Audio Tapers for Windows Volume‑Control Applications

Abstract

The audio taper of a volume control slider determines how the voltage level of the audio output signal changes as the user moves the slider through each increment in its range. In Windows® XP Service Pack 2 (SP2) and later service packs, the audio taper that Windows uses is stored as a table in the system registry and is available to third-party on-screen volume displays and other volume-control applications. By using this table, these applications can implement volume controls that behave much like the native Windows controls.

This paper provides information about how to implement audio volume-level controls for Windows XP SP2 and later service packs. It provides guidelines for developers of on-screen volume displays to implement volume controls with custom audio tapers that match those of the system-supplied displays for the keyboard volume buttons and Windows XP Media Center Edition remote controls.

**Note:** The procedures for implementing custom audio tapers that is discussed in this paper should not be used for applications that are running on Windows Vista® and later versions. Applications that run on those versions of Windows should instead use the audio endpoint volume interfaces, which automatically provide the appropriate audio tapers.

This information applies only to the Windows XP operating system.

For the latest information, see:   
 <http://www.microsoft.com/whdc/device/audio/VolTable.mspx>

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2006–2008 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Vista, and Windows XP are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

| **Date** | **Change** |
| --- | --- |
| October 27, 2008 | Updated supported Windows versions and performed minor editing. |
| January 16, 2008 | Updated trademarking, removed outdated reference, added Web page information. |
| January 30, 2006 | First publication |

Contents

[Introduction 3](#_Toc212887004)

[Volume Controls and Loudness 4](#_Toc212887005)

[Audio Taper versus Linear Taper 5](#_Toc212887006)

[Changes in Windows XP with SP2 7](#_Toc212887007)

[Implementation Details 7](#_Toc212887008)

[Application Developers 7](#_Toc212887009)

[Audio Driver Developers 8](#_Toc212887010)

[Enabling and Disabling the Volume Table 9](#_Toc212887011)

[Sample Code 10](#_Toc212887012)

[Call to Action 15](#_Toc212887013)

[Resources 16](#_Toc212887014)

# Introduction

When a user adjusts audio volume, ideally there should be a linear relationship between the perceived change in loudness and the distance that the audio control slider has moved, regardless of the control’s starting position. However, the relationship between output voltage and loudness is not linear. A simple linear relationship between the position of the control and the output voltage results in a significantly nonlinear relationship between the slider position and the perceived volume.

An audio taper is a curve that specifies appropriate voltage levels for the entire range of the volume control so that the relationship between the position of the slider and the perceived audio volume is approximately linear over the range of the control.

Beginning with Windows® XP Service Pack 2 (SP2), Windows stores the audio taperfor its volume controls as a volume table in the system registry. The Windows keyboard volume buttons and the Windows XP Media Center Edition remote controls use the volume table to govern the behavior of their volume controls. Applications can implement volume controls that use this audio taper, so that they behave much like the native Windows controls. The appropriate implementation depends on the Windows version, as follows:

* For Windows XP with SP2 and later service packs, applications read the volume table from the registry and use that data to create a custom audio taper for their controls. This paper describes how to use the volume table to implement custom audio tapers and includes sample code.
* For Windows Vista and later versions, applications should not implement custom audio tapers. They should instead use the audio endpoint volume interfaces, which automatically provide the correct behavior. For more information, on these interfaces, see “Endpoint Volume Controls.”

# Volume Controls and Loudness

Typically, a volume control in an on-screen volume display appears in the form of a slider. The user moves the slider up and down by pushing the volume-up or volume-down button on the keyboard or remote control. Figure 1 shows the on-screen volume display for the remote control in Windows XP Media Center Edition.



Figure 1. On-screen volume display

In Figure 1, the volume display appears on the right side of the screen. Similar to the volume displays on many televisions, a Windows on-screen volume display pops up when the user begins to adjust the volume and automatically disappears several seconds after the user makes the last adjustment. The display is typically semi-transparent.

For the best user experience, each change in slider position should produce a change in perceived loudness that is proportional to the distance that the slider travels. For a particular travel distance, the amount by which the perceived loudness increases or decreases should be the same regardless of whether the slider movement occurs in the lower, upper, or middle portion of the slider's range of movement. Perceived loudness varies approximately linearly with the logarithm of the audio signal power. The term *audio taper* indicates that the curve that maps volume slider position to signal voltage produces a linear relationship between slider position and perceived loudness.

By reading the volume table from the registry, third-party on-screen volume display applications can use the same audio taper as Windows does in the displays for its keyboard and remote-control volume buttons. Third-party applications can improve the user’s experience by providing volume controls that behave consistently with those of the operating system.

Use of the volume table in the registry is optional, but recommended. In Windows XP with SP2 and later service packs, volume-control applications that do not use the table continue to run, but behave as if they were running in an earlier version of Windows without volume table support.

Windows includes a volume-control application that lets users control the volume of the system audio mix and of the individual audio sources that contribute to that mix. However, the behavior of this application depends on the Windows version, as follows:

* In Windows XP and earlier versions of Windows, including Windows XP SP2, the volume control application—SndVol32—does *not* use the volume table in the registry. Instead, slider movements in SndVol32 produce linear changes in output voltage, which means that perceived loudness varies nonlinearly with slider movements.
* In Windows Vista and later versions, the system-supplied volume control application reads the volume table from the system registry and implements volume controls with audio tapers.

# Audio Taper versus Linear Taper

The term *audio taper* originally referred to the tapered shape of the resistive element in a potentiometer that is used as a volume control in an audio electronics device. An audio-tapered resistive element is widest at the zero-volume position and narrowest at the maximum-volume position. The potentiometer controls the voltage level of the audio signal that the device plays through its speakers. The tapering produces a roughly linear relationship between the position of the potentiometer wiper and the perceived loudness at the speakers. Note that the tapering produces a *non*linear relationship between the wiper position and the voltage.

In contrast, a resistive element that has a *linear taper* has a uniform width over the potentiometer wiper’s range of movement. Therefore, the voltage at the wiper varies linearly with the wiper position.

In a Windows on-screen volume display, the relationship of the volume-slider position to the voltage level of the audio signal can effectively be tapered to provide a roughly linear relationship between slider position and perceived loudness.

In the initial release of Windows XP Media Center Edition, the taper of the volume control in the Windows on-screen volume display is defined by a fixed table that can be changed only by rebuilding parts of the operating system. In addition, the taper is linear, as shown on the left side of Figure 2.

Figure 2. Linear taper

On the left side of Figure 2, the output voltage level of the audio digital-to-analog converter (DAC) increases linearly as the volume slider moves from its minimum position to its maximum position (labeled *Min* and *Max*). Label *V****FS*** on the vertical axis represents the full-scale DAC output voltage.

However, perceived loudness varies approximately as the logarithm of the power of the audio signal, as shown on the right side of Figure 2. Therefore, movement of the slider over an interval near the minimum setting results in a relatively large change in perceived loudness, but slider movement over an interval of the same width near the maximum setting causes a relatively small change in perceived loudness.

On the right side of Figure 2, loudness on the vertical axis is measured in decibels (dB) relative to the full-scale power setting (at 0 dB). The loudness curve intersects the vertical axis at minus infinity, but only the portion of the curve from 0 dB to ‑96 dB appears in the figure. The decision to show only this portion of the curve is somewhat arbitrary, but -96 dB conveniently represents the power at the next-to-lowest output level of a 16-bit DAC relative to the full-scale power. In other words, this value is 20\*log10(1/65535).

Because small changes in slider position near the minimum setting in Figure 2 result in large changes in loudness, the user might find the volume difficult to control over this region. Relatively small slider movements can push the volume well above or below the desired level. A more ideal volume control would provide a linear relationship between slider position and loudness.

## Changes in Windows XP with SP2

Beginning with Windows XP SP2, a volume table in the registry specifies the taper of the volume controls in the on-screen displays for the keyboard and remote-control volume buttons. The table can specify the taper for a volume slider that varies the perceived loudness approximately linearly with changes in slider position, as shown on the right side of Figure 3.



Figure 3. Audio taper

For loudness to vary linearly with the slider position, the DAC voltage must vary nonlinearly with position, as shown on the left side of Figure 3. The curve asymptotically approaches 0 volts as the slider moves leftward from the maximum setting. The usual convention is that the voltage is exactly zero at the minimum slider position.

# Implementation Details

This section provides implementation details for developers of volume-control applications. It also presents guidelines for developers of audio drivers to follow in implementing volume controls.

## Application Developers

A volume slider can be implemented as a Windows control known as a *trackbar*. Part of the trackbar specification is the number of ticks from the minimum position to the maximum position. Therefore, the travel range of a trackbar with *n* ticks is divided into *n*-1 uniform intervals. Windows specifies the slider position as a discrete tick number in the range 0 to *n*-1. For more information about the trackbar control, see “Resources” at the end of this paper.

The volume table in Windows XP SP2 and SP3 contains 26 entries that correspond to a volume slider that is implemented as a trackbar with 26 ticks over its range of movement. The table size might change in future Windows XP updates. The value of the first table entry is 0, which indicates that the minimum slider position corresponds to 0 volts. The last table entry is 65535, which indicates that the maximum slider position corresponds to the full-scale voltage—V**FS**. The remaining table entries define a curve similar to the one shown on the left side of Figure 3.

By representing the full-scale voltage—V**FS**—as 65535, the volume table is consistent with the **mixer** API, which assigns the value 65535 to the maximum volume setting. The **mixer** API assigns 0 to the minimum volume setting, and values between 0 and 65535 map linearly to intermediate voltage levels. Therefore, the volume table expresses the volume settings in a convenient form that can be passed directly to the **mixer** API without conversion by the volume-control application.

An application can get or set an audio adapter’s volume level by calling **mixer** API **mixerGetControlDetails** and **mixerSetControlDetails** functions. These functions use the MIXERCONTROLDETAILS structure to pass control information between the application and audio device. For more information, see the Platform Software Development Kit (SDK) documentation.

In Windows XP with SP2 or SP3, the registry location of the volume control table is the HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Multimedia\Audio\  
VolumeControl key. This key contains two values that are named **VolumeTable** and **EnableVolumeTable**:

* **VolumeTable** is a REG\_BINARY value that contains the volume table as a sequence of monotonically increasing 32-bit ULONG values. Each 32-bit value represents a value in the range 0 to 65535.
* **EnableVolumeTable** is a REG\_DWORD value that enables and disables use of the volume table. A value of zero disables table use. A nonzero value enables use of the table.

Each volume table entry must have a size of exactly 4 bytes, so that the total size of the table must be a multiple of 4 bytes. The table must contain at least 11 entries but never more than 201 entries. The number of table entries is implied by the total size of the table. Volume-control applications that honor these conventions will continue to operate correctly over later Windows releases and upgrades. As mentioned earlier, Windows XP SP2 and SP3 populate this table with 26 entries.

The on-screen volume display mirrors the relationship of the table value that most closely represents the current volume level to the total number of values in the table. After the application uses the **mixer** API to retrieve the current volume level, the application finds the lowest position in the table whose value is greater than or equal to this level. This table position corresponds to the position that is shown on the volume display. The table size indicates the total number of positions in the display.

**Important:** The volume-table registry settings are managed by Windows components. Application software must not modify these registry settings.

## Audio Driver Developers

The Windows multimedia **mixer** API sends KSPROPERTY\_AUDIO\_VOLUMELEVEL property requests to the audio driver to control and monitor the volume-level setting of the wave output device. To avoid having the **mixer** API misrepresent volume settings to applications, the KSPROPERTY\_AUDIO\_VOLUMELEVEL handler in an audio driver should represent volume levels according to the guidelines in the Windows Driver Kit (WDK) documentation.

Some hardware vendors have violated these guidelines in their drivers in an attempt to compensate for the linear tapers used by volume-control applications in older Windows versions. The property handlers in these drivers bias their volume-level settings to affect an audio-tapered volume control in an application that implements a linear-tapered volume control. However, this unspecified driver behavior can produce unforeseen side-effects. In Windows XP with SP2 and later, when the biased volume settings that these drivers produce are used by applications that implement volume controls with audio tapers, the result is an exaggerated audio taper and therefore a poor user experience.

A vendor that chooses to bias the volume settings in violation of the guidelines can at least try to reduce the risk of undesirable behavior. In particular, a driver that might otherwise bias its volume settings should check the system registry to determine whether the volume table is enabled. In Windows XP with SP2 and later service packs, if the volume table is enabled, the driver can avoid biasing its volume settings and therefore work compatibly with applications that implement volume controls with audio tapers.

Several methods can be used to determine whether the volume table is enabled. One difficulty that the driver might encounter is that the part of the registry that contains the table might not be available early in the Windows boot process. Depending on the driver's existing design, the developer can choose among the following options for determining whether the table is enabled:

* At the time that the driver receives its first KSPROPERTY\_AUDIO\_VOLUMELEVEL property request, it can check whether the registry is ready. If the registry is not ready, the driver can continue to use its default volume behavior, which preferably follows the published guidelines. The next time that the driver receives a KSPROPERTY\_AUDIO\_VOLUMELEVEL request, it can try again to access the registry.
* The developer can write a co-installer that reads and propagates the registry settings from HKEY\_LOCAL\_MACHINE\SOFTWARE to the device registry key, which is ready during system boot.
* If the audio solution includes a user-mode process (for example, an NT service or some other process that is always running), then this process can communicate the setting to the driver. The user-mode process can also be useful for monitoring changes in the volume-table registry settings.

For more information about co-installers and installing NT services, see the WDK.

**Note:** For Windows Vista and later, audio drivers should never implement their own audio taper. Instead, they should implement the kernel streaming volume interfaces as accurately as possible. For more information, see “Audio Devices Design Guide.”

## Enabling and Disabling the Volume Table

Associated with the volume table in the registry is a registry value that enables and disables use of the volume table. If an on-screen volume display application finds the table disabled, it ignores the table and behaves as if it were running in an earlier version of Windows that does not support the table. The sample code presented later in this paper shows how to determine whether the table is enabled or disabled.

Users who download Windows XP SP2 to their Home, Professional, or Media Center Edition systems find that, although the service pack creates a volume table in the registry, the table is disabled by default. Therefore, users will notice no change in the behavior of their on-screen volume display applications after they download the service pack. The reason for disabling the table by default is to avoid surprising users with possibly unexpected user-interface changes. (For example, the audio driver might bias its speaker volume settings, as described in the preceding section.) However, systems manufacturers can enable the volume table in the systems that they produce.

In Windows Media Center Edition (MCE) 2005, the volume table is enabled by default, and on-screen volume display applications that use the table present volume controls with audio tapers to users. This policy is consistent with the prominent role that is played by volume controls in a media-centric operating system such as Windows MCE 2005. Typical users should find the audio-tapered controls similar in behavior to the on-screen volume displays they are accustomed to seeing on their television sets.

Windows Vista and later versions include the volume table in the registry, and it is enabled by default.

# Sample Code

The following sample code is written in C++. It implements a **CVolumePositions** class that a volume-control application can use to map volume-control positions to output voltage levels, and vice versa. Voltage levels are expressed in a format that is directly compatible with the Windows multimedia **mixer** API. The application calls the **CVolumePositions::GetPosition** method to convert a volume-level value from the **mixer** API to a volume-control position. In Windows XP with SP2 and later service packs, the **CVolumePositions** class retrieves the volume table from the system registry. If the **CVolumePositions** class runs in an earlier version of Windows that does not support this table, the class provides the application with a default table that defines a linear taper that mimics the on-screen volume display’s behavior in these versions of Windows:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Volume positions class

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CVolumePositions

{

public:

CVolumePositions();

~CVolumePositions();

int GetPosition(ULONG volume);

int ivolumesLim; // Current size of the volume

// table in use

private:

void \_RegReadSettings();

void \_RefreshSettings();

//

// The default volume table. The volume table contains

// values to program the mixer API. If no table is stored

// in the registry then this table is used.

//

const static ULONG \_volumesDefault[];

//

// The registry key and value names for our settings.

//

const static WCHAR \_szVolumesKey[];

const static WCHAR \_szVolumesName[];

const static WCHAR \_szVolumesEnableName[];

//

// Following are used to validate the registry settings.

//

const static int \_cvolumesMax; // Maximum allowed size of

// a custom table

// (exclusive)

const static int \_cvolumesMin; // Minimum allowed size of

// a custom table

// (exclusive)

const static ULONG \_volumeLast; // Highest possible volume

// level

//

// Registry settings

//

HKEY \_hkeyVolumes; // Handle to registry key

// containing settings

HANDLE \_hVolumesChanged; // Registry-change event:

// Reg API sets this whenever

// settings key is written.

ULONG \*\_volumesCustom; // Custom volumes table read from

// registry

//

// Current settings that are in effect

//

const ULONG \*\_volumes; // Pointer to current volume table

// in use (default or custom)

};

//-----------------------------------------------------------

//

// The registry key and value names for our settings.

//

//-----------------------------------------------------------

const WCHAR CVolumePositions::\_szVolumesKey[] =

L"Software\\Microsoft\\Multimedia\\Audio\\VolumeControl";

const WCHAR CVolumePositions::\_szVolumesName[] =

L"VolumeTable";

const WCHAR CVolumePositions::\_szVolumesEnableName[] =

L"EnableVolumeTable";

//-----------------------------------------------------------

//

// Limitations on the registry settings, used to validate the

// settings. Table must have from 11 to 201 entries. Highest

// possible value is 65535. (Note the max and min constants

// below are exclusive, meaning the count of volumes must not

// equal these values.)

//

//-----------------------------------------------------------

const int CVolumePositions::\_cvolumesMax = 202;

const int CVolumePositions::\_cvolumesMin = 10;

const ULONG CVolumePositions::\_volumeLast = 65535;

//-----------------------------------------------------------

//

// The default table is compatible with the behavior of

// previous Windows releases. It has min volume, twenty-four

// equal increments in voltage gain leading up to max volume.

//

//-----------------------------------------------------------

const ULONG CVolumePositions::\_volumesDefault[] = {

0,2621,5243,7864,10486,13107,15728,18350,20971,23593,26214,

28835,31457,34078,36700,39321,41942,44564,47185,49807,

52428,55049,57671,60292,62914,65535

};

CVolumePositions::CVolumePositions() :

\_hkeyVolumes(NULL),

\_hVolumesChanged(NULL),

\_volumesCustom(NULL)

{

ULONG error;

// Attempt to open the registry key containing our settings

// and register a registry-change notification event.

// Failure to open the registry key is not critical and

// will cause this object to use hard-coded settings.

// Failure to register a notification event also is not

// critical and means that this object will not respond to

// settings changes.

error = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, \_szVolumesKey, 0,

KEY\_QUERY\_VALUE | KEY\_NOTIFY,

&\_hkeyVolumes);

if (ERROR\_SUCCESS == error)

{

\_hVolumesChanged = CreateEvent(NULL, TRUE, FALSE, NULL);

if (\_hVolumesChanged)

{

error = RegNotifyChangeKeyValue(\_hkeyVolumes, FALSE,

REG\_NOTIFY\_CHANGE\_LAST\_SET,

\_hVolumesChanged, TRUE);

}

}

// Initial read of settings

\_RegReadSettings();

return;

}

CVolumePositions::~CVolumePositions()

{

if (\_volumesCustom) LocalFree(\_volumesCustom);

if (\_hVolumesChanged) CloseHandle(\_hVolumesChanged);

if (\_hkeyVolumes) RegCloseKey(\_hkeyVolumes);

}

int CVolumePositions::GetPosition(ULONG volume)

{

\_RefreshSettings();

// Starting at the lowest position, find the first

// position that has higher or equal volume.

for (int i = 0; i < ivolumesLim-1; i++)

{

if (volume <= \_volumes[i]) break;

}

return i;

}

void CVolumePositions::\_RegReadSettings()

{

// Attempt to read any existing custom volume positions

// from the registry but if there are any errors, just

// use the hard-coded defaults instead.

// Free any existing custom table.

if (\_volumesCustom) LocalFree(\_volumesCustom);

\_volumesCustom = NULL;

if (\_hkeyVolumes)

{

DWORD cbValue;

DWORD typeValue;

DWORD dwEnable;

ULONG error;

// Check whether the table is enabled.

cbValue = sizeof(dwEnable);

error = RegQueryValueEx(\_hkeyVolumes,

\_szVolumesEnableName, NULL, &typeValue,

(BYTE\*)&dwEnable, &cbValue);

if ((ERROR\_SUCCESS == error) &&

(REG\_DWORD == typeValue) &&

(sizeof(dwEnable) == cbValue) &&

(0 != dwEnable))

{

// Get the size of the table data.

error = RegQueryValueEx(\_hkeyVolumes, \_szVolumesName,

NULL, &typeValue, NULL, &cbValue);

if ((ERROR\_SUCCESS == error) &&

(REG\_BINARY == typeValue) &&

(0 == (cbValue % sizeof(\_volumesCustom[0]))))

{

// Allocate memory to hold the table.

\_volumesCustom = (DWORD\*)LocalAlloc(LPTR, cbValue);

if (\_volumesCustom)

{

BOOL fValid = FALSE;

// Read the table from the registry.

error = RegQueryValueEx(\_hkeyVolumes,

\_szVolumesName, NULL, &typeValue,

(BYTE\*)\_volumesCustom, &cbValue);

if ((ERROR\_SUCCESS == error) &&

(REG\_BINARY == typeValue) &&

(0 == cbValue % sizeof(\_volumesCustom[0])))

{

// The number of volume positions in the table

int cvolumes = cbValue/sizeof(\_volumesCustom[0]);

// Validate the custom volume positions in the

// table. There must be an acceptable number of

// entries, entries must be monotonically

// increasing, and no value can be greater than

// the maximum volume level.

if (cvolumes > \_cvolumesMin &&

cvolumes < \_cvolumesMax)

{

ivolumesLim = cvolumes;

for (int ivolumes = 1; ivolumes < ivolumesLim;

ivolumes++)

{

if (\_volumesCustom[ivolumes] <=

\_volumesCustom[ivolumes-1])

break;

if (\_volumesCustom[ivolumes] > \_volumeLast)

break;

}

if (ivolumes == ivolumesLim) fValid = TRUE;

}

}

if (!fValid)

{

LocalFree(\_volumesCustom);

\_volumesCustom = NULL;

}

}

}

}

}

// If we have custom volumes, use those.

// If not, use hard-coded volumes.

if (\_volumesCustom)

{

\_volumes = \_volumesCustom;

}

else

{

\_volumes = \_volumesDefault;

ivolumesLim = ARRAYSIZE(\_volumesDefault);

}

}

void CVolumePositions::\_RefreshSettings()

{

ASSERT(\_volumes);

//

// If the registry API has signaled the \_hVolumesChanged

// event, then the settings key has been written since we

// last read it. In that case, reset the event, re-register

// the event, and re-read our settings.

//

if (\_hVolumesChanged &&

(WAIT\_OBJECT\_0 == WaitForSingleObject(\_hVolumesChanged,

0)))

{

ResetEvent(\_hVolumesChanged);

RegNotifyChangeKeyValue(\_hkeyVolumes, FALSE,

REG\_NOTIFY\_CHANGE\_LAST\_SET,

\_hVolumesChanged, TRUE);

\_RegReadSettings();

}

}

The ARRAYSIZE macro, which generates a count of the number of elements in an array **a**, is defined as follows:

#define ARRAYSIZE(a) (sizeof(a)/sizeof(a[0]))

Before it uses the volume table in the registry, the preceding sample code verifies the following:

* The volume table exists.
* The volume table is enabled.
* The table data is of type REG\_BINARY.
* The table size is a multiple of 4 bytes.
* The number of table entries is greater than 10 but less than 202.
* All table entries have values in the range 0 to 65535.
* Successive table entries have monotonically increasing values.

If any of the preceding conditions are not satisfied, the sample code uses a hard-coded, default volume table instead of the volume table in the registry.

In addition, the sample code receives notifications of any change to the registry key that contains the volume table. If any table entries change or if table use is enabled or disabled, the **CVolumePositions** class updates its internal volume table accordingly. Therefore, a volume-control application that uses this class can immediately reflect changes that were made to the volume table in the registry at run time.

# Call to Action

Manufacturers and developers who implement applications or drivers for Windows XP SP2 and later service packs should take the following steps:

System Manufacturers

Update the volume-control applications for your systems to use the volume table in the system registry when it is running on Windows XP SP2 and later service packs.

Device Manufacturers

Update the volume-control applications for your audio devices to use the volume table in the system registry when running on Windows XP SP2 and later service packs.

Update drivers that bias audio settings to compensate for a lack of audio taper in earlier versions of Windows to disable that bias on Windows XP SP2 and later service packs if the volume table is enabled.

Application Developers

Update your volume-control applications to use the volume table in the system registry in Windows XP SP2 and later service packs.

If you have questions about using the volume table in the system registry in Windows XP SP2 and later service packs, send e-mail to [wmadrv@microsoft.com](mailto:wmadrv@microsoft.com?subject=Volume%20table%20in%20system%20registry).

# Resources

Windows Hardware and Driver Central

The Windows Hardware and Driver Central (WHDC) Web site at <http://www.microsoft.com/whdc/default.mspx> provides a wide range of information on hardware and driver development, including numerous white papers and information about the WDK and the Windows Logo Program.

Windows Driver Kit

The WDK contains the documentation and tools that are used to implement device drivers. For information on how to obtain the WDK, see the WDK Web site at <http://www.microsoft.com/whdc/driver/WDK/aboutWDK.mspx>

Trackbar

<http://msdn2.microsoft.com/en-us/library/bb760145(VS.85).aspx>

Endpoint Volume Controls

<http://msdn2.microsoft.com/en-us/library/bb331828(VS.85).aspx>

Audio Devices Design Guide

<http://msdn2.microsoft.com/en-us/library/ms790360.aspx>