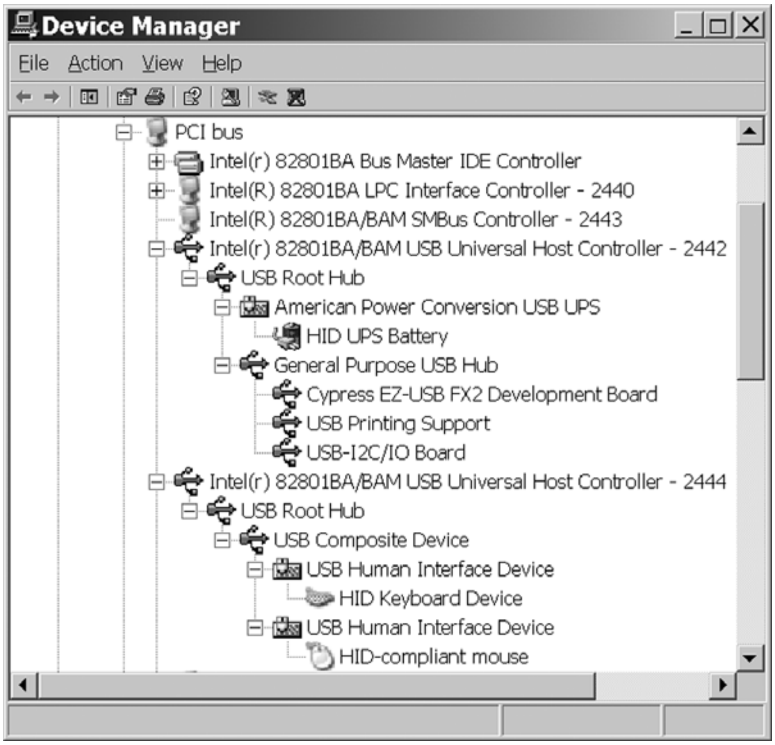
9

Matching a Driver to a Device

On detecting a newly attached USB device, the operating system needs to decide what driver to assign to the device. This chapter shows how Windows uses INF files to select a driver. I also show how the Windows Device Manager and the system registry store information about devices and their drivers.

Using the Device Manager

Windows' Device Manager displays information about all installed devices and presents a user interface for enabling, disabling, and uninstalling devices and updating or changing a device's assigned driver. For developers, the Device Manager is useful for showing whether the correct driver is assigned and successfully installed and for providing a way to force Windows to forget what it knows about a device and start fresh.



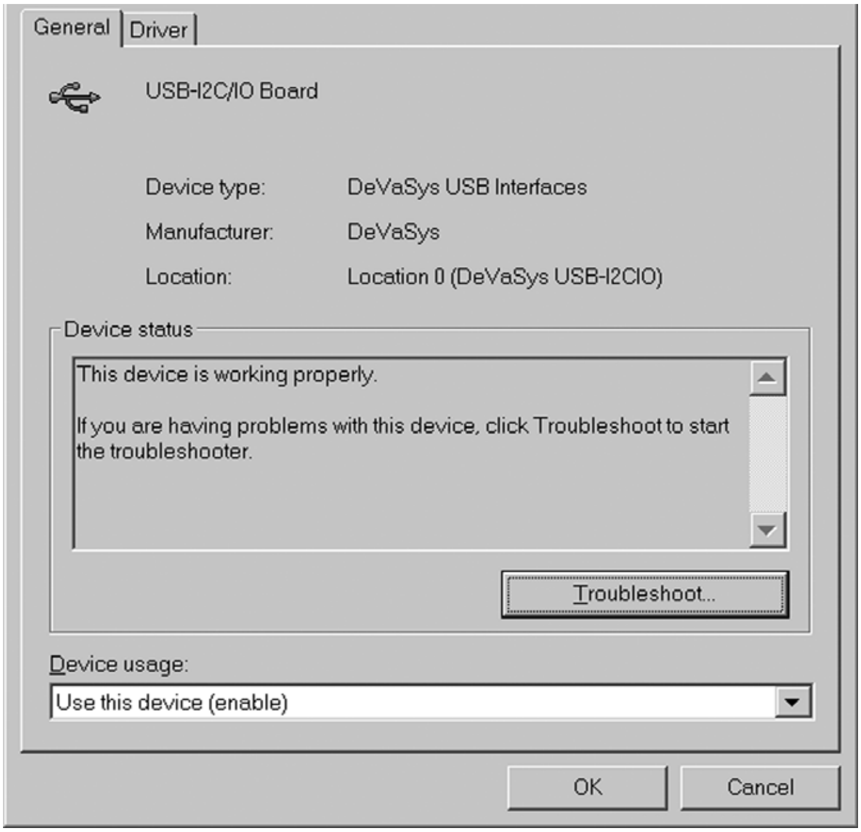
PCI bus
Intel(r) 82801BA Bus Master IDE Controller
Intel(R) 82801BA LPC Interface Controller - 2440
Intel(R) 82801BA/BAM SMBus Controller - 2443
Intel(r) 82801BA/BAM USB Universal Host Controller - 2442
USB Root Hub
American Power Conversion USB UPS
HID UPS Battery
General Purpose USB Hub
Cypress EZ-USB FX2 Development Board
USB Printing Support
USB-I2C/IO Board
Intel(r) 82801BA/BAM USB Universal Host Controller - 2444
USB Root Hub
USB Composite Device
USB Human Interface Device
HID Keyboard Device
USB Human Interface Device
HID-compliant mouse

Figure 9-1: Device Manager's option to view devices by connection quickly shows which devices connect to which hubs and host controllers.

Viewing Devices

To view the Device Manager, in Windows XP, right-click on My Computer, click Manage, and in the Computer Management pane, select Device Manager. Or click Start and select Settings > Control Panel > System > Hardware > Device Manager. Or save some clicks by creating a shortcut to the file *devmgmt.msc* in *Windows\System32*.

The Device Manager's View menu offers four ways to view information: as devices by type and by connection and as resources by type and by connection. Viewing devices by connection (Figure 9-1) shows the physical connections from each host controller and root hub, through any additional hubs, to the attached devices. To view information about a device, including



USB-I2C/IO Board

Device type: DeVaSys USB Interfaces

Manufacturer: DeVaSys

Location: Location 0 (DeVaSys USB-I2CIO)

Device status

This device is working properly.

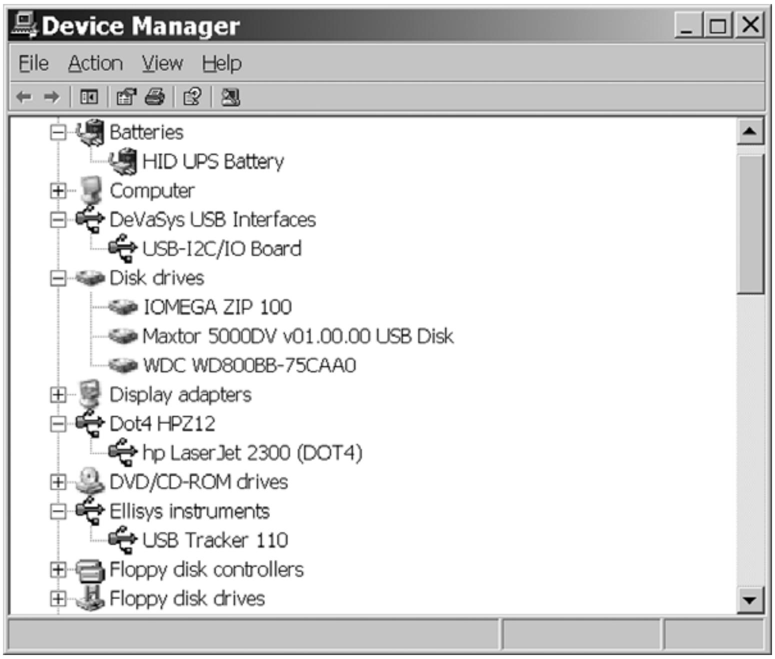
If you are having problems with this device, click Troubleshoot to start the troubleshooter.

Device usage: Use this device (enable)

Figure 9-2: Device Manager's Properties screens provide more information about a device, including what driver the operating system has assigned to the device.

its driver(s) and any problem the operating system has detected with the device, right-click the device's listing and select Properties (Figure 9-2).

Viewing devices by type (Figure 9-3) groups devices according to their functions, with little regard to hardware interface. The Class key(s) in the registry determine what category or categories a device appears in. Many devices fit into standard categories such as Disk Drives, Keyboards, and Modems. Some devices are in multiple categories. For example, a keyboard may appear under both Human Interface Devices and Keyboards. The USB category lists host controllers, hubs, and some other devices. A device with a vendor-specific driver can have its own category or use the USB category.



Batteries
HID UPS Battery
Computer
DeVaSys USB Interfaces
USB-I2C/IO Board
Disk drives
IOMEGA ZIP 100
Maxtor 5000DV v01.00.00 USB Disk
WDC WD800BB-75CAA0
Display adapters
Dot4 HPZ12
hp LaserJet 2300 (DOT4)
DVD/CD-ROM drives
Ellisys instruments
USB Tracker 110
Floppy disk controllers
Floppy disk drives

Figure 9-3: Device Manager also has an option to view devices grouped by type, or function.

Viewing resources by connection or by type shows the memory and I/O addresses and interrupt request (IRQ) lines assigned to each host controller. It's unlikely you'll need this information when developing USB devices, drivers, or applications.

An exclamation point over a device's icon means that the host had a problem communicating with the device or finding a driver. An X over an icon means that the device is present but disabled, possibly by the user.

By default, the Device Manager shows only attached USB devices. To view devices that have been removed but whose drivers are still installed, set the following system environment variable:

```
DEVMGR_SHOW_NONPRESENT_DEVICES=1
```

To set the variable, in Windows' Control Panel, click System > Advanced > Environment Variables, enter the variable's name, and set its value. Then in

Device Manager, click View and check the option to Show Hidden Devices. You may need to reboot after setting the environment variable.

Property Pages

Each listing in the Device Manager has Property Pages that provide additional information about a device and the ability to control the device and its driver. To view the Property Pages, double-click the device's listing. You can request to enable or disable the device or view, update, roll back, or uninstall the device's driver. A Details page provides additional information, including various system IDs, any filter drivers or coinstallers the device uses, and power capabilities.

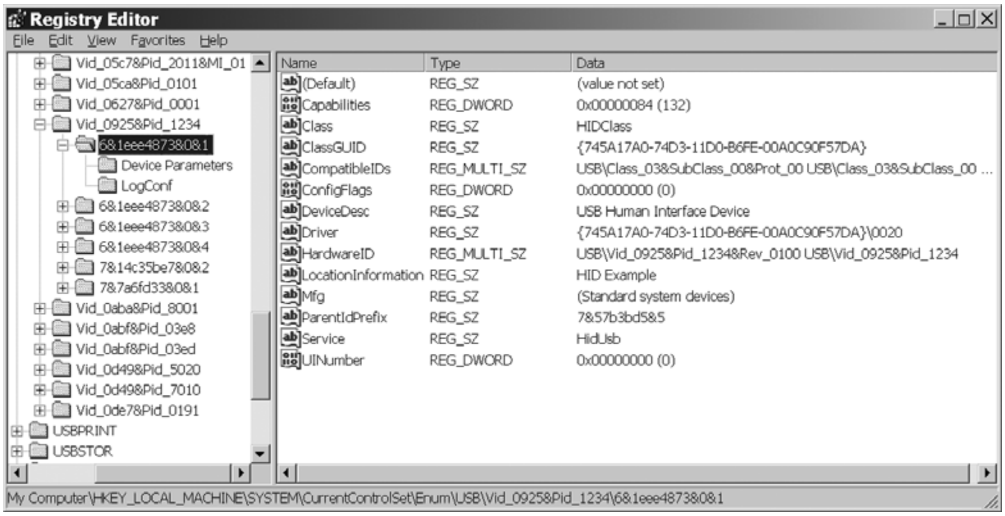
Device Information in the Registry

The system registry is a database that Windows maintains for storing critical information about the hardware and software installed on a system. The registry stores information about all devices that have been installed, whether or not they're currently attached. When a new device is enumerated, Windows stores information about the device in the registry.

Some of the information about USB devices in the registry comes from the bus drivers, which obtain the information from the devices. Other information is from the INF file that the operating system selects when assigning a driver to a device.

You can view the registry's contents using Windows' *regedit* utility. (From the Start menu, select Run and enter *regedit*.) You can also use regedit to edit the registry's contents, but making registry changes this way isn't recommended and is seldom necessary. The Windows Platform SDK documents API functions that enable applications to read and write to the registry. Typically, device installation is the only time it's necessary to change device information in the registry. A request to uninstall a device via the Device Manager or another application also results in changes to the registry.

The system registry is a vital and essential component of Windows. It's so important that Windows maintains multiple backup copies in case the cur-



Registry Editor

```
Vid_05c7&Pid_2011&MI_01
Vid_05ca&Pid_0101
Vid_0627&Pid_0001
Vid_0925&Pid_1234
    6&1eee4873&0&1
        Device Parameters
        LogConf
    6&1eee4873&0&2
    6&1eee4873&0&3
    6&1eee4873&0&4
    7&14c35be7&0&2
    7&7a6fd33&0&1
Vid_0aba&Pid_8001
Vid_0abf&Pid_03e8
Vid_0abf&Pid_03ed
Vid_0d49&Pid_5020
Vid_0d49&Pid_7010
Vid_0de7&Pid_0191
USBPRINT
USBSTOR
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Capabilities	REG_DWORD	0x00000084 (132)
Class	REG_SZ	HIDClass
ClassGUID	REG_SZ	{745A17A0-74D3-11D0-B6FE-00A0C90F57DA}
CompatibleIDs	REG_MULTI_SZ	USB\Class_03&SubClass_00&Prot_00 USB\Class_03&SubClass_00 ...
ConfigFlags	REG_DWORD	0x00000000 (0)
DeviceDesc	REG_SZ	USB Human Interface Device
Driver	REG_SZ	{745A17A0-74D3-11D0-B6FE-00A0C90F57DA}\0020
HardwareID	REG_MULTI_SZ	USB\Vid_0925&Pid_1234&Rev_0100 USB\Vid_0925&Pid_1234
LocationInformation	REG_SZ	HID Example
Mfg	REG_SZ	(Standard system devices)
ParentIdPrefix	REG_SZ	7&57b3bd5&5
Service	REG_SZ	HidUsb
UINumber	REG_DWORD	0x00000000 (0)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\Vid_0925&Pid_1234\6&1eee4873&0&1

Figure 9-4: A hardware key contains information about an instance of a device with a specific Vendor ID and Product ID.

rent copy becomes unusable. Be extremely careful about making changes to the registry. Windows' System Restore utility can restore the registry to an earlier state. Just viewing the registry is safe, however.

The registry's data has a tree structure. Each node on the tree is a registry *key*. Each key can have entries with assigned values and subkeys that in turn may have entries and subkeys. Information about the system's hardware and installed software is under the HKEY_LOCAL_MACHINE key, with information about USB devices under several subkeys: the hardware key, the class key, the driver key, and the service key.

The Hardware Key

The hardware key, also called the instance key or device key, stores information about an instance of a specific device. Hardware keys are under the enumerator (Enum) key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Enum
```

Under the Enum key is a USB key. Each subkey of the USB key contains the Vendor ID and Product ID of a USB device. Figure 9-4 shows the entry for a device with a Vendor ID of 0925h and Product ID or 1234h. Under each of these keys may be one or more hardware keys, with each hardware key identifying an instance of the device. Table 9-1 lists some of the entries under the hardware key.

A device without a USB serial number gets a new hardware key every time the device attaches to a port the device hasn't been attached to previously. If you physically remove the device and attach a different device with identical descriptors to the same port, the operating system doesn't know the difference so there is no new hardware key. Devices with USB serial numbers have one hardware key per physical device, without regard to what port the device is attached to.

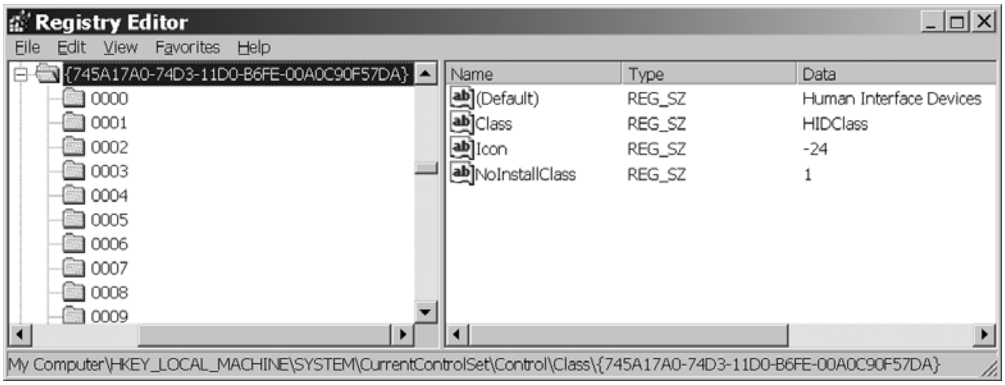
A USB device may also have one or more keys for additional enumerators such as HID, USBPRINT, and USBSTOR. For example, a UPS back-up device with a HID interface can have a key in the Enum\USB branch to name the HidUsb service and a key in the Enum\HID branch to name the HidBatt service.

The Class Key

The class key stores information about device setup class and the devices that belong to it. The class keys are under this registry key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\
Class
```

The name of a class key is the device setup GUID for the class. This is the same as the value stored in the hardware key for devices in the class, under ClassGUID. Figure 9-5 shows the class key for the HID class. The class key contains a friendly name for the setup class, the class name from the header file that defines the GUID, and an index value that specifies the icon to use in Device Manager and other windows that display setup information. Applications can retrieve the index of the mini-icon for a class by calling SetupDiGetClassBitmapIndex. A vendor-specific class installer or co-installer can provide a vendor-specific icon.



{745A17A0-74D3-11D0-B6FE-00A0C90F57DA}
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009

Name	Type	Data
(Default)	REG_SZ	Human Interface Devices
Class	REG_SZ	HIDClass
Icon	REG_SZ	-24
NoInstallClass	REG_SZ	1

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{745A17A0-74D3-11D0-B6FE-00A0C90F57DA}

Figure 9-5: The class key for the HID class includes a friendly name for the class and an index to an icon.

Optional entries in the class key can affect what users see on device installation. If NoInstallClass is present and not equal to zero, users will never need to manually install devices in the class. If SilentInstall is present and not equal to zero, the Plug and Play manager will install devices in the class without displaying dialog boxes or requiring user interaction. If NoDisplayClass is present and not equal to zero, the Device Manager doesn't display devices of the class.

UpperFilters and LowerFilters entries can specify upper filter and lower filter drivers that apply to all devices in the class.

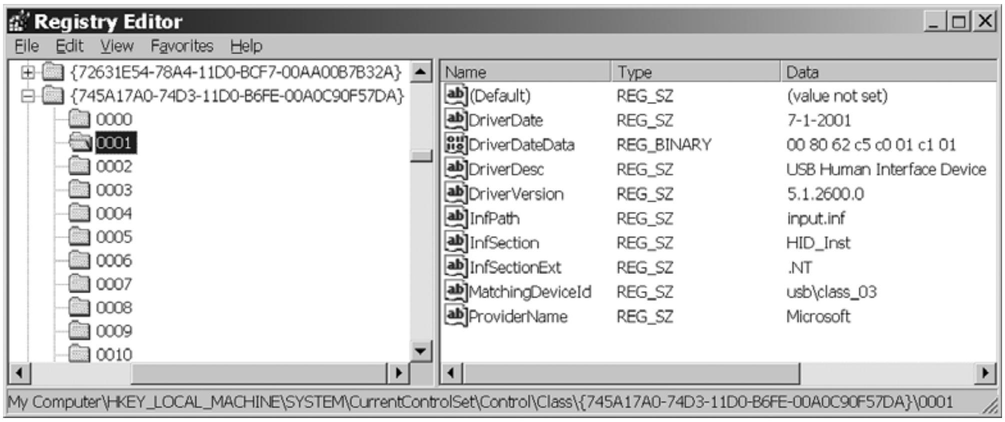
The Driver Key

Under the class key, each device in a class has a driver key, also called a software key. In the hardware key for a device instance, the Driver entry names a device setup GUID that matches a class key and a device instance number that matches a driver subkey under the class key. Figure 9-6 shows the key for a generic HID-class device. Table 9-2 lists some of the entries for a driver key.

The driver key contains the name of the INF file that in turn names the driver files for the device.

Table 9-1: These are some of the entries in a USB device's hardware key.

Key	**Description**	**Source of Information**
Class	Name of the device's setup class	INF file (from *devguid.h*)
ClassGUID	GUID of the device's setup class	INF file (from *devguid.h*)
DeviceDesc	Device Description	INF file, Models section, device description entry
HardwareID	ID string containing the device's Vendor ID and Product ID	Device descriptor
CompatibleIDs	ID string(s) containing the device's class and (optional) subclass and protocol	Device and interface descriptors
Mfg	Device manufacturer	INF file, Manufacturer section, manufacturer name entry
Driver	Name of the device's driver key	System registry, under CurrentControlSet\Control\Class
Location Information	"USB Device" or iProduct string	Bus driver or string descriptor
Service	Name of the device's Service key	System registry, under HKLM\System\ CurrentControlSet\Services


Registry Editor
File Edit View Favorites Help
{72631E54-78A4-11D0-BCF7-00AA00B7B32A}
{745A17A0-74D3-11D0-B6FE-00A0C90F57DA}
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010

Name	Type	Data
(Default)	REG_SZ	(value not set)
DriverDate	REG_SZ	7-1-2001
DriverDateData	REG_BINARY	00 80 62 c5 c0 01 c1 01
DriverDesc	REG_SZ	USB Human Interface Device
DriverVersion	REG_SZ	5.1.2600.0
InfPath	REG_SZ	input.inf
InfSection	REG_SZ	HID_Inst
InfSectionExt	REG_SZ	.NT
MatchingDeviceId	REG_SZ	usb\class_03
ProviderName	REG_SZ	Microsoft

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{745A17A0-74D3-11D0-B6FE-00A0C90F57DA}\0001

Figure 9-6: The driver keys under each class key have information about the drivers assigned to instances of devices in the class.

Table 9-2: The driver key contains information about the driver assigned to a device.

Key	Description	Source of Information
DriverDate	Date of the driver file	INF file, Version section, DriverVer directive
DriverDesc	Driver description	INF file
DriverVer	Driver version	INF file, Version section, DriverVer directive
InfPath	Name of INF file	INF file name
InfSection	Name of the driver's DDInstall section	INF file
InfSectionExt	"Decorated" extension used in INF file (.NT, etc.)	INF file
MatchingDeviceID	The hardware or compatible ID used to assign the driver	Device descriptor and INF file
ProviderName	The provider of the driver	INF file, Provider string

The Service Key

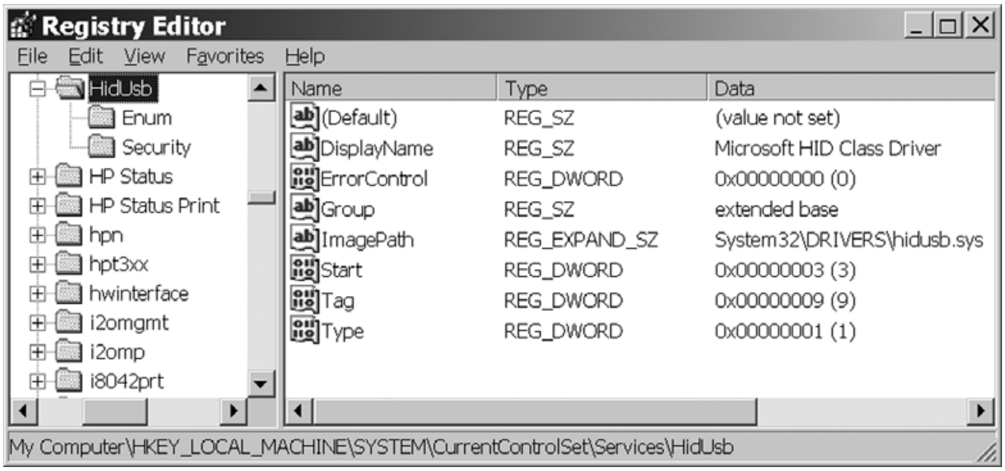
A service key has information about a driver's files, including where they are stored and how to load them. Service keys are in this branch:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
```

There are service keys for each host controller type, hubs, and classes such as storage (USBSTOR), printers (USBPRINT), and HIDs (HidBatt, HidServ, HidUsb). Figure 9-7 shows the Service key for HidUsb.

Inside INF Files

A device setup information file, or INF file, is a text file that contains information about one or more devices in a device setup class. The devices may be from one or more manufacturers. The file tells Windows what driver or drivers to use and contains information to store in the registry. Windows includes INF files for the drivers provided with the operating system. The files are in the *%SystemRoot%\inf* folder. Any new INF files for added devices are copied to this folder as well. By default, the folder is hidden. If you don't see it in Windows Explorer, select Tools > Folder Options > View,



Name	Type	Data
(Default)	REG_SZ	(value not set)
DisplayName	REG_SZ	Microsoft HID Class Driver
ErrorControl	REG_DWORD	0x00000000 (0)
Group	REG_SZ	extended base
ImagePath	REG_EXPAND_SZ	System32\DRIVERS\hidusb.sys
Start	REG_DWORD	0x00000003 (3)
Tag	REG_DWORD	0x00000009 (9)
Type	REG_DWORD	0x00000001 (1)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\HidUsb

Figure 9-7: The service key names a driver's file.

then under Hidden Files, select *Show hidden files and folders*. Do not select *Hide file extensions for known file types.*

On first attachment, after retrieving descriptors from a USB device, Windows looks for a match between the information in the descriptors and the information in the system's INF files.

This section doesn't attempt to document every nuance of INF-file creation. Instead, I use an example INF file to show the kinds of information an INF file can contain. The Windows DDK documentation has more details. Examining the INF files included with Windows is another way to learn about the kinds of things contained in the files and how the information is structured.

Listing 9-1 shows an INF file for the Ellisys USB Explorer protocol analyzer, which uses a vendor-specific driver. (The analyzer has a USB interface that communicates with the PC running the analyzer software.) This INF file is suitable for use under Windows 98, Windows Me, Windows 2000, and Windows XP.

```
; Copyright (C) 1999-2004 Ellisys. All rights reserved.

[Version]
Signature="$CHICAGO$"
DriverVer=01/29/2004,2.0.1600.0
Provider=%Provider%
Class=EllisysProtocolAnalyzers
ClassGUID={D8854594-A4EF-480e-B8D8-CBDDADB4F3B4}

[ClassInstall]
AddReg=ClassAddReg

[ClassInstall32]
AddReg=ClassAddReg

[ClassAddReg]
HKR,,,,"%ClassName%"
HKR,,Icon,,-20

[Manufacturer]
%Manufacturer%=Models

[DestinationDirs]
DefaultDestDir=10,System32\Drivers

[SourceDisksNames]
1=%SourceDisk%,,,.

[SourceDisksFiles]
ellex200.sys=1

[Models]
%DeviceDesc%=Install,USB\VID_0ABA&PID_8002

[Install]
CopyFiles=Install.CopyFiles
AddReg=Install.AddReg

[Install.CopyFiles]
ellex200.sys,,,2
```

Listing 9-1: The INF file for the Ellisys USB Explorer 200 protocol analyzer (Sheet 1 of 2).

```
[Install.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,ellex200.sys

[Install.NT]
CopyFiles=Install.CopyFiles

[Install.NT.Services]
AddService=ellex200,2,Install.NT.AddService

[Install.NT.AddService]
DisplayName=%SvcDesc%
ServiceType=1
StartType=3
ErrorControl=1
ServiceBinary=%10%\system32\drivers\ellex200.sys

[Strings]
ClassName="Ellisys protocol analyzers"
Provider="Ellisys "
Manufacturer="Ellisys"
SourceDisk="USB Explorer 200 Installation Disk"
DeviceDesc="USB Explorer 200"
SvcDesc="USB Explorer 200 Driver (ellex200.sys)"
```

Listing 9-1: The INF file for the Ellisys USB Explorer 200 protocol analyzer (Sheet 2 of 2).

Syntax

The contents of an INF file must follow a few syntax rules.

- The information is arranged in sections, with each section containing one or more items. The section name is in square brackets [ ]. Some of the sections (Version, Manufacturer) are standard sections that every INF file has. Other sections match values specified in other sections. For example, if the Manufacturer section designates the manufacturer as Lakeview, the INF file will also have a [Lakeview] section. The sections can be in any order, but the order of the items within a section can be critical.

- A semicolon (;) indicates a comment.
- A backslash (\) at the end of a line acts as a line continuator, unless it's enclosed in quotes ("\").
- Text enclosed in percent symbols (%sampletext%) refers to a string. For example, you might have the following item:

  ```
  Provider=%Provider%
  ```

  with an item in the Strings section that defines the provider string:

  ```
  Provider="Ellisys"
  ```
- Some items set the value of an entry. For example, this item specifies a device's class:

  ```
  Class=EllisysProtocolAnalyzers
  ```
- Some items provide information to store in the system registry. This item stores the name of a device's driver:

  ```
  HKR,,NTMPDriver,,ellex200.sys
  ```

Sections

Each section of an INF file has a role in helping Windows find a file that matches a device, load the appropriate drivers, and store information about the device in the registry. The discussion that follows explains the purpose of each section in the example INF file with the aim of showing the types of information an INF file can provide.

Copyright Comment

To pass the tests in the Chkinf utility (described later in this chapter), an INF file must have a comment that contains the word *copyright*:

```
; Copyright (C) 1999-2004 Ellisys. All rights
reserved.
```

Version

The Version section is the file's header. Every INF file must have this section. The Version section in the example has these items:

```
[Version]
Signature="$CHICAGO$"
DriverVer=01/29/2004,2.0.1600.0
Provider=%Provider%
Class=EllisysProtocolAnalyzers
ClassGUID={D8854594-A4EF-480e-B8D8-CBDDADB4F3B4}
```

The Signature directive specifies what operating systems the INF file is intended for. For devices that use WDM drivers, the value can be $Windows 95$, $Windows NT$, or $Chicago$, no matter which operating system the PC is using. Chicago was a name used when Windows 95 was under development and its use is still valid under later editions of Windows. The value is case-insensitive.

The DriverVer directive gives the date and version number for the driver(s) named in the INF file. In selecting a driver, all else being equal, Windows will select the more recent driver. A DriverVer directive can also appear in a DDInstall section to provide information that applies only to the driver(s) in that section. Windows 2000 and Windows XP must have a DriverVer directive in the Version section and may have DriverVer directives in DDInstall sections. Windows 98 and Windows Me don't recognize DriverVer directives in the Version section, so any DriverVer directives for these Windows editions must be in the DDInstall sections.

The Provider directive names the creator of the INF file. In the example, %Provider% is a string defined later in the file.

The Class directive specifies the class for devices installed with the INF file. The example specifies the vendor-specific class EllisysProtocolAnalyzers.

The ClassGUID directive is the device setup GUID to store in the device's Class key in the registry. A vendor-specific driver can use the GUID for USB devices or a vendor-specific GUID. The example uses a vendor-specific GUID.

ClassInstall32

The ClassInstall32 section installs a new class in the Class section of the registry. This section is processed only if a device's class hasn't been installed previously. This section should exist only in INF files for devices in vendor-specific device setup classes.

The example ClassInstall32 section has one item:

```
[ClassInstall32]
Addreg=Class.AddReg
```

The Addreg directive adds a class description to the registry. In the example, the directive's value refers to the Class.Addreg section. This section provides a class description in the %ClassName% string and an index value for an icon to display in Device Manager:

```
[ClassAddReg]
HKR,,,"%ClassName%"
HKR,,Icon,,-20
```

A negative Icon value refers to an icon defined in Windows' *setupapi.dll*. A positive Icon value refers to an icon to be extracted from a class installer DLL or property page DLL.

HKR stands for HKEY_ROOT, which is the base registry key for the section that AddReg appears in. In this example, the information is stored under the device's Class key.

The example also has a section titled [ClassInstall] for Windows 98 systems.

Manufacturer

The Manufacturer section identifies one or more groups of devices and an Install section for each group. Every INF file must have this section.

In the example, %Manufacturer% is a string defined later in the file, and Models is the name of a section that identifies the manufacturer's devices. Models is the generic name for the section. The name can be more specific, such as *CypressMice* or *PhilipsAudio*, and an INF file with multiple Models sections must of course use a different name for each section.

```
[Manufacturer]
%Manufacturer%=Models
```

DestinationDirs

The DestinationDirs section names the folder or folders that any CopyFiles, RenFiles, and DelFiles items in the INF file will use. A *dirid* value of 10 specifies the Windows folder. The Windows DDK documentation lists other dirid values. Windows 98 documentation uses the term LDID (logical disk identifier) instead of dirid.

```
[DestinationDirs]
DefaultDestDir=10,System32\Drivers
```

SourceDisksNames

The SourceDisksNames section provides a text description for the installation disk(s). In the entry below, there is one source disk whose name is the %SourceDisk% string defined later in the file. An entry can also specify a volume label and serial number for the disk.

```
[SourceDisksNames]
1=%SourceDisk%,,,.
```

SourceDisksFiles

The SourceDisksFiles section names any file(s) to install from the installation disk. If a file isn't in the disk's root directory, the entry can specify a subdirectory.

```
[SourceDisksFiles]
ellex200.sys=1
```

Models

The Manufacturer section names one or more Models sections. Each Models section contains one or more entries that match a device description to a DDInstall section and hardware ID.

In the example, the device description is the %DeviceDesc% string defined later in the file. Install is the name of the INF file's DDInstall section, which has installation instructions for the device and can add device information to the registry. USB\VID_0ABA&PID_8001 is the hardware ID that identifies the device by its Vendor ID and Product ID. Following the hardware ID, an entry can provide one or more compatible IDs that provide alternate, typically more general, ways to identify devices that use the same driver. The section *Using Device Identification Strings* later in the chapter has more about hardware and compatible IDs.

```
[Models]
%DeviceDesc%=Install,USB\VID_0ABA&PID_8001
```

The example INF file has two sets of Install sections. One set (Install, Install.CopyFiles, and Install.AddReg) is for Windows 98 and Windows Me. The other set (Install.NT, Install.CopyFiles, Install.NT.Services, and Install.NT.AddService) is for Windows 2000 and Windows XP. Both sets use the same Install.CopyFiles section. The section names that contain *.NT* are known as *decorated* sections:

```
[Install]
CopyFiles=Install.CopyFiles
AddReg=Install.AddReg

[Install.CopyFiles]
ellex200.sys,,,2

[Install.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,ellex200.sys

[Install.NT]
CopyFiles=Install.CopyFiles

[Install.NT.Services]
AddService=ellex200,2,Install.NT.AddService

[Install.NT.AddService]
DisplayName=%SvcDesc%
ServiceType=1
StartType=3
ErrorControl=1
ServiceBinary=%10%\system32\drivers\ellex200.sys
```

In the example's Install section, the CopyFiles directive names the Install.CopyFiles section, which specifies the driver file to copy (*ellex200.sys*). A flag value of 2 tells Windows not to allow the user to skip copying the file. The AddReg directive names the Install.AddReg section, which provides information to add to the registry. In the Install.AddReg section, DevLoader names the device loader associated with the device and NTMPDriver names the driver.

The DDInstall section for Windows 2000 and Windows XP is Install.NT. The CopyFiles directive names the same Install.CopyFiles section used for Windows 98 and Windows Me. These Windows editions don't require the DevLoader and NTMPDriver entries, so there is no AddReg section.

Windows 2000 and Windows XP require two additional sections: Install.NT.Services and Install.NT.AddService. The Services section specifies a ServiceName for the driver (*ellex200*), assigns the service as the Plug-and-Play function driver for the device (flags = 2), and names an AddService section that specifies how and when the driver's services are loaded (Install.NT.AddService).

The AddService section in the example has five entries. DisplayName specifies a friendly name for the service. ServiceType = 1 indicates that the entry is for a kernel-mode device driver. StartType = 3 to start the driver on enumeration. ErrorControl = 1 to display a warning and proceed if there is an error when loading or initializing the device. ServiceBinary specifies the location of the driver named in the CopyFiles section. This section can have many additional directives that are optional or required only for some device and drivers.

Strings

The Strings section defines all of the strings that other sections refer to.

```
[Strings]
ClassName="Ellisys protocol analyzers"
Provider="Ellisys "
Manufacturer="Ellisys"
SourceDisk="USB Explorer 200 Installation Disk"
DeviceDesc="USB Explorer 200"
SvcDesc="USB Explorer 200 Driver (ellex200.sys)"
```

Using Device Identification Strings

To identify possible drivers for a device, Windows searches the system's INF files for a device identification string that matches a string created from information in the device's descriptors. There are three categories of device identification strings: device IDs, hardware IDs, and compatible IDs.

Identification Strings Obtained from a Device

Every USB device has at least one device ID, which the hub driver creates from the Vendor ID, Product ID, and revision number in the device descriptor. A device ID for a USB device has one of these forms:

```
USB\VID_xxxx&PID_yyyy&REV_zzzz
USB\VID_xxxx&PID_yyyy
```

The values in xxxx, yyyy, and zzzz are four characters each: xxxx is the idVendor value, yyyy is the idProduct value, and zzzz is the bcdDevice value. The idVendor and idProduct values are hexadecimal values, except for Windows Me, which uses decimal, and bcdDevice is in BCD format.

For example, a device with VID = 0925h, PID = 1234h, and bcdDevice = 0310 has this device ID:

```
USB\VID_0925&PID_1234&REV_0310
```

Devices with multiple interfaces can specify a driver for each interface. In this case, the device has multiple device IDs, one for each interface. A device ID for an interface has one of these forms:

```
USB\VID_xxxx&PID_yyyy&REV_zzzz&MI_ww
USB\VID_xxxx&PID_yyyy&MI_ww
```

The values in xxxx, yyyy, and zzzz are the same as in the previous device IDs. The 2-character value in ww equals bInterfaceNumber in the interface descriptor for one of the device's interfaces. For example, a composite device that functions as a mouse and keyboard might have entries in two Models sections, one for the keyboard (interface 00) and one for the mouse (interface 01):

```
[LAKEVIEW_KEYBOARD]
%USB\VID_0925&PID_0801&MI_00.DeviceDesc%=
HID_Inst,, USB\VID_0925&PID_0801&MI_00
```

```
[LAKEVIEW_MOUSE]
%USB\VID_0925&PID_0801&MI_01.DeviceDesc%=
HID_Inst,, USB\VID_0925&PID_0801&MI_01
```

A HID-class device whose report descriptor contains more than one top-level collection can have a device ID for each collection. A device ID for a collection can have any of these forms, where bb indicates the collection number:

```
USB\VID_xxxx&PID_yyyy&REV_zzzz&Colbb
USB\VID_xxxx&PID_yyyy&Colbb
USB\VID_xxxx&PID_yyyy&REV_zzzz&MI_ww&Colbb
USB\VID_xxxx&PID_yyyy&MI_ww&Colbb
```

In addition to a device ID, some drivers create one or more compatible ID strings for a device. A compatible ID can identify a device by its class code and any subclass and protocol codes in the device descriptor. A compatible ID uses one of the following forms:

```
USB\CLASS_aa&SUBCLASS_bb&PROT_cc
USB\CLASS_aa&SUBCLASS_bb
USB\CLASS_aa
```

The values aa, bb, and cc match values in the device descriptor and are two characters each: aa is the bDeviceClass value, bb is the bDeviceSubclass value, and cc is the bDeviceProtocol value. The values are expressed in hexadecimal, except for Windows Me, which uses decimal.

For example, the class code for HIDs is 03h, so HID-class devices have the following compatible ID:

```
USB\Class_03
```

For some compatible IDs, Windows defines descriptive names such as USBSTOR_BULK or GENERIC_USB_PRINTER.

A compatible ID in an INF file indicates a less desirable but acceptable match. Compatible IDs enable Windows to find and load a driver if the INF files don't contain a matching device ID. A vendor's INF file should not contain a compatible ID.

Obtaining Identification Strings from an INF File

In an INF file, each entry in a Models section has one hardware ID and zero or more compatible IDs. The hardware ID is listed first, followed by any compatible IDs, with commas separating the IDs.

A hardware ID can have any of several forms. It can have one of the forms described above for a device ID for a device, interface, or HID collection. INF files provided with Windows may contain hardware IDs that use the compatible-ID formats described above to identify a device by class or descriptive name.

Finding a Match

In looking for the best match between the information retrieved from a device and the information in INF files, Windows assigns a rank to every match found, with a lower rank indicating a better match (Table 9-3). NT-based operating systems, which include Windows 2000 and Windows XP, give a much lower rank to "trusted" drivers. These are drivers whose catalog (*.cat*) file has a digital signature that indicates that the driver has passed Windows Hardware Quality Labs (WHQL) testing. Chapter 17 has more about WHQL testing. A trusted driver is also called a signed driver. Windows 98 doesn't check for trusted drivers.

In an NT-based operating system, the best match is a device ID that matches a hardware ID in a trusted INF file. The second-best match is a device ID that matches a compatible ID in a trusted INF file. Next is a match between a compatible ID from the device and a hardware ID in a trusted INF file, followed by a match of compatible IDs from the device and a trusted INF file. Only if there are no matches at all with a trusted INF file will an NT-based operating system consider an ID from an untrusted INF file.

If Windows can't find a match, it starts the Found New Hardware wizard and gives the user a chance to specify a location (such as a CD drive) to look for the INF file.

Composite devices, which have multiple interfaces, are a special case. Because each interface may require a different driver, selecting a driver using

Table 9-3: Windows assigns a rank to each INF file that matches a device ID or compatible ID from the device.

Rank (Hex)	ID from Descriptors	ID from INF file	Trusted Driver?	"Decorated" INF section?
0000–0FFF	Device	Hardware	yes	Yes. (All trusted drivers have decorated INF sections for NT-based OS's.)
1000–1FFF	Device	Compatible	yes	
2000–2FFF	Compatible	Hardware	yes	
3000–3FFF	Compatible	Compatible	yes	
8000–8FFF	Device	Hardware	no	yes*
9000–BFFF	Compatible	Compatible	no	yes*
C000–CFFF	Device	Hardware	no	no
D000–FFFE	Compatible	Compatible	no	no
FFFF	Worst-case match. Used by components such as co-installers			
*Considered only by NT-based operating systems (Windows 2000, Windows XP).				

only the Vendor ID and Product ID isn't always sufficient. If there is no better match, Windows XP uses the compatible ID USB\COMPOSITE, which results in loading the USB common class generic parent driver. This driver creates a set of device and compatible IDs for each interface, and Windows can then assign a driver to each interface. In earlier Windows editions, the bus or hub driver handles this task.

Windows comes with hundreds of INF files, and a new device may come with its own INF file. To speed up searching, during device installation, Windows creates a PNF (precompiled INF) file and stores it in the same folder as the device's INF file. The PNF file contains much of the same information as the INF file but in a format that enables quicker searching.

Do You Need to Provide an INF File?

Not every device requires its own INF file. Many devices that use only the system's class drivers can use the INF file that Windows provides for the class. These are some INF files for USB classes in Windows XP:

Class	INF File
Audio	wdmaudio.inf
Human Interface Device (HID)	input.inf (hiddev.inf in Windows 98)
Hub	usb.inf
Mass Storage	usbstor.inf
Printer	usbprint.inf
Smart Card	smartcrd.inf
Still Image	sti.inf

Because Windows XP and later prefer trusted drivers, if you provide an untrusted driver for a device in a supported class, Windows XP and later won't use your driver and instead will select a compatible ID from the class's INF file. An INF file is considered part of the driver package, so if you attempt to provide an untrusted INF file that assigns a trusted driver to your device, Windows XP and later will prefer a system-provided INF file over your INF file. Any change to the contents of a trusted INF file causes an INF file to become untrusted, so you can't add your device to an existing INF file without causing the INF file to become untrusted.

When the best match is an unsigned driver, operating-system settings control whether Windows blocks installation, installs the driver with a warning, or installs with no warning. To change the setting, in Windows Control Panel, click System > Hardware > Driver Signing.

A device that uses a class driver can have a custom INF file with vendor-specific strings that display in the Device Manager. For example, the entry for a HID can be a vendor-specific string such as "My Marvelous HID" instead of the default "USB Human Interface Device." But using a custom INF file under Windows XP and later requires the device and INF file to pass WHQL tests.

The INF files provided with Windows typically contain sections with manufacturer-specific information. When a device passes the WHQL tests, Microsoft often adds the device's sections to an existing INF file or adds a manufacturer-specific INF file to the files distributed with Windows.

Some devices, such as modems, must provide their own INF files. The Windows DDK has examples. A device with a vendor-specific driver must also have its own INF file.

Tools and Diagnostic Aids

Microsoft provides several tools to help in creating and testing INF files: *GenInf* for creating files, *ChkInf* for testing a file's structure and syntax, and a log file of events that occur during device installation.

GenInf is a wizard that asks questions about your device and uses the information to create an INF file. The documentation warns that the created file is a skeleton that may not be fully valid and is likely to need additions or revisions. In particular, the generated INF files do not support older Windows editions or create multi-platform INF files.

ChkINF is a Perl script that requires a Perl interpreter, which you can download free from *www.activeware.com* and other sources. The script runs from a command prompt and creates an HTML page that annotates an INF file with errors and warnings.

When a device is detected, Windows uses Setup and device-installation functions to select a matching INF file and install the device's drivers. The functions also log events and errors in a text file stored in *%SystemRoot%\setupapi.log*. The log can be useful when debugging problems with device installations. The Windows DDK documentation has more about how to use the logging capability.

Tips for Using INF Files

Here are some tips for using and experimenting with INF files:

Use a Valid Vendor ID

Firmware that you make available outside of a controlled environment must use a Vendor ID assigned by the USB-IF. My example code uses the Vendor ID of 0925h, which is assigned to my company, Lakeview Research. The owner of the Vendor ID is responsible for ensuring that each product and version has a unique Vendor ID/Product ID pair. Borrowing someone else's Vendor ID can lead to conflicts if the owner of the ID uses the same values for a different device.

Finding INF Files

On installing a device with a new INF file, Windows copies the INF file to *%SystemRoot%\inf* and may rename the file *oem*.inf* and create a .PNF file named *oem*pnf*, where * is a number. To find INF files that contain a specific Vendor ID and Product ID, search from Windows' Start menu > Search > For Files or Folders. Browse to the *%SystemRoot%\inf* folder and search for the text *VID_xxxx&PID_yyyy*, where *xxxx* is the device's vendor ID and *yyyy* is the product ID.

Removing Device Information

When experimenting with different settings in an INF file, you may find that at times Windows is using information stored in the system registry from a previous version of the INF file. If you want Windows to use a different or changed INF file for a device (because you want to change the driver or device description, for example), you may need to tell Windows to forget what it knows about the device. With the device installed, right-click its listing in the Device Manager, and select Uninstall. Delete any unwanted INF and PNF files that contain your device's Vendor ID and Product ID. You can then remove the device and reattach it, and Windows will start fresh in searching for a driver. (If this approach fails, you may need to delete the unwanted INF and PNF files and registry keys manually.)

To cause Windows 98 to forget what it knows about a device, you may need to rebuild the driver information database. In the *%SystemRoot%\inf* folder, rename *drvdata.bin* to *drvdata.xxx* and rename *drvidx.bin* to *drvidx.xxx*. By renaming the files rather than deleting them, you can restore them if necessary.

INF File Names

The INF files that ship with Windows all have file names with no more than eight characters plus the 3-character extension. Microsoft says that this is due to "technical issues with the product install," but that INF files added after Windows is installed may use longer file names.

What the User Sees

What the user sees on the screen after attaching a USB device can vary depending on the Windows edition, the contents of the device's INF file, the driver's location, whether the driver has a co-installer and is digitally signed, and whether the device has been attached and enumerated previously and has a serial number.

Device and Class Installers

Device and class installers provide functions relating to device installation. The installers are DLLs (dynamic link libraries). Windows provides default installers for devices in supported device setup classes. A device vendor can provide a co-installer that works along with a class installer to support operations that are specific to one or more devices in a class. A co-installer can add information to the registry, request additional configuration information from the user, provide device-specific Properties pages for the Device Manager to display, and perform other tasks relating to device installation. A vendor-defined device setup class can have its own class installer. The Windows DDK documentation has information about writing installers and co-installers.

Searching for a Driver

On boot up or device attachment, after retrieving a device's descriptors, the operating system searches for a matching hardware key. If a key exists, the operating system has what it needs to assign a driver to the device. The hardware key's Driver entry points to the driver key, which names the INF file.

The hardware key's Service entry points to the service key, which has information about the driver files.

On first attachment, there is no matching hardware key and Windows searches its INF files for a match. If the device uses a vendor-specific driver, Windows won't find an INF file and will start the New Device Wizard. The user can let Windows search for a driver or specify what disk and/or folder to search. If your driver is signed and you want to eliminate the need for users to specify the driver's location, you can provide an installation program that uses the API function SetupCopyOEMInf to copy your INF file to the INF folder on the user's system.

On finding a matching INF file, Windows copies the file to *%SystemRoot%\inf* (if the file isn't already there), loads the driver(s) specified in the file if necessary, adds the appropriate keys to the system registry (which also adds the device to the Device Manager), and may display a message to inform the user that the device has been installed.

After installing a device, when installing additional devices that are identical except for the serial number, Windows behaves differently depending on whether the driver is digitally signed. When the driver is signed, Windows uses administrative privileges to install the driver for additional devices after the first, even if the current user doesn't have these privileges. If the driver is unsigned, Windows uses the privileges of the current user in deciding whether to install the driver for additional devices.

When re-attaching a previously attached device, whether Windows finds a driver key can depend on whether the device's descriptors include a USB serial number string. If the device doesn't have a serial number, the hardware key will be found only if the device is re-attached to a port where the device was attached previously. If the device has a serial number, the hardware key will be found no matter which port the device attaches to.