DirectSetup

This section provides information about the DirectSetup component of the DirectX® Programmer's Reference in the Platform Software Development Kit (SDK). Information is divided into the following groups:

- About DirectSetup
- DirectSetup Overview
- DirectSetup Reference

About DirectSetup

DirectSetup is a simple application programming interface (API) that provides you with a one-call installation for the DirectX components. This is more than merely a convenience; DirectX is a complex product, and its installation is an involved task. You should not attempt to manually install DirectX.

In addition, DirectSetup provides an automated way to install the appropriate Microsoft® Windows® registry information for applications that use the DirectPlayLobby object. This registry information is required for the DirectPlayLobby object to enumerate and start the application.

DirectSetup includes the following API functions: **DirectXRegisterApplication**, **DirectXUnRegisterApplication**, **DirectXSetup**, **DirectXSetupSetCallback**, and **DirectXSetupCallbackFunction**. The functions **DirectXRegisterApplication**, **DirectXUnRegisterApplication**, **DirectXSetup**, and **DirectXSetupSetCallback** are provided by Microsoft. The function **DirectXSetupCallbackFunction** is an optional function supplied by applications that use DirectSetup.

DirectSetup Overview

This section contains general information about the DirectSetup component. The following topics are discussed:

- What's New In DirectSetup For DirectX 5?
- Using the DirectXSetup Function
- The Default Setup Process With DirectXSetup
- Customizing Setup With the DirectSetup Callback Function
- Preparing a DirectX Application for Installation
- Enabling AutoPlay

What’s New In DirectSetup For DirectX 5?

DirectSetup now supports a callback function that provides notification of various types of events that occur during the setup of DirectX. This allows developers to customize the setup interface. For details, see Customizing Setup With the DirectSetup Callback Function.

Also new in this version of DirectSetup is the ability of DirectPlayLobby applications to remove registration information. For details, see **DirectXUnRegisterApplication**.

Using the DirectXSetup Function

Applications and games that depend on DirectX use the **DirectXSetup** function to install their system components into an existing Windows installation. It optionally updates the display and audio drivers to support DirectX during the DirectX installation process. This process is designed to happen smoothly, without adversely affecting the user’s system. Older drivers are upgraded whenever possible to prevent reduced performance or stability of all DirectX applications on a computer.

DirectXSetup is provided to each application from Dsetup.dll, DSetup16.dll, and Dsetup32.dll. Therefore all three of these files are included with your product’s setup program. You can find the declarations for DirectSetup in Dsetup.h.

Note

The **DirectXSetup** function overwrites system components from previous versions of DirectX. For example, if you install DirectX 5 on a system that already has DirectX 3 components, all DirectX 3 components will be overwritten. Because all DirectX components comply with Component Object Model (COM) backward compatibility rules, software written for DirectX 3 will continue to function properly.

DirectX 5 requires the installation of all components. Previous versions of DirectX allowed the installation of individual DirectX components. However, the amount of disk space saved by this was minimal. Current DirectX components are tightly integrated together for maximum performance. Hence, they all need to be installed for any one of them to work.

The DirectX Programmer’s Reference of the Platform SDK contains the \Redist directory. Setup programs that use the **DirectXSetup** function must distribute the appropriate files from this directory as specified in the End User License Agreement (EULA).

The Default Setup Process With DirectXSetup

The **DirectXSetup** function can tell when DirectX components, display drivers, and audio drivers need to be upgraded. It can also distinguish whether or not these components can be upgraded without adversely affecting the Windows operating system. This is said to be a “safe” upgrade. It is important to note that the upgrade is safe for the operating system, not necessarily for the applications running on the computer. Some hardware-dependent applications can be negatively affected by an upgrade that is safe for Windows.

By default, the **DirectXSetup** function performs only safe upgrades. If the upgrade of a device driver may adversely affect the operation of Windows, the upgrade is not performed.

During the setup process, DirectSetup creates a backup copy of the system components and drivers that are replaced. These can typically be restored in the event of an error.

When display or audio drivers are upgraded, the **DirectXSetup** function utilizes a database created by Microsoft to manage the process. The database contains information on existing drivers that are provided either by Microsoft, the manufacturers of the hardware, or the vendors of the hardware. This database describes the upgrade status of each driver, based on testing done at Microsoft and at other sites.

Customizing Setup With the DirectSetup Callback Function

DirectSetup for DirectX 5 allows developers to specify a setup callback function. In the DirectSetup documentation, the callback function is referred to as **DirectXSetupCallbackFunction**. However, the actual name of the callback function is supplied by the application setup program.

If it is provided, **DirectXSetupCallbackFunction** is called once for each DirectX component and device driver that can be upgraded by the **DirectXSetup** function. Note that the callback function is completely optional. It does not have to be provided.

If a callback function is not provided by the setup program, **DirectXSetup** will display status and error information by calling the **MessageBox** function. If a callback is provided, the information that would be used by **DirectXSetup** to display the status with **MessageBox** is passed as parameters to the **DirectXSetupCallbackFunction** callback function. The callback function can use this information to display a status or error message using **MessageBox**. It can also implement a custom user interface to display the status or error message.

Uses of the DirectSetup Callback Function

DirectXSetupCallbackFunction can be used to:

- *Display a user interface that is customized for the application*. A game, for example, could display the progress of DirectX installation as a flying saucer descending toward a planet. When setup is complete, the saucer could land and an amusing alien could disembark carrying a sign reading, “Success!”
- *Suppress the display of status and error messages.* Designers of programs that are for novice users may want to suppress error messages so that they can be handled by the setup program. This requires a larger-than-normal development effort for the setup program, but may be appropriate for the target audience.
- *Automatically handle status and error conditions.* Setup programs that suppress error messages should handle them automatically. If the users are not given error information, they should not be required to intervene when an error occurs.
- *Allow the user greater control when status and error messages occur.* It is often appropriate to allow knowledgeable users greater control than normal over the setup process. Caution should be taken, however, when using this approach. Allowing users greater-than-normal control over the setup process increases the chances that their system will be adversely affected during or after the setup.
- *Override the default behavior of the **DirectXSetup** function.* Although this is not recommended, it can be done. The user is typically notified when a setup program does this.

Providing a Callback Function to DirectSetup

If a setup program provides a callback function to DirectSetup, it does so by calling the **DirectXSetupSetCallback** function before the **DirectXSetup** function. A pointer to the callback function is passed as a parameter to the **DirectXSetupSetCallback** function. See **DirectXSetupSetCallback**, and An Example Callback Function for details.

Interpreting DirectSetup Flags in the Callback Function

When the callback function **DirectXSetupCallbackFunction** is called by the **DirectXSetup** function, it is passed a parameter that contains the reason that the callback function was invoked. If the reason is DSETUP_CB_MSG_CHECK_DRIVER_UPGRADE, the *pInfo* parameter points to a structure containing flags that summarize the **DirectXSetup** function’s recommendations on how the upgrade of DirectX components, display drivers, and audio drivers should be performed. For a chart that summarizes the callback function flags, see **DirectXSetupCallbackFunction**. The structure member containing the flags is called **UpgradeFlags** .

The flags passed through the **UpgradeFlags** member of the structure that is pointed to by the *pInfo* parameter of the callback function are present when the *Reason* parameter of the callback function is

DSETUP_CB_MSG_CHECK_DRIVER_UPGRADE. They occur in the following combinations:

Primary Upgrade Flags

These flags are mutually exclusive. One of them is always present in the **UpgradeFlags** structure member.

DSETUP_BC_UPGRADE_FORCE

DSETUP_BC_UPGRADE_KEEP

DSETUP_BC_UPGRADE_SAFE

DSETUP_BC_UPGRADE_UNKNOWN

Secondary Upgrade Flags

Any or all of these flags may be present in the **UpgradeFlags** structure member.

DSETUP_BC_UPGRADE_CANTBACKUP

DSETUP_BC_UPGRADE_HASWARNINGS

Device Active Flag

This flag is present in the **UpgradeFlags** structure member if the device whose driver is being upgraded is active. This flag may be present in combination with any of the others.

DSETUP_BC_UPGRADE_DEVICE_ACTIVE

Device Class Flags

These flags are mutually exclusive. One of them is always present in the **UpgradeFlags** structure member.

DSETUP_BC_UPGRADE_DISPLAY

DSETUP_BC_UPGRADE_MEDIA

Every time the *Reason* parameter has the value DSETUP_CB_MSG_CHECK_DRIVER_UPGRADE, the **UpgradeFlags** member of the structure pointed to by *pInfo* contains one Primary Upgrade Flag, zero or more Secondary Upgrade Flags, zero or one Device Active Flag, and one Device Class Flag.

If the **UpgradeFlags** member is set to DSETUP_CB_UPGRADE_KEEP, the DirectX component or device driver can't be upgraded. Performing an upgrade would cause Windows to cease to function properly. The **DirectXSetup** function will not perform an upgrade on the component or driver.

A value of DSETUP_CB_UPGRADE_FORCE in the **UpgradeFlags** structure member means that the component or driver *must* be upgraded for Windows to function properly. The **DirectXSetup** function will upgrade the driver or component. It is possible that the upgrade may adversely affect some programs on the system. When the **DirectXSetup** function detects this condition, the **UpgradeFlags** member will be set to DSETUP_CB_UPGRADE_FORCE | DSETUP_CB_UPGRADE_HAS_WARNINGS, where the symbol | represents a bitwise **OR** operation. When this occurs, the **DirectXSetup** function will perform the upgrade, but issue a warning to the user.

Components and drivers are considered safe for upgrade if they will not adversely affect the operation of Windows when they are installed. In this case, the **UpgradeFlags** member will be set to DSETUP_CB_UPGRADE_SAFE. It is possible that the upgrade can be safe for Windows, but still cause problems for programs installed on the system. When **DirectXSetup** detects this condition, the **UpgradeFlags** member will contain the value DSETUP_CB_UPGRADE_SAFE | DSETUP_CB_UPGRADE_HAS_WARNINGS. If this occurs, the default action for the **DirectXSetup** function is to not perform the upgrade.

If no callback is provided, the **DirectXSetup** function calls the Win32® API function **MessageBox** to get input from the user if needed. Typically, however, the **DirectXSetup** function will perform the default action without notifying the user. The function **DirectXSetupCallbackFunction** is supposed to return the same values that **MessageBox** would return if it were used.

The **MessageBox** function displays a message and some buttons for user response. When it is called, flags are passed to it that indicate what buttons should be present and which is the default button. These same flags are passed to the function **DirectXSetupCallbackFunction** in the *MsgType* parameter. These flags are the same flags that can be passed to the **MessageBox** function through its *uType* parameter.

The callback function should return what **MessageBox** would return if it were used. For instance, a callback function can be called with the flags in the *MsgType* parameter set to MB_YESNO | MB_DEFBUTTON1, where the | symbol is a bitwise **OR** operation. If **MessageBox** were called with these flags, it would present the user with a dialog box containing the **Yes** and **No** buttons. The default button is the **Yes** button. The callback should do something that is functionally equivalent to that. In this example, the return value of **MessageBox** would be the ID of the button that the user selected, either IDYES or IDNO. The return value of the callback function should be whichever one of these two the user selects.

A more complete discussion of the flags and the appropriate return values is contained in the Platform SDK documentation for the **MessageBox** function.

The following code is a function that can be used by DirectSetup callback functions. It illustrates the process of determining the ID of the default button for any allowable set of input flags.

```
INT DefaultButton(DWORD MsgType)
{
    INT iDefaultButton=0;

    switch (MsgType & 0x0F0F)
    {
        case MB_OK          | MB_DEFBUTTON1:
        case MB_OKCANCEL    | MB_DEFBUTTON1:
            iDefaultButton = IDOK;
        break;

        case MB_OKCANCEL    | MB_DEFBUTTON2:
```

```
        case MB_RETRYCANCEL | MB_DEFBUTTON2:
        case MB_YESNOCANCEL | MB_DEFBUTTON3:
            iDefaultButton = IDCANCEL;
        break;

        case MB_ABORTRETRYIGNORE | MB_DEFBUTTON1:
            iDefaultButton = IDABORT;
        break;

        case MB_RETRYCANCEL      | MB_DEFBUTTON1:
        case MB_ABORTRETRYIGNORE | MB_DEFBUTTON2:
            iDefaultButton = IDRETRY;
        break;

        case MB_ABORTRETRYIGNORE | MB_DEFBUTTON3:
            iDefaultButton = IDIGNORE;
        break;

        case MB_YESNO       | MB_DEFBUTTON1:
        case MB_YESNOCANCEL | MB_DEFBUTTON1:
            iDefaultButton = IDYES;
        break;

        case MB_YESNO       | MB_DEFBUTTON2:
        case MB_YESNOCANCEL | MB_DEFBUTTON2:
            iDefaultButton = IDNO;
        break;
    }

    return iDefaultButton;
}
```

In this example, the function uses bitwise **OR** operations to determine what kind of dialog box the **MessageBox** function would display, and which button is the default. A callback function can use a similar method to determine what value it should return.

Overriding DirectSetup Flags in the Callback Function

The function **DirectXSetupCallbackFunction** can override some of the default behaviors of the **DirectXSetup** function through its return value. As an example, the default behavior for **DirectXSetup** is to not install a component if the *UpgradeType* member of the *pInfo* parameter of the function **DirectXSetupCallbackFunction** is set to DSETUP_CB_UPGRADE_SAFE | DSETUP_CB_UPGRADE_HAS_WARNINGS (where the | symbol indicates a bitwise **OR** operation). In this case, the *MsgType* parameter of the callback function is set to MB_YESNO | MB_DEFBUTTON2. If the callback function accepts the default, it will return IDNO. If it wants to override the default, the callback function should return IDYES. If it does override the default, the user will be notified by the **DirectXSetup** function.

An Example Callback Function

A simple setup program could contain a callback function along the lines of the following:

```
DWORD MySetupCallback(
    DWORD Reason,
    DWORD MsgType,
    char *szMessage,
    char *szName,
    DSETUP_CB_UPGRADEINFO *pUpgradeInfo)
{
    if (MsgType==0)        // ignore status messages
        return ID_OK;

    return MessageBox(MyhWnd, szMessage,
                "My Application Name", MsgType);
}
```

This example ignores all status messages, but displays error messages by calling the **MessageBox** function.

The address of the callback function in the example above would be passed to DirectSetup prior to calling the **DirectXSetup** function. The following code gives an example of how this is done.

```
// Set the callback function.
DirectXSetupSetCallback(MySetupCallback);
// Start the setup of DirectX components and drivers.
if (SUCCESS(DirectXSetup(hWndParent, NULL,
                    DSETUP_DIRECTX))
    {
        // The installation succeeded.
    }
    else
    {
        /* Installation failed. Handle the error in MySetupCallback.
        Do any additional cleanup needed right here. */
    }
```

Preparing a DirectX Application for Installation

At some point during the development of your application, you will need to create a setup program that installs your application and the DirectX files on a user's system. This program will determine the amount of disk space required to install your application and copy the appropriate DirectX files to the user's computer. You also need to create a directory on your distribution medium in which you will place all the application's files and any additional DirectX components. The following topics describe these steps:

- Creating the Setup Program
- Testing the Setup Program

Creating the Setup Program

The DirectX Programmer's Reference in the Platform SDK includes an example setup program that you can use as a model for your application's setup program. The setup program is called Dinstall, and it is located in the \Dxsdk\Sdk\Samples\Setup directory. It installs a sample DirectX application called Rockem. It also demonstrates one way to configure the **DirectXSetup** function.

Û To adapt the Dinstall.c program to your needs

1. In an editor, open Dinstall.c.
2. Search for the text "copy_list".
3. Edit the list of files in this structure to contain the names of the files you want copied to the user's computer during installation.
4. If necessary, modify Dinstall.c so that it installs files in subdirectories on the user's hard disk. Currently, Dinstall installs files only in the default directory.
5. Search for two locations in Dinstall.c that contain the text "IDS_DISK_MSG".
6. Add some code that checks whether there is enough free hard disk space to install your application on the user's computer. Dinstall does not currently check this.

The *lpszRootPath* parameter of **DirectXSetup** specifies the path to the Dsetup*.dll files (Dsetup.dll, Dsetup16.dll, and Dsetup32.dll) and the Directx directory on your distribution media. These dynamic-link libraries and this directory should be located in the same directory as the Dinstall executable after it is compiled, unless there is an overwhelming reason to do otherwise. If all these files and directories are located in the same directory, the value of the *lpszRootPath* parameter should be set to NULL. This ensures that if the path changes when the files are placed on a compact disc or floppy disks from the root of the application, the **DirectXSetup** function still works properly.

For example, suppose Dinstall.exe, Dsetup*.dll, and the Directx directory are located in an application directory called D:\Funstuff during the testing phase. Then, when you burn the files on a compact disc, suppose you put them in the root. If the *lpszRootPath* parameter is set to "\FUNSTUFF", the setup program (Dinstall.exe) will not function from the compact disc. However, if the *lpszRootPath* parameter is set to NULL, the setup program will function in both cases, because the path to Dsetup*.dll, and the Directx directory are still in the current directory.

If you decide to place the Dsetup*.dll files and the Directx directory somewhere other than in the directory that contains Dinstall.exe, you must pass the appropriate parameters to **DirectXSetup** and load Dsetup.dll correctly. The *lpszRootPath* parameter of **DirectXSetup** should contain the full path to Dsetup.dll. In addition, you need to use the **LoadLibrary** and **GetProcAddress** Win32 functions in your setup program to locate Dsetup.dll.

The content of the **Setup** dialog box is determined by data supplied in the Dinstall.rc resource file.

Û To display your application's name and graphics

1. In an editor, open Dinstall.rc.
2. Search for all occurrences of "Rockem" and change them to the name of your application.
3. The graphics that are displayed in the **Setup** and **Reboot** dialog boxes are called Signon.bmp and Reboot.bmp in the resource file. Either rename your bitmap files these names, or change the names in the resource file to match the names of your bitmaps.
4. The icon for the Dinstall executable is called Setup.ico in the resource file, and it is specified by SETUP_ICON. Either set the name of your icon file to Setup.ico, or change the name in the resource file to match the name of your icon file.
5. If appropriate for your application, change the default directory in which your application is installed. To do this, search for "IDS_DEFAULT_GAME_DIR" (it is located in two places in the resource file) and change the path of the default directory.

After you have modified the Dinstall.c and Dinstall.rc files to fit your application's needs, compile them into the Dinstall.exe executable file. You can rename this executable file to anything you want (Setup.exe, for example).

Testing the Setup Program

Before you commit your application to a compact disc or floppy disks, you should test your setup program. Do this by creating an application directory that contains all of your application’s files, the setup program, and the DirectX files and drivers.

Û To set up the application directory

1 Create a directory that includes all your application's files. Be sure to create any subdirectories that are needed. Place the appropriate application files in the subdirectories.

2 Copy the setup program you wrote to the root of your application directory.

3 At the MS-DOS prompt, use the **xcopy** command to copy the Redist directory on the DirectX compact disc to the root of your application directory. For example, if your application's root directory is D:\Fungame, and the E: drive is your CD-ROM drive, type the following:

xcopy /s e:\redist\*.* d:\fungame

Note

The root of your application directory should include the entire contents of the Redist directory distributed on the DirectX Programmer's Reference on the Platform SDK. This is essential to ensure that the **DirectXSetup** function and the Dxsetup.exe file work properly.

Enabling AutoPlay

If you are building an AutoPlay compact disc title, copy the Autorun.inf file in the root directory of the DirectX Programmer's Reference in the Platform SDK compact disc to the root of your application directory. This text file contains the following information:

```
[autorun]
OPEN=SETUP.EXE
```

If your application's setup program is called Setup.exe, you will not have to make any changes to this file; otherwise, edit this file to contain the name of your setup program. For more information, see **Autorun.inf**.

DirectSetup Reference

This section contains reference information for the API elements that DirectSetup provides. Reference material is divided into the following categories:

- Functions
- Structures
- Return Values

Functions

This section contains information on the following global functions used with DirectSetup:

- **DirectXRegisterApplication**
- **DirectXSetup**
- **DirectXSetupGetVersion**
- **DirectXSetupSetCallback**

- **DirectXSetupCallbackFunction**
- **DirectXUnRegisterApplication**

DirectXRegisterApplication

The **DirectXRegisterApplication** function registers an application as one designed to work with DirectPlayLobby.

int WINAPI DirectXRegisterApplication(
 HWND *hWnd*,
 LPDIRECTXREGISTERAPP *lpDXRegApp*
);

Parameters

hWnd
: Handle to the parent window. If this parameter is set to NULL, the desktop is the parent window.

lpDXRegApp
: Address of the **DIRECTXREGISTERAPP** structure that contains the registry entries that are required for the application to function properly with DirectPlayLobby.

Return Values

If this function is successful, it returns TRUE.

If it is not successful, it returns FALSE. Use the **GetLastError** Win32 function to get extended error information.

Remarks

The **DirectXRegisterApplication** function inserts the registry entries needed for an application to operate with DirectPlayLobby. If these registry entries are added with **DirectXRegisterApplication**, they should be removed with **DirectXUnRegisterApplication** when the application is uninstalled.

Many commercial install programs will remove registry entries automatically when a program in uninstalled. However, such a program will only do so if it added the registry entries itself. If the DirectPlayLobby registry entries are added by **DirectXRegisterApplication**, commercial install programs will not delete the registry entries when the application is uninstalled. Therefore, DirectPlayLobby registry entries that are created by **DirectXRegisterApplication** should be deleted by **DirectXUnRegisterApplication**.

Registry entries needed for DirectPlayLobby access can be created without the use of the **DirectXRegisterApplication** function. This, however, is not generally recommended. See *Registering Lobby-able Applications* in the DirectPlay® documentation.

See Also

DirectXUnRegisterApplication

DirectXSetup

The **DirectXSetup** function installs one or more DirectX components.

int WINAPI DirectXSetup(
  HWND *hWnd***,**
  LPSTR *lpszRootPath***,**
  DWORD *dwFlags*
);

Parameters

hWnd
: Handle to the parent window for the setup dialog boxes.

lpszRootPath
: Pointer to a string that contains the root path of the DirectX component files. This string must specify a full path to the directory that contains the files Dsetup.dll, Dsetup16.dll, and Dsetup.dll32. This directory is typically Redist. If you are certain the current directory contains Dsetup.dll and the Directx directory, this parameter can be NULL.

dwFlags
: One or more flags indicating which DirectX components should be installed. A full installation (DSETUP_DIRECTX) is recommended.

DSETUP_D3D	Obsolete. DirectX 3 programs that use this flag will install all DirectX components.
DSETUP_DDRAW	Obsolete. DirectX 3 programs that use this flag will install all DirectX components.
DSETUP_DDRAWDRV	Installs display drivers provided by Microsoft.
DSETUP_DINPUT	Obsolete. DirectX 3 programs that use this flag will install all DirectX components.
DSETUP_DIRECTX	Installs DirectX runtime components as well as DirectX-compatible display and audio drivers.
DSETUP_DIRECTXSETUP	Obsolete. DirectX 3 programs that use this

	flag will install all DirectX components.
DSETUP_DPLAY	Obsolete. DirectX 3 programs that use this flag will install all DirectX components.
DSETUP_DPLAYSP	Obsolete. DirectX 3 programs that use this flag will install all DirectX components.
DSETUP_DSOUND	Obsolete. DirectX 3 programs that use this flag will install all DirectX components.
DSETUP_DSOUNDDRV	Installs audio drivers provided by Microsoft.
DSETUP_DXCORE	Installs DirectX runtime components. Does not install DirectX-compatible display and audio drivers.
DSETUP_TESTINSTALL	Performs a test installation. Does not actually install new components.

Return Values

If this function is successful, it returns SUCCESS.

If it is not successful, it returns an error code. For a list of possible return codes, see Return Values .

Remarks

Before you use the **DirectXSetup** function in your setup program, you should ensure that there is at least 15 MB of available disk space on the user's system. This is the maximum space required for DirectX to set up the appropriate files. If the user's system already contains the DirectX files, this space is not needed.

See Also

Using the DirectXSetup Function.

DirectXSetupGetVersion

The **DirectXSetupGetVersion** function retrieves the version number of the DirectX components that are currently installed.

```
INT WINAPI DirectXSetupGetVersion(
  DWORD  *pdwVersion            // receives the version number
  DWORD  *pdwRevision           // receives the revision number
);
```

Parameters

pdwVersion

Pointer to a **DWORD**. The **DirectXSetupGetVersion** function will fill the **DWORD** with the version number. If this parameter is NULL, it is ignored.

pdwRevision

Pointer to a **DWORD**. The **DirectXSetupGetVersion** function will fill the **DWORD** with the revision number. If this parameter is NULL, it is ignored.

Return Values

If this function is successful, it returns non-zero.

If it is not successful, it returns zero.

Remarks

The **DirectXSetupGetVersion** function can be used to retrieve the version and revision numbers before or after the **DirectXSetup** function is called. If it is called before the **DirectXSetup** function is invoked, it gives the version and revision numbers of the DirectX components that are currently installed. If it is called after the **DirectXSetup** function is called, but before the computer has been rebooted, it will give the version and revision numbers of the DirectX components that will take effect after the computer is restarted.

The version number in the *pdwVersion* parameter is composed of the major version number and the minor version number. The major version number will be in the 16 most significant bits of the **DWORD** when this function returns. The minor version number will be in the 16 least significant bits of the **DWORD** when this function returns. The version numbers can be interpreted as follows:

DirectX Version	**Value Pointed At By *pdwVersion***
DirectX 1	0x00040001
DirectX 2	0x00040002
DirectX 3	0x00040003
DirectX 4	0x00040004
DirectX 5	0x00040005

The version number in the *pdwRevision* parameter is composed of the release number and the build number. The release number will be in the 16 most significant bits of the **DWORD** when this function returns. The build number will be in the 16 least significant bits of the **DWORD** when this function returns.

The following sample code fragment demonstrates how the information returned by **DirectXSetupGetVersion** can be extracted and used.

```
DWORD dwVersion;
DWORD dwRevision;
if (DirectXSetupGetVersion(&dwVersion, &dwRevision))
{
  printf("DirectX version is %d.%d.%d.%d\n",
        HIWORD(dwVersion), LOWORD(dwVersion),
        HIWORD(dwRevision), LOWORD(dwRevision));
}
```

Version and revision numbers can concatenated into a 64 bit quantity for comparison. The version number should be in the 32 most significant bits and the revision number should be in the 32 least significant bits.

See Also

DirectXSetup

DirectXSetupSetCallback

The **DirectXSetupSetCallback** sets a pointer to a callback function that is periodically called by **DirectXSetup**. The callback function can be used for setup progress notification and to implement a custom user interface for an application's setup program. For information on the callback function, see **DirectXSetupCallbackFunction**. If a setup program does not provide a callback function, the **DirectXSetupSetCallback** function should not be invoked.

```
INT WINAPI DirectXSetupSetCallback(
  DSETUP_CALLBACK  Callback        // pointer to the callback function
);
```

Parameters

Callback

Pointer to a callback function.

Return Values

Currently returns zero.

Remarks

To set a callback function, **DirectXSetupSetCallback** must be called before the **DirectXSetup** function is called.

The name of the callback function passed to **DirectXSetupSetCallback** is supplied by the setup program. However, it must match the prototype given in **DirectXSetupCallbackFunction**

See Also

DirectXSetupCallbackFunction, **DirectXSetup**

DirectXSetupCallbackFunction

DirectXSetupCallbackFunction is a placeholder name for a callback function supplied by the setup program. The callback function reports the status of the current installation process. It also can provide information for use by the **MessageBox** function.

```
DWORD DirectXSetupCallbackFunction(
  DWORD  Reason,              // reason for the callback
  DWORD  MsgType,             // same as MessageBox
  char *szMessage,            // message string
  char *szName                // depends on Reason
  void *pInfo                 // upgrade information
);
```

Parameters

Reason

Reason for the callback. It can be one of the following values.

DSETUP_CB_MSG_BEGIN_INSTALL

DirectXSetup is about to begin installing DirectX components and device drivers.

DSETUP_CB_MSG_BEGIN_INSTALL_DRIVERS

DirectXSetup is about to begin installing device drivers.

DSETUP_CB_MSG_BEGIN_INSTALL_RUNTIME

DirectXSetup is about to begin installing DirectX components.

DSETUP_CB_MSG_BEGIN_RESTORE_DRIVERS

DirectXSetup is about to begin restoring previous drivers.

DSETUP_CB_MSG_CANTINSTALL_BETA

A pre-release beta version of Windows 95 was detected. The DirectX component or device driver can't be installed.

DSETUP_CB_MSG_CANTINSTALL_NOTWIN32

The operating system detected is not Windows 95 or Windows NT®. DirectX is not compatible with Windows 3.*x*.

DSETUP_CB_MSG_CANTINSTALL_NT

The DirectX component or device driver can't be installed on versions of Windows NT prior to version 4.0.

DSETUP_CB_MSG_CANTINSTALL_UNKNOWNOS

The operating system is unknown. The DirectX component or device driver can't be installed.

DSETUP_CB_MSG_CANTINSTALL_WRONGLANGUAGE

The DirectX component or device driver is not localized to the language being used by Windows.

DSETUP_CB_MSG_CANTINSTALL_WRONGPLATFORM

The DirectX component or device driver is for another type of computer.

DSETUP_CB_MSG_CHECK_DRIVER_UPGRADE

Driver is being considered for upgrade. Verification from user is recommended.

DSETUP_CB_MSG_INTERNAL_ERROR

An internal error has occurred. Setup of the DirectX component or device driver has failed.

DSETUP_CB_MSG_NOMESSAGE

No message to be displayed. The callback function should return.

DSETUP_CB_MSG_NOTPREINSTALLEDONNT

The DirectX component or device driver can’t be installed on the version of Windows NT in use.

DSETUP_CB_MSG_PREINSTALL_NT

DirectX is already installed on the version of Windows NT in use.

DSETUP_CB_MSG_SETUP_INIT_FAILED

Setup of the DirectX component or device driver has failed.

MsgType

Contains flags that control the display of a message box. These flags can be passed to the **MessageBox** function. An exception is when *MsgType* is equal to zero. In that case, the setup program can display status information but should not wait for input from the user.

szMessage

A localized character string containing error or status messages that can be displayed in a message box created with the **MessageBox** function.

szName

The value of *szName* is NULL unless the *Reason* parameter is DSETUP_CB_MSG_CHECK_DRIVER_UPGRADE. In that case, *szName* contains the name of driver to be upgraded.

pInfo

Pointer to a structure containing upgrade information. When *Reason* is DSETUP_CB_MSG_CHECK_DRIVER_UPGRADE, the setup program is in the process of upgrading a driver and asking the user whether the upgrade should take place. This structure contains information about the upgrade in its **UpgradeType** member, which can have the following values.

DSETUP_CB_UPGRADE_CANTBACKUP

The old system components can’t be backed up. Upgrade can be performed, but the components or drivers can’t be restored later.

DSETUP_CB_UPGRADE_DEVICE_ACTIVE

The device is currently in use.

DSETUP_CB_UPGRADE_DEVICE_DISPLAY

The device driver being upgraded is for a display device.

DSETUP_CB_UPGRADE_DEVICE_MEDIA

The device driver being upgraded is for a media device.

DSETUP_CB_UPGRADE_FORCE

Windows may not function correctly if the component is not upgraded. The upgrade will be performed.

DSETUP_CB_UPGRADE_HASWARNINGS

DirectXSetup can upgrade the driver for this device, but doing so may affect one or more programs on the system. *szMessage* contains the names of which programs may be affected. Upgrade not recommended.

DSETUP_CB_UPGRADE_KEEP

The system may fail if this device driver is upgraded. Upgrade not allowed.

DSETUP_CB_UPGRADE_SAFE

DirectXSetup can safely upgrade this device driver. Upgrade recommended. A safe upgrade will not adversely affect the operation of Windows. Some hardware-dependent programs may be adversely affected.

DSETUP_CB_UPGRADE_UNKNOWN

DirectXSetup does not recognize the existing driver for this device. This value will occur frequently. Upgrading may adversely affect the use of the device. It is strongly recommended that the upgrade not be performed.

DSETUP_CB_UPGRADE_UNNECESSARY

The existing device driver is newer than the driver being installed. An upgrade is not recommended.

Return Values

The return value should be the same as the **MessageBox** function, with one exception. If this function returns zero, the **DirectXSetup** function will act as if no callback function was present. That is, it will perform the default action for upgrade of the DirectX component or driver.

Remarks

The name of the **DirectXSetupCallbackFunction** is supplied by the setup program. The **DirectXSetupSetCallback** function is used to pass the address of the callback function to DirectSetup.

If *MsgType* is equal to zero, the setup program may display status information, but it should not wait for user input. This is useful for displaying ongoing status information.

See Also

MessageBox, **DirectXSetupSetCallback,** Customizing Setup With the DirectSetup Callback Function

DirectXUnRegisterApplication

The **DirectXUnRegisterApplication** function deletes the registration of an application designed to work with DirectPlayLobby.

int WINAPI DirectXUnRegisterApplication(
  HWND *hWnd*,
  LPGUID *lpGUID*
 );

Parameters

hWnd
Handle to the parent window. Set this to NULL if the desktop is the parent window.

lpGUID
Pointer to a GUID that represents the DirectPlay application to be unregistered.

Return Values

If the function succeeds, the return value is TRUE meaning that the registration is successfully deleted.

If the function fails, the return value is FALSE.

Remarks

The **DirectXUnRegisterApplication** function removes registry the entries needed for an application to work with DirectPlayLobby. An uninstall program should only use **DirectXUnRegisterApplication** if it used **DirectXRegisterApplication** when the application was installed.

See Also

DirectXRegisterApplication

Structures

This section contains information about the following structures used with DirectSetup:

- **DIRECTXREGISTERAPP**
- **DSETUP_CB_UPGRADEINFO**

DIRECTXREGISTERAPP

The **DIRECTXREGISTERAPP** structure contains the registry entries needed for applications designed to work with DirectPlayLobby.

```
typedef struct _DIRECTXREGISTERAPP {
    DWORD  dwSize;
    DWORD  dwFlags;
    LPSTR  lpszApplicationName;
    LPGUID lpGUID;
    LPSTR  lpszFilename;
    LPSTR  lpszCommandLine;
    LPSTR  lpszPath;
    LPSTR  lpszCurrentDirectory;
} DIRECTXREGISTERAPP, *PDIRECTXREGISTERAPP, *LPDIRECTXREGISTERAPP;
```

Members

dwSize
: Size of the structure. Must be initialized to the size of the **DIRECTXREGISTERAPP** structure.

dwFlags
: Reserved for future use.

lpszApplicationName
: Name of the application.

lpGUID
: Globally unique identifier (GUID) of the application.

lpszFilename
: Name of the executable file to be called.

lpszCommandLine
: Command-line arguments for the executable file.

lpszPath
: Path of the executable file.

lpszCurrentDirectory
: Current directory. This is typically the same as **lpszPath**.

DSETUP_CB_UPGRADEINFO

The **DSETUP_CB_UPGRADEINFO** structure is passed as a parameter to the **DirectXSetupCallbackFunction**. It only contains valid information when the *Reason* parameter is DSETUP_CB_MSG_CHECK_DRIVER_UPGRADE. Callback functions can use it to get status information on the upgrade that is about to be done.

```
typedef struct _DSETUP_CB_UPGRADEINFO
{
  DWORD UpgradeFlags;
} DSETUP_CB_UPGRADEINFO;
```

Members

UpgradeFlags
A flag indicating the status of the upgrade. See the *pInfo* parameter of the **DirectXSetupCallbackFunction** function for details.

See Also

DirectXSetupCallbackFunction

Return Values

The **DirectXSetup** function can return the values listed blow. It can also return a standard COM error.

DSETUPERR_SUCCESS	Setup was successful and no restart is required.
DSETUPERR_SUCCESS_RESTART	Setup was successful and a restart is required.
DSETUPERR_BADSOURCESIZE	A file's size could not be verified or was incorrect.
DSETUPERR_BADSOURCETIME	A file's date and time could not be verified or were incorrect.
DSETUPERR_BADWINDOWSVERSION	DirectX does not support the Windows version on the system.
DSETUPERR_CANTFINDDIR	The setup program could not find the working directory.
DSETUPERR_CANTFINDINF	A required .inf file could not be found.
DSETUPERR_INTERNAL	An internal error occurred.
DSETUPERR_NOCOPY	A file's version could not be verified or was incorrect.
DSETUPERR_NOTPREINSTALLEDONNT	The version of Windows NT on the system does not contain the current version of DirectX. An older version of DirectX may be present, or DirectX may be absent altogether.

DSETUPERR_OUTOFDISKSPACE

The setup program ran out of disk space during installation.

DSETUPERR_SOURCEFILENOTFOUND

One of the required source files could not be found.

DSETUPERR_UNKNOWNOS

The operating system on your system is not currently supported.

DSETUPERR_USERHITCANCEL

The **Cancel** button was pressed before the application was fully installed.