

ITProTM
SERIES

WindowsITPro

 **eBooks**

Keeping Your Business
SAFE from Attack:

Patch Management

By Jeff Felling

Microsoft[®]



Microsoft®

Contents

Chapter 4 Microsoft Patching Technologies	52
Decoding a Software Patch	53
Discovering the Installer Version	53
How the Patch Installs	55
Microsoft's Most Common Patch Engines	60
Update.exe	60
Hotfix.exe	65
Ohotfix.exe	66
Normal Updates and Administrative Updates	67
Normal Updates	67
Administrative Updates	68
Integrating Office Patches into the Install Sources	70
Obtaining Ohotfix.exe	71
Dahotfix.exe	71
Off the Beaten Track: Older and Unique Update Engines	71
Vgxupdate.exe	71
Iexpress	72
Installing Multiple Hotfixes with Qchain Technology	72
Installer Wrap-Up	72

Chapter 4:

Microsoft Patching Technologies

When you download a patch from the Microsoft Web site and run it, you are running an installer application configured to install the patch. Today, Microsoft relies on several different patching engines to apply security updates to its products, although the company is making an effort to reduce this number. Depending on the type of patch management software that you use, you might find yourself having to work with these different types of installers. For example, you might need to create custom deployment scripts such as logon scripts to install patches, you might need to customize deployment packages of a product like Microsoft Systems Management Server (SMS), or you might just find yourself needing to test a new patch or troubleshoot a failed deployment. Understanding how these different installers work will make it much easier to do tasks like these.

Almost every Microsoft patch is wrapped up into a self-extracting executable that you download from the Microsoft security Web site, then run on the target system to install the patch. However, depending on the patch installer chosen to package the patch, you might be able to customize the deployment, such as quietly deploying the patch without user interaction or suppressing a reboot after the patch has been installed. To take advantage of these custom installation options, you will need to know the type of patch installer that was used to create the patch, as well as the command-line switches for that specific installer. Many patch management software packages absolve you from worrying about these details, but even if you use a sophisticated product like SMS to deploy your patches, you might find yourself needing to learn the parameter syntax for the different installers to create specialized patch deployment packages.

This chapter is not intended to promote manual patching in lieu of using patch management software. Due to many patch installers, the increasing frequency of patch releases, and the variety of OS platforms and software versions that must be supported, using robust patch management software is a must. Also, patching a computer system is not necessarily a one-time process. When you install a new software component or if Microsoft updates the patch, you might need to reapply the patch. For example, if you install Microsoft IIS, apply several IIS security updates, and later uninstall and reinstall IIS, you will need to reinstall the related security updates. Plus, knowing which patches relate to each component is difficult. So when you install a new component, rescanning the entire system for missing patches is a good idea. Use a full-featured patch management software that fully scans the files installed on a system, as opposed to simply checking the registry for installed patches. This way, through regular scanning, you can ensure that all systems are up to date with the latest security updates.

This chapter takes a look at Microsoft's most popular patch installer engines and their command-line syntax to explain how to use these programs for customized deployments.

The technologies and techniques discussed in this chapter make up the core essence of installing Microsoft updates which consists of

- Downloading a wrapper that includes the installer and the patched files
- Decompressing the files and starting the patch installer
- Replacing the vulnerable files with the updated files

Historically many of the major Microsoft product families (such as Windows, Office, SQL Server, Internet Explorer (IE), and Windows Media Player—WMP) used different patch installer engines, which made using a single method to deploy the patches difficult. Today Microsoft uses the update.exe patch installer for packaging its Windows and IE security updates and service packs and OHotFix for deploying Office updates. You will encounter other installers if you support SQL Server or older platforms such as Windows NT 4.0.

Decoding a Software Patch

Most Microsoft security updates consist of a software patch to correct an identified vulnerability in the original code. After Microsoft releases a new security update and you have deemed it appropriate to your environment, you need to do a few things to install the software patch. To begin you must download the appropriate software patch from the Microsoft Web site to a computer on your network, then copy the file to the target computer that you want to install the patch on. These files are almost always self-extracting executables. This means to install the patch you simply run the patch file by double-clicking it or executing it from a script or command line. Running the patch file from the command prompt with additional command-line parameters lets you control the patch installation, for example suppressing reboots or installing quietly in the background without user interaction. Most patch management software deployment programs use Microsoft's original patch files and many use a combination of these command-line parameters to deploy the software.

Discovering the Installer Version

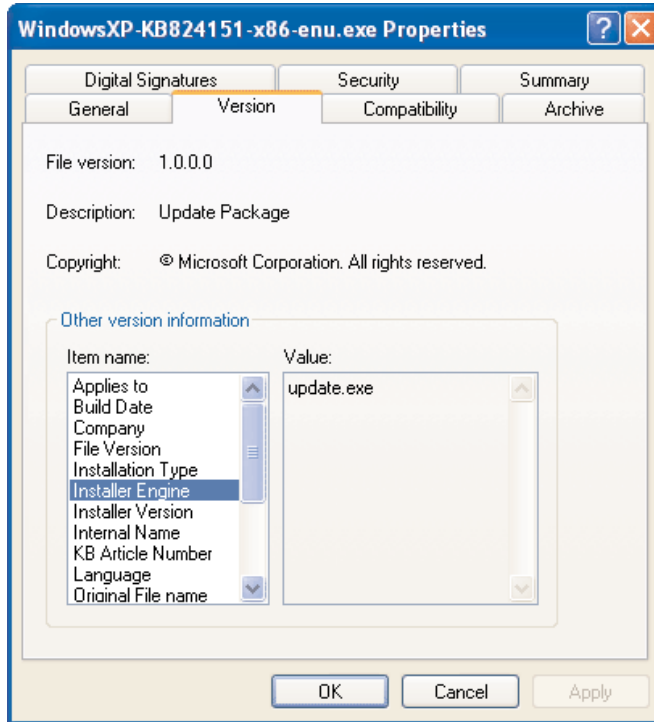
You can discern a lot about the patch from its name. The name might even help you determine which patch installer a particular update uses. You can identify patches created using the update.exe installer by their consistent naming convention. For example, when you download the Microsoft security update MS04-030 for Windows XP, you download a file named *WindowsXP-KB824151-x86-enu.exe*. The name is delimited by dashes (-) into four fields:

- The first field is the product name.
- The second field is the name of the Microsoft Knowledgebase article that describes the vulnerability that this patch fixes.
- The third field contains the name of the platform on which the patch is compiled.
- The fourth field is the language version of the patch.

In addition to the name, you might also be able to discover the patch installer type from its file properties. Select the patch, open its properties dialog box, click the Version tab, and select the Installer Engine, as Figure 4-1 shows.

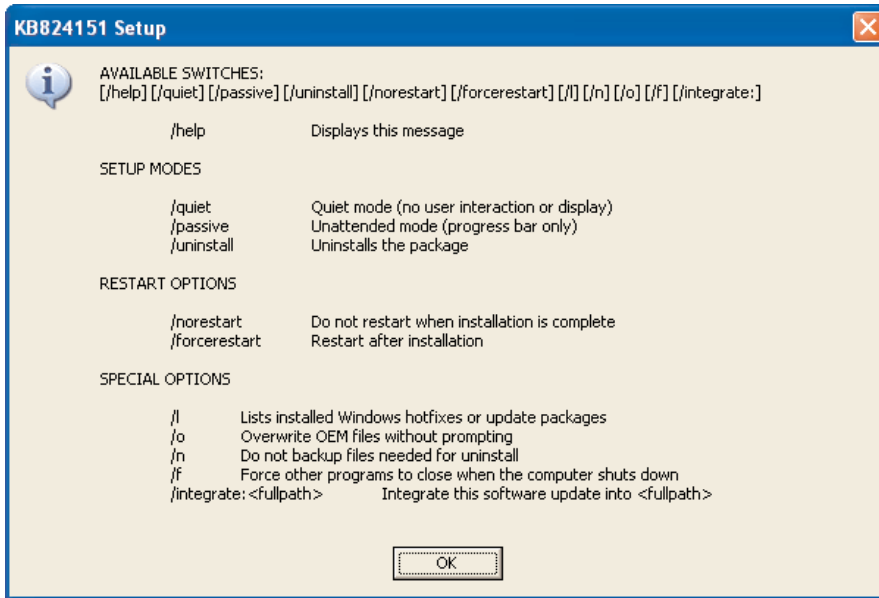
Figure 4-1

Viewing patch properties to discover its installer type



Here you can see that the Installer Engine is *update.exe*. Another less direct way to tell the installer type is to start a command window and run the patch with the */help* parameter, as Figure 4-2 shows.

Figure 4-2
Viewing the `/help` parameter and other available switches



Although this method does not tell you the name of the installer, it might provide the syntax of the supported parameters, from which can infer the installer used with that patch.

How the Patch Installs

The patch software file that you download is a self-extracting executable that wraps the patch installer together with the patched files. When you run this program, the wrapper decompresses the patched files and starts the patch installer. The wrapper accepts optional command-line parameters that it passes through to the installer program. For example, to install the MS04-030 XP patch quietly and suppress the reboot you can execute the wrapper

```
WindowsXP-KB824151-x86-enu.exe /quiet /norestart
```

When you execute the wrapper, it decompresses the patch files into a temporary folder on your hard drive. The location of these files depends on the patch installer engine used. For example, the patch associated with the security bulletin MS04-030 uses the update.exe installer that creates a new directory on the primary partition of your hard disk and names the folder a random string (e.g., `c:\aecb766510779209e2087587e1838a`).

After the wrapper decompresses the patch files, it launches the patch installer. Depending on the version of the installer, it might perform patch applicability checks before installing the patch. For example, the update.exe installer checks the date of the patch against the date of previously installed service packs. If the patch was released after the service pack, the installer will install the patch. Otherwise the patch installation will quit.

The wrapper decompresses the patch files into subdirectories depending on the patch installer. Figure 4-3 shows an example of the directory structure for the XP security update MS04-030, which uses the update.exe installer.

Figure 4-3

Showing the directory structure for the MS04-030 security update

```

C:\WINDOWS\system32\cmd.exe
C:\aebc766510779209e2087587e1838a>dir /S
Volume in drive C has no label.
Volume Serial Number is E8D8-35BE

Directory of C:\aebc766510779209e2087587e1838a

11/06/2004 11:28 AM <DIR>      .
11/06/2004 11:28 AM <DIR>      ..
11/06/2004 11:28 AM <DIR>      rtmqfe
11/06/2004 11:28 AM <DIR>      sp1qfe
07/20/2004 02:02 PM           7,168 spmsg.dll
09/19/2004 12:22 PM        169,984 spuninst.exe
11/06/2004 11:28 AM <DIR>      update
                2 File(s)          177,152 bytes

Directory of C:\aebc766510779209e2087587e1838a\rtmqfe

11/06/2004 11:28 AM <DIR>      .
11/06/2004 11:28 AM <DIR>      ..
11/06/2004 11:28 AM <DIR>      ip
08/26/2004 10:39 AM    1,172,992 msxml3.dll
09/18/2004 09:31 PM         30,720 xpsp1hfm.exe
                2 File(s)          1,203,712 bytes

Directory of C:\aebc766510779209e2087587e1838a\rtmqfe\ip

11/06/2004 11:28 AM <DIR>      .
11/06/2004 11:28 AM <DIR>      ..
08/26/2004 10:39 AM    240,640 httpext.dll
                1 File(s)           240,640 bytes

Directory of C:\aebc766510779209e2087587e1838a\sp1qfe

11/06/2004 11:28 AM <DIR>      .
11/06/2004 11:28 AM <DIR>      ..
11/06/2004 11:28 AM <DIR>      ip
08/26/2004 10:54 AM    1,172,992 msxml3.dll
                1 File(s)          1,172,992 bytes

Directory of C:\aebc766510779209e2087587e1838a\sp1qfe\ip

11/06/2004 11:28 AM <DIR>      .
11/06/2004 11:28 AM <DIR>      ..
08/26/2004 10:54 AM    242,688 httpext.dll
                1 File(s)           242,688 bytes

Directory of C:\aebc766510779209e2087587e1838a\update

11/06/2004 11:28 AM <DIR>      .
11/06/2004 11:28 AM <DIR>      ..
09/19/2004 09:47 AM           390 branches.inf
07/20/2004 02:40 PM           4,092 eula.txt
09/24/2004 01:15 AM    11,484 KB824151.CAT
09/19/2004 12:22 PM    21,504 spcustom.dll
07/20/2004 02:02 PM    654,336 update.exe
09/24/2004 01:18 AM           444 update.ver
09/24/2004 12:27 AM           485 updatebr.inf
09/24/2004 01:07 AM           5,338 update_RTMQFE.inf
09/24/2004 01:07 AM           7,305 update_SP1QFE.inf
                9 File(s)           705,378 bytes

Total Files Listed:
        16 File(s)      3,742,562 bytes
        17 Dir(s)      20,520,853,504 bytes free

C:\aebc766510779209e2087587e1838a>

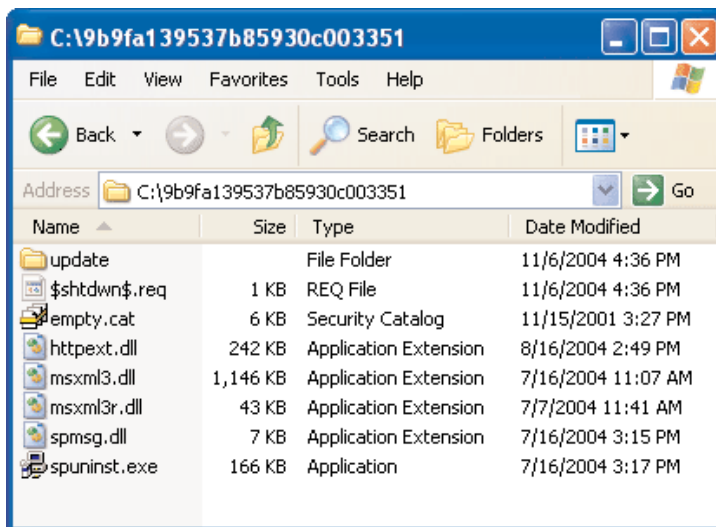
```

The *update* directory contains the patch installation engine *update.exe* and its support files. This patch also includes two patch specific folders, *rtmqfe* and *splqfe*, which contain the new patched software files. With Windows Server 2003 and XP Service Pack 2 (SP2) patches, you will more frequently see the folders with the nomenclature GDR, which stands for general distribution release. The GDR and Quick Fix Engineering (QFE) folders (among others) can coexist within one patch under a multibranch-aware file structure. This method allows multiple installation scenarios in the same package. GDR files represent security updates as released through Windows Update and QFE represent hotfixes released by Microsoft Product Support.

These file names and versions will be the same files listed as affected files in the Security Bulletin for that patch. (More information about decoding Microsoft Security Bulletins is in Chapter 2.) Continuing to use MS04-030 as an example, there are two files that need updated: *msxml3.dll* and *httpext.dll*. Notice that both the *rtmqfe* and *splqfe* folders contain the same files, but that the *httpext.dll* is a different size. This means that the release to manufacturing (RTM) version of XP requires a different version of the update than a XP SP1 version. Imagine how many versions of *httpext.dll* must be updated, tested, and tracked by Microsoft for all the different OS platforms, processor platforms, and languages—and this is for only one update!

The Windows 2000 (Win2K) patch for the same security update also uses the *update.exe* engine but contains only one folder (*update*) in addition to the patch files, which in Figure 4-4 shows. It requires only one folder because the Win2K version does not provide multibranch support. For most organizations, simply downloading the patches from the Windows Update Web site will suffice. However, if your organization is running specifically one version (QFE or GDR), then you will want to work with your Microsoft support team to ensure that you continue to receive the proper versions of your updates.

Figure 4-4
Viewing the update folder and files for a Win2K patch



Deploying this update requires quite a bit of overhead. Uncompressed, the two crucial files in this example patch (*msxml3.dll* and *httpext.dll*) together are about 1380KB in size. But this entire update takes up about 3560KB of space. This package is over two-and-a-half-times the size of the files that make up the patch, with the bulk coming from two slightly different versions of the same files plus the patch installer. Microsoft compresses this entire patch directory into one self-extracting executable to reduce its size to around 942KB, but this package size can be reduced even more. In the future, Microsoft might include the patch installer with the base Windows OS and remove it from each patch.

In addition to the patch installer and patch files, the wrapper usually contains a configuration file for the installer. In the previous example for MS04-030, notice that the update directory contains the patch engine executable *update.exe* and two configuration files named *update_RTMQFE.inf* and *update_SP1QFE.inf*. Ordinarily you'll find one *update.inf*, but because this wrapper patches two different XP builds, Microsoft included two configuration files—one for each build. Figure 4-5 shows a sample configuration file for the update.exe installer.

Figure 4-5
Showing a configuration file for the update.exe installer

```

update_RTMQFE.inf - Notepad
File Edit Format View Help

[[Version]
    Signature="$Windows NT$"
    NtBuildUpdate=2600
    MaxNtBuildUpdate=2600
    NtMajorVersionToUpdate=5
    NtMinorVersionToUpdate=1
    MaxNtMajorVersionToUpdate=5
    MaxNtMinorVersionToUpdate=1
    MinNtServicePackVersion = 0
    MaxNtServicePackVersion = 0
    LanguageType=%LangTypevalue%
    ThisServicePackVersion = 0
    CatalogFile = %SP_SHORT_TITLE%.cat

[ProductCatalogsToInstall]
    %SP_SHORT_TITLE%.cat, update\%SP_SHORT_TITLE%.cat

[ProductInstall.ReplaceFilesIfExists]
    CopyFiles=System32.Files
    CopyFiles=Cache.Files

[ProductInstall.CopyFilesAlways]
    CopyFiles=CopyAlways.System32.Files

[ProductInstall.ProfessionalFiles]
    CopyFiles=Professional.Inetsrv.Files
    CopyFiles=Professional.Cache.Files

[ProductInstall.GlobalRegistryChanges.Install]
    AddReg=Product.Add.Reg

[ProductInstall.GlobalRegistryChanges.ReInstall]
    AddReg=Product.Add.Reg

[Save.Reg.For.Uninstall]
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%
    HKLM,SOFTWARE\Microsoft\Updates\Windows XP\SP%SERVICE_PACK_NUMBER%\%SP_SHORT_TITLE%

[Product.Add.Reg]
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Installed", 0x10001, 1
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Comments", 0, %SP_TITLE%
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Backup Dir", 0, ""
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Fix Description", 0, %SP_TITLE%
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Installed By", 0, ""
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Installed on", 0, ""
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Service Pack", 0x10001, %SERVICE_PACK_NUMBER%
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%, "Valid", 0x10001, 1
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%\File 1\, "Flags", 0, ""
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%\File 1\, "New File", 0, ""
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%\File 1\, "New Link date", 0, ""
    HKLM,SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix\%SP_SHORT_TITLE%\File 1\, "Old Link date", 0, ""
    HKLM,SOFTWARE\Microsoft\Windows\CurrentVersion\Setup, "LogLevel", 0x10001, 0

[DestinationDirs]
    System32.Files=11 ; %windir%\system32 (replace if exist)
    CopyAlways.System32.Files=11 ; %windir%\system32 (copy even if don't exist)
    Cache.Files=65619 ; %windir%\system32\Dllcache (replace if exist)
    Professional.Cache.Files=65619 ; %windir%\system32\Dllcache (replace if exist)
    Professional.Inetsrv.Files=11, "inetsrv"

[CopyAlways.System32.Files]
    xpsp1hfm.exe, RTMQFE\xpsp1hfm.exe

[System32.Files]
    msxml3.d11, RTMQFE\msxml3.d11

[Cache.Files]
    msxml3.d11, RTMQFE\msxml3.d11

[Professional.Cache.Files]
    httpext.d11, RTMQFE\fp\httpext.d11

[ArchiveCatalogFilesOnly]
    %SP_SHORT_TITLE%.cat

[SourceDisksNames]
    1=ServicePackSourceFiles%

```

This file contains all the instructions on how update.exe should install the patch, including registry keys to update and file locations to copy the new files.

The configuration file also tells you where in the registry the update installation will be recorded. In the HKEY LOCAL MACHINE hive under the Software, Microsoft, Windows NT, CurrentVersion, Hotfix key, Microsoft adds the KB article number (in this case KB824151) for the patch. Browse to

this registry key on your own computer and review the list of patches that have been applied. Avoid lower-function patch management software that merely queries this registry key to see which patches are applied. Instead choose a robust patch management software that compares the date and size (or checksum) of an installed file with the patched file to ensure that the updated file is installed. This approach of checking the file is important as the following example explains.

Let's say you apply this patch to a computer and its registry is updated to reflect this update. Later you install (or reinstall) a component like IIS that copied files from the Windows source CD-ROM. The files copied from the source CD-ROM might overwrite existing patched files. However, the registry would not be updated to reflect this, because the IIS installation does not know about the hotfix. Subsequent scans by patch management software that only verify this key would erroneously report that the patch is indeed installed.

Microsoft's Most Common Patch Engines

The most common Microsoft patch engines are `update.exe` and `ohotfix.exe`. You might also encounter older or other product specific patch installers such as `hotfix.exe`, `dahotfix.exe`, `IExpress`, and even one-off derivatives such as `vgxupdate`. The following sections briefly describe each of these installers.

Update.exe

`Update.exe` is currently Microsoft's preferred software patch installation engine to install patches for Windows 2003, XP, and Win2K OSs, Microsoft Exchange Server, and IE. `Update.exe` can add or delete files, registry keys, and back up files before patching them. `Update.exe` supports single file deployment to the deployment of hundreds of files from a service pack. For example, Microsoft uses `update.exe` to install XP SP2.

An update consists of three key parts. The first part is the installer application named *update.exe*. To perform the patching, this 650KB program updates the registry and copies the updated files. The second part is the configuration file, commonly named *update.inf*, which tells `update.exe` how to install the patch and where to locate the files. The final part consists of the updated files to install on the target system.

`Update.exe` supports different parameters to customize the patch installation. Figure 4-2 shows the supported `update.exe` parameters that you can invoke through the patch wrapper executable. As Figure 4-2 summarizes, running

```
WindowsXP-KB824151-x86-enu.exe /quiet /norestart
```

instructs the installer to quietly install the patch and not to restart to computer when finished.

A summary of these parameters follows:

`/help`

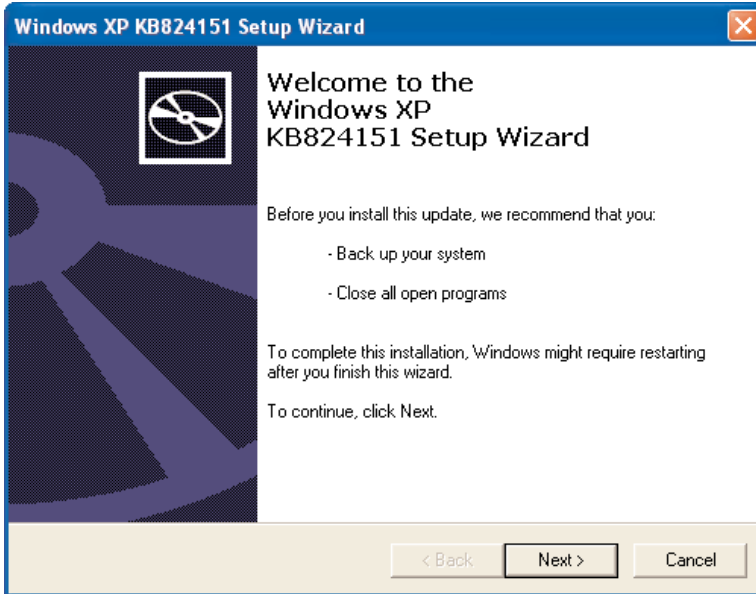
Displays a help dialog box listing the parameters supported by the patch installer.

`/quiet` or `/Q`

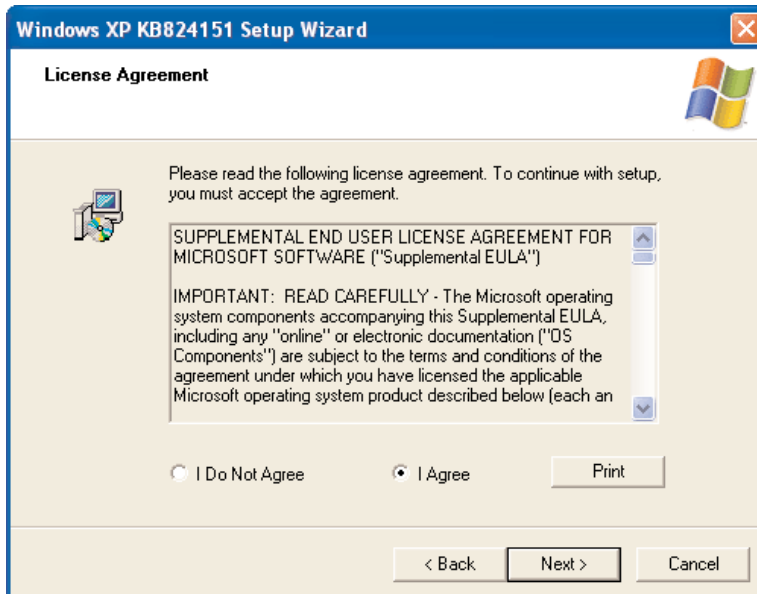
When you simply double-click a downloaded patch executable to begin installation, it might recommend you back up current files or ask you to present an End User License Agreement (EULA), as Figures 4-6 and 4-7 show.

Figure 4-6

Viewing the wizard's recommendation to back up current files

**Figure 4-7**

Viewing the patch's EULA



62 Keeping Your Business Safe from Attack: Patch Management

Executing the path with the `/quiet` parameter suppresses these prompts. This option is useful when you want to deploy the patch in the background as a part of a logon script or other action and you don't want to interfere with the current user of the system.

`/passive or /U`

Similar to the quiet parameter, specifying the passive parameter instructs the patch engine to deploy the patch without user intervention. However, whereas quiet suppresses all output, passive still shows a progress bar notifying the user of installation progress. This parameter can come in handy when you are deploying larger patches and want to keep your users apprised of the progress. After you execute a patch installation with the `/passive` parameter, the user will see dialog boxes informing them of the stages of the patch installation, including backing up files, copying new files, finalizing the patch installation, then following with an immediate reboot if needed. The user will not be prompted for any interaction during a passive installation.

`/uninstall`

The uninstall parameter uninstalls the patch, if possible. Not all patches can be uninstalled. You can also remove an update from the Control Panel Add or Remove Programs applet. (On a pre-XP SP2 you will see the updates listed alongside your other programs. With XP SP2, Microsoft hides the list of updates and you must enable the checkbox Show updates to display the updates.) From this applet, select the update you want to remove and click the Change/Remove button.

`/norestart or /Z`

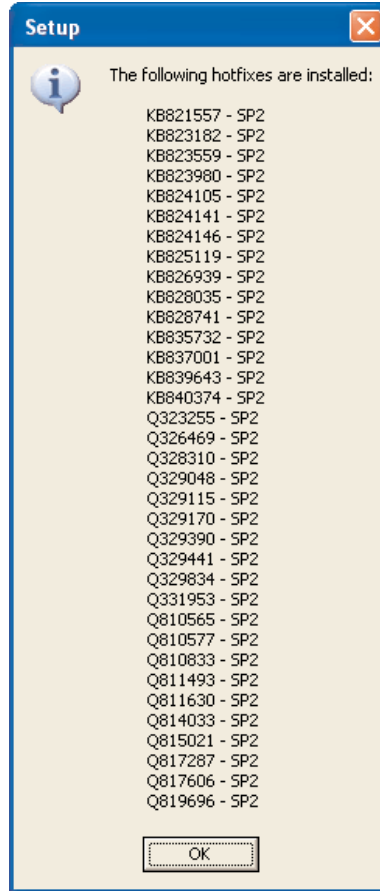
A patch might or might not require a restart after installation. Specify the `/norestart` parameter and the system will not restart after the patch is installed. Patches released since May 2002 include Qchain functionality; you can use this parameter to automatically suppress a restart. The end of this chapter discusses using Qchain technology to install multiple patches between reboots. When you use this parameter, remember that some patches require the computer to be restarted for the patch to take affect, so if you suppress the restart during the patch installation, don't forget to restart the computer soon after.

`/forcerestart`

The `/forcerestart` parameter restarts the computer after the patch installation regardless how the patch configuration file specifies a reboot.

`/L`

The `/L` parameter lists all the installed patches on a computer as recorded in the registry, which Figure 4-8 shows.

Figure 4-8*Displaying all the installed patches on a computer*

Be aware that this example only provides a query of the registry for installed patches; you should not regard this list as a guarantee that the patched files are installed. Instead, you need to use a patch management program that checks the file information against a database of known patches to see whether the patches are indeed installed.

/O

Many times computer hardware vendors write specialized drivers for their hardware that supersedes builtin Windows drivers. For example, a laptop vendor might include custom network adapter or USB drivers. Windows keeps track of the source of these OEM files, and when you install a patch that wants to replace one of these OEM drivers, Windows might ask whether to replace the OEM file with the patched file. The /O parameter instructs the installer to overwrite any OEM files it encounters with the new patched files without prompting the user.

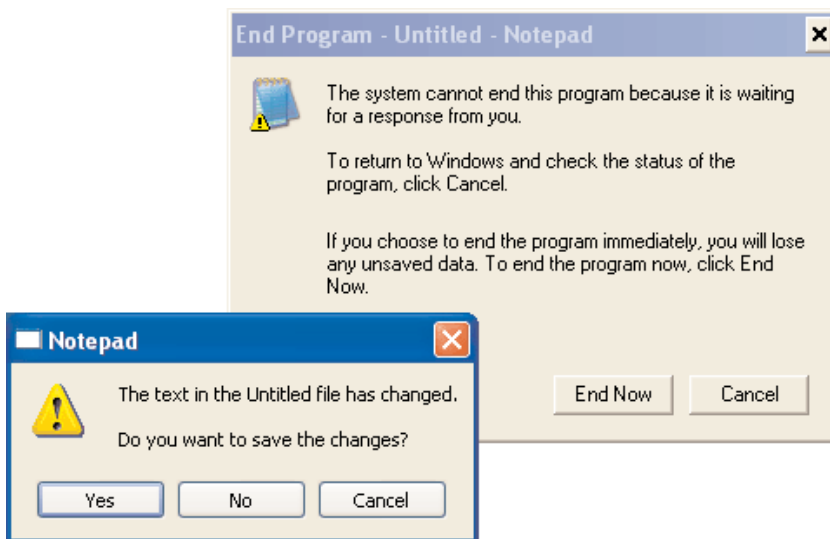
/N

The /N parameter prevents the patch update engine from backing up the original files it replaces. Although this parameter saves disk space and decreases the time of the update installation (especially for large service pack installations), it disables your ability to uninstall the patch.

/F

During a computer restart Windows prompts the active user to save any open documents. Sometimes these prompts interrupt the restart process and leave the system at a prompt waiting for someone to click Yes to save the open document, No, or Cancel, as Figure 4-9 shows.

Figure 4-9
Prompting the user to save file changes



Specify the /F parameter to force Windows to close any open programs. Note that the user will lose any unsaved work on the computer if the open applications are forced closed.

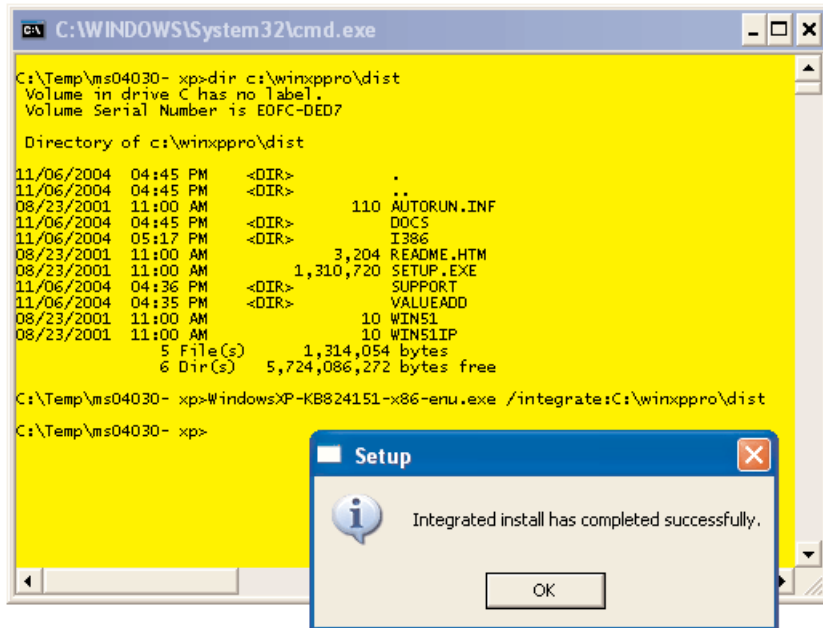
/integrate:<fullpath>

Most of the time you will be installing patches on individual computers in your environment. But when you use the original Windows installation files (available on the Windows CD-ROM or the i386 directory from a network share) to build out a new computer, it will not be patched with any updates. The integrated parameter lets you install a patch into a Windows source file directory. This means that any computer built with these files (or later installations of optional components that use this source) will use the patched files. To use the integrated switch, you need to point to the source root folder for the Windows installation files (e.g., the folder immediately before the i386 folder). For example, if your XP source files are at c:\winxpro\dist, then you use the command

```
WindowsXP-KB824151-x86-enu.exe /integrate:c:\winxpro\dist
```

The installer copies the necessary files and upon completion displays a success dialog box, which Figure 4-10 shows.

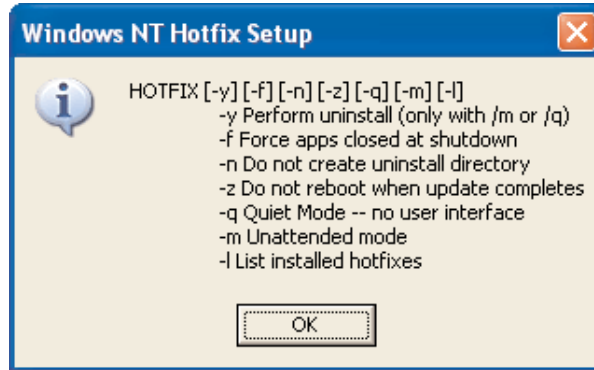
Figure 4-10
Signaling the successful installation of patch files



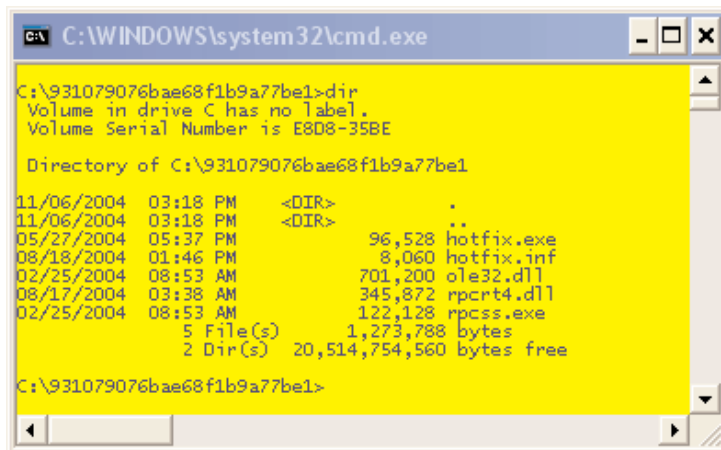
Update.exe is the predominate installer for the latest patches. However if you support earlier platforms like NT 4.0, you will still encounter other patch installers like hotfix.exe.

Hotfix.exe

The predecessor to update.exe is a program called hotfix.exe, which is still used for NT 4.0 patch deployment. New patches for NT 4.0 reflect the update.exe naming convention (e.g., WindowsNT4Server-KB873350-x86-ENU.exe) and can consistently be decoded. But if you run the /help parameter on this patch, you'll see that it uses the hotfix engine, as Figure 4-11 shows.

Figure 4-11*Using the /help parameter to reveal the hotfix patch installer*

The decompressed patch tells the whole story, as Figure 4-12 shows. Here you can see the installer engine *hotfix.exe*, its configuration file *hotfix.inf*, and the three updated files. The *hotfix.inf* file looks a lot like the *update.inf* file that *update.exe* uses. The *hotfix.inf* file tells how to install the patch files and how to update the registry with any patch related information. The temporary directory location and patch installer parameters for the hotfix installer are similar to those of *update.exe*.

Figure 4-12*Decompressing the patch to reveal its patch installer information*

***O*hotfix.exe**

Microsoft uses *ohotfix.exe* to deploy patches to Microsoft Office products. *OHotfix* differs from *update.exe* and *hotfix.exe*. Whereas *update.exe* and *hotfix.exe* install the patches, *ohotfix.exe* only brokers the installation of a patch for the target computer. *Ohotfix.exe* relies on Windows Installer patch files (designated by the *.msp* file extension) to install the updated files. A benefit of this system

is that to scan and install multiple patches at one time, you can use `ohotfix.exe` to reference a folder containing several `.msp` files. OHotFix will scan the target computer and apply only the necessary patches. `ohotfix.exe` offers no command-line parameters. All the instructions for how to use `ohotfix.exe` are contained in the configuration file `ohotfix.ini`, which is well documented. In this file you can specify how to log the installation, whether to show the OHotFix UI or suppress it (quiet mode), how to handle reboots, and other patch-related settings. But first, let's look at the two different ways to run `ohotfix.exe`-based patches.

Normal Updates and Administrative Updates

OHotFix supports two classes of updates that it calls Normal Updates and Administrative Updates. Normal Updates are similar to updates installed with `update.exe` and consist of installing one patch on a target system. Administrative Updates are a centralized method for applying multiple patches to a target system. First, let's examine Normal Updates.

Normal Updates

You needn't do anything special to deploy OHotfix updates on a single system beyond executing the downloaded patch on the target system. Like other patching engines, the Microsoft Office patches are contained in a self-extracting executable that wraps the OHotFix installer and the patch files. When you execute the wrapper, the patch files are decompressed to the user's temporary directory. (To find your temp directory, you can issue a `set` command from the command prompt or in the Start, Run box type in `%temp%`. In this directory, the patch installer creates a directory named `IXP000.TMP` that contains the patch engine `ohotfix.exe`, its configuration file `ohotfix.ini`, a helper DLL, and the patch. The patch is not a folder containing the updated files as with `update.exe`. Instead, the individual updated files are packaged into a Windows Installer patch file format with a `.msp` extension. The `.msp` extension works with the Windows Installer system (previously known as Microsoft Installer—MSI). After decompressing the files, the original wrapper executable runs the `ohotfix.exe` program which scans the newly created patch directory for any `.msp` files and installs them when applicable.

The OHotFix patches support several command-line switches, which Figure 4-13 shows, and are described as follows:

/Q (set to quiet mode)

Execute the patch from a command line or script with the `/Q` parameter and the patch will be extracted and installed quietly without user interaction.

/T:<fullpath> (specify a temporary working directory)

You can specify the temporary working folder where the patch will be uncompressed. By default this folder is `%temp%\IXP000.TMP`.

/C (to uncompress the files only)

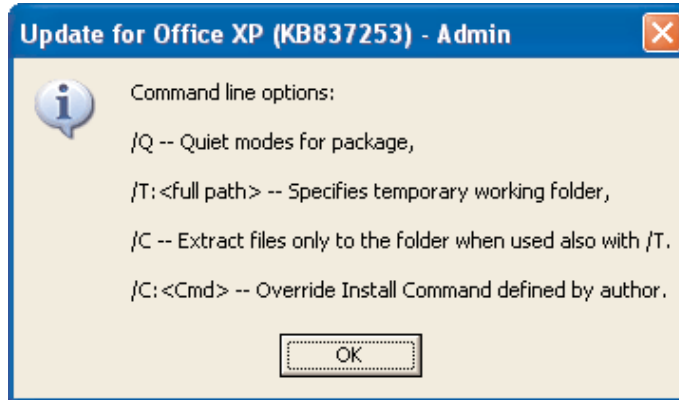
This will extract the files only to the path specified with the `/T` switch (which you must also use). The patch will not be installed.

/C:<Cmd>

This parameter lets you override the Install command defined by the patch author.

Figure 4-13

Displaying command-line switches that OHotFix supports



This two-tiered operation opens up administrator managed deployment opportunities not available with other Microsoft patch engines. Let's take a look at the possibilities. For example, for one Office update Microsoft provides two files named

```
Office2003-kb838905-client-enu.exe
Office2003-kb838905-fullfile-enu.exe
```

The naming convention is similar to update.exe except for the third field that contains the descriptor *client* and *fullfile*. The Microsoft Office update Web page describes the *client* file as the installer of choice for basic installations. The process for installing the client patch is straightforward: simply download and double-click the executable and it will install the patch in one swift action.

However, the *fullfile* program is targeted at administrators. When you run this program, it prompts you to specify a target directory to decompress the files. Inside this directory is one .msp file (in our example, the downloadable wrapper file *Office2003-kb838905-fullfile-enu.exe* decompresses to a single file named *gdipplus-FullFile-GLB.msp*.) The wrapper executable exists without attempting to install this file or starting ohotfix.exe. This feature is for administrators and will be described in more detail in the next section.

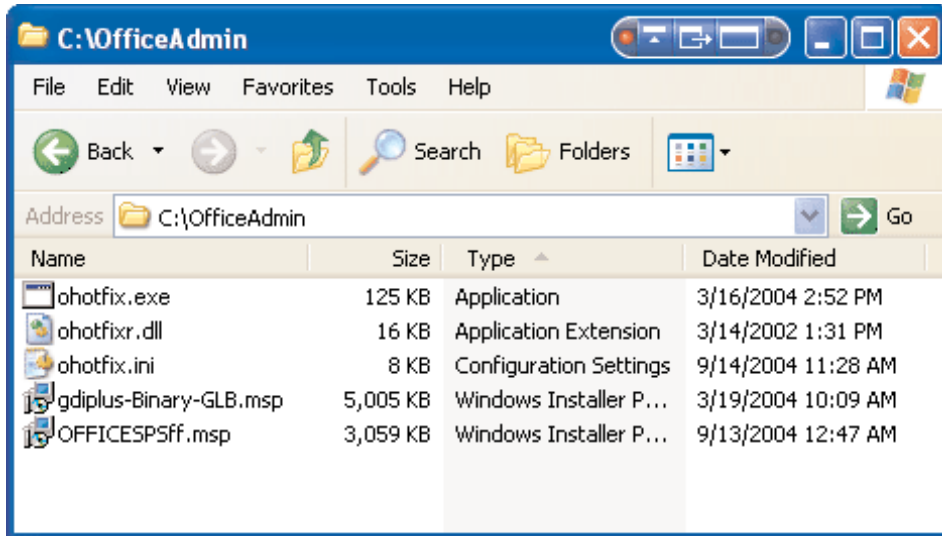
Administrative Updates

You can use OHotFix to deploy multiple Office updates from a centralized location for Administrative Updates. Copy multiple .msp files into a commonly accessible directory, then use a logon script or other means of running ohotfix.exe on each target system. OHotFix will run with the configuration specified in ohotfix.ini and will scan the target system for applicability with each .msp file. If OHotFix determines that a patch is needed, it will attempt to run the .msp file which will install the patch on the target system. OHotfix logs the installation history and you can specify the verbosity of the logging in the ohotfix.ini configuration file. By default OHotFix logs status messages to %temp%\ohotfix.

For example, Figure 4-14 shows an example of a folder containing OHotfix and two .msp files representing two different patches to Microsoft Office.

Figure 4-14

Viewing a folder containing OHotFix and .msp patch files



Anytime you run OHotfix from this folder it will attempt to install both of these patches. However before it runs the patch, it scans the target system to see whether the relevant Office product is installed or whether the patch has already been installed. When OHotFix runs, nothing might appear to happen but if you review the OHotFix logs, similar to Figure 4-15, you'll see that both patch files were evaluated and rejected because the software was not installed or because a newer patch was installed. In this manner you can copy new .msp files into this folder and anytime OHotFix is run it will sweep the directory and execute the .msp files it finds. No manual updating of log files is required.

Figure 4-15
Reviewing the OhotFix logs

```

OHotfix(00001).log - Notepad
File Edit Format View Help
Starting the log on 11/06/2004 at 19:34:08...
Loading the resource .dll...
Reading the settings...
The settings are:
  IsNormalUpdate=1
  AdminusesShortFileNames=1
  AdminPath=""
  MessageTitle="Update for office XP (KB837253)"
  CopyFilesToRerun=1
  ShowSuccessDialog=1
  RebootInQuietMode=0
  RunSetupWatson=1
  RequireAllPatchesNeeded=0
  RequireAllProductsAtMinVersion=0
  MinProductVersion=""
  MinMsiVersion="2.0"
  UpgradeMsi=1
  MsiUpgradeURL="http://go.microsoft.com/fwlink/?LinkId=20610"
  OHotfixLogLevel="v"
  MsiLogLevel="v"
  OHotfixUILevel="n"
  MsiUILevel="n"
  FeatureToCheck=""
Making sure only this instance of the bootstrapper is running...
Making sure the environment is okay for patching...
Getting the patches...
Getting the products to patch...
Seeing if patch C:\OfficeAdmin\OFFICESPSff.msp is needed...
  Product {90150409-6000-11D3-8CFE-0050048383C9}: Not installed. Cannot patch.
  Product {90170409-6000-11D3-8CFE-0050048383C9}: Not installed. Cannot patch.
  ...
  Product {90160409-6000-11D3-8CFE-0050048383C9}: Not installed. Cannot patch.
  Product {91160409-6000-11D3-8CFE-0050048383C9}: Not installed. Cannot patch.
Seeing if patch C:\OfficeAdmin\gdipplus-Binary-GLB.msp is needed...
  Product {90150401-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {901C0401-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  ...
  Product {91440409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {90A10409-6000-11D3-8CFE-0150048383C9}: Microsoft Office OneNote 2003
Version 11.0.6360.0: Already has a patch that supersedes this patch.
  Product {91A10409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {901A0409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {90E00409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {90180409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {90110409-6000-11D3-8CFE-0150048383C9}: Microsoft Office Professional
Edition 2003 Version 11.0.6361.0: Already has a patch that supersedes this patch.
  Product {91110409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {91E30409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {90190409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {91190409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {90120409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.
  Product {91120409-6000-11D3-8CFE-0150048383C9}: Not installed. Cannot patch.

```

Integrating Office Patches into the Install Sources

Additionally, ohotfix provides the capability (which is similar to the update.exe *integrate* parameter) to integrate your office updates into your source files. To do this for Office products, update the ohotfix.ini file and specify the location of the *AdminPath* variable including the name of the target MSI file that needs to be upgraded. For example, if your Administrative update for Office 11 is on your deployment server at c:\OfficeAdmin\office11, then specify your admin path as

```
AdminPath=c:\OfficeAdmin\office11\pro11.msi
```

Obtaining Ohotfix.exe

Ohotfix.exe is included in each Office client patch. However, if you want to set up a centralized location for all of your Office patches, you can download ohotfix.exe separately from the Microsoft Web site. (An Office XP version is available from the Web at <http://download.microsoft.com/download/OfficeXPPProf/Install/4.71.1015.0/W982KMeXP/EN-US/offinst.EXE>).

Dahotfix.exe

Microsoft uses another patch installer engine for updating Microsoft Data Access Components (MDAC) called the MDAC Hotfix Installer. Microsoft uses this installer, which Figure 4-16 shows, to install SQL Server and MDAC updates.

Figure 4-16

Viewing the Microsoft Data Access Components Hotfix Installer



This installer consists of an application named dahotfix.exe and a configuration file named dahotfix.ini. The configuration file and command-line switches for the self-extracting executable are similar to those used for ohotfix.exe and described in the previous section. An example of an update that uses this installer is the MDAC update ENU_Q832483_MDAC_x86.exe released January 13, 2004 and released under MS04-003.

Off the Beaten Track: Older and Unique Update Engines

This chapter has covered the major update engines. But you might come across some of Microsoft's older update engines, update engines that have been renamed, and update engines that were designed to deploy only one update.

Vgxupdate.exe

The latest security update for IE 6 SP1 (IE6.0sp1-KB833989-x86-ENU.exe) released on September 20, 2004 uses the vgxupdate.exe update engine to update the Vector Graphics engine vgx.dll.

Iexpress

Microsoft released the IExpress Deployment Kit to help generate user specific profiles and custom hotfixes and patches aimed primarily at administrators seeking to create custom deployments of Outlook (using the Microsoft Outlook 98 Deployment Kit—ODK) and IE (using the Internet Explorer Administration Kit- IEAK). The most recent IE updates have migrated to use update.exe.

Installing Multiple Hotfixes with Qchain Technology

Installing a security update might require you to restart the target computer. If you install four to five security updates (which is now common during a patch deployment session), those updates might include patches that individually require a computer restart. Deploying the patches independently might necessitate multiple sequential reboots, which can dramatically increase the computer's downtime.

System restarts are sometimes needed to free up a file that otherwise might be in use. For example, if you need to patch a system file that is in use, the OS might already have locked it as “in use” and prevent the patch from being installed. When you deploy a patch that must replace this locked file, the system recognizes this and prompts you to restart the computer. When the computer is restarted the file is unlocked, and before it can be locked again, it is replaced with the new file.

So what happens when you have multiple patches that might update the same files? Several years ago, Microsoft released a program named qchain.exe which addresses this problem. Qchain keeps track of the files that a patch updates allowing you to install multiple updates without having to worry about file version conflicts. Essentially this functionality lets you install multiple updates without restarting the computer between each installation. A system restart is required after the last patch installation. All updates released since May 18, 2001 that use update.exe include Qchain functionality built in. When you specify the /Z or /norestart parameter (described earlier in the Update.exe section), you instruct Windows to suppress the reboot and prepare for subsequent update installations.

For example, to deploy multiple updates you can use this syntax (example taken from the Microsoft Web site):

```
%PATHTOFIXES%\WindowsXP-KB#####-x86-LLL.exe /quiet /norestart
%PATHTOFIXES%\WindowsXP-KB#####-x86-LLL.exe /quiet /norestart
%PATHTOFIXES%\WindowsXP-KB#####-x86-LLL.exe /quiet /forcerestart
```

Installer Wrap-Up

Microsoft has used several patch installers for its different products over the years. Today Microsoft is standardizing on update.exe and Windows Installer. However, you might encounter some of these older installers or other obscure installers. Even though most patch management software abstracts you from needing to deal with the installers on a per-patch basis, if you ever need to customize a patch installation or troubleshoot a failed patch installation, you might find yourself tinkering with these individual installation programs.

Points to remember:

- Update.exe and the Windows Installer are the most common Microsoft installers.
- Most patch management software uses the same parameters that you use when you manually install the patch from the command line.

- You don't need to use qchain.exe on patches later than May 18, 2001 and you can install multiple patches with the /norestart parameter to suppress the reboot.
- The Office patch installer is quite different from the Windows patch installer. To install multiple Office patches, you can run one executable that in turn will install a series of patches located in a folder.

The next chapter explores the Windows Update and Office Update solutions. These solutions help individuals keep their systems up to date.