

ANNUAL GRAPHICS ISSUE!

#202 JULY 1993

Dr. Dobb's

JOURNAL

**UNLEASHING THE POWER OF
COMPUTER GRAPHICS**

**MORPHING
PALETTE MAPPING
IMAGE PROCESSING
COLOR MODELS**

**OUR OWN
WINDOWS MULTIMEDIA
CLASS LIBRARY**

**D-ROM
TENSIONS**

**DEBUGGING
REAL-TIME
PROGRAMS**

**DISTRIBUTED
COMPUTING**

(\$4.95 CANADA)



8351 16562 8

UNIVERSITY OF WASHINGTON

JUN 17 1993

LIBRARIES

ENGINEERING
PERIODICAL
DISPLAY

**ALGORITHMS
FOR DATA
COMPRESSION**



A Multimedia Class Library for Windows

Encapsulating the Media Control Interface

John Musser

Microsoft Windows' Media Control Interface (MCI) is a standard command language for communicating with multimedia devices—CD, Waveform and MIDI audio, AVI, videodiscs, video overlay devices, audio mixers, and the like. However, even though both Microsoft and Borland provide with their compilers large, comprehensive class libraries—the Microsoft Foundation Class library and ObjectWindows Library, respectively—neither provide object support for multimedia or Windows API multimedia extensions.

This article addresses this gap by showing how to design and implement a comprehensive C++ class library that enhances the MCI interface to multimedia devices. This hierarchy employs encapsulation, inheritance, and polymorphism to create a flexible and extensible framework for controlling multimedia devices under Windows. The result is a set of objects that make programming multimedia easier, more robust, concise, and maintainable. A simple client program, MciMan, demonstrates the use of the Waveform audio and AVI video classes. The class library and client program are compiler independent and can be used with any Windows 3.1 compatible C++ compiler, including Borland's C/C++ and Microsoft's Visual C++. The AVI portions of this code require the digitalv.h #include file, Video for Windows drivers, and runtime

John is a senior developer at Personal Media International, a startup specializing in multimedia entertainment software. He can be reached via CompuServe at 70621,1460.

DLLs that come with the Video for Windows package and its SDK. They're available free of charge from the Windows Extensions forum on CompuServe (GO WINEXT). Vfwrun.zip contains the runtime DLLs and vfwdk.zip is the Video for Windows Development Kit.

MCI Overview

The MCI specification, released in 1991 by Microsoft and IBM, defines a set of base commands that can be applied to any general device, and extended commands for specific device types. The specification is designed to be extensible so that other devices may be added. For example, the AVI specification was

added in 1992. Another example is MediaVision, which supplies an MCI driver that provides audio-mixing capabilities using extended commands specific to this device type. The specification as documented in the Windows SDK identifies 11 device types. Some drivers are supplied with either Windows or the SDK, while others are provided by the device supplier. Table 1 describes most of the currently available MCI drivers. As of this writing, however, not all the device types listed actually had MCI drivers available.

MCI divides all multimedia devices into one of two device types: simple or compound. The basic difference is that sim-

Device Type	Description	Driver Files	Driver Source
animation	Plays Autodesk Animator (.flc/.fli) files	MCIAAP.DRV	Autodesk
	Plays MacroMind Director (.mmm) files	MCIMMP.DRV	MDK, Visual Basic
cdaudio	Controls compact disc audio	MCICDA.DRV	Windows 3.1 SDK, MDK
dat	Controls Digital Audio Tape deck		
digitalvideo	Plays AVI video files	MCIavi.DRV	Video for Windows
other	An undefined MCI device		
overlay	Controls a video overlay device—analog video in a window	MCIVBLST.DRV	Creative Labs
scanner	Controls an image scanner		
sequencer	Plays MIDI audio files	MCISEQ.DRV	Retail Windows
vcr	Controls a video cassette recorder		
videodisc	Controls the Pioneer LD-V4200 videodisc player	MCIPIONR.DRV	Windows 3.1 SDK, MDK
waveaudio	Plays and records waveform audio files	MCIWAVE.DRV	Retail Windows

Table 1: Several Windows MCI devices.

We slash interface development time across DOS, UNIX, POSIX, VMS... (and we can prove it!)



C-Programmers: See for yourself how Vermont Views® can help you create powerful user interfaces—whatever your environment!

If you want to create sophisticated user interfaces—and save tremendous time and effort doing it—Vermont Views is exactly what you need.

Vermont Views isn't just a common interface package.

It's a deep, flexible, menu-driven screen designer supported by a C library of over 580 functions. It lets you create the ultimate user interfaces for complex database applications - in a fraction of the time it would take to code it yourself!

With Vermont Views, you create screens interactively. Designing is fast and creative. And changes—both tiny adjustments and huge reworks—are incredibly easy.

Pull down menus, window-based data entry forms with tickertape or memo fields, scrollable regions, choice lists, context sensitive help All these interface objects (and more) are immediately accessible. Your

DOS applications can have full mouse control, and work in graphics as well as text modes! And with Vermont Views, even terminal-based applications can have the elegant features usually found only on micros.

Create prototypes fast, applications even faster.

With most systems, you have to throw away your prototypes when coding begins. But with Vermont Views, *prototypes become the actual application.*

Menus, data-entry forms, and all screen features are usable in the final applications without change.

NEW! Vermont Views PLUS

An enhanced version of Vermont Views for DOS. Vermont Views PLUS provides the interface power needed for developing multi-mega-byte, graphic-enhanced C applications. Best of all it supports all popular DOS extenders and graphics libraries. VVPLUS also includes full source code for all libraries.

It's the universal solution.

Vermont Views operates completely independent of hardware, operating system, and database.

Any interface you create can be ported easily among DOS, UNIX, XENIX, POSIX, QNX, OS/2, and VMS.

You can use Vermont Views with any database that has a C-language interface (including Oracle, Informix, db_Vista, and C-Tree). You can run on PCs, DEC, NCR, HP, AT&T, and other systems.

No runtime fees or royalties.

Don't take our word for it—put Vermont Views to the test. Call or fax now for your personal, free demonstration kit. Or order Vermont Views with our 60-day, money back guarantee.

Either way, you'll immediately see how Vermont Views can slash your interface development time.

For MS/PCDOS, Vermont Views is \$495. Vermont Views PLUS with MemEx, GraphEx and full source code is \$795. For UNIX, POSIX, and VMS, prices start at \$1,795, depending on machine class.



**Vermont
Creative
Software**

1 Pinnacle Meadows, Richford, VT 05476
Phone (802)-848-7731 FAX (802)-848-3502
Please Mention "Offer 396"

Call 1-800-848-1248 for a free demo of the ultimate application interface!

CIRCLE NO. 529 ON READER SERVICE CARD

(continued from page 84)

ple devices do not use data files, whereas compound devices usually do. CD audio and videodisc players are simple devices; waveform audio, MIDI sequencers, and AVI are all compound devices.

MCI provides two basic programming interfaces: a command-string interface that allows the use of ordinary text strings such as *play cdaudio* to control multimedia devices (this very open approach is well suited to scripting and authoring applications); and a command-message interface that uses C-language structures and a Windows-style message-passing model for device control. The MCI classes I describe here use the com-

mand-message interface because it's more efficient and better suited to general programming.

A single function, *mciSendCommand()*, is used along with a set of "polymorphic" arguments to access the command-message interface. The *mciSendCommand()* function takes:

- A WORD device identifier (analogous to a handle).
- A message-type constant prefixed with MCI_ (such as MCI_PLAY).
- A DWORD value set to one or more flags usually specifying which elements of a given structure contain valid values.

- A far pointer to a data structure containing values to be sent or returned. (The structure is often specific to each message type.)

Nearly all the member functions in this library make at least one call to *mciSendCommand()*. But don't be fooled by its simplicity: It has many options, constants, flags, messages, structures, and return values (and our goal is to hide these).

MCI Class Library

The MCI class hierarchy is designed to encapsulate and enhance the MCI interface and to get the most benefit from the least code. It is part of a C++ library that will be used as part of the basis for the commercial multimedia products we are currently developing. As such, it was designed for a specific immediate purpose, but also had to be capable of handling unknown future requirements. This influenced both the design and the implementation. The design had to be flexible, extensible, and (most of all) practical. Because our needs are for specific devices only, not all MCI devices are directly implemented although adding support for additional devices is a short process. Figure 1 diagrams the MCI class hierarchy, giving an overview of this single-inheritance tree.

All MCI commands are classified into one of four categories: system, required, basic, and extended. Commands specified for the first two types (such as open, close, and status) require support by all device types and are good candidates for member-function definitions in the base class(es) of the MCI class hierarchy. The basic commands, including play, stop, and seek, are supported by most device types and can also be placed at or near the top of the tree. Extended commands that support specific devices can be implemented farther down the hierarchy.

The root node for the MCI class hierarchy, *MCIDevice*, serves as the base class for both the *CompoundDevice* class

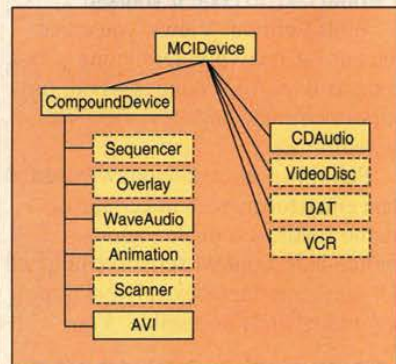


Figure 1: MCI class hierarchy.



PC-lint 5.0 presents C Bug # 502

```

int main()
{
    char ch1 = '\x0F';
    char ch2 = '\xFF';
    unsigned u = (ch1 << 8) | ch2;
    /* some computation */
    u = u & ~ch2;
    printf( "%x\n", u );
    return 0;
}
  
```

It was the programmer's intent to form a byte pair, do some computation, and then remove the second byte; but it's not working. Why does this program not print "F00"? Call if you need a hint. Refer to Bug #502.

PC-lint will catch this and many other C bugs. Unlike your compiler, PC-lint looks across all modules of your application for bugs and inconsistencies.

More than **330 messages**. Includes optional Strong Type Checking and flow of control analysis. More than **105 options** for complete customization. Suppress error messages, locally or globally, by symbol name, by message number, by filename, etc. Check for **portability** problems. Alter size of scalars. Adjust format of error messages. Automatically generate ANSI prototypes for your K&R functions.

Attn: Power users with huge programs.

PC-lint 386 uses DOS Extender Technology to access the full storage and flat model speed of your 386.

PC-lint 386 - \$239

PC-lint DOS - OS/2 - \$139

Mainframe & Mini Programmers

FlexeLint in obfuscated source form, is available for Unix, OS-9, VAX/VMS, QNX, IBM VM/MVS, etc. Requires only K&R C to compile but supports ANSI. Call for pricing.

Gimpel Software

3207 Hogarth Lane, Collegeville, PA 19426

CALL TODAY (215) 584-4261 Or FAX (215) 584-4266

30 Day Money-back Guarantee.

PA add 6% sales tax.

PC-lint and FlexeLint are trademarks of Gimpel Software

IBM
PS/2
AIX • MACH-Based AIX
OS/2 & SOM (planned)

Intel
x86-Pentium
Extended DOS • Windows • NT
Solaris • UNIX SVR3-4 • OSF

Multi-Platform

IBM
ESA
AIX

MIPS
Embedded Applications
BSD UNIX

Motorola
680x0
Embedded Applications • SunOS

AT&T
Hobbit
Pen Point

Intel
i860
UNIX SVR4 • APX
Embedded Applications

AMD
Am29K
Embedded Applications

HP-PA
HP/UX • CenterLine

Sun
SPARC
SunOS • Solaris 1, 2 • CenterLine
Embedded Applications

MetaWare[®]
INCORPORATED



*Multi-Language, Multi-Platform
Compiler Products for
Professional Software Developers*

Professional developers today know that portability and superior code quality are more than luxuries — they're a matter of survival. Better, faster chips and more robust operating environments are making their appearance on what seems to be a daily basis. MetaWare has a solid understanding of how these issues affect your business. Our 32-bit high-performance multi-platform compilers ensure that investments you make with us today will be safe for you tomorrow. **High C/C++**™ and **Professional Pascal**® are the clear choices for multi-platform software development. Call today!

CIRCLE NO. 158 ON READER SERVICE CARD

2161 Delaware Avenue • Santa Cruz, CA 95060-5706 • (408) 429-META (6382) • FAX (408) 429-WARE (9273)

MetaWare, the MetaWare logo, and High C and Professional Pascal, are registered trademarks, and High C/C++ and High C++ are trademarks, of MetaWare Incorporated. Other names are trademarks of their respective companies. © Copyright 1993 MetaWare Incorporated

(continued from page 86)

and all simple device classes. It provides the basic open/close/play type commands, status commands for querying a device's state, set commands to configure devices, and a number of other miscellaneous functions. The data items needed (all private) are:

- An integer device id.
- An error-status value.
- A string pointer to the device name.
- An optional handle to a parent window.

The constructor (see Listing One, page 102) does not automatically open the device—doing so would mean that the device would be opened and potentially unavailable to other applications from the time this object is constructed until it's destroyed. This is also true for all the derived classes: A device is not opened until explicitly told to do so with an *Open()* call. It is closed either through a call to the *Close()* member function or when the object is destroyed and the destructor is invoked. (Constructing a *CompoundDevice* with a filename is treated as if it were an *Open()* call and that file is immediately opened.) Keeping devices open only as long as necessary is generally polite behavior in a multitasking environment such as Windows.

The constructor for *MCIDevice* and any derived simple device (such as *CDAudio*) takes a single argument, a string value specifying the appropriate MCI device name. A default value is provided for each of the simple device types so that an application will rarely have to explicitly provide this value. The only exception might be on a system using nonstandard or additional device names that do not match the usual MCI device-name strings defined in the [mci] section of the SYSTEM.INI file. *MCI-Device* is not an abstract class and can therefore be constructed and used directly. It is designed, however, to serve as the basis for derived classes supporting unique device types.

Most of the member functions have a direct mapping to a corresponding MCI command. These functions almost always require fewer arguments than either the command-message or the command-string interfaces because some of the necessary information has been encapsulated in the object. Each member function first initializes any necessary data structures, sets the appropriate flags and arguments, and calls *mciSendCommand()*, occasionally more than once.

The function's return code is passed back to the object's client and is also saved in order to maintain an error state

for later reference. The *ErrorMessage()* member function calls *mciGetErrorString()* to get the error description, and displays a standard Windows error-message dialog box to the user. This class and all derived classes highlight one of the benefits of using C++: the ability to use destructors to perform automatic cleanup. It is important that applications properly close all MCI devices and files they have opened. By utilizing the C++ destructor mechanism we can automatically trigger the closing of any open devices before an application exits and thereby improve reliability and robustness (not to mention avoiding any extra general-protection-fault-type messages).

A sampling of functions built upon the MCI set, status, and capability commands are implemented to show how these access operations can be implemented. *MCIDevice* provides the virtual functions *Set()*, *Status()*, and *GetDevCaps()* for this purpose, but makes these protected because they are not intended to be directly called. Making these public forces the user to be familiar with the numerous MCI_SET and MCI_STATUS constant values needed for each specific option. Instead, these are used as the mechanism within publicly defined inline functions defined for each of the set and status options. For example, the *Length()* function, which returns the length of the current device, uses the *Status()* function with the MCI_STATUS_LENGTH option to get its value. (It should be pointed out that a type of simple runtime type identification can be achieved by using information functions such as *DeviceType()* and *CompoundDevice()*. The *DeviceType()* function returns a unique constant identifier for each device type, and therefore each class. *CompoundDevice()* returns a Boolean value of True if the given object is a compound device.)

One other notable method, *SetParent()*, is used to assign the handle of a window designated as the parent of this MCI object. *Play()* checks to see if a parent window has been assigned to it and if so, stores this handle in the parameter block given to *mciSendCommand()* and sets the MCI_NOTIFY bit of the corresponding flags value. This causes MCI to post the MM_MCINOTIFY message to the specified window when the operation is completed. When this occurs, the window procedure's *lParam* is set to the device id that initiated the callback, and the *wParam* gives the status of the operation, which can be success, failure, superseded, or aborted. MCI allows the Notify option to be used with all commands, and this library uses

it specifically as an option to the *Play()* function. If needed, this option can be added to any or all of the other functions for these classes.

A Simple MCIDevice

CDAudio is a simple class for this "simple" device. In fact, all the basic functionality for simple devices is provided in the *MCIDevice* base class. Therefore, *CDAudio* doesn't need to provide additional functions of its own. It can instead rely on the operations inherited from its parent. The MCI command set for *CDAudio* defines a few additional options for some of the base commands, three of which are implemented here. These are *Eject()* to eject the disc, and *SetTimeFormatMSF()* and *SetTimeFormatTMSF()* for setting the time format (T=tracks, M=minutes, S=seconds, F=frames). Each is implemented as an inline function that passes the appropriate flags to the *Set()* function. By allowing just the time format to be set through these functions, we can avoid inadvertently trying to set a device to a time format that it will not accept. This lets the programmer know which formats are acceptable for each device by looking at its class definition, which cannot be determined without the documentation using the C interface.

This *CDAudio* device can be opened, closed, played, stopped, queried for its status, and so on. All common operations are inherited from the *MCIDevice* base class. Other simple devices such as *Videodisc* and *VCR* can be implemented similarly by deriving from *MCIDevice*, specifying the appropriate device-type string for the constructor and adding any device-specific operations.

CompoundDevice

A compound device is an MCI device that uses files. Therefore, the *CompoundDevice* class and its descendants must be able to handle a filename on open. Keep in mind that the device id is more like an element id, one of which is allocated for each open compound-device file. (A filename for a compound device is known as the "device element.") The constructor for *CompoundDevice* takes two optional arguments: the name of the file to open, and the name of the device type to be opened. The second argument is immediately passed to the base-class constructor. The first argument, if given, is stored internally and then used to open the file. This approach follows the *iostream* model, in which a filename passed to the constructor automatically opens the file. The implementation could easily be changed to not open

the file and require a subsequent call to `Open()`. Not requiring a filename allows the object to be constructed first and later supply a filename on the call to `Open()`.

The `Open()` and `Close()` virtual functions are redefined in order to deal with filenames. `Open()` saves the filename as part of the object's data and then passes this along in the `lpstrElementName` field of the `mciOpenParms` structure given to `mciSendCommand()`. The `Close()` command uses `MCIDevice::Close` to close the device and then frees any memory associated with the filename.

An example of a straightforward derivation from `CompoundDevice` is the `Wave` class. Windows supports digital audio through waveform audio files that typically use the .WAV file extension. The MCI support for waveform audio is fairly complete, allowing recording and playback of formats ranging from 8-bit mono at 11 KHz to 16-bit stereo at 44 KHz, depending on the audio card installed. The `Wave` class does not need to override most of the virtual functions inherited from `CompoundDevice`. Functions such as `Play`, `Stop`, `Pause`, `Resume`, and `Seek` can all be used as is. Three additional status functions have been added to allow the user to query the format of the current file: `Channels()`, which returns 1 if the file is mono and 2 if it is stereo; `BitsPerSample()`, which returns either 8 or 16; and `SamplesPerSecond()`, which returns 11025, 22050, or 44100 to reflect the sampling rate. These functions use extended MCI commands added as part of the waveform audio-command set.

Video as a Data Type

Audio Video Interleaved (AVI) provides scalable, software-only (with optional hardware support), full-motion digital video. The video sequences can contain images, audio, and palettes. The audio can be synchronized with the video within 1/30th of a second. Currently, application support for AVI playback is only available through an MCI interface—a low-level playback is provided in the Video for Windows Development Kit. Microsoft provides an MCI driver, `MCI.AVI.DRV`, which implements a subset of the MCI digital-video command set for AVI.

To support this new (and large) command set, our AVI class implements a number of new functions. A `Window()` function was added, which takes a handle to a parent window to allow playback as the child of a specific window rather than the default behavior which is to create its own frame window. `Put-Destination()` was added to allow the

playback window to be positioned at a specific rectangle within the parent window. Other added functions include: `Step()`, which moves forward or backward a specified amount; `Seek()`, which moves to a given position; `Update()`, which displays the current frame; and `Signal()`, which notifies the parent when a particular position has been reached. This implementation could be further enhanced to redefine some of the base-class virtual functions for extensions that cannot be handled by the inherited versions, such as having a `Play()` that handles options to play back in a window or full screen. (Alternatively, these can be set using the `Configure()` function,

which pops up a dialog box to set these and other options.)

This class could be modified to use three different window handles: one for the recipient of the `MM_MCINOTIFY` notify message after `Play()`, one to receive the `MM_MCISIGNAL` message when a `Signal()` position is reached, and another to be used as the parent window for display purposes using the `Window()` function. The present design uses up to two concurrent window handles: One is used for the `Window()` parent—this value is not stored as class data; the other, the `bWndParent` data member, is used for receiving both types of notification messages.

Mouse, Menu Bar (and More) in Epsilon 6.5!

Lugaru Software proudly presents Epsilon version 6.5. This latest release of our award-winning programmer's editor is now available for DOS and OS/2, and it's got mouse support, a menu system, and many other great features. We're constantly improving Epsilon, making it more useful to you, and this is the latest result.

Mouse Support

We designed the mouse to be as non-intrusive as possible, so that when you're not using the mouse, you won't notice it's there. We don't reserve valuable screen space for scroll bars—Epsilon's scroll bars pop up when you move the mouse over them. And Epsilon automatically hides the mouse cursor when you're typing, so it doesn't get in your way.

On EGA and VGA displays under DOS, Epsilon uses special techniques to display a graphical arrow cursor in "text mode." With no jumping block cursor, the mouse is easier to position, but Epsilon avoids the overhead of graphics mode, which can slow down screen updates.

You can now run commands from a pull-down menu bar. Select commands with the mouse, or access the menu from the keyboard. Like Epsilon's scroll bars, the menu doesn't take up any screen space when you're not using it—Epsilon pops up the menu bar when you move the mouse past the top of the screen. Naturally, you can make the menu bar or the scroll bars permanent if you prefer, or customize the mouse and menu system in many other ways. It's easy to change the menu—just edit a special file. Epsilon can automatically fill in the keyboard equivalents for the commands you add.

Improved C code

Epsilon can scan your C source files for function definitions, and jump directly to a function's definition when you type its name. Or while editing a function call, you can press a key and

jump instantly to the function's definition. Now, in Epsilon 6.5, you can do the same thing with C variables, typedefs, macros, and structure and union tags.

You can choose to record only definitions, as in previous versions, or declarations as well. This is especially handy for library header files, where you might have only the declarations, not the definitions, of functions and variables.

Epsilon provides automatic indenting and reindenting of your C code. It does sophisticated reverse-parsing of your program to determine the correct indentation, unlike some other editors that only look at the previous line, and often misindent. Now Epsilon is smarter about indenting some relatively uncommon C constructs, and it offers more variables for customizing it to your indenting style.

You can now double-click with the mouse on a subroutine or variable name, and Epsilon will jump to its definition. This is especially convenient when browsing source files—you can navigate using only the mouse.

Customize the screen

New variables let you customize the screen's appearance, moving the echo area or mode lines, or adding borders around windows.

Epsilon can display more background colors (including brighter whites) on EGA and VGA systems. Now a total of 256 foreground/background combinations are available, twice as many as before.

Much more

The new version has lots of other

improvements as well, too many to mention here. And of course, Epsilon 6.5 has all the features of previous Epsilon versions:

- Multitasks compilers, even in DOS.
- Scans compiler errors (customizable).
- Full undo/redo retained after file saves.
- Unlimited number of files or windows.
- No built-in limits on line length or file size.
- Ultra-fast multi-file search.
- File comparison commands compare by lines or by characters.
- Switch among 80x25, 80x50, 4 other VGA video modes, with a keypress.
- Fully utilizes EMS & XMS memory & upper memory blocks.
- EMACS-style command set.
- Interactively customizable keyboard.
- Record keystroke sequences.
- Context sensitive help autoconfigures.
 - Fast C-like extension language.
 - Source code to all commands.
 - Regular expression (wildcard) search & tagged replace.
 - Permanent and temporary bookmarks.
 - And much more!

Epsilon 6.5 is available now for \$250. Updates are only \$45 from version 6.0, or \$60 from any previous version.

Now is the perfect time to try Epsilon for yourself. We offer a 60-day money-back guarantee. If you're not totally thrilled with Epsilon, just send it back and receive a full refund. Why not call us now?

Lugaru's
epsilon
programmer's editor

Only \$250
for DOS, OS/2,
SCO Unix or ISC Unix.
60-day money-back guarantee.
Order Now: (412) 421-5911

Lugaru Software, Ltd. 5843 Forbes Avenue Pittsburgh, PA 15217 Phone (412) 421-5911 Fax 412.421.6371
Copyright © 1993 Lugaru Software, Ltd. All Rights Reserved.

CIRCLE NO. 144 ON READER SERVICE CARD

MciMan

I've included a sample client application, MciMan, which is available electronically (see "Availability," page 5). MciMan shows how an application might use the MCI class library. Specifically, MciMan uses the *AVI* and *Wave* classes to allow the user to work with either type of device. A simple menu interface allows you to select, play back, pause, resume, and stop the device type. A Notify toggle under the Command menu causes the user to be notified by a modal message box when the next Play command is completed. (This uses the MCI_NOTIFY option.) One file of each type can be open simultaneously.

MciMan works by creating one instance each of the *Wave* and *AVI* classes as global objects. When the user selects which type to make currently active from the MCI Device menu, a pointer to that global object is assigned to a *CompoundDevice** variable. This pointer is then used for all subsequent operations and relies on the virtual-function mechanism to invoke the appropriate class function based upon the current selected type. Therefore, when the Play menu item is selected, the *Play()* function is invoked using this pointer and either the *Wave::Play()* or the *AVI::Play()* function will be executed. When the program is exited, the destructors for

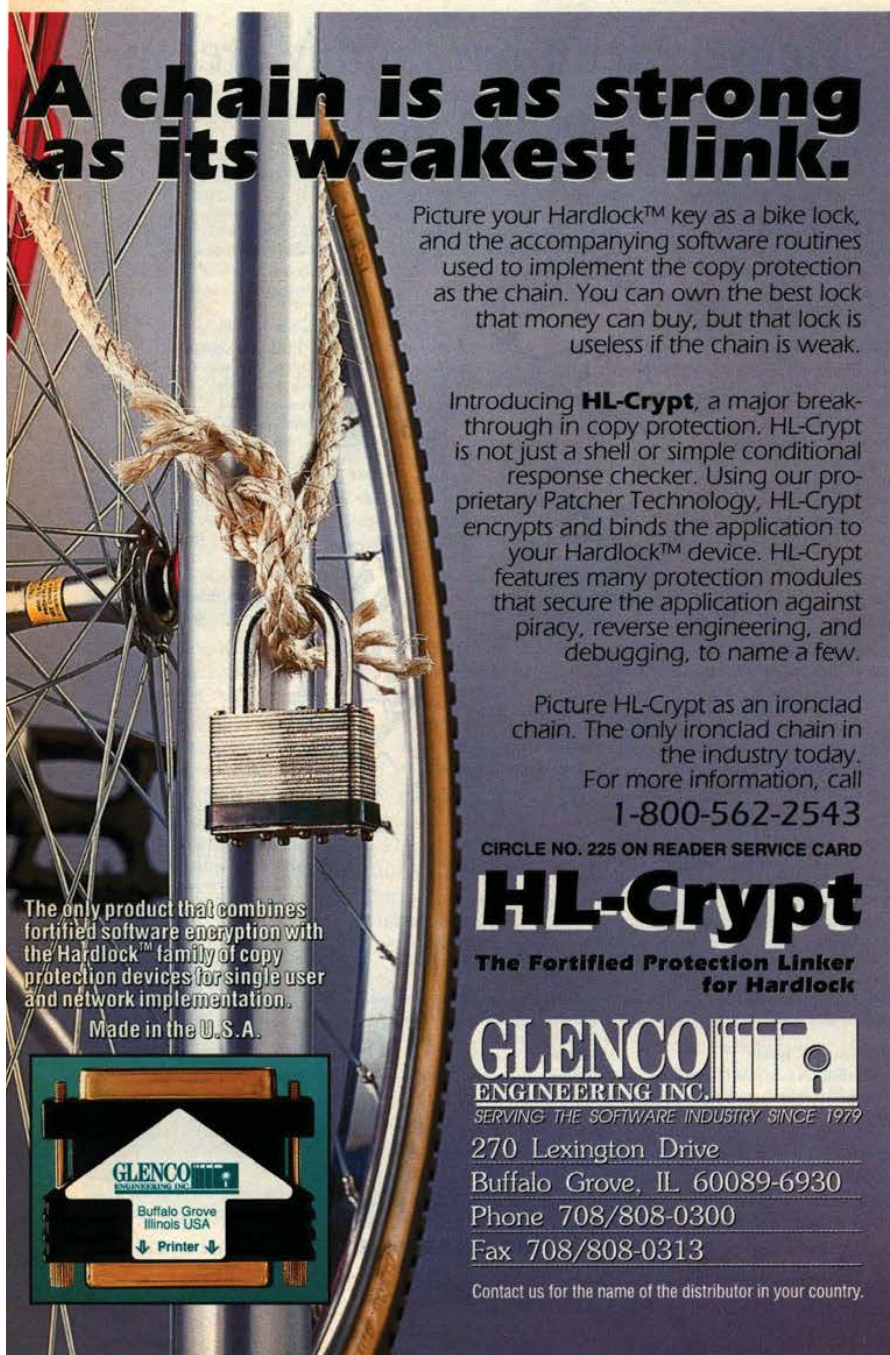
these two objects are invoked, and all devices and files are automatically closed—no special cleanup is needed in the WM_DESTROY or other similar code block.

Future Directions

Although the current implementation of the MCI class hierarchy is fairly comprehensive, it does not cover all devices nor does it implement all of the commands and their variants. In addition to MCI's large command set, the number of MCI devices is also increasing. You can, in fact, write your own MCI device drivers with the Microsoft Device Driver Kit (DDK). And as with any software library, there's always room for one more feature.

A good example of an additional device is the MCI driver that Apple is expected to supply for playing QuickTime for Windows video clips. Depending on the command set this driver supports, it may make sense to create a *Digital-Video* class which could serve as the parent to both *AVI* and *QuickTime* subclasses. This class could be used to provide a set of functions common to these two specific devices, thus sharing code and giving a standard API for both types. (Client programs could then use these interchangeably.)

In any case, you now have a class library that uses C++ and the MCI command set to build a solid framework of multimedia objects for Windows. The MCI class library hides the arcane syntax of the MCI command-message interface (the messages, ids, structures, double casts, and flags) and instead provides a set of objects that give applications a cleaner, more flexible, and robust interface for controlling a variety of multimedia devices.



A chain is as strong as its weakest link.

Picture your Hardlock™ key as a bike lock, and the accompanying software routines used to implement the copy protection as the chain. You can own the best lock that money can buy, but that lock is useless if the chain is weak.

Introducing **HL-Crypt**, a major breakthrough in copy protection. HL-Crypt is not just a shell or simple conditional response checker. Using our proprietary Patcher Technology, HL-Crypt encrypts and binds the application to your Hardlock™ device. HL-Crypt features many protection modules that secure the application against piracy, reverse engineering, and debugging, to name a few.

Picture HL-Crypt as an ironclad chain. The only ironclad chain in the industry today. For more information, call **1-800-562-2543**

CIRCLE NO. 225 ON READER SERVICE CARD

HL-Crypt


The Fortified Protection Linker for Hardlock

GLENCO

ENGINEERING INC.
SERVING THE SOFTWARE INDUSTRY SINCE 1979
270 Lexington Drive
Buffalo Grove, IL 60089-6930
Phone 708/808-0300
Fax 708/808-0313

The only product that combines fortified software encryption with the Hardlock™ family of copy protection devices for single user and network implementation.

Made in the U.S.A.



Contact us for the name of the distributor in your country.

Products Mentioned

Video For Windows
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052-6399
\$199.00; Runtime DLL and SDK
available free of charge from
CompuServe (GO WINEXT)
System Requirements:
Playback: 386 SX, Windows 3.1,
sound card, mouse, VGA
Capture: 50 Mbytes free hard-
disk space, video-capture board

DDJ

(Listing begins on page 102.)

Vote for your favorite feature/article.
Circle Reader Service **No. 8**.

Listing One (Text begins on page 84.)

```

/*-----*/
* Mci.cpp: C++ class hierarchy for Windows multimedia objects using MCI.
* $Author: John Musser $
* $Date: 24 Feb 1993 19:37:02 $
* $Copyright: Personal Media International Inc., 1993 $
* Description: A set of classes to encapsulate the command-message interface
* to MCI. Currently implemented: CDAudio, Waveform audio, AVI.
* Two base classes MCIDevice and CompoundDevice provide most of
* the basic functionality needed for derived MCI types.
/*-----*/

#include "MCI.h"
#include <memory.h>
#include <string.h>

#define MCI_BUFSIZE 128 // used for char array sizing

/*----- * MCIDevice functions * -----*/
MCIDevice::MCIDevice(LPSTR lpszDevice)
{
    wDeviceID = NULL;
    hWndParent = NULL;
    dwErrState = NULL;
    lpszDeviceType = NULL;
    if (lpszDevice)
        SetDeviceType(lpszDevice);
}

MCIDevice::~MCIDevice()
{
    if (wDeviceID)
        Close();
    if (lpszDeviceType)
        delete [] lpszDeviceType;
}

LPSTR
MCIDevice::Info(DWORD dwFlags)
{
    MCI_INFO_PARMS mciInfoParms;
    static char cBuf[MCI_BUFSIZE];
    if (!wDeviceID)
        return (LPSTR)NULL;
    mciInfoParms.lpstrReturn = cBuf;
    mciInfoParms.dwRetSize = MCI_BUFSIZE;
    dwErrState = mciSendCommand(wDeviceID, MCI_INFO, dwFlags,
        (DWORD) (LPMCI_INFO_PARMS) &mciInfoParms);
    return mciInfoParms.lpstrReturn;
}

DWORD
MCIDevice::Set(DWORD dwFlags, DWORD dwExtra)
{
    MCI_SET_PARMS mciSetParms;

```

```

    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    if (dwFlags & MCI_SET_TIME_FORMAT)
        mciSetParms.dwTimeFormat = dwExtra; // dwExtra is format
    dwErrState = mciSendCommand(wDeviceID, MCI_SET, dwFlags,
        (DWORD) (LPMCI_SET_PARMS) &mciSetParms);
    return dwErrState;
}

DWORD
MCIDevice::Status(DWORD dwFlags, DWORD dwItem, DWORD dwExtra)
{
    MCI_STATUS_PARMS mciStatusParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciStatusParms.dwItem = dwItem;
    // Determine which extra struct fields to set based on flags.
    if (dwFlags & MCI_TRACK)
        mciStatusParms.dwTrack = dwExtra;
    dwErrState = mciSendCommand(wDeviceID, MCI_STATUS, dwFlags,
        (DWORD) (LPMCI_STATUS_PARMS) &mciStatusParms);
    return mciStatusParms.dwReturn;
}

DWORD
MCIDevice::GetDevCaps(DWORD dwItem)
{
    MCI_GETDEVCAPS_PARMS mciGetDevCapsParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciGetDevCapsParms.dwItem = dwItem;
    dwErrState = mciSendCommand(wDeviceID, MCI_GETDEVCAPS, MCI_GETDEVCAPS_ITEM,
        (DWORD) (LPMCI_GETDEVCAPS_PARMS) &mciGetDevCapsParms);
    return mciGetDevCapsParms.dwReturn;
}

void
MCIDevice::SetDeviceType(LPSTR lpszDevice)
{
    if (lpszDeviceType)
        delete [] lpszDeviceType;
    lpszDeviceType = new _Tchar [lstrlen(lpszDevice)+1];
    lstrcpy(lpszDeviceType, lpszDevice);
}

void
MCIDevice::ErrorMessageBox()
{
    char szErrorBuf[MAXERRORLENGTH];
    if (mciGetErrorString(dwErrState, (LPSTR)szErrorBuf, MAXERRORLENGTH))
        MessageBox(hWndParent, szErrorBuf, "MCI Error", MB_ICONEXCLAMATION);
    else
        MessageBox(hWndParent, "Unknown MCI Error", "MCI Error",
            MB_ICONEXCLAMATION);
}

DWORD
MCIDevice::Open(LPSTR lpszDevice /* = NULL */)
{
    MCI_OPEN_PARMS mciOpenParms;
    if (wDeviceID) // Already open don't do it again
        return MCI_WARN_ALREADY_OPEN;
    if (lpszDevice) // Type given, save it in private data
        SetDeviceType(lpszDevice);
    else
        if (!lpszDeviceType) // Make sure we have a type to open.
            return MCI_ERR_NO_DEVICENAME; // had to give here or in ctor.
    mciOpenParms.lpstrDeviceType = lpszDeviceType;
    dwErrState = mciSendCommand(NULL, MCI_OPEN, MCI_OPEN_TYPE,
        (DWORD) (LPMCI_OPEN_PARMS) &mciOpenParms);

    if (!dwErrState)
        wDeviceID = mciOpenParms.wDeviceID;
    return dwErrState;
}

DWORD
MCIDevice::Close()
{
    MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    if (!dwErrState = mciSendCommand(wDeviceID, MCI_CLOSE, MCI_WAIT,
        (DWORD) (LPMCI_GENERIC_PARMS) &mciGenericParms)))
        wDeviceID = NULL;
    return dwErrState;
}

DWORD
MCIDevice::Stop()
{
    MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID, MCI_STOP, MCI_WAIT,
        (DWORD) (LPMCI_GENERIC_PARMS) &mciGenericParms);
}

DWORD
MCIDevice::Pause()
{
    MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID, MCI_PAUSE, MCI_WAIT,
        (DWORD) (LPMCI_GENERIC_PARMS) &mciGenericParms);
}

DWORD
MCIDevice::Resume()
{
    MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID, MCI_RESUME, MCI_WAIT,
        (DWORD) (LPMCI_GENERIC_PARMS) &mciGenericParms);
}

DWORD
MCIDevice::Play(LONG lFrom, LONG lTo)
{
    MCI_PLAY_PARMS mciPlayParms;

```

Real-Time Multitasking

for Microsoft C, Borland C, Turbo Pascal

Develop Real-Time Multitasking Applications with RTKernel!

RTKernel is a professional, high-performance real-time multitasking kernel. It runs under MS-DOS and supports Microsoft C, Borland C++, Turbo Pascal, and Stony Brook Pascal. RTKernel is a library you can link to your application. It lets you run several functions / procedures as parallel tasks. RTKernel offers these advanced features:

- pre-emptive, event-/interrupt-driven scheduling
- number of tasks only limited by RAM
- task-switch time of 6 µsecs (33-MHz 486)
- speed independent of the number of tasks
- up to 64 priorities (changeable at run-time)
- time-slicing can be activated
- timer interrupt rate of 0.1 to 55 ms
- high-resolution interval timer (1 µsec)
- activate or suspend tasks out of interrupt handlers
- programmable interrupt priorities
- supports math coprocessor and emulator
- inter-task communications using semaphores, mailboxes, and message-passing
- keyboard, hard disk, floppy, disk idle times usable by other tasks
- interrupt handlers for keyboard and COM ports included with source code
- supports up to 36 COM ports (DigiBoard and Hostless boards)
- full support of NS16550 UART chip
- fast inter-network communication using Novell's IPX services
- runs under MS-DOS 3.0 to 6.0, DR-DOS, LANs, or without operating system
- call DOS without re-entrance problems
- resident multi-tasking applications (TSRs)
- runs Windows or DOS Extenders as a task
- supports CodeView and Turbo Debugger
- ROMable
- full source code available
- no run-time royalties
- free technical support by phone or fax

RTKernel-C (MSC 6.0/7.0/8.0, BC++ 1.0/2.0/3.x) \$495 (Source Code: add \$445)

RTKernel-Pascal (TP/BP 5.x/6.0/7.0, SBP 6.x) \$445 (Source Code: add \$375)

For international orders, add \$30 for shipping and handling. MasterCard, Visa, check, bank transfer, COD accepted.



FREE DEMO DISK

On Time
MARKETING

Professional Programming Tools

Karolinenstrasse 32 · 20357 Hamburg · Germany
Phone +49 - 40 - 43 74 72 · Fax +49 - 40 - 43 51 96 · CompuServe 100140,633

CIRCLE NO. 622 ON READER SERVICE CARD

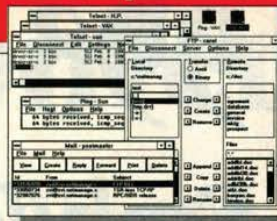

```

DWORD dwFlags = 0L;
if (!wDeviceID)
    return MCI_ERR_NOT_OPEN;
if (hWndParent) {
    mciPlayParams.dwCallback = (DWORD) (LPVOID) hWndParent;
    dwFlags |= MCI_NOTIFY;
}
if (lFrom != current) {
    mciPlayParams.dwFrom = lFrom;
    dwFlags |= MCI_FROM;
}
if (lTo != end) {
    mciPlayParams.dwTo = lTo;
    dwFlags |= MCI_TO;
}
dwErrState = mciSendCommand(wDeviceID, MCI_PLAY, dwFlags,
    (DWORD) (LPMCI_PLAY_PARMS) &mciPlayParams);
return(dwErrState);
}
DWORD
MCIDevice::Seek(LONG lTo)
{
    MCI_SEEK_PARMS mciSeekParams;
    DWORD dwFlags = 0L;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    switch(lTo) {
        case start:
            dwFlags = MCI_SEEK_TO_START;
            break;
        case end:
            dwFlags = MCI_SEEK_TO_END;
            break;
        default:
            mciSeekParams.dwTo = (DWORD) lTo;
            dwFlags = MCI_TO;
    }
    dwErrState = mciSendCommand(wDeviceID, MCI_SEEK, dwFlags,
        (LONG) (LPMCI_SEEK_PARMS) &mciSeekParams);
    return dwErrState;
}
DWORD
MCIDevice::StopAll()
{
    return mciSendCommand(MCI_ALL_DEVICE_ID, MCI_STOP, 0, NULL);
}
/*----- * CompoundDevice functions * -----*/
CompoundDevice::CompoundDevice(LPSTR lpszFile, LPSTR lpszDevice)
: MCIDevice(lpszDevice)
{
    lpszFileName = NULL;
    if (lpszFile)
        Open(lpszFile);
}
CompoundDevice::~CompoundDevice()
{
    if (wDeviceID)
        Close();
    if (lpszFileName)
        delete [] lpszFileName;
}
DWORD
CompoundDevice::Open(LPSTR lpszFile)
{
    MCI_OPEN_PARMS mciOpenParams;
    DWORD dwFlags = 0L;
    if (wDeviceID) // If we're already open don't do it again
        return MCI_WARN_ALREADY_OPEN;
    lpszFileName = new _Tchar[strlen(lpszFile)+1];
    lstrcpy(lpszFileName, lpszFile);
    mciOpenParams.lpstrElementName = lpszFileName;
    mciOpenParams.lpstrDeviceType = lpszDeviceType;
    dwFlags = MCI_OPEN_ELEMENT | MCI_OPEN_TYPE;
    dwErrState = mciSendCommand(NULL, MCI_OPEN, dwFlags,
        (DWORD) (LPMCI_OPEN_PARMS) &mciOpenParams);
    if (!dwErrState)
        wDeviceID = mciOpenParams.wDeviceID;
    return dwErrState;
}
DWORD
CompoundDevice::Close()
{
    MCIDevice::Close();
    if (lpszFileName) {
        delete [] lpszFileName;
        lpszFileName = NULL;
    }
    return dwErrState;
}
/*----- * AVI functions * -----*/
DWORD
AVI::Update(HDC hdc)
{
    MCI_DGV_UPDATE_PARMS mciUpdateParams;
    DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciUpdateParams.hDC = hdc;
    dwFlags = MCI_DGV_UPDATE_HDC | MCI_DGV_UPDATE_PAINT;
    dwErrState = mciSendCommand(wDeviceID, MCI_UPDATE, dwFlags,
        (LONG) (LPMCI_DGV_UPDATE_PARMS) &mciUpdateParams);
    return dwErrState;
}
DWORD
AVI::PutDestination(RECT &rect)
{
    MCI_DGV_PUT_PARMS mciPutParams;
    DWORD dwFlags;

```

(continued on page 106)

TCP/IP for Windows



CHAMELEON™
100% DLL
NFS = RPC
SNMP

**Requires only 6KB of base memory
Implemented as 100% Windows DLL (not a TSR)**

- All applications are both client and server
- Works concurrently with Netware, LAN Manager, Vines etc.
- Up to 64 concurrent sessions
- Extensible SNMP Agent

Developer Tools:

Windows Socket API
Berkeley 4.3 Socket API
ONC RPC/XDR

NEW! NFS Client & Server

Applications:

TELNET (VT100, VT220),
TN3270, FTP, TFTP
SMTP/Mail, POP2, SNMP,
PING, BIND, Statistics,
and Custom

For overnight delivery call:
NETMANAGE™
(408) 973-7171
20823 Stevens Creek Blvd.
Cupertino, CA 95014 USA
Fax (408) 257-6405

CIRCLE NO. 805 ON READER SERVICE CARD

Copia International Ltd

AccSys Libraries



AccSys™
for
dBASE



AccSys™
for
Paradox

- | | |
|--|--|
| <input type="checkbox"/> Support Memo, MDX and NDX Index Files | <input type="checkbox"/> Twice the Speed & Half the Size of Engine |
| <input type="checkbox"/> Windows (DLL) and VisualBasic Support | <input type="checkbox"/> QuickBasic, VisualBasic & Windows (DLL) Support |

Both libraries are multi-platformed, have C source available, and offer full network support.

TECH INFO VIA FAX: 708/924-3030 DOC. NO. 889823

Copia International Ltd.

Wheaton, Illinois 60187 708/682-8898

SlickEdit™, the multi-platform editor, gives you freedom of choice!

"...overall, the easiest editor to learn and use."
—PC Week



Special Offer! Buy SlickEdit for DOS—alone or bundled with our SlickEdit for OS/2 or Windows NT—and receive the new full-GUI SlickEdit for Windows 3.1 in May!

"SlickEdit is the only editor I use."
—Dave Cutler, Director of Windows NT development

SlickEdit has all the CUA standards you expect in a graphical Windows application—plus full on-line documentation, unlimited free tech support, and a full 30-day money-back guarantee. These are just a few more of its features:

- Fully programmable with new C-style macro language
- Truly emulates text-mode editors such as Brief and Emacs
- Optional one file per window
- Multiple clipboard support
- Optional CUA marking
- We've kept our command line!

MS-DOS	} \$195 each, or any two for only \$295!
OS/2	
Windows NT	
Windows 3.1	
386/486 UNIX	} \$425 per CPU
Sun Sparc	
IBM AIX RS6000	
HP9000, DG Avilion, Silicon Graphics, EP/IX, Dynix	

CALL



For more information, call
(919) 831-0662
or fax: (919) 831-0101

microEdge, Inc.
P.O. Box 18038
Raleigh, NC
27619-8038
USA

MS-DOS, OS/2, Windows, Windows NT, Sun Sparc, HP-9000, DG Avilion, Silicon Graphics, EP/IX, Dynix, AIX RS6000, Brief, Epsilon, and UNIX are trademarks of their respective manufacturers

CIRCLE NO. 160 ON READER SERVICE CARD

Listing One (Listing continued, text begins on page 84.)

```

if (!wDeviceID)
    return MCI_ERR_NOT_OPEN;
mciPutParms.rc = rect;
dwFlags = MCI_DGV_PUT_DESTINATION | MCI_DGV_RECT;
dwErrState = mciSendCommand(wDeviceID, MCI_PUT, dwFlags,
    (LONG) (LPMCI_DGV_PUT_PARMS) &mciPutParms);
return dwErrState;
}
DWORD
AVI::Signal(DWORD dwPosition)
{
    MCI_DGV_SIGNAL_PARMS mciSignalParms;
    DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    if (!hWndParent) // we need a parent to get the message
        return MCI_ERR_NO_PARENT;
    mciSignalParms.dwCallback = (DWORD) (LPCVOID) hWndParent; // MCI_SIGNAL recipient
    mciSignalParms.dwPosition = dwPosition; // when to send
    mciSignalParms.dwUserParam = (DWORD) (LPCVOID) this; // self will be lParam
    dwFlags = MCI_DGV_SIGNAL_AT | MCI_DGV_SIGNAL_USERVAL;
    dwErrState = mciSendCommand(wDeviceID, MCI_SIGNAL, dwFlags,
        (LONG) (LPMCI_DGV_SIGNAL_PARMS) &mciSignalParms);
    return dwErrState;
}
DWORD
AVI::Configure()
{
    MCI_GENERIC_PARMS mciGenericParms;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    return dwErrState = mciSendCommand(wDeviceID, MCI_CONFIGURE, 0,
        (DWORD) (LPMCI_GENERIC_PARMS) &mciGenericParms);
}
DWORD
AVI::Que(DWORD dwTo)
{
    MCI_DGV_CUE_PARMS mciCueParms;
    DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciCueParms.dwTo = dwTo;
    dwFlags = MCI_DGV_CUE_OUTPUT | MCI_TO;
    dwErrState = mciSendCommand(wDeviceID, MCI_CUE, dwFlags,
        (LONG) (LPMCI_DGV_CUE_PARMS) &mciCueParms);
    return dwErrState;
}
DWORD
AVI::Step(DWORD dwFrames, BOOL bReverse)
{
    MCI_DGV_STEP_PARMS mciStepParms;
    DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciStepParms.dwFrames = dwFrames;
    dwFlags = MCI_DGV_STEP_FRAMES;
    if (bReverse)
        dwFlags |= MCI_DGV_STEP_REVERSE;
    dwErrState = mciSendCommand(wDeviceID, MCI_STEP, dwFlags,
        (LONG) (LPMCI_DGV_STEP_PARMS) &mciStepParms);
    return dwErrState;
}
DWORD
AVI::SetAudioVolume(DWORD dwVolume)
{
    MCI_DGV_SETAUDIO_PARMS mciSetAudioParms;
    DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciSetAudioParms.dwValue = dwVolume;
    dwFlags = MCI_DGV_SETAUDIO_ITEM | MCI_DGV_SETAUDIO_VOLUME;
    dwErrState = mciSendCommand(wDeviceID, MCI_SETAUDIO, dwFlags,
        (LONG) (LPMCI_DGV_SETAUDIO_PARMS) &mciSetAudioParms);
    return dwErrState;
}
DWORD
AVI::SetSpeed(DWORD dwSpeed)
{
    MCI_DGV_SET_PARMS mciSetParms;
    DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciSetParms.dwSpeed = dwSpeed;
    dwFlags = MCI_DGV_SET_SPEED;
    dwErrState = mciSendCommand(wDeviceID, MCI_SET, dwFlags,
        (LONG) (LPMCI_DGV_SET_PARMS) &mciSetParms);
    return dwErrState;
}
DWORD
AVI::Window(HWND hWnd)
{
    MCI_DGV_WINDOW_PARMS mciWindowParms;
    DWORD dwFlags;
    if (!wDeviceID)
        return MCI_ERR_NOT_OPEN;
    mciWindowParms.hWnd = hWnd;
    dwFlags = MCI_DGV_WINDOW_HWND;
    dwErrState = mciSendCommand(wDeviceID, MCI_WINDOW, dwFlags,
        (LONG) (LPMCI_DGV_WINDOW_PARMS) &mciWindowParms);
    return dwErrState;
}
}

```

End Listing