CHAPTER 1

Playing AVI Files With MCI

This chapter describes how to play Video for Windows AVI files using the MCI interface. It contains the following topics:

- MCI Overview
- Using the MCI Command Interface
- Using the MCI String Interface
- Handling MCI notification
- Playing AVI files using MCI

Sample code for AVI playback is in the MOVPLAY1.C and MOVPLAY2.C files. MOVPLAY1.C uses the MCI command interface while MOVPLAY2.C uses the string interface. Both applications look the same to the end user, they just illustrate the different methods of using MCI to send commands.

MCI Overview

MCI provides a high-level interface to control various media devices through generalized commands such as play, pause and stop as well as through specific command sets defined for different device types. MCI uses the MCIAVI.DRV driver to handle AVI playback.

Your application uses MCI commands from the Digital Video Command Set to control MCIAVI.DRV. Since most of the work is done by the commands and not by MCI directly, the interface to MCI itself is very simple and just passes commands down to MCIAVI. In fact, MCI only has five functions that applications use for MCI operation. Of these five functions, the following two functions are commonly used for sending commands:

mciSendCommand

Sends a command message to MCI.

mciSendString

Sends a string command to MCI.

Your application must link with MMSYSTEM.LIB to use MCI. It must also include the MMSYSTEM.H and DIGITALV.H files. The MMSYSTEM.H file included with the SDK for Microsoft Windows defines the prototypes for these functions and defines the messages, flags, constants, and data structures needed for their use. The DIGITALV.H file defines the digital video command set specifically used to control MCIAVI. For a summary of the command messages and command strings used with MCIAVI, see Chapter 7, “MCI Command Strings for MCIAVI” and Chapter 8, “MCI Command

Messages for MCIAVI.”

For full information on the MCI commands see the *Microsoft Multimedia Programmer's Reference* and the *Microsoft Multimedia Programmer's Guide* of the Windows 3.1 Software Development Kit. For full information on the MCI Digital Video Command Set see the *Digital Video Command Set for the Media Control Interface* standards update.

Using the MCI Command Interface

One method of sending MCI commands to MCIAVI uses **mciSendCommand** to send a *command message*. Command messages include a message corresponding to the command, a set of flags, and a data structure defining the parameters for that command. This function has the following syntax:

DWORD mciSendCommand(*wDeviceID, wMessage, dwParam1, dwParam2* **)**

The *wDeviceID* parameter defines the device ID of the MCI device that will receive the command. (This parameter is returned for the open command, which does not require the device ID.) The *wMessage* parameter specifies the message your application wants to send. The *dwParam1* parameter defines the flags for the command, and *dwParam2* points to a data structure for the command. This function returns 0 on success or an MCI error code on failure.

The programming example for sending a command message has MCIAVI.DRV play an AVI file. The command message for playing an AVI file is MCI_PLAY. For this command, MCIAVI.DRV accepts the following flags in *dwParam1:*

MCI_FROM

Indicates the **dwFrom** field of the structure identified by *dwParam2* specifies a starting position for the file.

MCI_TO

Indicates the **dwTo** field of the structure identified by *dwParam2* specifies an ending position for the file.

MCI_DGV_PLAY_WINDOW

Indicates playing should occur in the window associated with a device instance (the default).

MCI_MCIAVI_PLAY_FULLSCREEN

Indicates playing should use a full-screen display, typically with a 320x200 resolution.

The MCI_PLAY command uses the following data structure to pass information (*dwParam2* points to this structure):

```
typedef struct {
    DWORD  dwCallback;
    DWORD  dwFrom;
    DWORD  dwTo;
} MCI_DGV_PLAY_PARMS;
```

Prior to using **mciSendCommand** to send the MCI_PLAY message, your application allocates the memory for the data structure, initializes the fields it wants to use, and sets the flags corresponding to the fields used in the data structure. (If your application does not set a flag for a data structure field, MCI drivers ignore the data structure field.) For example, the following function plays a movie from the starting position specified by *dwFrom* to the an ending position specified by *dwTo* (if either position is 0 then it is considered not used):

```
DWORD PlayMovie(WORD wDevID, DWORD dwFrom, DWORD dwTo)
{
    MCI_DGV_PLAY_PARMS   mciPlay;    // play params
    DWORD                dwFlags = 0;

    // check dwFrom, if it is != 0 then set parameters and flags
    if (dwFrom){
        mciPlay.dwFrom = dwFrom; // set parameter
        dwFlags |= MCI_FROM;     // set corresponding flag to validate field
    }

    // check dwTo, if it is != 0 then set parameters and flags
    if (dwTo){
        mciPlay.dwTo = dwTo;    // set parameter
        dwFlags |= MCI_TO;      // set corresponding flag to validate field
    }

    // send the PLAY command and return the result
    return mciSendCommand(wDevID, MCI_PLAY, dwFlags,
            (DWORD)(LPVOID)&mciPlay);
}
```

Using the MCI String Interface

Another method of sending MCI commands to MCIAVI uses **mciSendString** to send a *command string.* This function uses text strings to represent the command. It has the following syntax:

DWORD mciSendString(*lpstrCommand*, *lpstrReturnString*, *wReturnLength*, *hCallback***)**

The *lpstrCommand* parameter specifies a far pointer to the actual command string. Each command string includes a command, a device identifier, and command arguments. Arguments are optional on some commands and required on other commands. A command string has the following form:

command device_id arguments

The *command* parameter represents the command name (for example, **open**, **close**, or **play**). The *device_id* parameter identifies the MCI driver or a file. The *arguments* parameter indicates any flags and values associated with the command.

When your application opens MCIAVI, it uses a device name, a keyword from the [MCI] section of the SYSTEM.INI file, or filename as the *device_id* used to identify the MCI device. Your application can avoid using the formal *device_id* argument in subsequent commands by specifying the **alias** flag when it opens MCIAVI. (The alias your application wants to use in subsequent commands is specified after the **alias** flag.) Most string examples in this section use an alias.

The *lpstrReturnString* parameter of **mciSendString** specifies a far pointer to a buffer for return information. (Your application can set it NULL if a command does not return information.) The *wReturnLength* parameter specifies the size of the return buffer, or 0 if no buffer is specified. The *hCallback* parameter specifies a window handle if your applications wants to receive MCI notify messages.

For example, the string **play** command used with MCIAVI.DRV has the following definition and arguments:

play *items*	Starts playing the video sequence. The following optional *items* modify the command:	
	from *position*	Specifies a starting position for the play.
	to *position*	Specifies an ending position for the play.
	fullscreen	Specifies playing should use a full-screen display.
	window	Specifies playback should be in the window associated with the device instance (the default).

The following example uses the string interface to send the **play** command with the **from** and **to** flags:

```
DWORD PlayMovie(LPSTR lpstrAlias, DWORD dwFrom, DWORD dwTo)
{
   char    achCmndBuff[128];

   wsprintf(achCmndBuff, "play %s from %u to %u", lpstrAlias, dwFrom, dwTo);

   return mciSendString(achCommandBuff, NULL, 0, NULL);
}
```

When using the string interface, all values passed with the command and all return values are text strings so your application needs conversion routines to translate from variables to strings or back again. For example, the following fragment gets the size of an AVI sequence and uses the size to allocate memory for a RECT structure:

```
void GetSourceRect(LPSTR lpstrAlias, LPRECT lpRect)
{
   char    achRetBuff[128];
   char     achCommandBuff[128];

   // build the command string "where name source"
   sprintf(achCommandBuff, "where %s source", lpstrAlias);


   SetRectEmpty(lpRect);    // clear the RECT


   // send the command
   if (mciSendString(achCommandBuff, achRetBuff,
           sizeof(achRetBuff), NULL)== 0){


      // The rectangle is returned in our buffer as "x y dx dy" and we
      // know that x and y are both 0 since this is the source rectangle.
      // The following lines translate the string into the RECT structure.
      char    *p;
      p = achRetBuff;          // point to the return string
      while (*p != ' ') p++;    // go past the x (0)
      while (*p == ' ') p++;    // go past spaces
      while (*p != ' ') p++;    // go past the y (0)
      while (*p == ' ') p++;    // go past spaces

      // now get the width
      for ( ; *p != ' '; p++)
         lpRect->right = (10 * lpRect->right) +
                     (*p - '0');

      while (*p == ' ') p++;    // go past spaces

      // now get the height
      for ( ; *p != ' '; p++)
         lpRect->bottom = (10 * lpRect->bottom) +
                     (*p - '0');

   }

}
```

Choosing the mciSendCommand or mciSendString Interface

Since there are two interfaces to send commands to MCIAVI, you must select the most appropriate one for your application’s needs. With the command interface, your application must fill a data structure and make sure that the flags it sets match the data structure fields it uses. With the string interface, however, your application must handle the conversion of string data for anything that might be variable within the application. Your application might choose to mix the two methods for the most efficient operation. For straightforward commands your application might use the string interface, and for commands that return information or commands your application passes information (such as window or palette handles), it might use the command interface.

You will probably find the string interface the easiest command set to understand and read. While the structure of the string commands is simple, it still retains the capabilities of the message commands to control MCI devices. This makes the command set extremely useful in planning your application and discussing the MCI capabilities of your application.

The examples in the rest of this chapter use a combination of the string and command interfaces. You can find other examples of using MCI with the digital video command set in the MOVPLAY1.C and MOVPLAY2.C files. For examples using command messages, see MOVPLAY1.C. For examples using command strings, see MOVPLAY2.C.

Handling MCI Notification

Whichever interface your application uses, it can have MCI send notification messages when an action completes. With the string interface, your application requests notification by adding the **notify** flag to the string command it sends. It prepares to receive the notification messages by setting the *hCallback* parameter to its window handle. With the command interface, your application requests notification by adding **MCI_NOTIFY** to the flags sent in *dwParam1*. It prepares to receive the notification messages by setting the **dwCallback** field associated with *dwParam2* to the callback window handle. In both cases the callback window procedure must be able to handle the MM_MCINOTIFY message.

An MCI notification message indicates one of the following results:

- The command completed successfully—MCI_NOTIFY_SUCCESSFUL
- The command was superseded—MCI_NOTIFY_SUPERCEDED
- The command was aborted—MCI_NOTIFY_ABORTED
- The command fails—MCI_NOTIFY_FAILURE

The MOVPLAY sample applications uses notification on the play command to determine when playing stops at the end of the sequence. Once started this way, the sequence plays independently of MOVPLAY and MCI notifies MOVPLAY when the sequence completes. MOVPLAY uses the notification message to rewind the sequence. The main window procedure of MOVPLAY includes the following fragment to handle the notification:

```
case MM_MCINOTIFY:
   // Check the status of an AVI movie that might have been playing.
   // By using MCI_NOTIFY, we will get the MCI_NOTIFY_SUCCESSFUL flag
   // if the play completes on it's own.
   switch(wParam){
      case MCI_NOTIFY_SUCCESSFUL:
         // Playing finished, let's rewind and clear our flag
         fPlaying = FALSE;
         mciSendCommand(wMCIDeviceID, MCI_SEEK,
            MCI_SEEK_TO_START, (DWORD)(LPVOID)NULL);
         return 0;
   }
```

The following fragment shows how the notify flag is used with the **play** command. To use the previous fragment to process the notification message, the handle to the window procedure containing it is specified in *hwnd.*

```
MCI_DGV_PLAY_PARMS    mciPlay;
DWORD                 dwFlags;

mciPlay.dwCallback = MAKELONG(hwnd, 0);
dwFlags = MCI_NOTIFY;

mciSendCommand(wMCIDeviceID, MCI_PLAY, dwFlags, (DWORD)(LPSTR)&mciPlay);
```

For the string interface, the following line uses **mciSendString** to send the **play** command and request notification. The *hwnd* parameter in it would also specify the handle to the window procedure containing the handler for notification.

```
mciSendString("play movie notify", NULL, 0, hwnd);
```

Playing AVI files with MCI

To play an AVI file, your application will perform the following actions:

1. Open the AVI file
2. Set up the playback window
3. Play the AVI sequence
4. Optionally change the playback state
5. Optionally get playback information
6. Close the AVI file

Opening an AVI File

To open an AVI file, your application sends the **open** command to MCIAVI. This command lets your application specify the file. If desired, your application can also specify information about the window used for playback.

If your application plans on opening multiple AVI files, it might open the MCIAVI driver initially by specifying the driver identifier and then open each file separately. This saves time because MCI will not load the MCIAVI driver for each file open command.

If your application will open multiple files, it should include routines like **initAVI** and **termAVI** found in MOVPLAY2.C. The application would use **initAVI** during its initialization and **termAVI** during its termination.

```
// Initialize the MCIAVI driver. This returns TRUE if OK, FALSE on error.
BOOL    initAVI(void)
{
   return mciSendString("open avivideo", NULL, 0, NULL) == 0;
}
```

```
// Close the MCIAVI driver
void termAVI(void)
{
   mciSendString("close avivideo", NULL, 0, NULL);
}
```

When your application uses a filename to open a device, MCI uses the file extension to locate the driver. For example, the following fragment opens MCIAVI using the file "YOSEMITE.AVI" and the alias movie. Subsequent commands for this file can use the alias movie to reference it.

```
if (mciSendString("open yosemite.avi alias movie", NULL, 0, NULL) == 0){

   // open is OK

} else {

   // handle the error

}
```

The **open** command has options to set some playback window characteristics. These options are covered in the next section. For a full example of using the **open** command, see the **fileOpenMovie** function in MOVPLAY1.C and MOVPLAY2.C.

Setting up the Playback Window

Your application can specify several options to define the playback window for playing the AVI sequence. The following options are available to your application:

- Use the default pop-up window of MCIAVI for playing
- Specify a parent window and window style that MCIAVI can use create the playback window
- Specify a playback window for MCIAVI to use for playback
- Play the AVI sequence on a full screen display

If your application does not specify any window options, MCIAVI creates a default window for playing the sequence. MCIAVI creates this playback window for the **open** command but it does not display the window until your application either sends a command to display the window or sends a command to play the file. This default playback window is a sizable pop-up window with a caption, a thick frame, a system menu, and a minimize box.

The application can also specify a parent window handle and a window style when it opens MCIAVI. When opened this way, MCIAVI creates a window based on these specifications instead of the default pop-up window. Your application can specify any window style available for the **CreateWindow** function. Those styles that require a parent window, like WS_CHILD, should include a parent window handle. The following fragment shows how to use the **open** command to set a parent window and create a child of that window:

```
MCI_DGV_OPEN_PARMS    mciOpen;

mciOpen.lpstrElementName = lpstrFile;  // set the file name
mciOpen.dwStyle = WS_CHILD;            // set the style
mciOpen.hWndParent = hWnd;             // give a window handle

if (mciSendCommand(0, MCI_OPEN,
   (DWORD)(MCI_OPEN_ELEMENT|MCI_DGV_OPEN_PARENT|MCI_DGV_OPEN),
   (DWORD)(LPSTR)&mciOpen) == 0){

    // open is OK, continue operation

}
```

Your application can also create its own window and supply the handle to MCIAVI with the **window** command. MCIAVI uses this window instead of the one it might have created for playback. The following fragment finds the dimensions needed to play an AVI file, creates a window corresponding to that size, and has MCIAVI to play the file in the window:

```
HWND        hwnd;
MCI_DGV_RECT_PARMS mciRect;

// Get the movie dimensions with MCI_WHERE
mciSendCommand(wDeviceID, MCI_WHERE, MCI_DGV_WHERE_SOURCE,
    (DWORD)(LPSTR)&mciRect);


// Create the playback window. Make it bigger for the border.                hwndMovie =
CreateWindow("mywindow", "Playback",
                    WS_CHILD|WS_BORDER, 0,0,
                    mciRect.rc.right+(2*GetSystemMetric(SM_CXBORDER)),
                    mciRect.rc.bottom+(2*GetSystemMetric(SM_CYBORDER)),
                    hwndParent, hInstApp, NULL);


if (hwndMovie){
    // Window created OK, make it the playback window.

    MCI_DGV_WINDOW_PARMS    mciWindow;

    mciWindow.hWnd = hwndMovie;
    mciSendCommand(wDeviceID, MCI_WINDOW, MCI_DGV_WINDOW_HWND,
                (DWORD)(LPSTR)&mciWindow);

}
```

When MCIAVI creates the playback window or obtains window handle from your application, it does not display the window until your application either plays the sequence or sends a command to display the window. Your application can use the **window** command to display the window without playing the sequence. The "**window movie state show**" command displays the window using the command string interface. The following fragment shows how to display the window using the command message interface:

```
MCI_DGV_WINDOW_PARMS    mciWindow;

mciWindow.nCmdShow = SW_SHOW;    // set command - see ShowWindow()
mciSendCommand(wDeviceID, MCI_WINDOW, MCI_DGV_WINDOW_STATE,
        (DWORD)(LPSTR)&mciWindow;
```

Your application can also play an AVI sequence full screen instead of in a window. To play full screen, modify the **play** command with the **fullscreen** flag. (Use the MCI_MCIAVI_PLAY_FULLSCREEN flag for the message interface.) When your application uses this flag, MCIAVI uses a 320x240 full screen format for playing the sequence. For example, "**play movie fullscreen**" plays a movie full screen.

With the **fullscreen** flag, movies with 160x120 dimensions play back centered in the 320x240 screen. If your application wants to play these moves in a full 320x240 screen, it can use "**play movie fullscreen by 2**" command to stretch the 160x120 movie to full screen.

Playing the AVI Sequence

Playing an AVI sequence is straightforward using the MCI **play** command. This command can play the entire sequence or portions of it. The previous examples show how use the play command with **mciSendString** and **mciSendCommand**.

Changing the Playback State

Your application can control many of the play back capabilities of MCIAVI. The **pause**, **resume**, **stop,** and **seek** commands let your application control the video sequence. Using these, the application can pause a play in progress, seek to a location within the video sequence, and resume play from that point. The following string command examples show how to use these commands:

```
// assume the file was opened with the alias 'movie'

// pause playing
mciSendString("pause movie", NULL, 0, NULL);

// resume play
mciSendString("resume movie", NULL, 0, NULL);

// stop play
mciSendString("stop movie", NULL, 0, NULL);

// seek to the beginning
mciSendString("seek movie to start", NULL, 0, NULL);
```

Your application can use the **seek** command to move the play position to the beginning, the end, or an arbitrary position in the AVI file. There are two seek modes for the MCIAVI driver—exact or non-exact—which affect the seek position. When seek exactly is enabled (**seek exactly on**), MCIAVI seeks exactly to the frame your application specifies. This might cause a delay if the file is temporally encoded and your application does not specify a key frame. With seek exactly disabled (**seek exactly off**), MCIAVI seeks to the nearest key frame in a temporally encoded file. Your application can change the seek mode with the **set** command. The following example shows how to use the string interface to change the seek mode:

```
// Set seek mode with the string interface
// assume the file was opened with the alias 'movie'
void SetSeekMode(BOOL fExact)
{
   if (fExact)

      mciSendString("set movie seek exactly on", NULL, 0, NULL);

   else

      mciSendString("set movie seek exactly off", NULL, 0,
            NULL);
}
```

Other MCI commands let your application alter the play other than altering the control flow of the play. For example, an AVI sequence by default plays at its normal rate of speed. Your application can change the play rate to speed up or slow down the playback. The **speed** flag for the **set** command lets your application control the play rate. For AVI sequences, a speed value of 1000 is considered normal. Thus, to play a movie at half-speed, your application can use the command string "**set movie speed 500.**" Alternatively, it can use "**set movie speed 2000**" to play the sequence at twice the normal rate.

The **setaudio** command lets your application control the audio portion of an AVI sequence. You application can mute audio during playback, or in the case of multiple audio stream files, select the audio stream played. For example, the "**setaudio movie off**" command string turns audio off during playback. The "**setaudio movie stream to n**" command string specifies the audio stream number (specified by **n**) played for the sequence.

MCIAVI has a dialog box to control some of its playback options. Some of the important option available to the user include selection of windowed or full screen playback, selection of the seek mode, and zooming the image. Your application can have MCIAVI display this dialog box with the **configure** command. For more information on this dialog box, see “MCI String Messages for MCIAVI.”

Obtaining Playback Information

Your application can get the status on the playback of an AVI sequence with the **status** command. This command obtains information on the state of the audio, state of the video, mode of the play, position of the play, seek mode, as well as other parameters. Your application might monitor playback so that it can update the state and position of the play in a routine that gets called through a timer call-back. The information returned by the **status** command can depend on the time format used. The device can specify the return values for position, length, and start in milliseconds or frames. Your application can use the **set** command to set alternate time formats and modes. The following fragment shows an example of such a function:

```
MCI_DGV_SET_PARMS       mciSet;

MCI_DGV_STATUS_PARMS    mciStatus;

// put in frame mode
mciSet.dwTimeFormat=MCI_FORMAT_FRAMES;
mciSendCommand(wDeviceID, MCI_SET,
            MCI_SET_TIME_FORMAT,
            (DWORD)(LPSTR)&mciSet);

mciStatus.dwItem = MCI_STATUS_MODE;
mciSendCommand(wDeviceID, MCI_STATUS,
            MCI_STATUS_ITEM,
            (DWORD)(LPSTR)&mciStatus);

// Update mode based on mciStatus.dwReturn

// If it is playing then get the position
if (mciStatus.dwReturn == MCI_MODE_PLAY){
        mciStatus.dwItem = MCI_STATUS_POSITION;
        mciSendCommand(wDeviceID, MCI_STATUS, MCI_STATUS_ITEM,
                        (DWORD)(LPSTR)&mciStatus);

        // update the position from mciStatus.dwReturn
}
```

Closing the AVI File

When finished with a file, your application closes it with the **close** command. With the string interface a "**close movie**" command is sent, with the command interface a **MCI_CLOSE** command is used and all parameters may be NULL.