

Windows Multimedia

Some Multimedia Devices

- **Some multimedia devices:**
 - Waveform audio device (sound card)
 - converts microphone & other analog audio to digitized samples (ADC)
 - can be stored as .WAV files
 - can be played back (DAC)
 - Also usually has a MIDI device
 - Musical Instrument Digital Interface
 - plays musical notes in response to short binary messages (MIDI codes)
 - can be attached to a MIDI input device (music keyboard)
 - CD Audio through the CD-ROM drive
 - Video for Windows device (AVI video device)
 - plays movie/animation files (.AVI)
 - QuickTime and MPEG movies
 - Video capture boards (different compression schemes)
 - Laserdisc players & video cassette recorders
 - Others (DVD)

Win32 MM Support & Documentation

- Extensive Win32 API support for multimedia devices
 - Low-level support
 - High-level support
- MSDN online documentation:
 - Platform SDK/Graphics & Multimedia Services/Multimedia Reference
 - Platform, SDK, & DDK/Platform SDK/Reference/Multimedia Command Strings
 - Visual Studio .NET Help (Index):
- Visual Studio Help on “MCI Command Strings”

Media Control Interface

- MCI (Media Control Interface)
 - High level multimedia control functions
 - Has commands common to all multimedia hardware
 - Possible since most use record/play metaphor
 - Open a device for input or output
 - If input, record; If output, play
 - When done, close the device
 - Some MCI Device Names:
 - cdaudio, waveaudio, sequencer (MIDI), videodisc, vcr, overlay (analog video in a window), dat (digital audio tape), digitalvideo
 - Categorized as “simple” or “compound”
 - Simple devices don’t use files (e.g., cdaudio)
 - Compound devices do (e.g., waveaudio use .WAV files)

Two Forms of MCI

- Send command messages (like Windows messages) to MCI
 - (need to include bit-encoded flags and C data structures)
- Send text strings to MCI
 - Good for use from scripting languages with string functionality and simple to use
 - MCI converts the strings to command messages

Sending Strings to MCI

- **mcisendstring() function:**

```
error = mcisendstring(sCmd, sRetStr, iReturn, hCallback);
```

 - sCmd--the MCI command string (specifies command & device)
 - sRetStr--return string buffer (NULL if none used)
 - iReturn--size of return string buffer (0 if none used)
 - hCallback--Handle to Callback window (NULL if none used)
 - Window to display info if “notify” flag was specified in cmd string
 - Usually NULL
 - Returns 0 if command is successful, an error code if not
 - Error code can be used as a parameter to mciGetErrorString()
 - Many command strings possible
- See MSDN online help
- In .NET, see help on:
 - mciSendString, mciGetErrorString
 - MCI Command Strings

Using Win32 Functions (like MCI) From .NET

- MCI is not directly accessible from .NET
- Also `mciSendString()` is C++, not C#
- But we can still use MCI and other Win32 API functions from .Net languages
- Key is to use “Platform Invocation Services”
 - “Interop Services”
 - A generalized mechanism that allows calling functions that are imported from DLLs
 - Drawbacks:
 - Code is no longer managed code
 - And it’s no longer platform independent

Win32 from .NET, continued

- Must include: `System.Runtime.InteropServices;`
- And then prefix any declarations of Win32 API functions to be used with:
`[DllImport(“xxx.dll”)]`
 - `DllImport`: A storage-class attribute:
 - A Microsoft-specific extension to C/C++
 - Enables importing functions, data, objects from a DLL
 - Where `xxx.dll` is the DLL that contains the function
 - For MCI functions that DLL is `winmm.dll`
- Also the declaration must include `public, static, extern` to be usable from a .NET application
- And then use equivalent .NET language data types for the parameters and for the type returned by the function

mciSendString() in .NET Unmanaged Code

- Its VC++ the parameter types are:
 - LPCTSTR, LPTSTR, UINT, HANDLE
- And it returns MCIERROR: a C++ DWORD
- Corresponding C# parameter types would be:
 - string, string, uint, IntPtr
 - In C# DWORD is implemented as an int
- So declare mciSendString as:

```
[DllImport("winmm.dll")]  
public static extern int mciSendString  
(string sCmd, string sRetStr, uint iReturn, IntPtr hCallback);
```

Some MCI Command String Commands:

- open -- initializes a multimedia device
- play-- starts playing an open device
- stop -- stops playing from an open device
- record -- starts recording to a device
- seek -- move to a specified position on device
- save -- saves an MCI file
- close -- closes a device and associated resources
- set -- establish control settings for the device
- status -- returns information in 2nd parameter
- Some device types:
 - cdaudio -- Audio CD played on system's CD-ROM
 - waveaudio -- .WAV audio files
 - AVIVideo -- .AVI video files

Some Example Command Strings

```
"open cdaudio"  
"play cdaudio"  
"close cdaudio"  
"open new type waveaudio alias mysound"  
"record mysound"  
"stop mysound"  
"save mysound mysound.wav"  
"open myvoice.wav alias voiceclip" //can open different types of media files like this  
"open AVIVideo!myclip.avi alias vidclip" //or specify a specific type  
  - the ! separates dev_name from file_name  
"play voiceclip" "stop voiceclip" "close voiceclip"  
"play vidclip" "stop vidclip" "close vidclip"  
"set mysound time format milliseconds"  
"status mysound length" -- Returns duration of mysound in milliseconds  
"set cdaudio time format tmsf"  
  - tmsf means tracks, minutes, seconds, frames (default format is msf)  
"play cdaudio from 01:00:00:00 to 02:05:06:00"  
  - tt=track (1-99), mm=minute (0-59), ss=second (0-59), ff=frame (0-74)  
  - a frame is 1/75 of a second  
"status cdaudio position" -- Returns position on audio CD in current-time-format  
"status cdaudio length track xx" -- Returns current-time-format length of CD track xx
```

Examples

- MCI-PlayCD
 - “Play” Button
 - Opens and plays cdaudio device
 - ”Stop” Button
 - Stops and closes cdaudio device
- mciSendString-Test
 - User can enter different command strings in a text box
- MCI-Record-Play
 - Must have a microphone attached to the computer
 - “Begin Recording” and “End Recording” buttons
 - Open, record, and save microphone input to a .WAV file
 - “Begin Playback” and “End Playback” buttons
 - Plays back the .WAV file

Retrieving Data from MM Commands

- Some `mciSendString()` commands provide data
 - Returned in second parameter: `szRetStr`
 - Example: “status” command
- Also `mciGetErrorString(err, errStr, lengErrStr);`
 - `err` was the value returned by `mciSendString()`
 - `errStr` will contain text describing the error
- Problem: a C# string cannot grow dynamically
 - Need another “dynamic” string-like data type to hold the data returned in the 2nd parameter
 - `StringBuilder` class (in `System.Text`) does the job
 - An instance of this class represents a string-like object whose value is a “mutable” sequence of characters
 - So it can be used to “receive” a return string object in a method
 - One constructor:
`StringBuilder sb = new StringBuilder(initLength);`

Using StringBuilder with MM in .NET

- Declare 2nd parameter as type `StringBuilder`
- For example:

```
[DllImport("winmm.dll")]
public static extern int mciSendString
(string sCmd, StringBuilder sRetStr, int iLength, IntPtr
hCallback);
```
- Then use it, for example:

```
StringBuilder sb = new StringBuilder(256);
string s = “status cdaudio length”
int error = mciSendString(s, sb, 256, IntPtr.Zero);
```
- Then Convert returned `StringBuilder` object to a string to be able to display it or use it, for example:

```
string sRet = sb.ToString();
```
- Don’t forget using `System.Text`; at top of application
- Example Pgm: [mcisendstring-mcierrorstring-strbuilder](#)

Using Windows Media Player

- A control that enables playing video/sound in many formats
 - MPEG, AVI, WAV, MIDI, etc.
- Add it to Visual Studio Tool Box
 - “Tools” | “Choose Toolbox Items” ↗ “Choose Toolbox Items” dialog box
 - Click “COM Components” tab and scroll down to “Windows Media Player”
 - Click “OK” and note the control is added to the toolbox
 - Drag it to form and use it just like other controls
 - Important property: URL – specifies media player control object to be opened and played
- Example Program: MediaPlayer
 - Menu Item “Open” starts Common File Dialog Box
 - Chosen file is loaded into Media Player Control