with a reference in the name to the reason for the modifications will help you manage the variances. This way, the Word .MSI application can have a unique install for both the Accounting Department and the Warehouse Department without having to create entirely separate packages for each department. If you are deploying a commercially available application via Windows Installer, a method of creating transforms is sometimes included. Now you will understand why I've been using Microsoft Office products as the examples for this chapter. The transform tools are readily available for you to use while you get a good understanding of Windows Installer.



 The Office XP resource kit and toolbox Web site (<http://www.microsoft.com/office/ork/xp/appndx/appc00.htm>) has the utilities to do customizations. The Office resource kit tools are located about halfway down the page. The executable is called OrkTools.exe. (And yes, the executable is a Windows Installer setup.)

After the resource kit is installed, you can begin the customization by opening the Microsoft Office Custom Installation Wizard. The welcome splash screen, which Figure 7.4 shows, gives you the overview.

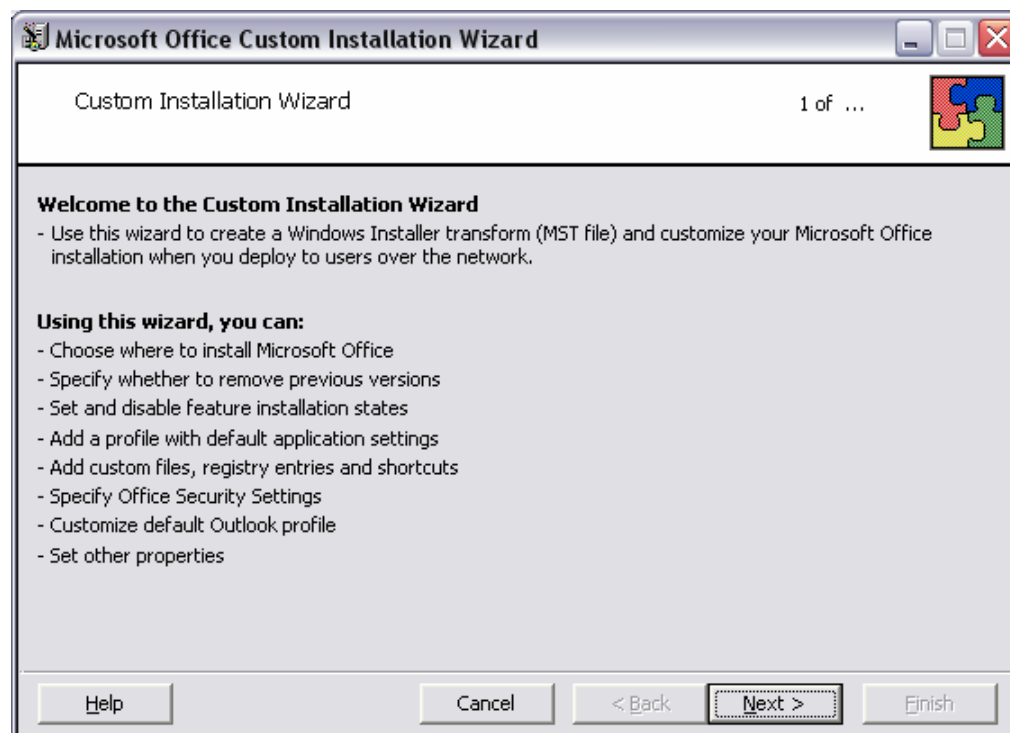


Figure 7.4: This screen gives you an overview of the functions that the utility will perform.

The next series of screens guides you through the process of selecting the base .MSI file, naming the MST file and location, and selecting various options. You don't need to save the .MST file to the same location as the .MSI file. In fact, if you are going to do many transforms for the base application, you might want to create a separate directory in which you can save the transforms. I've selected the word.MSI file stored on my server for customizing and have instructed the program to save the transform file called WordTest.MST to the C:\RTP directory. For this example, I've changed the feature installation state for the Help for WordPerfect Users to Not Available, as Figure 7.5 shows. All other features remain as they are currently configured.

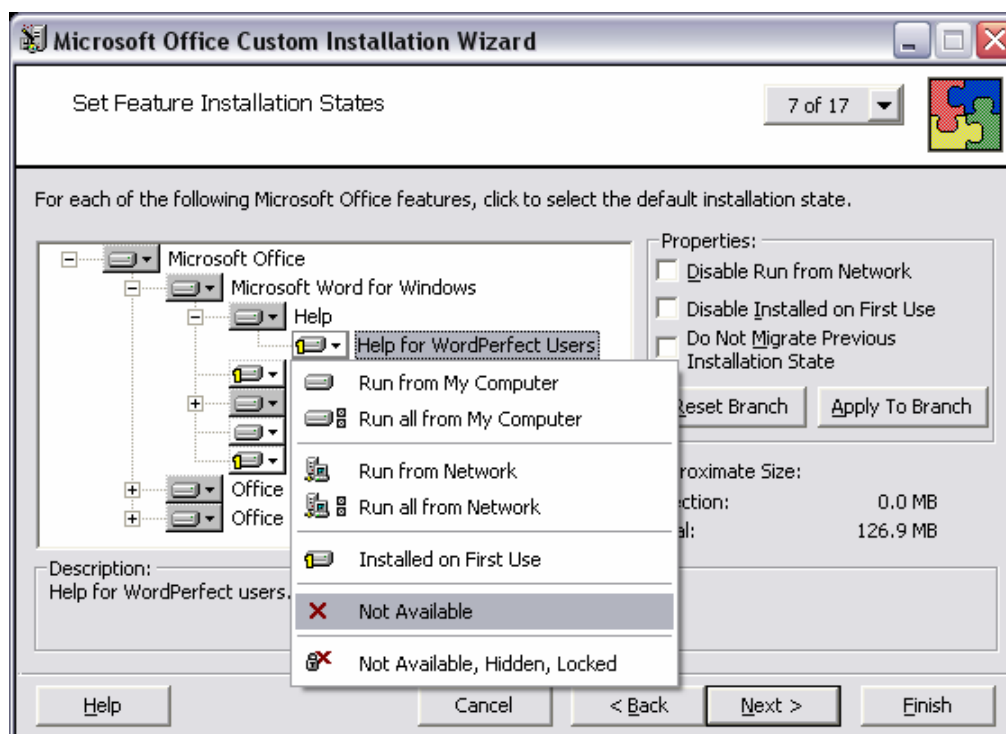



Figure 7.5: Disabling the feature Help for WordPerfect Users.

One type of Microsoft Office-specific file needs a quick mention at this point; the OPS file. This file contains default custom user settings for applications that are applied to a computer. These settings are mostly the configuration options in the Tools, Options menu and include such common items as toolbar use, menu options, and positioning of toolbars. You can create the file with the Profile Wizard from the resource kit, and import the settings into the transform. If you're not familiar with or don't want to use the Profile Wizard (an administrator-level function), you can get the same results from the Save My Settings Wizard (which is for the user level). What is configured is dependent on the INI file settings. (See the Help file in the Profile Wizard for more details.) In this case, I've opted to use the Microsoft default settings, as Figure 7.6 illustrates.



Figure 7.6: Default settings page.

The next major screen, which Figure 7.7 shows, gives you the option to customize several user settings for the products that the transform supports. The screen indicates that *These settings are applied to all users on the computer and overwrite existing settings*. After the transform is applied, the settings become active.

 Be careful when you are in this screen. If you change a setting and decide that the setting isn't correct, simply backing out or canceling the action does not really mean the change has not been written to the transform.

Be sure that what you have changed is what you really need to use. Click Help and review the information in the screen for more details.



Figure 7.7: In this screen, most of the configuration options are changed in the Office XP transform editor.

Subsequent screens step you through adding or removing files, adding or removing registry settings, modifying security settings, specifying any server locations, including additional programs, and choosing any setup properties. You then reach the end, and by clicking Finish, save the .MST file. At the end of this transform-creation process, a sample command line is given to you about how to include the transform, as Figure 7.8 shows.



Figure 7.8: The summary screen provides a sample of the MSI command line needed to include the .MST file.

For a complete discussion about properties within Windows Installer, download the Windows Installer software development kit (SDK) from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/wiport_6gf9.asp (or <http://www.microsoft.com/windows/reskits/webresources> and select Windows Installer SDK).

If you needed to create other transforms that are similar but have minor adjustments, open the same .MST file, make your adjustments, and save the transform with a different name. You can use this method to easily create multiple transforms that have different default save locations. Transform creation GUIs are different between applications and vendors. The previous example was used to help show the process. Different vendors will create different transform utilities (if at all), and even within the same vendor, the various products could have their own GUIs that differ greatly from each other. Which features you are able to modify depends on what the developer is willing to create for you. Normally, if the option is part of the install process, the transform GUI will have the same option defined.

Can the MST file be viewed without the transform utility? Only if you enjoy looking at garbled data. The Office XP resource kit has a Windows Installer Transform Viewer that lets you view the data in a Notepad format. You need to specify the base package (.MSI file) and the associated transform (.MST file). The WordTest transform that I created (which had only a single change within it) produced Listing 7.1 from the Transform Viewer. All the settings listed in the file will be applied during installation of the Office.MSI file if the transform is specified in the setup command line. You can also specify the MSI file in the GPO customization tab when you define the package properties in the GPO.

Take some time to review the listing. You can see the GUIDs of the product codes clearly listed along with the setup properties. The section in which the Help for WordPerfect Users change is listed is boldfaced in this listing. You can use listings such as this for troubleshooting problems.

```
Enforce Validation Flags: False
Base package: S:\Office10\WORD.MSI
    ProductCode: {901B0409-6000-11D3-8CFE-0050048383C9};
ProductVersion: 10.0.2627.01; UpgradeCode: {001B0000-6000-11D3-8CFE-0050048383C9}

Transform: C:\RTP\WordTest.MST
Expected values - ProductCode: {901B0409-6000-11D3-8CFE-0050048383C9};
ProductVersion: 10.0.2627.01; UpgradeCode: {001B0000-6000-11D3-8CFE-0050048383C9}
    Validate Major Version
    Validate Version is Equal
    Validate UpgradeCode
    Ignore Error of AddExistingRow
    Ignore Error of DeleteNonExistingRow
    Ignore Error of AddExistingTable
    Ignore Error of UpdateNonExistingRow

CREATED - OCW_Opt_Props DisplayName (s128*)      Value (S0)  ActualName
(s128)      Type (I2)
+  ALLUSERS      2      ALLUSERS      8
+  ARPCOMMENTS      ARPCOMMENTS 12
+  ARPCONTACT      ARPCONTACT 12
+  ARPHELPLINK http://www.microsoft.com/support      ARPHELPLINK 12
+  ARPHELPTTELEPHONE      ARPHELPTTELEPHONE 12
+  ARPNOMODIFY      ARPNOMODIFY 14
+  ARPNOREMOVE      ARPNOREMOVE 15
```



```

+ ARPNOREPAIR          ARPNOREPAIR 16
+ COMPLETEINSTALLDESCRIPTION <Default>
  COMPLETEINSTALLDESCRIPTION 11
+ CUSTOMINSTALLDESCRIPTION <Default>
  CUSTOMINSTALLDESCRIPTION 11
+ DEFAULTREMOVECHOICEDESCRIPTION <Default>
  DEFAULTREMOVECHOICEDESCRIPTION 11
+ DISABLESCMIGRATION    DISABLESCMIGRATION 17
+ NOFEATURESTATEMIGRATION NOFEATURESTATEMIGRATION 18
+ RUNFROMSOURCEINSTALLDESCRIPTION <Default>
  RUNFROMSOURCEINSTALLDESCRIPTION 11
+ RUNFROMSOURCETEXT <Default> RUNFROMSOURCETEXT 11
+ StrContactInfo        StrContactInfo 12
+ TRANSFORMSSECURE      TRANSFORMSSECURE 19
+ TYPICALINSTALLDESCRIPTION <Default>
  TYPICALINSTALLDESCRIPTION 11
+ TYPICALINSTALLTEXT    <Default> TYPICALINSTALLTEXT 11
+ TYPICALUPGRADEDESCRIPTION <Default>
  TYPICALUPGRADEDESCRIPTION 11
+ TYPICALUPGRADETEXT    <Default> TYPICALUPGRADETEXT 11

CREATED - OCW_Strings  Name (s64*) Value (s128)
+ Msi_ProductVersion  10.0.2627.01
+ Msi_UpgradeCode     {001B0000-6000-11D3-8CFE-0050048383C9}

ALTERED - Feature Feature      Feature_Parent  Title Description
          Display  Level Directory_ Attributes Lock (I2)
OCW_DisableRFS (I2) OCW_DisableJIT (I2)
<> WORDHelpForWPFiles 100{1}
4{20}

DATA CHANGE - _Validation  Table Column      Nullable  MinValue
          MaxValue  KeyTable  KeyColumn  Category  Set      Description
+ Feature  Lock Y  0  1  Category  Set      Lock
+ Feature  OCW_DisableJIT Y  0  1
  OCW_DisableJIT
+ Feature  OCW_DisableRFS Y  0  1
  OCW_DisableRFS
+ OCW_Opt_Props ActualName N
  Identifier      Actual Name of OCW_Opt_Props property-like
record
+ OCW_Opt_Props DisplayName N
  Identifier      Display Name of OCW_Opt_Props property-like
record
+ OCW_Opt_Props Type N Integer
  Type of OCW_Opt_Props property-like record
+ OCW_Opt_Props Value Y Text
  Value of OCW_Opt_Props property-like record
+ OCW_Strings Name N Identifier
  Name of OCW_Strings property-like record
+ OCW_Strings Value N Text Value
of OCW_Strings property-like record

DATA CHANGE - ActionText  Action      Description Template
+ OCW_Create_AppDataFolder Removing temporary working files

DATA CHANGE - Binary  Name  Data
+ OCW_CPYF.DLL Binary.OCW_CPYF.DLL

```



```

DATA CHANGE - CustomAction      Action      Type      Source      Target
+   OCW_Create_AppDataFolder      1      OCW_CPYF.DLL
+   _OCW_Create_AppDataFolder@4
+   OCW_INST_LOC_NEW      51      INSTALLLOCATION
+   [ProgramFilesFolder]\Microsoft Office
+   OCW_INSTALL_MP_ADDED_CONTENT      51      OCW_INSTALL_MP_ADDED_CONTENT
+   yes
+   OCW_INSTALL_MR_ADDED_CONTENT      51      OCW_INSTALL_MR_ADDED_CONTENT
+   yes
+   OCW_INSTALL_UP_ADDED_CONTENT      51      OCW_INSTALL_UP_ADDED_CONTENT
+   yes
+   OCW_INSTALL_UR_ADDED_CONTENT      51      OCW_INSTALL_UR_ADDED_CONTENT
+   yes
+   OCW_SET_COMPANY_NAME      51      COMPANYNAME RealTimePublishers

DATA CHANGE - Feature      Feature      Feature_Parent      Title      Description
      Display      Level      Directory_      Attributes
<>   WORDHelpForWPFiles      100{1}
      4{20}

DATA CHANGE - InstallExecuteSequence      Action      Condition
      Sequence
+   OCW_Create_AppDataFolder      (NOT AppDataFolder) AND
+   (OCW_DestDirUsesAppData OR OCW_UserSettingsUsesAppData OR
+   OCW_AddFileUsesAppData OR OCW_ShortcutUsesAppData)      799
+   OCW_INSTALL_MP_ADDED_CONTENT      $OCW_Added_Content_MP=3 AND (NOT
+   OCW_AC_REGKEYPATH_FOUND_MP)      1399
+   OCW_INSTALL_MR_ADDED_CONTENT      $OCW_Added_Content_MR=3 AND (NOT
+   OCW_AC_REGKEYPATH_FOUND_MR)      1399
+   OCW_INSTALL_UP_ADDED_CONTENT      $OCW_Added_Content_UP=3 AND (NOT
+   OCW_AC_REGKEYPATH_FOUND_UP)      1399
+   OCW_INSTALL_UR_ADDED_CONTENT      $OCW_Added_Content_UR=3 AND (NOT
+   OCW_AC_REGKEYPATH_FOUND_UR)      1399
+   OCW_SET_COMPANY_NAME      NOT DONTUSEOCIWORNAME      799

DATA CHANGE - InstallUISequence      Action      Condition      Sequence
+   OCW_Create_AppDataFolder      (NOT AppDataFolder) AND
+   (OCW_DestDirUsesAppData OR OCW_UserSettingsUsesAppData OR
+   OCW_AddFileUsesAppData OR OCW_ShortcutUsesAppData)      999
+   OCW_SET_COMPANY_NAME      NOT DONTUSEOCIWORNAME      999

DATA CHANGE - LaunchCondition      Condition      Description
+   ReleaseType <> 1      [CANNOTRUNCIWTRANSFORMS]

DATA CHANGE - Property      Property      Value
+   CIWVersion      10.0.3228
+   COMPANYNAME      RealTimePublishers
+   DISABLEFEATURECONDITIONS      Yes
+   OCW_INST_LOC_NEW      [ProgramFilesFolder]\Microsoft Office
+   REMOVELPK      0
+   TransformType      Customization
<>   SKIPREMOVEPREVIOUSDIALOG      1{0}

```

Listing 7.1: Transform viewer output.

Could you manually make modifications to the .MST file and execute it that way? You could. However, unless you are sharp on the Windows Installer variables and values for the application as well as the interrelationships of the entries, I suggest sticking with the GUI method.

Windows Installer Patches

Remember that the .MST file is used for initial setup tweaking. After you have an application out to your clients and you want to make adjustments, Microsoft says that you need to create an .MSP file. The patch file (.MSP) modifies the .MSI information. The advantage of the .MSP file is the ability to modify the .MSI file without requiring a fresh install of the application. The patch can be a simple data change or a complete upgrade. Whereas the .MSI file contains all the databases and data stream of the application, the .MSP contains a database transform that modifies the .MSI database. Patches are generally created by the initial developer of the application and deployed using your usual deployment methods.

Although you, as the deployment administrator, might never create a patch file, understanding how it works when you get one from your application vendor is needed for your troubleshooting skills. In shops in which multiple job functions are combined, the packaging side of your duties might require you to create a patch file.

Fixing Damaged Applications

In the past couple of chapters, one of the many minor points that was raised is how to repair an application remotely. If a user removes files accidentally or other applications overwrite files needed for another application, how do you deal with the resulting damage? If you write your own distribution method, this situation can reduce the number of hairs on your head very quickly. Commercially available delivery systems (CADSs) have numerous methods for handling repairs. Some work, some don't. Windows Installer addresses this issue.

Two main functions are available in Windows Installer files when working on damaged applications: Rollback and Repair (or Verify). Rollback is putting a system back to a known pre-install state. During application installations, if the install fails, Windows Installer controlled setup will attempt to remove all the changes back to a known state. Windows Installer keeps track of the changes as they occur and will create a rollback file for replaced files. After an install is complete, the rollback files are removed from the system. Rollbacks are generally performed automatically by the setup logic.

A rollback is not always possible. If changes to a schema are performed, the changes are in the system for good or bad. If an application is running with custom programs, the logic of the custom program needs to account for the rollback possibilities. You are at the mercy of the developer in those cases.

Repair is a part of the verify process. When a product is launched, it automatically does a component verification based on the keypath data of the component. If the keypath is not in a known state, Windows Installer attempts to replace the keypath data based on the source location in the component.


A little clarification on this point is needed. As always, the level of performance that the MSI process does is dependent on the level of intelligence that the developer applies to the package. If the developer of the MSI package does not specify full repair abilities (such as version and corruption checking), the repair option will check for existence of the keypath and apply a repair process only if the keypath is not there.

You can also trigger a repair function from the command line. By specifying the repair property within the parameters of your command, you can force the application to do a full repair.

However, if the user has customized the application and you have not taken this fact into consideration, the user's tweaks could be removed and replaced with the data from the .MSI and .MST files. Of course, that may be the goal of the repair. Using this feature works great in training situations in which you want to return an application to a known state between classes.




Be sure to test the repair function and fully understand what is being done before you trigger it globally.

 Check out http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/wiport_6gf9.asp for details about repair functions. There are several considerations for using this feature properly, including the ability to prevent rollbacks.

Windows Installer Elevated Privileges

A major advantage of Windows Installer packages is the ability to elevate the security context of the install process. If your users are locked down, running installations at their security level could prevent the successful installation. With Windows Installer, you can adjust the security context for the duration of the package installation. Because users of Win95, Win98, and ME already have administrative rights on their systems, elevating privileges is not available to those folks. For the rest of the Windows users, this ability solves many problems for distributions. Right now, to fully use the elevation principle, you need to be using Group Policy distributions within Win2K Server or .NET. Other deployment methods are beginning to support the elevation concept but implement it differently. Check the documentation on your CADs for the details.

 To turn on the elevated rights option, adjust the local registry (the same key applies to the HKEY_LOCAL_MACHINE hive):

HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Installer

AlwaysInstall Elevated = REG_DWORD:00000001

Creating a Windows Installer File

If you have a package that is not a native Windows Installer-based application and would like to take advantage of the functionality of the Installer functions, there are many methods available to do the conversions. Several third-party applications are available and more are coming along all the time.

 You can get a list of third-party applications that support .MSI files on the Web. Authoring tools are also included in the lists. To get you started, check out <http://www.appdeploy.com>, <http://www.microsoft.com>, and <http://www.zdnet.com>.

Many of the major software companies provide support for creating Windows Installer files in some form or another. If you decide to start creating your own Windows Installer setup applications, make sure you fully understand your needs and review the products available before you settle on a packages. A few of the companies that have Windows Installer creation capabilities include Lanovation, Wise Solutions, InstallShield, Microsoft (of course), and Veritas.

Lanovation Prism Deploy provides the ability to create Installer packages. After you create a Prism Deploy package and save it as a Portable Windows Change (PWC) file, you can create the Installer file. This method gives you some advantages over a standard Windows Installer package. You can use the snapshot abilities of Prism Deploy to get an accurate package. Using the Adobe Acrobat 5.0 package created in Prism Deploy from a previous chapter, I converted the PWC package into an .MSI package.

To do so, open the Prism Deploy Editor, and select the PWC file of the package, in this case, the AdobeFive0_1 package was selected. From the Package menu, select Create Windows Installer File, as Figure 7.9 shows.

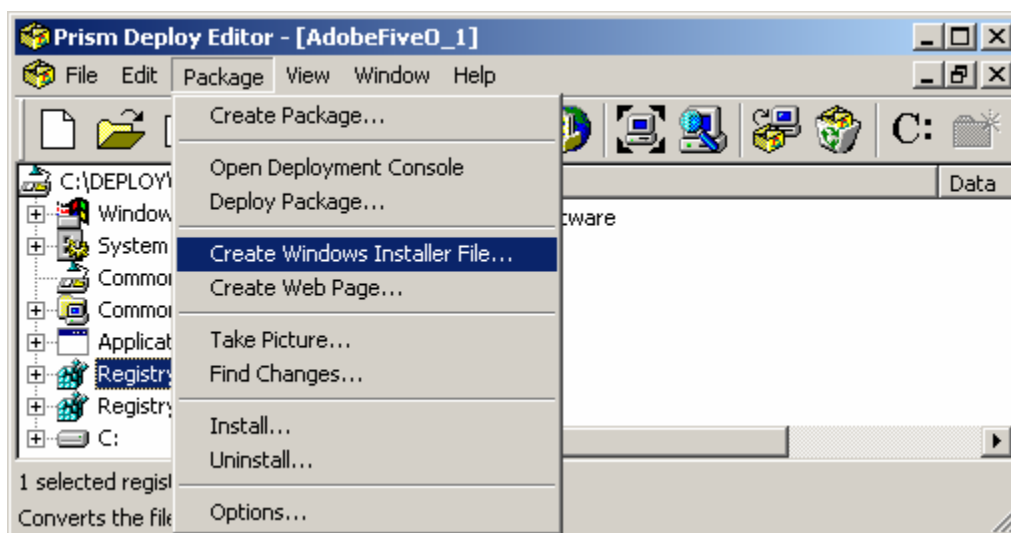
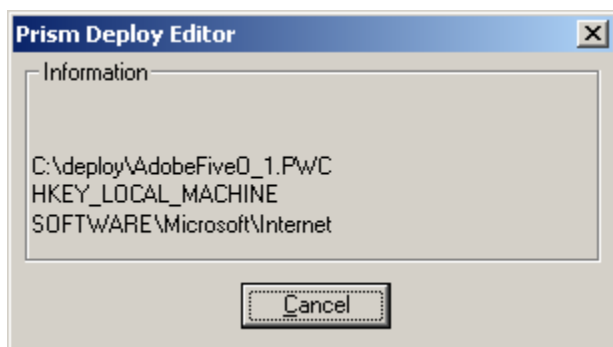


Figure 7.9: Starting the Windows Installer creation process within Prism Deploy.

The process will then ask for basic author data such as package name and the name of the manufacturer (company name). After entering this data and clicking Next, the editor begins the process of converting the PWC formatted file into the Installer format. Status messages similar to those that Figure 7.10 shows give you a progress report.



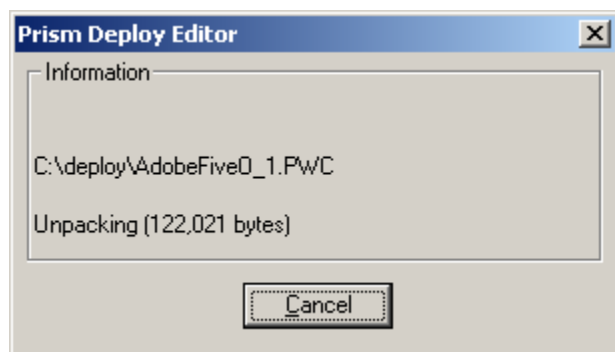


Figure 7.10: Status messages presented during the Prism Deploy Windows Installer conversion process.

After a few minutes, the Windows Installer Conversion Results window appears with a status of the package process. Any errors listed should be investigated and resolved before you use the resultant .MSI file. In this case, the messages listed in Figure 7.11 are warning messages that are indicating differences between the Prism Deploy setup and the Windows Installer process.

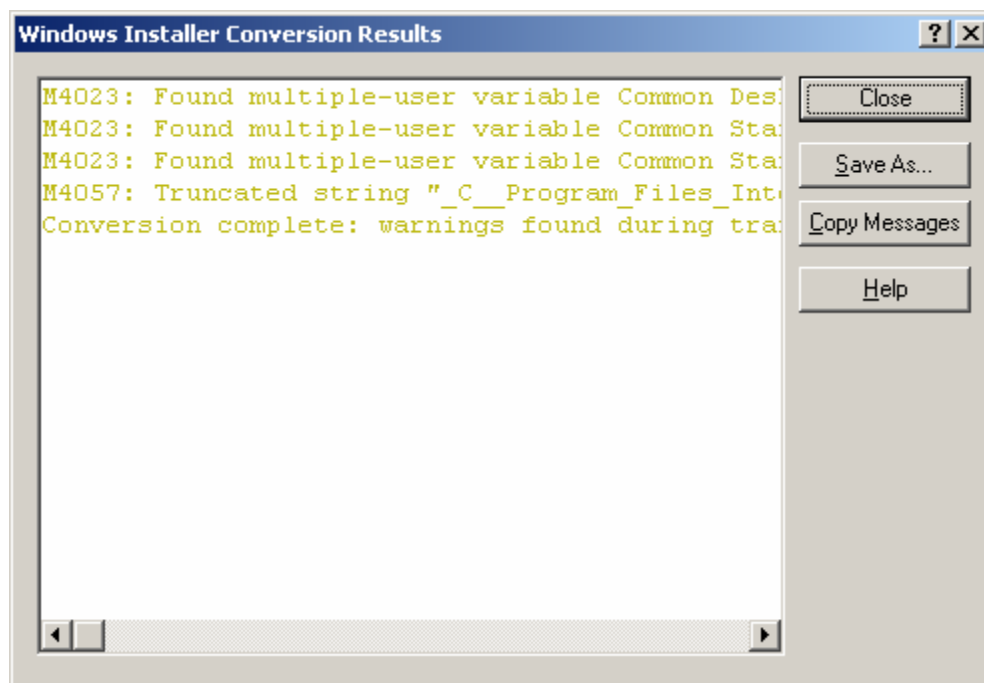


Figure 7.11: The results screen from the Prism Deploy Windows Installer conversion process.

Clicking Close closes the conversion window. When you complete the conversion process, you end up with an .MSI file in the chosen directory, as Figure 7.12 illustrates.

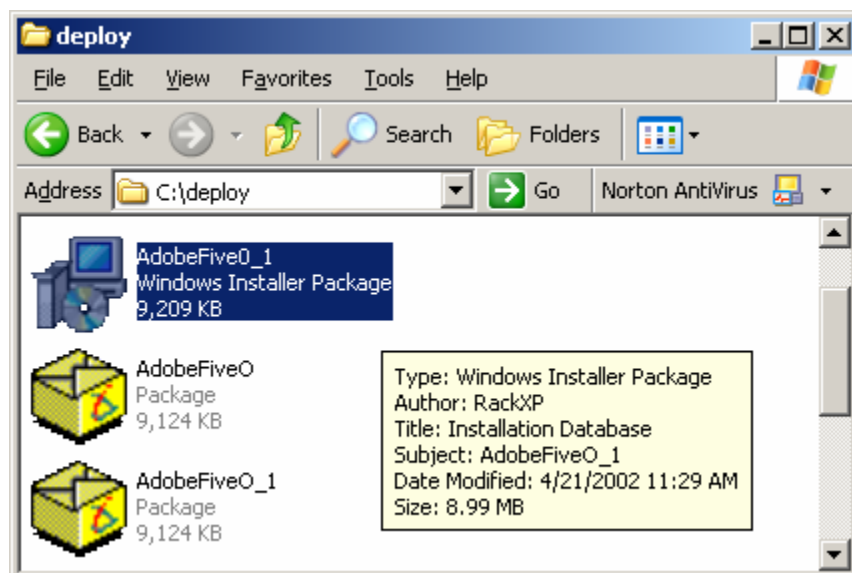


Figure 7.12: Snapshot showing the Prism PWC and Installer packages.

Running an Installer File

After you have the package created, you need to be able to run the application on the target system. The command-line format to do so is rather simple

```
msiexec /i \\server\share\filename options
```

To install the Adobe package I just created, the target machine would run


```
msiexec /i \\server\sharename\AdobeFive0_1.msi /qb
```


where the /qb option sets the user interface to the basic level. You can include logging options, an .MST file, and parameters for the properties in the command line or included files. For a complete list of the options and the requirements for each, review the Installer Web site for details (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/wiport_6gf9.asp). There is a large amount of data available about Windows Installer. The preceding information will hopefully give you enough of an understanding to start delving into the data pile.

Resource Kits

Whether you need to create a simple distribution package or just run a simple script to your workstations, you might need some utilities to create the package for you. Using the proper resource kit for the OS you are targeting is a time-saving method as many functions are already created for you. As the software distribution guru of your company, you should be aware of the various utilities in the resource kits.


Many of the utilities found in older resource kits work within new OSs. Some utilities written for NT 3.51 still work with Windows XP. Additionally, some of the older utilities that worked so well in the older technologies have been updated for each succeeding OS as needed. As with any utility you work with, test it to your satisfaction before you release it into the production world. If you haven't played with the resource kits for your OS, you should take a few hours away from the demands of the day and play. Excuse me, I meant you should enhance your contribution to the departmental goals by proactively fine-tuning your skills and enhancing your knowledge of OS functionalities to meet your quarterly objectives in a timely manner. Everyone knows you're playing productively, it just looks better on the status reports using larger phrasings.

 Two good Microsoft sites to visit for resource kit information include <http://www.microsoft.com/windows/reskits/default.asp> and <http://www.microsoft.com/windows/reskits/webresources/default.asp>.

 Resource kits are a collection of utilities that have been written to perform various functions. Not all the tools are from Microsoft programmers, and support for the tools is not always available. Microsoft states throughout the resource kit documentation that support is not provided and the products are used as is. Microsoft is mainly doing distribution of the tools that have proven effective from the company's viewpoint. Although most are safe and don't cause problems, you need to prove that to yourself first. With that word of caution, we can proceed.

After you start using the tools, several will become staples in your growing arsenal of utilities. Some help with distributions, some with customizations, some with data gathering, and others with data manipulation. In just the Win2K Server resource kit, more than 300 tools are included. Many of the resource kits (and many of the individual tools) are available on the Microsoft Web site for free.

Some of the tools available in a resource kit are not functional tools but references to documentation or information for your use. Much of the information files can be found in other areas of the Web. Keep searching for those pearls of wisdom. Overall, the kits are essentially a requirement for anyone doing serious support work for a Windows environment. Because the tools in the resource kits are not fully supported, their structure and user interfaces can be unique. It was quite common when the resource kits first came out to notice typographical errors in the Help files and instructions. But the tools work well and are worth using. Many of the tools are command-line driven, some are GUI-based. Most can be used remotely to distant machines or combined with other resource kit tools to create custom utilities for your needs. As an example of the mix that you can do with resource kit tool versions, I've included a simple report-gathering script for fun.

 One point before we get into the script example: The process used as an example can be done via numerous other methods. There are other ways to obtain the same data. The point here is to highlight the ease of use the resource kit tools can provide to solve a problem. With a skeleton script in place, adding resource kit tools into the method can give you fast data to solve problems. When creating application packages to solve problems when your CADS does not provide the functionality you need for the task, a resource kit utility may be just what you need.

Here is the scenario for our example: You need to identify whether a certain hotfix has been applied to the NT workstations that are currently online. Additionally, you'll need to gather the information from the hotfix registry key. (An assumption is made that the userid being used to run the script has administrative authority on the target machines in the queried domain.) As a modular quick-run script, the script is actually two scripts: Listing 2 shows SPRAY.BAT and Listing 3 shows HTFX.BAT.

```
REM ** SPRAY.BAT **
@echo off
REM ** Generic dynamic spray script.
REM ** Input variables.
REM ** %1 is the name of the domain to check.
REM ** %2 is the name of the batch file to spawn.

REM ** Housekeeping. Create a report directory.
```



```
REM ** If a report dir exists, clean it out.
if not exist .\report md .\report
if exist .\report\*.txt del .\report\*.txt

REM ** Create the source to use.
REM ** Assumes the common name of workstations start
REM ** With \\NT??????? for a max of 10 characters.
net view /domain:%1 | findstr \\NT > %1.txt

REM ** The loop generator.
for /f "tokens=1" %i in (%1.txt) do call %2.bat %i

REM ** Report time.
echo . > .\report\Report.txt
echo Total machines searched >> .\report\Report.txt
find /c /i "\\NT" %1.txt >> .\report\Report.txt

echo . >> .\report\Report.txt
echo Total entries found >> .\report\Report.txt
find /c /i "Report" .\report\good.txt >> .\report\Report.txt

echo . >> .\report\Report.txt
echo Total Bad connections >> .\report\Report.txt
find /c /i "failed" .\report\BadC.txt >> .\report\report.txt

REM ** Pop up the final report summary
start notepad .\report\report.txt

:End
```

Listing 7.2: The SPRAY.BAT file.

```

REM ** HTFX.BAT **
@echo off
REM ** Check for connection to the target machine
If NOT exist %1\c$\boot.ini goto :BadC

REM ** Remove the slashes from the NETVIEW generated name.
set MCHN=%1
echo %MCHN%
set MCHN=%MCHN:~2,10%

REM ** Do the work. Save the output to a file for each machine.
REM ** This allows research per machine if needed.
dumpel -l system -e 4359 -m NTServicePack -f .\report\%MCHN%.txt -s %1

REM ** Find the trigger keyword for the report
find "Q300845" .\report\%MCHN%.txt >> .\report\Good.txt

REM ** The searched for string is checked, grab the entire HOTfix list
for reference.
regdmp -m %1 "\Registry\Machine\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Hotfix"
goto :end

:BadC
REM ** Report that a NETVIEW found machine could not be connected.
echo %MCHN% connection failed >> .\report\BadC.txt

:end

```

Listing 7.3: The HTFX.BAT file.

The process is started by calling SPRAY.BAT with two operands—the domain to review and the work script to feed the machine names into. SPRAY.BAT is used as a shell for other small scripts so that you have the same front end and only need to adjust the core work script.

The NET VIEW command is used by the script to gather the list of machines available in the domain browser list for the target domain. The browser list is not 100 percent complete, it shows only machines that have contacted the browser and announced their presence. As there can be different browser masters depending on the network, using NET VIEW will show only the machines that are online and talking to the same browser master that you are. In practice, the list shows between 95 percent and 99 percent of the online machines when you do a NET VIEW statement. It depends on your network and segmentation factors.

The NET VIEW statement pipes the results to a FIND command that searches only for machine names beginning with \NT, assuming all the NT workstations are named NT*something*. The results of the FIND command are fed into a *domain.txt* file for later use. SPRAY.BAT then parses the *domain.txt* file one entry at a time, and feeds the machine name to the core work batch file called HTFX.BAT.

HTFX.BAT uses a simple test to see whether it can talk to the target machine. It checks for the existence of the C:\boot.ini file on the target machine. There are only four results possible from the check:

1. The target machine is online and reachable
2. The target machine is not online
3. The script does not have authority on the target machine



4. The target machine does not have a boot.ini file

The script only needs the first condition (online and reachable). Any other condition is not a concern of the script. Failed connections are recorded in a bad connection log file. After HTFX.BAT confirms that the target can be reached, the resource kit utility DUMPEL is run. This utility is useful and has been around since the NT 3.5 resource kit. The Win2K Server version (dated 12/99) is the one used for this example. This utility can obtain data from event logs and parse the information into different formats. This ability lets you place the event log information into spread sheets or other databases for massaging as you need. DUMPEL has options for changing the separator character, changing the output results, filtering, and selecting which log to use.

This script grabs any system events related to hotfix installs and sends the results to a text file that starts with the machine name. This information all goes into a directory called REPORT for ease of data collection. As the script is tailored to search for a particular hotfix number (Q300845), a quick search to the saved off file is done. After the FIND /C command completes the filtering, another resource kit utility is called to dump the registry for all hotfixes. The REGDMP utility (from Supplement 4 of the NT resource kit) is called in case the event logs happened to wrap and the installed hotfixes have rolled off. REGDMP is one of several REG* utilities that provide a simple means of gathering registry data or manipulating the registry contents.

After the data is gathered for the target machine, HTFX.BAT ends and returns control back to SPRAY.BAT. After the entire *domain.txt* file has been individually processed, SPRAY.BAT does a fast count of the results and presents them in the form of a notepad report. If you have a CADS, the hotfix data may already be available in any inventory report the CADS generates. Additionally, most CADS will provide methods to gather the same data dynamically from inside their processing methods.

REGDMP and DUMPEL are both resource kit utilities that were originally designed for older NT-based OSs. For this example, I ran the script from a Windows XP Pro workstation to an NT 4.0 SP6 workstation, a Win2K Server, a Win2K workstation, and to another Windows XP system. All four systems returned the proper data as expected without problems. Thus, don't limit yourself to only the resource kit for your particular OS.



You can remotely install services by using other resource kit tools: the command-line INSTSRV tool, which is great for scripting needs, or the GUI-based SRVINSTW tool, which guides you through the process.



If you need to remotely install a service, the SRVINSTW.EXE tool can do the trick. The steps are pretty easy:

1. Copy the source files for the service to the remote machine. Place them in the proper directories depending on the needs of the server.
2. Start SRVINSTW and follow the prompts. (Figure 7.17 shows the startup screen.)

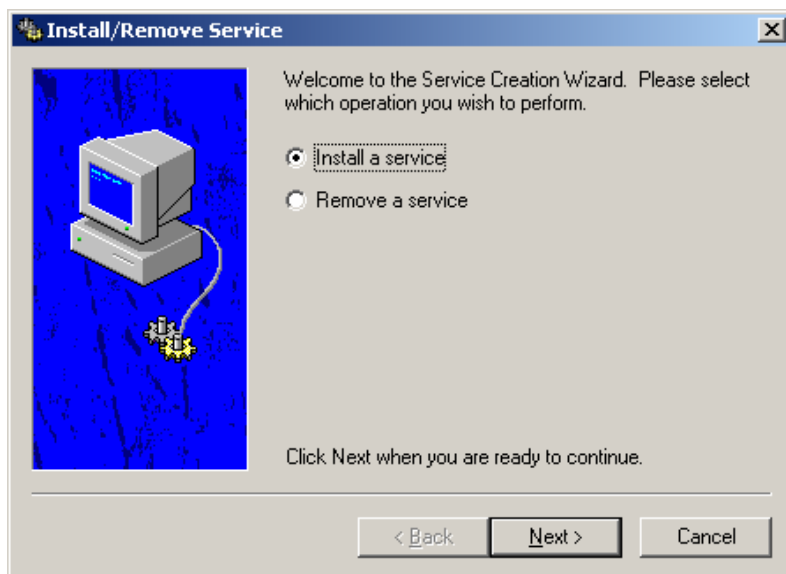


Figure 7.17: The startup screen for SRVINSTW.exe.

3. When the service asks for the source location, be sure to use the remote machine's point of view. Files placed on the remote machine's disk should be referenced by C:\, not a UNC or network connection, as Figure 7.18 shows.



Figure 7.18: Be sure to specify the proper source location from the remote machine's point of view.


4. After the wizard completes, start the service via normal remote service install methods.

Another resource kit tool that you may have already used is the Remote Command Service. RCMD is a UNIX-like command-line tool that provides access to a client. It requires installation of a service (RCMDSVC.EXE) on the client, and the service must be running on the target machine before you can start the service.

RCMD needs the RCMDSVC.EXE file copied to the target machine first, then the RCMDSVC service installed on the target machine. You can use either of the previously mentioned tools to install the service on the client if it isn't already installed. As RCMD logs you onto the machine using the userid you are currently using, your userid must have the proper access level for the target machine.

RCMD is a great scripting and troubleshooting tool. It is limited to command-line aspects only. Any command you enter from the machine you're on will run as if you were physically at the target machine. A Windows XP machine can gain access to any other Windows machine via RCMD.

For software distributions, you can use RCMD for triggering scripts for silent installs, activating AT commands, or making modifications to directories. At times you might find that performing functions on a target machine works better than attempting to do the function remotely. One example is installing software remotely when the source is closer to the target than to you. Using RCMD to gain access to the target machine, map a drive via the NET USE command to a source server, then activating the silent install can speed the process. If your install package was not part of your software deployment method and you had no other way to activate the install, RCMD is a fast solution. The goal is to have different methods to perform the needed functions.

 Be aware that RCMD is a little buggy and there are various versions out there. Commands entered via RCMD must run without user interaction on the target machine or the machine hangs. If a command calls up a GUI window, you will never see the window and the RCMD session is hung until the window is closed. That same hang condition happens if a pop-up window, such as an error message, happens. In scripting, you need to test the logic as RCMD might handle things a little differently than you expect.

The big consideration to remember with RCMD is to execute the commands locally with the local drive mapping logic. Because the RCMD session will not appear to the user, you can come in under the covers and execute PSTAT or TLIST or KILL commands for troubleshooting processes, but you can't see what the user is seeing. It is not a remote-control service, just a remote execution. The overhead for RCMD is lower than for remote-control services.

When scripting, you can copy a batch file to a known location on the target machine, then have your script call the batch file as if you were on the system. The command

```
rcmd \\machine c:\temp\run.bat
```

will connect to *machine*, execute the c:\temp\run.bat command and return focus back to your machine to continue on with your script. This process is a down-and-dirty method to make simple changes or activate processes on remote machines.

Resource Kit tools are not just focused on the distribution world. You can find utilities for managing AD, monitoring the network, monitoring API calls, verifying your cluster, manipulating the registry, manipulating services, and gathering hardware data. As I mentioned earlier, there are more than 300 utilities in just the Win2K Server resource kit. Instead of trying to cover all the utilities from all the resource kits, I've listed some of my favorites that have proven useful at one point or another within my software-distribution experiences (see Table 7.1).

Utility	Comments
APIMON	Monitor API calls; great for troubleshooting
DELSRV	Delete services installed on clients
DIRUSE	Obtain data on a directory structure; great for checking for size limits when space is an issue
DRIVERS	Locate data on installed drivers; great for pre-checking systems when preparing packages
DUREG	Check the size of registries—not just the space, but the actual data size
EXTRACT	Expand cab files
GETSID	Obtain the SID of two different userids
GPRESULT	Determine the impact that a GPO has on a userid
HEAPMON	Troubleshoot unknown issues or occasional hangs in processes or systems
IFMEMBER	Use in logon scripts for group membership branch logic
INUSE	Replace system reserved files great if you create your own installers
NOW	Determine what time is it NOW
OH	Determine what handles are open; great for troubleshooting
OLEview	Check out OLE and COM stuff
PSTAT	Determine how many processes are running on a system; this utility helps locate problems
RPCDUMP	Check the RPC side of the network
RCMD	Get console access to a remote machine; has some limitations but works well for most command-line needs
Robocopy	Mirror and filter very quickly; only copies what it needs to (if the file exists as you directed, it skips copying it over); great copy utility
SCOPY	Copy the security aspects of directories and files
SOON	Run a process inside the next 24 hours (like the AT command)
TIMETHIS	Determine how long it took to run a process
Whoami	Check a userid prior to running a script
XCACLS	Modify security parameters for directories; good for new machines or resetting older machines to standards

Table 7.1: Some useful resource kit utilities.

Summary

It is good to have a standard deployment method and to use it as much as possible. Being aware of other options and possibilities expands your scope and abilities to perform those one-off jobs that creep up once in awhile. Many of the tools needed to do simple distributions or data gathering are available free on the Web.

Although every deployment method has its own interface and rules for use, Windows Installer has an increasing presence in the deployment field. As a result, even if you don't use it or plan to, you should still be familiar with it and how it works. Commercially available applications may only provide setups within the Installer framework, and understanding the technology will assist you in creating the application package and deployment design that best fits your environment.

Resource kits are a wealth of tools that are needed in every administrator's bag of tricks. The documentation may not be the best for all the tools and not every tool will fit into your environment. However, the little gems that you find will quickly become a staple for your daily administrative functions.

Copyright Statement

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

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

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

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

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