PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/06	A2	(11) International Publication Number: WO 98/45773
		(43) International Publication Date: 15 October 1998 (15.10.98)

(21) International Application Number: PCT/US98/06593

(22) International Filing Date: 3 April 1998 (03.04.98)

(30) Priority Data:
08/833,384 4 April 1997 (04.04.97) US

(71) Applicant: ANDREA ELECTRONICS CORPORATION [US/US]; 11–40 45th Road, Long Island City, NY 11101 (US).

(72) Inventors: JONES, Gary; 959 North Sage Drive, Pleasant Grove, UT 84062 (US). SHOELL, Leonard; 966 North 1020 East, Pleasant Grove, UT 84062 (US).

(74) Agents: KOWALSKI, Thomas, J. et al.; Frommer Lawrence & Haug LLP, 745 Fifth Avenue, New York, NY 10151 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR PROVIDING AUDIO UTILITY SOFTWARE FOR USE IN WINDOWS APPLICATIONS

(57) Abstract

A method is provided for allowing a WINDOWS computer user to effectively utilize audio applications in a WINDOWS 95 or WINDOWS NT operating system by individually setting the controls of the computer sound card by an improved dialog–based tabbed menu system. Upon loading and starting the computer software system of the present invention in the WINDOWS based computer, the WINDOWS 95 or NT operating system is queried for the type of computer sound card driver installed. Next, the computer sound card driver software is queried to determine the capabilities and requirements of a computer sound card and is adjusted to meet the specifics capabilities and requirements of the computer sound card detected. Further, the present software system allows the user to conveniently organize dialog–based tabbed menus for controlling the computer sound card settings with provision for simplifying the configuration of microphone related settings. Further, the present invention includes a built–in validation capability which alerts the user when improper related settings are detected and provides the user an easy method of correction.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

METHOD AND APPARATUS FOR PROVIDING AUDIO UTILITY SOFTWARE FOR USE IN WINDOWS APPLICATIONS

FIELD OF THE INVENTION

The present invention relates to method and apparatus for allowing a user to efficiently display and adjust computer sound card settings, such as the volume, playback, and recordings settings, which effect the quality of sound of audio applications run on a WINDOWS computer utilizing a WINDOWS 95 or WINDOWS NT operating system.

The invention further relates to a software system that automatically detects improper microphone related computer sound card settings after detecting the type of computer sound card installed in the computer and allows the user to quickly correct these improper settings by changing them individually or by forcing them collectively to nominal default values. The present invention allows the user to configure or re-configure the computer sound card settings and certain aspects of its dialog-based tabbed menu configuration at any time.

BACKGROUND OF THE INVENTION

As is to be appreciated, in the age of cyberspace, many different types of WINDOWS computers with WINDOWS 95 or WINDOWS NT operating systems are configured with many different computer sound cards, sound accessories, and sound applications software, each system requiring its own set of instruction manuals to fully describe the operation of the computer configuration. With many audio software applications, there exist an increasing need to simply the task the users must face when utilizing WINDOWS computers with the wide variety of available audio devices.

However, to utilize many of these audio applications now available in many computers equipped with audio devices such as headsets and handsets,

speakers, and musical input and output devices, the computer sound card settings must be properly configured by the computer user to satisfy the unique requirements of the particular computer system. Many times, the user must consult the computer sound card, audio accessory, audio software application, and operating system instruction manuals to properly configure the computer sound card.

Even an experienced user may be overwhelmed with all the different computer sound card settings, and to expect the novice to read all the associated instruction manuals and understand which settings are correct and how to set them for effective sound performance is unrealistic. As a result, many users shy away from instruction manuals and fumble by trial and error to find the proper sound card setting or instead opt not to utilize some or all of the audio capabilities of the computer audio applications requiring microphones.

Thus, there is a need for a method that simplifies the process of configuring the computer sound card settings associated with audio applications of WINDOWS computer systems. The prior art has failed to provide a relatively easy means to ensure proper computer sound card settings when audio devices are used with computer sound cards and audio software applications to receive, process, and output accurate audio signals relating to telephony systems involving microphones. The prior art has further failed to provide a system for organizing user configurable dialog-based tabbed menus for controlling computer sound card settings and for alerting the user of improper microphone related computer sound card settings and quickly correcting improper computer sound card settings via internal means.

OBJECTS AND SUMMARY OF THE INVENTION

An object of the present invention is to provide a method of configuring the computer sound card settings of the audio system of a WINDOWS computer for effective use with audio driven applications running under WINDOWS 95 or WINDOWS NT operating systems, which substantially reduces or eliminates problems with the prior art.

More specifically, it is an object of the present invention to provide a software system that enables effective usage of audio devices including a microphone in connection with a WINDOWS computer having varying computer sound cards in a WINDOWS 95 or WINDOWS NT environment.

Another object of the present invention is to provide a method for configuring a WINDOWS computer for audio applications, comprising the steps of: providing a WINDOWS computer having a WINDOWS 95 or WINDOWS NT operating system with means to install data stored in a medium; installing the data stored in the medium to operate in a dialog-based menu format utilizing a smart icon on a taskbar tray; querying the operating system to detect a computer sound card software driver; querying the computer sound card software driver to determine the type of computer sound card installed; and inputting default or user selected settings pertaining to the operation of the computer sound card detected.

Another object of the present invention is to provide a relatively easy to use software system that alerts the user of a WINDOWS computer and a WINDOWS 95 or WINDOWS NT operating system of improper computer sound card settings which may adversely effect microphone performance.

Another object of the present invention is to provide the user of a WINDOWS computer and a WINDOWS 95 or WINDOWS NT operating system a relatively easy to use software system that provides the user a means to quickly

correct improper computer sound card settings which may adversely effect microphone performance.

Another object of the present invention is to provide an enhanced user-friendly dialog-based tabbed menu system in a WINDOWS 95 or WINDOWS NT environment that contains the microphone related computer sound card settings required for proper microphone operation that typically include the balance, volume, and mute/select settings for the microphone related recording and playback functions utilized in audio applications.

Another object of the present invention is to provide a computerized system for controlling the computer sound card audio functions in a WINDOWS computer with WINDOWS 95 or WINDOWS NT operating system, comprising: a WINDOWS computer having a computer sound card means to provide a user the ability to perform certain audio applications; a data means installed in the computer to determine computer sound card capabilities and appropriate settings of the computer sound card; a plurality of tabbed dialog means displayed on a computer screen that allow the user to select which audio settings to manipulate for the certain audio application; an alert means displayed on the tabbed dialog means to notify the user of individual controls which are set improperly and may adversely effect microphone performance; a dialog pushbutton means to force all of the computer sound card microphone related settings to nominal default values; and an alert means displayed on the taskbar tray to notify the user of improper audio settings which may adversely effect microphone performance. In the preferred system, said audio functions typically comprise balance, volume, mute/select, and sometimes VU meter controls which are displayed in tabbed dialogs. In the preferred system, the tabbed dialog means comprise at least a microphone tabbed dialog and an options tabbed dialog or comprise a microphone tabbed dialog, a recording tabbed dialog, a

playback tabbed dialog, and an options tabbed dialog. It is preferred that the microphone tabbed dialog controls a gain of a microphone input and other settings effecting microphone performance, while the recording tabbed dialog typically individually and collectively controls a plurality of computer sound card audio input settings and the playback tabbed dialog typically individually and collectively controls a plurality of computer sound card output settings. In addition, it is preferred that the options tabbed dialog controls the display of the recording tabbed dialog, playback tabbed dialog, and smart icon alert means displayed on the taskbar tray.

More specifically, it is another object of the present invention to provide the preferred default dialog which contains secondary tabbed dialogs consisting of a microphone tabbed dialog and an options tabbed dialog, where the microphone tabbed dialog is comprised of a recording box and a playback box. The contents of the preferred recording box in the microphone tabbed dialog depends upon the characteristics and requirements of the computer sound card and includes a recording group and a microphone group, where the recording group is typically comprised of balance, volume, mute/select and VU meter controls and the microphone group is typically comprised of balance, volume, and mute/select controls. In addition, the contents of the preferred playback box in the microphone tabbed dialog depends upon the characteristics and requirements of the computer sound card and typically includes volume control, wave, and option groups where the volume control group is typically comprised of balance, volume, and mute/select controls, where the wave group is typically comprised of balance, volume, mute/select, and VU meter controls, and where the options group is typically comprised of a control to prevent speaker feedback into the microphone. The preferred default dialog is expandable by user selection to include a recording tabbed dialog and/or a playback

tabbed dialog. The contents of the recording tabbed dialog depends upon the characteristics and requirements of the computer sound card and is typically comprised of a recording group, a MIDI group, a CD audio group, a line-in group, and a microphone group where each group is typically comprised of a balance, volume, mute/select, and sometimes VU meter controls. The contents of the playback tabbed dialog depends upon the characteristics and requirements of the computer sound card and is typically comprised of a volume control group, a wave group, a MIDI group, a CD audio group, a line-in group, a microphone group, and a PC Speaker group where each group is typically comprised of a balance, volume, mute/select, and sometimes VU meter controls. An additional step of the method is to selectively establish a smart icon on the taskbar tray of the WINDOWS desktop to alert a user of an improper computer sound card setting possibly effecting microphone performance and to provide a set default means to allow the user to easily establish nominal values for all microphone related settings to promote effective microphone performance for audio applications using microphone input. The preferred options tabbed dialog allows the user to display the recording tabbed dialog and associated boxes and controls, the playback tabbed dialog and associated boxes and controls and/or a smart icon on the taskbar tray.

Other objects, features and advantages according to the present invention will become apparent from the following detailed description of the illustrated embodiments when read in conjunction with the accompanying drawings in which corresponding components are identified by the same reference numerals.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a representative screen display of one of the setup/installation steps illustrating a

picture of a portion of the computer hardware utilized by the present invention;

Fig. 2 is a representative screen display of one of the setup/installation steps illustrating installation progress indicators of the present invention;

Fig. 3 is a representative screen display illustrating the WINDOWS computer screen with a main dialog containing tabbed dialogs, buttons, and computer sound card identifier and a smart icon on the taskbar tray;

Fig. 4 is a representative screen display illustrating the features of the microphone tabbed dialog according to the present invention;

Fig. 5 is a representative screen display similar to Fig. 4, except an improper microphone related setting is illustrated;

Fig. 6 is a representative screen display illustrating the features of the options tabbed dialog according to the present invention;

Fig. 7 is a representative screen display similar to Fig.6, except the recording and playback tabbed dialogs are selected according to the present invention;

Fig. 8 is a representative screen display illustrating the features of the recording tabbed dialog according to the present invention;

Fig. 9 is a representative screen display similar to Fig. 8 except an alternate computer sound card driver is installed according to the present invention;

Fig. 10 is a representative screen display illustrating the features of the playback tabbed dialog according to the present invention;

Fig. 11 is a representative screen display similar to
Fig. 10 except an alternate computer sound card driver is installed according to the present invention; and

Fig. 12 is a flowchart illustrating the operations of the present invention with a WINDOWS 95 or WINDOWS NT operating
system.

BRIEF DESCRIPTION OF THE APPENDICES

Appendix A contains the source code for the computer system of the present invention and is incorporated by reference
herein. Appendix A contains eight modules, A1-A8.

Module A1 (Exhibit B) W contains an installation script of the present invention in a WINDOWS 95 or WINDOWS NT platform.

Module A2 (Exhibits C-D) contains an audio routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

Module A3 (Exhibits E-F) contains an hidden window routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

Module A4 (Exhibits G-H) contains an initialization dialogue routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

Module A4 (Exhibits I-J) contains a microphone control routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

Module A (Exhibits K-L) contains a microphone control dialogue routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

Module A6 (Exhibits M-N) contains a microphone property sheet routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

Module A7 (Exhibits O-P) contains an options dialogue routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

Module A8 (Exhibits Q-R) contains a standard system files routine of the present invention in C++ in a WINDOWS 95 or WINDOWS NT platform.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Fig. 1 illustrates a computer monitor 1, a computer base unit 2 having a hard disk drive (not shown), a CD drive 3, a keyboard (not shown), a mouse (not shown) and cable connections (not shown) from the base unit 2 to the monitor 1 to an electrical power supply (not shown). The present invention is preferably available on a CD 4 or floppy disk(s) 5, which is inserted into the CD drive 3 or floppy disk drive (not shown), and installed onto the hard disk (not shown) of the base unit 2, and which operates under either a WINDOWS 95 or WINDOWS NT operating system. The present invention operates on a WINDOWS computer with WINDOWS 95 or WINDOWS NT operating system and quickly identifies and provides an easy method of correction of microphone related sound problems that commonly plague audio device users.

In Fig. 2, as the information from the CD or floppy disk(s) is copied and installed onto the hard disk of the computer, progress is displayed by the progress indicators 10 and
11. After installation and upon startup, the present invention first queries the operating system to determine which computer sound card driver is installed. The computer sound card driver is then queried and software system of the present invention internally configures itself to support the detected computer sound card's specific capabilities and requirements. For instance in Figs. 3-8, and 10 the Sound Blaster 16 computer sound card is detected, while in Figs. 9 and 11, the ESS

AudioDrive computer sound card is detected. The software system of the present invention may be uninstalled from the hard disk of the base unit 2 using the combined uninstallation features of the present invention and the WINDOWS 95 or WINDOWS NT operating system.

In Fig. 3, the software system of the present invention has tabs for the quick selection of conveniently organized dialogs for displaying and adjusting computer sound card settings. When the software system of the present invention is installed and started for the first time, the system defaults to its simplest configuration with the desktop display 20 containing a main dialog 21 containing a close button 23, a about button 24, a computer sound card descriptor 22, and two tabbed dialogs: a selected tabbed dialog 25 for microphone related functions and a tabbed dialog 26 for options selection. If desired, the software system of the present invention will put a smart icon 28 on the taskbar tray 27 to be visible at all times in the lower right hand corner of the desktop display 20. The smart icon 28 performs a plurality of functions. When smart icon 28 is clicked on, the main dialog 21 is displayed within the desktop display 20. When the verification feature of the present computer system detects a computer sound card setting which may adversely effect microphone operation, the smart icon 28 changes its appearance on the taskbar tray 27 to alert the user of computer sound card settings which may adversely effect microphone performance.

In Fig. 4, the selected microphone tabbed dialog 30 is displayed within main dialog 21. The computer sound card microphone related settings and functions contained within the microphone tabbed dialog 30 include a set defaults pushbutton 35 with a possible warning icon 71 and a warning message 35 (both shown in Fig. 5) and a plurality of controls arranged into a recording box 36 and a playback box 37. All of the

controls contained in the recording box 36 and the playback box 37 are also contained in either the recording or the playback tabbed dialogs shown in Figs. 8 or 10 and have been put in the microphone tabbed dialog 30 to bring all the controls that effect microphone performance together in one place to make it convenient for the user to adjust the computer sound card settings that are relevant to microphone operation.

In Fig. 4, the recording box 36 is displayed within the tabbed dialog 30 and is comprised of the recording group 58 and the microphone group 59. The recording group 58 includes a recording label 38, which identifies a balance control 39, a volume control 40, a select checkbox 41, and a VU meter 42. The VU meter 42 displays sound levels when the computer sound card is receiving audio input and passing it to an active application such as a sound recorder or voice recognition program. The imbalance control 39, the volume control 40*, and the select checkbox 41 typically effect all computer sound card audio inputs and are called master controls. This group of master controls are typically fixed at nominal values by the computer sound card driver and not adjustable by the user. The microphone group 59 includes a microphone label 43, which identifies a balance control 44, a volume control 45, and a select checkbox 46. Most computer sound cards support only monaural microphones and thus the microphone balance control 44 is typically fixed by the computer sound card driver at the center position and is not adjustable by the user. The volume control 45 provides control of the gain of the microphone input signal. The select checkbox 46 either enables or mutes the microphone input. When the select checkbox 46 is checked, the microphone input is enabled and when the checkbox 46 is not checked, the microphone input is muted.

In Fig. 4, the playback box 37 is displayed within the

tabbed dialog 30 and is comprised of a volume control group 60, a wave group 61, and an options group 62. The volume control group 60 includes a volume control label 47, which identifies a balance control 48, a volume control 49, and a mute all checkbox 50. In the volume control group 60, the balance control 48, the volume control 49, and the mute all checkbox 50 typically effect computer sound card audio outputs regardless of the source of the signal and are called master controls. The wave group 61 includes a wave label 51 which identifies a balance control 52, a volume control 53, a mute checkbox 54, and a VU meter 55. The balance control 52, the volume control 53, and the mute checkbox 54 effect the audio played back from wave files. The VU meter 55 displays sound levels when wave files are played. The options group 62 includes an options label 56, which identifies a check to prevent speaker feedback into microphone checkbox 57. The check to prevent speaker feedback into microphone checkbox 57 of the options group 62 controls whether or not audio signals received from the microphone are routed directly to the computer sound card output. It is preferred that the check to prevent speaker feedback into microphone checkbox 57 be checked to avoid speaker feedback into the microphone by disabling the direct routing of the microphone input signal to the computer sound card output.

As shown in Fig. 5, the present software system has a built-in validation capability that displays a yellow alert symbol in a plurality of locations when one or more of the microphone related controls is set improperly. For example, when the select checkbox 46 is not in the preferred checked condition which prevents proper microphone operation, a yellow alert symbol 46 appears next to the incorrectly set select checkbox 72 and a yellow alert symbol 71 appears to the right of the set defaults button 35 next to a descriptive message 73 describing the nature of the detected problem. The set

defaults button 35 when operated forces all of the microphone related controls within the microphone tabbed dialog 30 to nominal settings which in turn causes the removal of any yellow alert symbols that have been displayed.

In Fig. 6, the options tabbed dialog 31 allows the user the ability to control a plurality of options available within the software system of the present invention, including the placement of the smart icon 28 on the taskbar tray 27 as illustrated in Fig. 3. The options menu is comprised of a display options box 80 and a system box 81. The display options box 80 is comprised of a show recording controls checkbox 82 and a show playback controls checkbox 83. The show recording controls checkbox 82 displays or hides the recording tabbed dialog 90 depending on whether or not the user selected the show the recording controls checkbox 82. Similarly, the show playback controls checkbox 83 displays or hides the playback tabbed dialog 91 depending on whether or not the user selected the show playback controls checkbox 83. If the user selects either or both checkboxes 82 and 83, the tab(s) of the recording tabbed dialog 90 and/or playback tabbed dialog 91 will be displayed as shown in Figs. 7-11. The system box 81 is comprised of the show the microphone control on the taskbar checkbox 84, which adds or removes the smart icon 28 from the taskbar tray 27 (shown in Fig. 3) depending on whether or not the user selects the show the microphone control on the taskbar show the microphone control on the taskbar checkbox 84. It is preferred that the checkbox 84 be selected at all times to assist proper operation of the microphone.

In Fig. 8, the recording tabbed dialog 90 provides the user control of all the recording features supported by the Sound Blaster 16 computer sound card and is comprised of a recording group 101, a MIDI group 102, a CD audio group 103, a line-in group 104, and microphone

group 105. In Fig. 9, an ESS AudioDrive computer sound card is installed as evidenced by the computer sound card descriptor 136 and accordingly the recording tabbed dialog 90 is comprised of a recording group 130, a linein group 131, a microphone group 132, a CD audio group 133, a synthesizer group 134, and an AuxB group 135. The difference in groups displayed in Fig. 9 from those displayed in Fig. 8 illustrate how the present invention adjusts according to the requirements and capabilities of the different computer sound cards installed in the WINDOWS computer.

In Fig. 8, the recording group 101 contains a recording label 106 identifying a balance control 107, a volume control 108, a mute all checkbox 109, and a VU meter 110. The recording group 101 functions identically to and adjusts the same computer sound card settings as the recording group 58 of the recording box 36 of the microphone tabbed dialog 30 shown in Fig. 4. The MIDI group 102 contains a MTDT label 111 identifying a balance control 112, a volume control 113, and a select checkbox 100 to adjust the computer sound card MTDT stereo input's balance, gain, and mute settings. The CD audio group 103 contains a CD audio label 115 identifying a balance control 116, a volume control 117, and a select checkbox 118 to adjust the computer sound card CD Audio stereo input's balance, gain, and mute settings. The line-in group 104 contains a line-in label 119 identifying a balance control 120, a volume control 121, and a select checkbox 122 to adjust the computer sound card line-in stereo input's balance, gain, and mute settings. The microphone group 105 contains a microphone label 123 identifying a balance control 124, a volume control 125, and a select checkbox 126. The microphone group 105 functions identically to and adjusts the same computer sound card settings as the microphone group 59 of the recording box 36 of the microphone tabbed dialog 30 shown in Fig. 4.

In Fig. 10, the playback tabbed dialog 91 provides the user control of all the playback features supported by the Sound Blaster 16 computer sound card and is comprised of a volume control group 141, a wave group 142, a MIDI group 143, a CD audio group 144, a line-in group 145, a microphone group 146, and a PC speaker group 147. In Fig. 11, an ESS AudioDrive computer sound card is installed as evidenced by the computer sound card descriptor 187 and accordingly the playback tabbed dialog 91 is comprised of a volume control group 180, a line-in group 181, a wave group 182, a microphone group 183, a CD audio group 184, a synthesizer group 185, and an AuxB group 186. The difference in groups displayed in Fig. 11 from those displayed in Fig. 10 illustrate how the present invention adjusts according to the requirements and capabilities of the different computer sound cards installed in the WINDOWS computer.

In Fig. 10, the volume control group 141 contains a volume control label 148, a balance control 149, a volume control 150, and a mute all control 151. The volume control group 141 functions identically to and adjusts the same computer sound card settings as the volume control group 60 of the playback box 37 of the microphone tabbed dialog 30 shown in Fig. 4. The wave group 142 contains a wave label 152 identifying a balance control 153, a volume control 154, a mute checkbox 155, and a VU meter 156 to adjust the computer sound card wave stereo output's balance, gain, and mute settings and display the sound level of wave files as they are played. The MIDI group 143 contains a MIDI label 157 identifying a balance control 158, a volume control 159, and a mute checkbox 160 to adjust the computer sound card MIDI stereo output's balance, gain, and mute settings. The CD audio group 144 contains a CD audio label 161 identifying a balance control 162, a volume control 163, and a mute checkbox 164 to adjust the computer sound card CD Audio stereo output's balance, gain, and mute settings. The

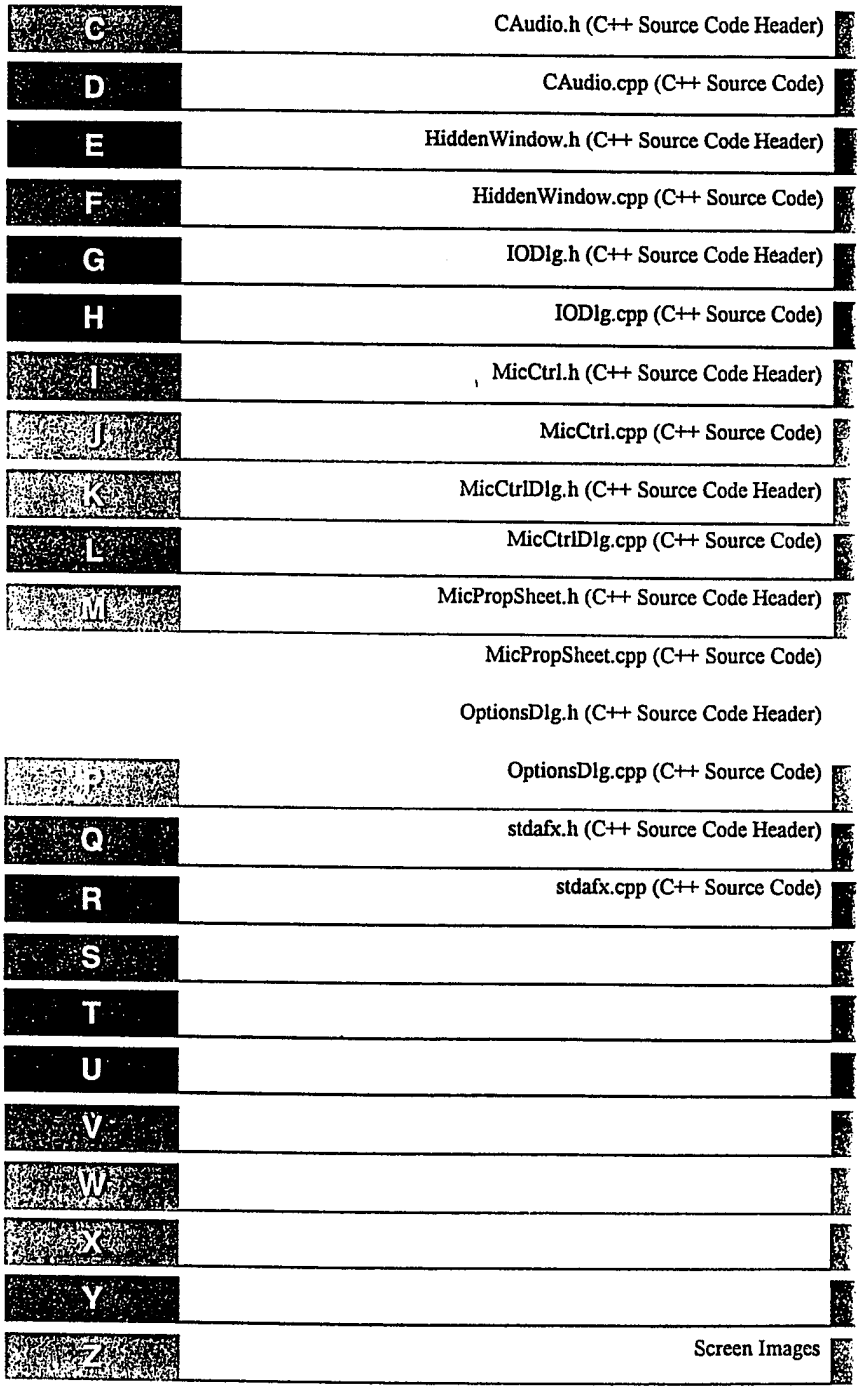
line-in group 145 contains a line-in label 165 identifying a balance control 166, a volume control 167, and 20a mute checkbox 168 to adjust the computer sound card line-in stereo output's balance, gain, and mute settings. The microphone group 146 contains a microphone label 169 identifying a balance control 170, a volume control 171, and a mute checkbox 172 to adjust the computer sound card microphone monaural output's gain and mute settings. The microphone balance control 170 is permanently set to the center position by the computer sound card driver and is not adjustable by the user. The microphone mute checkbox 172 performs the same function as the check to prevent speaker feedback into microphone checkbox 57 shown in Fig.4 to prevent speaker feedback into the microphone. The PC speaker group 147 contains a PC speaker label 173 identifying a balance control 174, a volume control 175, and a mute checkbox 176 to adjust the computer sound card PC speaker monaural output's gain and mute settings. The PC speaker balance control 174 is permanently set by the computer sound card driver to the center position and is not adjustable by the user.

Fig. 12 illustrates a block diagram of the present computer system 195 when operating in a WINDOWS 95 or WINDOWS NT environment. A WINDOWS operating system 190 is an event-based, message-driven operating system. During the execution of the present computer system 195, every time the user takes an action that affects a window screen by clicking on the mouse, the user's action triggers a user event 191. Each time the user event 191 is triggered, the WINDOWS operating system 190 sends a message 193 to the present computer system 195 so that the present computer system 195 can handle the user event 191. When the WINDOWS operating system 190 detects the user event 191 and generates the message 193 in response, it places the message 193 in a message queue 192 that is dedicated to the application program 195. When the present computer system 195 needs to determine whether

any user events 191 have occurred and what kind of event they are, the present computer system 195 gets the information it needs by retrieving message 194 that the WINDOWS operating system 190 has placed in the message queue 192.

To retrieve message 194, the present computer system 195 repeatedly checks the status of the message queue 192 by executing a series of statements that execute in a message pump loop 197. Each time the present computer system 195 receives the message 194, the window procedure (WndProc) 199 determines how the message 194 should be handled. The present computer system 195 then passes information relating to the message 194 to a message handler 200. The message handler 200 is specially designed to implement much of the present computer system's 195 individual behavior.

Furthermore, although preferred embodiments of the present invention and modifications thereof have been described in detail herein, it is to be understood that this invention is not limited to those precise embodiments and modifications, and that other modifications and variations may be affected by one skilled in the art without departing from the spirit and scope of the invention as defined by the appended claims.


SETUP.RUL (Installation Script)
C
CAudio.h (C++ Source Code Header)
D
CAudio.cpp (C++ Source Code)
E
HiddenWindow.h (C++ Source Code Header)
F
HiddenWindow.cpp (C++ Source Code)
G
IODlg.h (C++ Source Code Header)
H
IODlg.cpp (C++ Source Code)
I
MicCtrl.h (C++ Source Code Header)
J
MicCtrl.cpp (C++ Source Code)
K
MicCtrlDlg.h (C++ Source Code Header)
L
MicCtrlDlg.cpp (C++ Source Code)
M
MicPropSheet.h (C++ Source Code Header)
MicPropSheet.cpp (C++ Source Code)
OptionsDlg.h (C++ Source Code Header)
OptionsDlg.cpp (C++ Source Code)
Q
stdafx.h (C++ Source Code Header)
R
stdafx.cpp (C++ Source Code)
S
T
U
V
W
X
Y
Z
Screen Images

Andrea ELECTRONICS CORPORATION **PRODUCT SPECIFICATION** AudioCOMMANDER®1

Audio Utility Software

20 January 1997



Specifications/Features:

- CD ROM
- Runs under Windows 95/NT
- Works with all sound cards
- Fast dialog tab access to all functions
- Smart icon warns of improper settings
- Default button corrects most common sound problems
- Standard Windows 95/NT installation/uninstallation

AudioCOMMANDER®1 Overview

AudioCOMMANDER®1 is a Windows 95/NT utility which quickly identifies and corrects sound problems that commonly plague multimedia users—particularly those problems associated with microphone use.

When AudioCOMMANDER®1 is started, it first queries the operating system to determine which sound card driver is installed. The sound card driver is then queried and AudioCOMMANDER®1 configures itself to support the detected sound card's specific features and options.

AudioCOMMANDER®1 has tabs for quick selection of conveniently organized dialogs for controlling microphone, recording, and playback settings.

If desired AudioCOMMANDER®1 will put a smart icon on the Taskbar's system tray visible at all times in the lower right hand corner of the screen for alerting the user of improper settings.

The example dialog boxes presented in this specification are representative of what you will see if you have a SoundBlaster 16 (or compatible) sound card. Other sound cards may cause dialog boxes to look different

When AudioCOMMANDER®1 is started for the first time after installation, it defaults to its simpliest configuration displaying two tabs: one for microphone related functions, and one for selecting various options.

Microphone Tab

Recording Box

Recording Group

The Recording group's Balance/Volume/Select controls are master controls that effect all sound card audio inputs equally. This group of master controls are often grayed-out and not adjustable by the user. The VU meter displays sound levels when the sound card is receiving audio input and passing it to an active application such as a sound recorder/mixer or voice recognition program.

Microphone Group

The Microphone group's Volume slider provides control of the microphone's gain. The Select box enables the microphone input. Most sound cards support only monaural microphones and for this reason the microphone Balance control is usually grayed-out.

Playback Box

Volume Control Group

The Volume Control group's Balance/Volume/Mute All controls are master controls that effect the sound card's audio output equally for all audio sources.

Wave Group

The Wave group's Balance/Volume/Mute controls effect the sound card's audio output of wave files. The VU meter displays sound levels when wave files are played.

Options Group

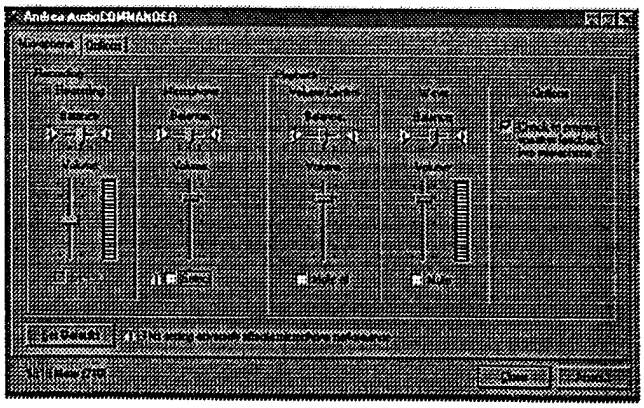
Normally this box should be checked to avoid speaker feedback into the microphone.

Yellow Alerts !

Yellow Alerts appear when one of the microphone related controls is set incorrectly for proper microphone operation. The Yellow Alert symbol appears next to the offending control(s) and to the right of the Set Defaults button.

Set Defaults Button

The Set Defaults button forces all of the microphone related controls to nominal settings which provide a good starting point for microphone enabled applications.



Options Tab

The Options tab provides user control the tabs over several AudioCOMMANDER®1 options including the placement of a smart icon on the taskbar's system tray.

Display Options Box

Show Recording Controls

The Show Recording Controls check box displays or hides the Recording tab and its associated controls.

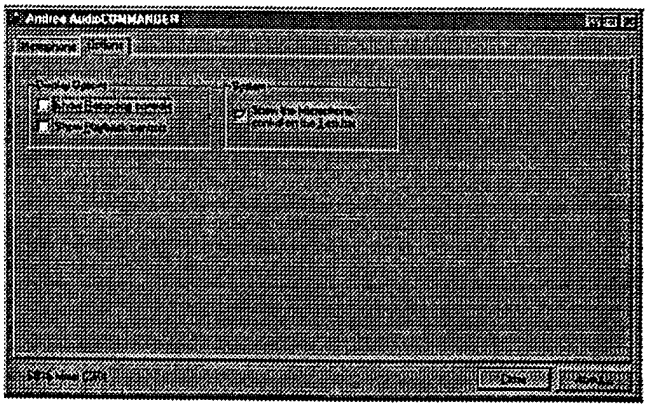
Show Playback Controls

The Show Playback Controls check box displays or hides the Playback tab and its associated controls.

System Box

Show the Microphone Control on the Taskbar

The Show the Microphone Control on the Taskbar check box adds or removes the AudioCOMMANDER®1 smart icon from the Taskbar's system tray.



Recording Tab

The Recording tab provides user control of all the sound card's recording settings whether or not they influence microphone performance.

Recording Group

The Recording group's Balance/Volume/Mute All controls are master controls that uniformly effect all sound card audio inputs equally. This group of master controls are often grayed-out and not adjustable by the user. The VU meter displays sound levels when the sound card is receiving audio input and passing it to an active application such as a sound recorder/mixer or voice recognition program. This group is identical to, and performs the same functions as, the Recording group which appears under the Microphone tab.

MIDI Group

The MIDI group's Balance/Volume/Select controls effect the sound card's MIDI stereo input left/right balance, gain, and muting functions.

CD Audio Group

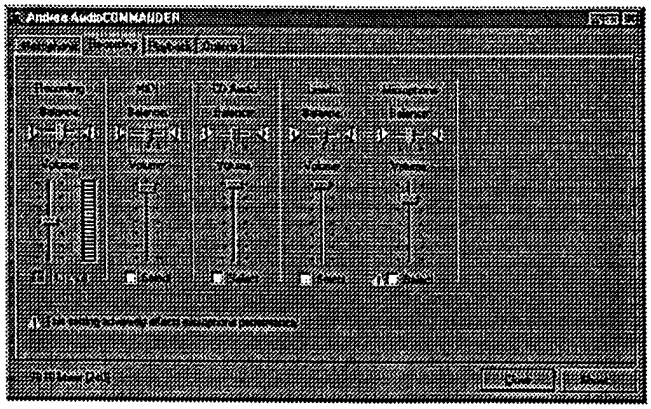
The CD Audio group's Balance/Volume/Select controls effect the sound card's CD Audio stereo input left/right balance, gain, and muting functions.

Line-In Group

The Line-In group's Balance/Volume/Select controls effect the sound card's Line-In stereo input left/right balance, gain, and muting functions.

Microphone Group

The Microphone group's Balance/Volume/Select controls effect the sound card's microphone input gain and muting functions. Since most sound cards support only monaural microphones, the microphone Balance control is usually grayed-out. This group is identical to, and performs the same functions as, the Microphone group which appears under the Microphone tab.



Playback Tab

The Playback tab provides user control of all the sound card's playback settings whether or not they influence microphone performance.

Volume Control Group

The Volume Control group's Balance/Volume/Mute controls are master controls that effect the sound card's output left/right balance, gain, and muting functions regardless of the audio source. This group is identical to, and performs the same functions as, the Volume Control group which appears under the Microphone tab.

Wave Group

The Wave group's Balance/Volume/Select controls effect the sound card's output left/right balance, gain, and muting functions of wave file audio sources. The VU meter indicates sound level when a wave file is being played back. This group is identical to, and performs the same functions as, the Wave group which appears under the Microphone tab.

MIDI Group

The MIDI group's Balance/Volume/Select controls effect the sound card's MIDI output left/right balance, gain, and muting functions. Note that this sound card's Mute box is grayed-out preventing the user from muting the MIDI output.

CD Audio Group

The CD Audio group's Balance/Volume/Select controls effect the sound card's output left/right balance, gain, and muting functions of CD Audio audio sources.

Line-In Group

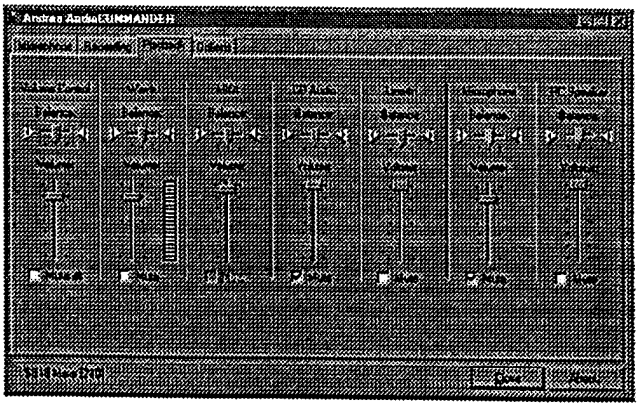
The Line-In group's Balance/Volume/Select controls effect the sound card's output left/right balance, gain, and muting functions of Line-In audio sources.

Microphone Group

The Microphone group's Balance/Volume/Select controls effect the sound card's output of Microphone audio sources. This microphone Mute box should be checked to prevent speaker feedback into the microphone.

PC Speaker Group

The PC Speaker group's Balance/Volume/Select controls effect the sound card's PC Speaker output left/right balance, gain, and muting functions.



```
  D:\Andrea\AudioCOMMANC  ,InstallBuild\Setup\setup.rul -  on Jan 13 11:32:22 1997

/*---------------------------------------------------------------------------*\
 *
 *    File Name:  SETUP.RUL
 *
 *  Description:  Andrea Electronics Corporation AudioCOMMANDER installation
 *                script.
 *
 *       Author:  Andrea Electronics Corporation              Date:  1-13-97
 *
 *     Comments:  This script performs a basic installation of AudioCOMMANDER
 *                to a Windows 95 or Windows NT platform.
 *
\*---------------------------------------------------------------------------*/

#define SIZE_REQ_HELP           0
#define SIZE_REQ_PROGRAM        300000

#define APP_NAME                "AudioCOMMANDER"
#define PROGRAM_GROUP_NAME      "Andrea"
#define APPBASE_PATH            "Andrea\\AudioCOMMANDER\\"
#define COMPANY_NAME            "Andrea"
#define PRODUCT_NAME            "AudioCOMMANDER"
#define PRODUCT_VERSION         "1.0"
#define PRODUCT_KEY             "AudioC"
#define UNINSTALL_KEY           "AudioCOMMANDER"
#define UNINSTALL_NAME          "AudioCOMMANDER 1.0"
#define APPBASE_DIR95           "Program Files\\"

declare

        // Global variable declarations.
        STRING  svGrp, szMsg, szFileSet, szTitle, szAppPath, szAppSharedDir;
        STRING  szProgram, szTemp, svUninstLogFile, szRegKey;
        STRING  svMainDirectory[ _MAX_STRING ];
        BOOL    bSpaceOk, bReshowSetupType;
        NUMBER  nResult, nStatusId, nType;

        BOOL    bIncludeProgram, bIncludeHelp;
        BOOL    bWinNT;

        // Function declarations.
        prototype SetupScreen();
        prototype CheckRequirements();
        prototype CheckSpaceRequirements( number, number, string );
        prototype CreateRegDBEntries();

program

//Beginning of program.
StartHere:
        Disable( BACKGROUND );

        // Set up the installation screen.
        SetupScreen();

        // Set installation info., which is required for registry entries.
        InstallationInfo( COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY
===> );
```

```
D:\Andrea\AudioCOMMANDE  nstallBuild\Setup\setup.rul -    : Jan 13 11:32:22 1997

// Create a Welcome dialog.
WelcomeDlg:
        Disable( BACKBUTTON );
        Welcome( "", 0 );
        Enable( BACKBUTTON );

        // Test target system proper configuration.
        CheckRequirements();

// Ask user for a destination location to install files.
GetTargetDirectory:

        svMainDirectory = TARGETDISK ^ APPBASE_DIR95 ^ APPBASE_PATH;

        szMsg = "";
        if ( AskDestPath( "", szMsg, svMainDirectory, 0 ) = BACK ) then
                goto WelcomeDlg;
        endif;

        szAppSharedDir = svMainDirectory ^ "System";

        nType = TYPICAL;

        bIncludeProgram = TRUE;
        bIncludeHelp = FALSE;

        // Check to see if target system meets space requirements.
        bSpaceOk = CheckSpaceRequirements( bIncludeProgram,
                                           bIncludeHelp,
                                           svMainDirectory );

        // Ask user to try again if not enough space available.
        if (bSpaceOk = FALSE) then goto GetTargetDirectory; endif;

// Ask user to select a folder name for the Start Menu.
FolderSelection:
        svGrp = PROGRAM_GROUP_NAME;

        // Allow user to modify folder name.
        if ( SelectFolder( "", svGrp, svGrp ) = BACK ) then
                goto GetTargetDirectory;
        endif;

FileTransferSetup:

        // Prepare InstallShield to record deinstallation information.
        DeinstallStart( svMainDirectory, svUninstLogFile, UNINSTALL_KEY, 0 );
        RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );

        // Define the "General" file set.
        szFileSet = "General";
        TARGETDIR = svMainDirectory;

        FileSetBeginDefine( szFileSet );

                SetStatusWindow( -1, "Copying program files..." );
```

```
D:\Andrea\AudioCOMMANDER_InstallBuild\Setup\setup.rul -     Jan 13 11:32:22 1997

                // Always copy README & related files, located at
                // the root level in the DATA.Z library file.
                CompressGet( "data.z", "*.*", COMP_NORMAL );

                if (bIncludeProgram) then
                        TARGETDIR = svMainDirectory ^ "Program";
                        CompressGet( "data.z", "program\\*.*", INCLUDE_SUBDIR );
                endif;

                if (bIncludeHelp) then
                        TARGETDIR = svMainDirectory ^ "HELP";
                        CompressGet( "data.z", "help\\*.*", INCLUDE_SUBDIR );
                endif;

        FileSetEndDefine( szFileSet );

DoFileTransfer:

        // Set up progress indicator and information gauge.
        Enable( STATUS );
        StatusUpdate( ON, 90 );

        // Perform the file set.
        nResult = FileSetPerformEz( szFileSet, 0 );

        switch (nResult)

        case FS_DONE: // Successful completion.

        case FS_CREATEDIR: // Create directory error.
                MessageBox( "Unable to create a directory under " + TARGETDIR + "."+
                          "Please check write access to this directory.", SEVERE );
                abort;

        default: // Group all other errors under default label.
                NumToStr( szTemp, nResult );
                MessageBox( "General file transfer error."+
                            "Please check your target location and try again."+
                            "\n\n Error Number:"+szTemp, SEVERE );

                abort;
        endswitch;

SetRegistryEntries:

        CreateRegDBEntries( );

        Delay(1);

CreateFolderIcons:

        SetStatusWindow( 95, "Creating Folder and Icons...." );

        CreateProgramFolder( svGrp );
        LongPathToShortPath( svMainDirectory );
        Delay(1);

        TARGETDIR = svMainDirectory;
```

```
D:\Andrea\AudioCOMMANDE  nstallBuild\Setup\setup.rul -    Jan 13 11:32:22 1997

        if (bIncludeProgram) then
                szProgram = TARGETDIR ^ "PROGRAM\\" ^ PRODUCT_KEY;
                AddFolderIcon( svGrp, APP_NAME, szProgram,
                               TARGETDIR ^ "PROGRAM",
                               "", 0, "", REPLACE );
        endif;

        AddFolderIcon( svGrp, "ReadmeFile",
                       "NOTEPAD.EXE " + TARGETDIR ^ "README.TXT",
                       TARGETDIR,
                       "", 0, "", REPLACE );

        szProgram = WINDIR ^ "UNINST.EXE";
        LongPathToShortPath( szProgram );
        LongPathToShortPath( svUninstLogFile );
        AddFolderIcon( svGrp, "unInstallShield",
                       szProgram + " -f" + svUninstLogFile,
                       WINDIR,
                       "", 0, "", REPLACE );

// Announce setup complete.

        SetStatusWindow( 100, "Installation complete." );
        Delay(2);
        Disable( STATUS );


        szMsg = "Setup is complete.  The next time you restart your computer, "+
                "the AudioCOMMANDER icon will automatically appear on your taskbar'
===> s "+
                "system tray. This feature may be disabled by selecting the "+
                "AudioCOMMANDER Options tab and unchecking 'Show the Microphone "+
                "control on the Taskbar'.\n\nPlease read the readme file for last "+
                "minute updates.";
        MessageBeep( 0 );
        MessageBox( szMsg, INFORMATION );

        exit;


/*---------------------------------------------------------------------------*\
 *
 * Function:  SetupScreen
 *
 *  Purpose:  This function will set up the screen look.  This includes
 *            colors, fonts, text to be displayed, etc.
 *
 *
 *    Input:
 *
 *  Returns:
 *
 * Comments:
\*---------------------------------------------------------------------------*/

function SetupScreen()
```

```
 D:\Andrea\AudioCOMMANDE   nstallBuild\Setup\setup.rul -    ı Jan 13 11:32:22 1997
       begin

       Enable( INDVFILESTATUS );

       SetTitle( COMPANY_NAME + " " + APP_NAME + " Setup", 28, WHITE );

       SetTitle( "Setup", 0, BACKGROUNDCAPTION ); // Caption bar text.

       Enable( BACKGROUND );

       end;

/*---------------------------------------------------------------------------*\
 *
 * Function:  CheckRequirements
 *
 *  Purpose:  This function will check all minimum requirements for the
 *            application being installed.  If any fail, then the user
 *            is informed and the installation is terminated.
 *
 *
 *    Input:
 *
 *  Returns:
 *
 * Comments:
\*---------------------------------------------------------------------------*/

function CheckRequirements()

       number  nvDx, nvDy;
       number nvResult;
       STRING szResult;

       begin

       // Determine if target system uses NT or Windows 95.
       GetSystemInfo( WINMAJOR, nvResult, szResult );
       bWinNT = TRUE;
       if (nvResult = 4) then
              bWinNT = FALSE; // Running Windows 95.
       endif;

       // Check screen resolution.
       GetExtents( nvDx, nvDy );
       if (nvDy < 480) then
              MessageBox( "This program requires VGA or better resolution.", WARN
===> ING );
              exit;
       endif;

       end;

/*---------------------------------------------------------------------------*\
 *
 * Function:  CheckSpaceRequirements
 *
```

```
 D:\Andrea\AudioCOMMANDE   nstallBuild\Setup\setup.rul -    1 Jan 13 11:32:22 1997

 *  Purpose:  This function will check space requirements based on the
 *            elements being installed.
 *
 *    Input:
 *
 *  Returns:
 *
 * Comments:
\*-------------------------------------------------------------------------*/

function CheckSpaceRequirements( bIncludeProgram,
                                 bIncludeHelp,
                                 szDir )

        number  nSizeRequired;

        begin

        nSizeRequired = 0;

        // Determine total size.
        if (bIncludeHelp) then
                nSizeRequired = nSizeRequired + SIZE_REQ_HELP;
        endif;

        if (bIncludeProgram) then
                nSizeRequired = nSizeRequired + SIZE_REQ_PROGRAM;
        endif;

        // Check space on target drive.
        bSpaceOk = TRUE;
        if (GetDiskSpace( szDir ) < nSizeRequired) then
                szMsg = "There is not enough space available on the disk\n" +
                        "'" + svMainDirectory + "' \n" +
                        "Please free up some space or change the target location\n"
===>  +
                        "to a different disk";
                MessageBeep(0);
                MessageBox( szMsg, WARNING );
                bSpaceOk = FALSE;
         endif;
         return bSpaceOk;

         end;


/*-------------------------------------------------------------------------*\
 *
 * Function:  CreateRegDBEntries
 *
 *  Purpose:  This function will create necessary keys and values.
  *
 *    Input:
 *
 *  Returns:
 *
 * Comments:
\*-------------------------------------------------------------------------*/
```

```
D:\Andrea\AudioCOMMANDE   nstallBuild\Setup\setup.rul -    : Jan 13 11:32:22 1997

function CreateRegDBEntries()

        string szKey[255], szValue[255];

        begin

        // Create PRODUCT_KEY key.

        RegDBSetDefaultRoot( HKEY_LOCAL_MACHINE );

        szKey = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";
        RegDBCreateKeyEx( szKey, "" );
        szValue = svMainDirectory + '"Program\\" + PRODUCT_KEY + " /tray";
        RegDBSetKeyValueEx( szKey, PRODUCT_NAME, REGDB_STRING, szValue, -1 );

        // Create PRODUCT_KEY key.

        RegDBSetDefaultRoot( HKEY_CURRENT_USER );

        szKey = "SOFTWARE\\" + COMPANY_NAME + "\\" + PRODUCT_KEY + "\\Settings";

        RegDBCreateKeyEx( szKey, "" );
        RegDBSetKeyValueEx( szKey, "ShowPlaybackControls",  REGDB_NUMBER, "0", -1 );
        RegDBSetKeyValueEx( szKey, "ShowRecordingControls",  REGDB_NUMBER, "0", -1
===> );

        end;
```

```
/* SOURCE_FILE ************************************************************
 *
 * File:    micctrl
 *
 * Title:
 *
 * Owner:   Leonard
 *
 *  Copyright (C) 1996 All Rights Reserved
 * ------------------------------------------------------------------------
 * Revision history (Most recent change first):
 *
 *  Leonard 21Oct96     Creation
 *
 *END*********************************************************************/

#include "stdafx.h"
#include "MicCtrl.h"
#include "caudio.h"

// #define VOLUME_MAX           128
// #define  VOLUME_TIC_FREQ     16
// #define  VOLUME_PAGE_SIZE    8
// #define  VOLUME_ADJUST       512
#define VOLUME_MAX          256
#define VOLUME_TIC_FREQ     43
#define VOLUME_PAGE_SIZE    43
#define VOLUME_ADJUST       256

#define BALANCE_MAX         64
#define BALANCE_TIC_FREQ    32
#define BALANCE_PAGE_SIZE   4

/* FUNCTION ***************************************************************
 *
 * Name:        CAudio - Constructor
 *
 *************************************************************************/
CAudio::CAudio()
{
    /*
     * Initialize all member variables
     */
    m_MeterRect.top = m_MeterRect.bottom = m_MeterRect.left = m_MeterRect.right = 0;
    m_iType             = 0;
    m_dwSource          = 0;
    m_dwLineID          = 0;
    m_dwComponentType   = 0;
    m_cChannels         = 0;
    m_dwVolControlID    = 0xffffffff;
    m_dwMuteControlID   = 0xffffffff;
    m_dwAGCControlID    = 0xffffffff;
    m_dwMeterControlID  = 0xffffffff;
    m_fdwControl        = 0;
    m_cMultipleItems    = 0;
    m_dwMstMuteControlID= 0xffffffff;
    m_fdwMstControl     = 0;
    m_cMstMultipleItems = 0;
    m_iMuteIndex        = 0;
```

```
    m_pBalance          = 0;
    m_pVolume           = 0;
    m_pMute             = 0;
    m_pTitle            = 0;
    m_pBalText          = 0;
    m_pVolText          = 0;
    m_pIcon1            = 0;
    m_pIcon2            = 0;
    m_pLine             = 0;
    m_pMuteWarn         = 0;
    m_pVolWarn          = 0;
    m_pAGC              = 0;
    m_lBalanceID        = 0;
    m_lVolumeID         = 0;
    m_lMuteID           = 0;
    m_lAGCID            = 0;

    m_fMuteMultiplexer  = FALSE;
    m_fInvertMute       = FALSE;
    m_fCheckForInvertMute = TRUE;

    m_pMasterControl    = 0;
    m_lLastMeter        = 0xffffffff;

//  brushBackground.CreateSolidBrush(GetSysColor(COLOR_BTNFACE));
//
//  WORD        Pattern[] = {   0xDD,
//                              0x77,
//                              0xDD,
//                              0x77,
//                              0xDD,
//                              0x77,
//                              0xDD,
//                              0x77
//                              };
//  bmBrush.CreateBitmap(8,8,1,1,Pattern);
//  brushDither.CreatePatternBrush(&bmBrush);
}

/* FUNCTION ***************************************************************
 *
 * Name:    CAudio - Destructor
 *
 ***************************************************************************/
CAudio::~CAudio()
{
}

/* FUNCTION ***************************************************************
 *
 * Name:    InitControls
 * Notes:
 *
 ***************************************************************************/
void CAudio::InitControls()
{
    /*
     * Setup the balance control, leave disabled if the device is not stereo
     */
    if (m_pBalance)
```

```
{
    m_pBalance->ShowWindow(SW_SHOW);
    m_pBalance->EnableWindow(TRUE);
    m_pBalance->SetRangeMax(BALANCE_MAX);
    m_pBalance->SetTicFreq(BALANCE_TIC_FREQ);
    m_pBalance->SetPageSize(BALANCE_PAGE_SIZE);
    m_pBalance->SetPos(BALANCE_MAX / 2);
    if (m_cChannels < 2)
    {
        m_pBalance->EnableWindow(FALSE);
    }
}

MIXERLINECONTROLS   mxLc;
MIXERCONTROL        mxCt;

/*
 * Setup the volume control
 */
if (m_pVolume)
{
    m_pVolume->ShowWindow(SW_SHOW);
    m_pVolume->EnableWindow(TRUE);
    m_pVolume->SetRangeMax(VOLUME_MAX);
    m_pVolume->SetTicFreq(VOLUME_TIC_FREQ);
    m_pVolume->SetPageSize(VOLUME_PAGE_SIZE);
    m_pVolume->SetPos(VOLUME_MAX / 2);
}

/*
 * Get the info necessary to get/set the volume information
 */
mxLc.cbStruct = sizeof(MIXERLINECONTROLS);
mxLc.dwLineID = m_dwLineID;
mxLc.dwControlType = MIXERCONTROL_CONTROLTYPE_VOLUME;
mxLc.cControls = 1;
mxLc.cbmxctrl = sizeof(MIXERCONTROL);
mxLc.pamxctrl = &mxCt;
if (mixerGetLineControls((HMIXEROBJ) FIRST_CARD, &mxLc
        , MIXER_GETLINECONTROLSF_ONEBYTYPE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
{
    if (mxCt.fdwControl & MIXERCONTROL_CONTROLF_DISABLED)
    {
        if (m_pBalance)
            m_pBalance->EnableWindow(FALSE);
        if (m_pVolume)
            m_pVolume->EnableWindow(FALSE);
    }
    else
    {
        m_dwVolControlID = mxCt.dwControlID;
        /*
         * Get the current volume and adjust the volume/balance sliders
         */
        GetVolume();
    }
}
else
{
    if (m_pBalance)
```

```
            m_pBalance->EnableWindow(FALSE);
        if (m_pVolume)
            m_pVolume->EnableWindow(FALSE);
    }

    /*
     * Show/enable all of the available controls
     */
    if (m_pTitle)
    {
        m_pTitle->ShowWindow(SW_SHOW);
        m_pTitle->EnableWindow(TRUE);
        m_pTitle->SetWindowText(m_sName);
    }
    if (m_pBalText)
    {
        m_pBalText->ShowWindow(SW_SHOW);
        m_pBalText->EnableWindow(TRUE);
    }
    if (m_pVolText)
    {
        m_pVolText->ShowWindow(SW_SHOW);
        m_pVolText->EnableWindow(TRUE);
    }
    if (m_pIcon1)
    {
        m_pIcon1->ShowWindow(SW_SHOW);
        m_pIcon1->EnableWindow(TRUE);
    }
    if (m_pIcon2)
    {
        m_pIcon2->ShowWindow(SW_SHOW);
        m_pIcon2->EnableWindow(TRUE);
    }
    if (m_pLine)
    {
        m_pLine->ShowWindow(SW_SHOW);
        m_pLine->EnableWindow(TRUE);
    }
    /*
     * Setup the Mute/Select controls
     */
    InitializeMute();
    /*
     * Setup the Auto Gain Control
     */
    InitializeAGC();
    /*
     * Setup the VU Meter
     */
    InitializeMeter();
}

/* FUNCTION ***************************************************************
 *
 * Name:    GetVolume
 * Notes:   Get the current volume information
 *
 **************************************************************************/
DWORD CAudio::GetVolume()
```

```
{
    DWORD   dwVolume = 0x8000;

    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_UNSIGNED    mxVol[2];

    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwVolControlID;
    mxDet.cChannels = m_cChannels;
    mxDet.cMultipleItems = 0;
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_UNSIGNED);
    mxDet.paDetails = &mxVol;
    if (mixerGetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_GETCONTROLDETAILSF_VA
LUE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
    {
        /*
         * if there is a volume control, set the volume/balance sliders
         */
        if (m_pVolume)
        {
            SetVolumePos(mxVol[0].dwValue, mxVol[1].dwValue);
        }
        /*
         * if mono, return the only volume.  If stero, return the highest
         * of the left and right channels
         */
        if (m_cChannels == 1)
            dwVolume = mxVol[0].dwValue;
        else
            dwVolume = max(mxVol[0].dwValue, mxVol[1].dwValue);
    }
    return(dwVolume);
}

.
/* FUNCTION ************************************************************
 *
 * Name:    SetVolumePos
 * Notes:   Set the position of the volume & balance controls
 *
 ***********************************************************************/
void CAudio::SetVolumePos(DWORD dwLeft, DWORD dwRight)
{
    WORD    wLeft = LOWORD(dwLeft);
    WORD    wRight = LOWORD(dwRight);
    WORD    wPos;

    /*
     * Set the balance position
     *
     * It is calculated as a percentage of the difference between the volume
     * of the left and right channels.
     */
    if (m_pBalance && m_cChannels > 1 && (wLeft || wRight))
    {
        if (wLeft == wRight)
        {
            wPos = BALANCE_MAX / 2;
        }
        else if (wLeft > wRight)
```

```
        {
            wPos = (WORD)FloatToWord(((((FLOAT)wRight)
                                / (FLOAT)wLeft)
                                * (FLOAT)(BALANCE_MAX / 2)));
        }
        else
        {
            wPos = (BALANCE_MAX) - (WORD)FloatToWord(((((FLOAT)wLeft)
                                / (FLOAT)wRight)
                                * (FLOAT)(BALANCE_MAX / 2)));
        }
        m_pBalance->SetPos(wPos);
    }

    /*
     * Set the volume position
     *
     * It is the maximum of the left and right channels
     */
    if (m_pVolume)
    {
        WORD    wVolume = (m_cChannels > 1 ? max(wLeft, wRight) : wLeft);
        m_pVolume->SetPos(VOLUME_MAX-(wVolume / VOLUME_ADJUST));
    }
}

.
/* FUNCTION ****************************************************************
 *
 * Name:    SetNewVolume
 * Notes:   Get the new volume from the dialog controls and send it to the driver
 *
 ***************************************************************************/
void CAudio::SetNewVolume()
{
    DWORD   dwVolume;

    if (!m_pVolume)
        return;

    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_UNSIGNED    mxVol[2];

    dwVolume = ((VOLUME_MAX - m_pVolume->GetPos()) * VOLUME_ADJUST);
    if (dwVolume > 0xffff)
        dwVolume = 0xffff;

    if (m_cChannels > 1)
    {
        /*
         * If stereo, calculate the left & right volumes based on the balance
         */
        CalcVolume(dwVolume, &mxVol[0].dwValue, &mxVol[1].dwValue);
    }
    else
    {
        mxVol[0].dwValue = dwVolume;
        mxVol[1].dwValue = dwVolume;
    }
```

```
    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwVolControlID;
    mxDet.cChannels = m_cChannels;
    mxDet.cMultipleItems = 0;
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_UNSIGNED);
    mxDet.paDetails = &mxVol;
    mixerSetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_SETCONTROLDETAILSF_VALUE
| MIXER_OBJECTF_MIXER);
}
.
/* FUNCTION *****************************************************************
 *
 * Name:    CalcVolume
 * Notes:   Calculate the left & right volume based on the balance
 *
 ****************************************************************************/
void CAudio::CalcVolume(DWORD dwVolume, DWORD* pdwLeft, DWORD* pdwRight)
{
    if (!m_pBalance)
    {
        *pdwLeft = *pdwRight = 0;
        return;
    }

    DWORD   dwBalance = m_pBalance->GetPos();

    /*
     * If the balance is in the middle the left & right volumes are the same.
     * Otherwise, one channel gets the indicated volume and the other gets a
     * percentage of the indicated volume based on the position of the
     * balance control.
     */
    if (dwBalance == BALANCE_MAX / 2)
    {
        *pdwLeft = *pdwRight = dwVolume;
    }
    else if (dwBalance < BALANCE_MAX / 2)
    {
        *pdwLeft = dwVolume;
        *pdwRight = (WORD)FloatToWord(((FLOAT)dwBalance
                    / (FLOAT)(BALANCE_MAX / 2)
                    * (FLOAT) dwVolume));
    }
    else
    {
        *pdwRight = dwVolume;
        *pdwLeft = (WORD)FloatToWord(((FLOAT)(BALANCE_MAX - dwBalance)
                    / (FLOAT)(BALANCE_MAX / 2)
                    * (FLOAT) dwVolume));
    }

    return;
}
.
/* FUNCTION *****************************************************************
 *
 * Name:    CheckDefaultVolume
 * Return:  TRUE - if the current volume is in the valid range
 *          FALSE - if the current volume is not in the valid range
 * Notes:   Valid is determined to be above 20%
```

```
 *
 ****************************************************************************/
BOOL CAudio::CheckDefaultVolume()
{
    BOOL    fReturn = TRUE;

    /*
     * If there is a dialog control, get the volume from it.
     * Otherwise, get the volume from the driver
     */
    if (!m_pVolume)
    {
        if (GetVolume() < 0x3333)
            fReturn = FALSE;
    }
    else
    {
        if (((VOLUME_MAX - m_pVolume->GetPos()) * VOLUME_ADJUST) < 0x3333)
            fReturn = FALSE;
    }

    /*
     * if there is a warning control on the dialog, show it
     */
    if (m_pVolWarn)
    {
        m_pVolWarn->ShowWindow(fReturn ? SW_HIDE : SW_SHOW);
    }

    return(fReturn);
}
.
/* FUNCTION ****************************************************************
 *
 * Name:    SetDefaultVolume
 * Notes:   Set the default volume, which is approximately 80%
 *
 ****************************************************************************/
void CAudio::SetDefaultVolume()
{
    if (!m_pVolume)
        return;

    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_UNSIGNED    mxVol[2];

    mxVol[0].dwValue = 0xd000;
    mxVol[1].dwValue = 0xd000;

    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwVolControlID;
    mxDet.cChannels = m_cChannels;
    mxDet.cMultipleItems = 0;
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_UNSIGNED);
    mxDet.paDetails = &mxVol;
    mixerSetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_SETCONTROLDETAILSF_VALUE
  MIXER_OBJECTF_MIXER);
}
.
/* FUNCTION ****************************************************************
```

```
 *
 * Name:     InitializeMute
 * Notes:    Initialize the Mute/Select control
 *
 ********************************************************************************/
void CAudio::InitializeMute()
{
    BOOL                fShow = FALSE;
    BOOL                fMultiple = FALSE;
    CString             sName;
    int                 i;
    MIXERLINECONTROLS   mxLc;
    MIXERCONTROL        mxCt;

    mxLc.cbStruct = sizeof(MIXERLINECONTROLS);
    mxLc.dwLineID = m_dwLineID;
    mxLc.cControls = 1;
    mxLc.cbmxctrl = sizeof(MIXERCONTROL);
    mxLc.pamxctrl = &mxCt;

    /*
     * Is this a multi-select item for all of the source lines?
     */
    mxLc.dwControlType = MIXERCONTROL_CONTROLTYPE_MIXER;
    if (mixerGetLineControls((HMIXEROBJ) FIRST_CARD, &mxLc
            , MIXER_GETLINECONTROLSF_ONEBYTYPE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
    {
        fMultiple = TRUE;
    }
    else
    {
        /*
         * Is this a Mutex type of multiple control?
         */
        mxLc.dwControlType = MIXERCONTROL_CONTROLTYPE_MUX;
        if (mixerGetLineControls((HMIXEROBJ) FIRST_CARD, &mxLc
                , MIXER_GETLINECONTROLSF_ONEBYTYPE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOER
ROR)
        {
            fMultiple = TRUE;
            m_fMuteMultiplexer = TRUE;
        }
    }
    if (fMultiple)
    {
        /*
         * This is a multiple item type of control, save the info in the master control
         */
        m_dwMstMuteControlID = mxCt.dwControlID;
        m_fdwMstControl = mxCt.fdwControl;
        m_cMstMultipleItems = mxCt.cMultipleItems;

        MIXERCONTROLDETAILS             mxDet;
        MIXERCONTROLDETAILS_LISTTEXT    mxList[10];

        /*
         * Get & save the IDs of the controls this multiple item controls
         */
        mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
        mxDet.dwControlID = m_dwMstMuteControlID;
```

```
        mxDet.cChannels = 1;
        mxDet.cMultipleItems = m_cMstMultipleItems;
        mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_LISTTEXT);
        mxDet.paDetails = mxList;
        if (mixerGetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_GETCONTROLDETAILS
F_LISTTEXT | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
        {
            for (i = 0; i < (int)m_cMstMultipleItems; i++)
            {
                m_dwMuxID[i] = mxList[i].dwParam1;
            }
        }
    }
    /*
     * Does the master control control multiple items?
     */
    else if (m_pMasterControl
                && (m_pMasterControl->m_fdwMstControl & MIXERCONTROL_CONTROLF_MULTIPLE))
    {
        /*
         * Yes, check the control IDs to see if the master controls this
         * control
         */
        for (i = 0; i < (int)m_pMasterControl->m_cMstMultipleItems; i++)
        {
            if (m_pMasterControl->m_dwMuxID[i] == m_dwLineID)
            {
                /*
                 * The master does control this control
                 */
                m_iMuteIndex = i;
                m_dwMuteControlID = m_pMasterControl->m_dwMstMuteControlID;
                m_fdwControl = m_pMasterControl->m_fdwMstControl;
                m_cMultipleItems = m_pMasterControl->m_cMstMultipleItems;
                m_fMuteMultiplexer = m_pMasterControl->m_fMuteMultiplexer;
                fShow = TRUE;
                sName.LoadString(IDS_SELECT);
                if (!m_pVolume && m_fCheckForInvertMute)
                    m_fInvertMute = TRUE;
                break;
            }
        }
    }

    /*
     * If not part of a multiple control, check for a single control
     */
    if (!fShow)
    {
        /*
         * Is there a mute control?
         */
        mxLc.dwControlType = MIXERCONTROL_CONTROLTYPE_MUTE;
        if (mixerGetLineControls((HMIXEROBJ) FIRST_CARD, &mxLc
                , MIXER_GETLINECONTROLSF_ONEBYTYPE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOER
ROR)
        {
            m_dwMuteControlID = mxCt.dwControlID;
            m_fdwControl = mxCt.fdwControl;
            m_cMultipleItems = mxCt.cMultipleItems;
```

```
            fShow = TRUE;
            sName.LoadString(m_pMasterControl ? IDS_MUTE : IDS_MUTE_ALL);
        }
        else
        {
            /*
             * Is there a single select control?
             */
            mxLc.dwControlType = MIXERCONTROL_CONTROLTYPE_SINGLESELECT;
            if (mixerGetLineControls((HMIXEROBJ) FIRST_CARD, &mxLc
                    , MIXER_GETLINECONTROLSF_ONEBYTYPE | MIXER_OBJECTF_MIXER) == MMSYSERR_
NOERROR)
            {
                m_dwMuteControlID = mxCt.dwControlID;
                m_fdwControl = mxCt.fdwControl;
                m_cMultipleItems = mxCt.cMultipleItems;
                fShow = TRUE;
                sName.LoadString(IDS_SELECT);
            }
        }

    }

    if (m_pMute)
        m_pMute->ShowWindow(SW_SHOW);

    /*
     * If we found a mute/select control, show the dialog control
     */
    if (fShow && m_pMute)
    {
        if (!(m_fdwControl & MIXERCONTROL_CONTROLF_DISABLED))
        {
            m_pMute->EnableWindow(TRUE);
            if (m_pVolume)
            {
                m_pMute->SetWindowText(sName);
            }
            GetMute();
        }
    }
}
.
/* FUNCTION *****************************************************************
 *
 * Name:    GetMute
 * Notes:   Get the mute value from the driver
 *
 ****************************************************************************/
BOOL CAudio::GetMute()
{
    BOOL    fValue = 0;
    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_BOOLEAN     mxMute[10];

    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwMuteControlID;
    mxDet.cChannels = 1;
    mxDet.cMultipleItems = (m_fdwControl & MIXERCONTROL_CONTROLF_MULTIPLE
                            ? m_cMultipleItems : 0);
```

```
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_BOOLEAN);
    mxDet.paDetails = mxMute;
    if (mixerGetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_GETCONTROLDETAILSF_VA
LUE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
    {
        fValue = mxMute[m_iMuteIndex].fValue;
        SetMute(fValue);
    }
    return fValue;
}

/* FUNCTION ************************************************************
 *
 * Name:    SetMute
 * Notes:   Set the mute dialog control
 *
 ************************************************************************/
void CAudio::SetMute(BOOL fMute)
{
    if (m_pMute)
    {
        if (m_fInvertMute)
            m_pMute->SetCheck(!fMute);
        else
            m_pMute->SetCheck(fMute);
    }
}

/* FUNCTION ************************************************************
 *
 * Name:    SetNewMute
 * Notes:   Get the new value from the dialog control and set the driver
 *
 ************************************************************************/
void CAudio::SetNewMute()
{
    if (!m_pMute)
        return;

    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_BOOLEAN     mxMute[10];


    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwMuteControlID;
    mxDet.cChannels = 1;
    mxDet.cMultipleItems = m_cMultipleItems;
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_BOOLEAN);
    mxDet.paDetails = mxMute;
    if (m_fMuteMultiplexer)
    {
        /*
         * If a Multipler control, only the one value can be sent to the driver
         */
        for (int i = 0; i < (int)m_cMultipleItems; i++)
        {
            mxMute[i].fValue = 0;
        }
    }
    else
```

```
    {
        /*
         * Get all of the current values
         */
        mixerGetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_GETCONTROLDETAILSF_VA
LUE | MIXER_OBJECTF_MIXER);
    }
    /*
     * replace the current value and send to the driver
     */
    if (m_fInvertMute)
        mxMute[m_iMuteIndex].fValue = !(m_pMute->GetCheck());
    else
        mxMute[m_iMuteIndex].fValue = m_pMute->GetCheck();
    mixerSetControlDetails((HMIXEROBJ)FIRST_CARD, &mxDet, MIXER_SETCONTROLDETAILSF_VALUE
| MIXER_OBJECTF_MIXER);
}

/* FUNCTION *************************************************************************
 *
 * Name:     CheckDefaultMute
 * Return:   TRUE - if the current value matches
 *           FALSE - if the current value does not match
 * Notes:    Check to see if the current mute value matches fMuteIn
 *
 ************************************************************************************/
BOOL CAudio::CheckDefaultMute(BOOL fMuteIn)
{
    BOOL    fReturn = TRUE;
    BOOL    fMute;

    /*
     * If there is a dialog control, get the value from it.
     * Otherwise, get the value from the driver.
     */
    if (!m_pMute)
    {
        fMute = GetMute();
    }
    else
    {
        fMute = m_pMute->GetCheck();
    }
    if (m_fInvertMute)
        fMute = !fMute;

    /*
     * If a multiple control then it is a Select instead of Mute, so the
     * value is opposite of mute.
     */
    if (m_fdwControl & MIXERCONTROL_CONTROLF_MULTIPLE)
    {
        if (fMuteIn == fMute)
            fReturn = FALSE;
    }
    else
    {
        if (fMuteIn != fMute)
            fReturn = FALSE;
    }
```

```
    /*
     * If there is a warning control on the dialog, hide/show it.

     */
    if (m_pMuteWarn)
    {
        m_pMuteWarn->ShowWindow(fReturn ? SW_HIDE : SW_SHOW);
    }

    return(fReturn);
}
.
/* FUNCTION ********************************************************************
 *
 * Name:    SetDefaultMute
 * Notes:   Set the supplied default
 *
 *******************************************************************************/
void CAudio::SetDefaultMute(BOOL fMuteIn)
{
    if (!m_pMute)
        return;

    BOOL    fMute;
    if (m_fdwControl & MIXERCONTROL_CONTROLF_MULTIPLE)
    {
        if (fMuteIn)
            fMute = FALSE;
        else
            fMute = TRUE;
    }
    else
    {
        if (fMuteIn)
            fMute = TRUE;
        else
            fMute = FALSE;
    }
    if (m_fInvertMute)
        m_pMute->SetCheck(!fMute);
    else
        m_pMute->SetCheck(fMute);
    SetNewMute();
}
.
/* FUNCTION ********************************************************************
 *
 * Name:    InitializeMeter
 * Notes:   Initialize the VU Meter
 *
 *******************************************************************************/
void CAudio::InitializeMeter()
{
    if (!m_pVolume)
        return;

    MIXERLINECONTROLS   mxLc;
    MIXERCONTROL        mxCt;

    mxLc.cbStruct = sizeof(MIXERLINECONTROLS);
```

```
    mxLc.dwLineID = m_dwLineID;
    mxLc.cControls = 1;
    mxLc.cbmxctrl = sizeof(MIXERCONTROL);
    mxLc.pamxctrl = &mxCt;
    mxLc.dwControlType = MIXERCONTROL_CONTROLTYPE_PEAKMETER;
    if (mixerGetLineControls((HMIXEROBJ) FIRST_CARD, &mxLc
            , MIXER_GETLINECONTROLSF_ONEBYTYPE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
    {
        if (!(mxCt.fdwControl & MIXERCONTROL_CONTROLF_DISABLED))
        {
            /*
             * There is a Meter control. Save the info and move the volume
             * controls to make room for it.
             */
            m_dwMeterControlID = mxCt.dwControlID;

            WINDOWPLACEMENT wp;

            if (m_pVolume)
            {
                brushDkGreen.CreateSolidBrush(RGB(0,0x80,0));
                brushLtGreen.CreateSolidBrush(RGB(0,0xff,0));
                brushYellow.CreateSolidBrush(RGB(0xff,0xff,0));
                brushRed.CreateSolidBrush(RGB(0xff,0,0));
                brushGray.CreateSolidBrush(RGB(0xc0, 0xc0, 0xc0));

                penDkGreen.CreatePen(PS_SOLID, 1, RGB(0,0x80,0));
                penLtGreen.CreatePen(PS_SOLID, 1, RGB(0,0xff,0));
                penYellow.CreatePen(PS_SOLID, 1, RGB(0xff,0xff,0));
                penRed.CreatePen(PS_SOLID, 1, RGB(0xff,0,0));
                penBlack.CreatePen(PS_SOLID, 1, RGB(0, 0, 0));

                wp.length = sizeof(WINDOWPLACEMENT);
                m_pVolume->GetWindowPlacement(&wp);

                m_MeterRect.top = wp.rcNormalPosition.top + 7;
                m_MeterRect.bottom = m_MeterRect.top + 85;
                m_MeterRect.left = wp.rcNormalPosition.left + 40;
                m_MeterRect.right = m_MeterRect.left + 16;

                wp.rcNormalPosition.left -= 10;
                wp.rcNormalPosition.right -= 10;
                m_pVolume->MoveWindow(&wp.rcNormalPosition, FALSE);
            }
            if (m_pVolWarn)
            {
                wp.length = sizeof(WINDOWPLACEMENT);
                m_pVolWarn->GetWindowPlacement(&wp);
                wp.rcNormalPosition.left -= 10;
                wp.rcNormalPosition.right -= 10;
                m_pVolWarn->MoveWindow(&wp.rcNormalPosition, FALSE);
            }
        }
    }
}

/* FUNCTION ****************************************************************
 *
 * Name:    IsLineActive
 * Notes:   Determine if the meter control is currently active
```

```
 *
 ****************************************************************************/
BOOL CAudio::IsLineActive()
{
    MIXERLINE   mxLine;

    mxLine.cbStruct = sizeof(MIXERLINE);
    mxLine.dwDestination = m_iType;
    if (m_dwSource == 0xffffffff)
    {
        mxLine.dwSource = 0;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_DESTINATIO
N | MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            return(FALSE);
        }
    }
    else
    {
        mxLine.dwSource = m_dwSource;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE | M
IXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            return(FALSE);
        }
    }
    return ((mxLine.fdwLine & MIXERLINE_LINEF_ACTIVE) ? TRUE : FALSE);
}
.
/* FUNCTION ****************************************************************
 *
 * Name:    UpdateMeter
 * Notes:   The meter has changed, draw the new values
 *
 ****************************************************************************/
void CAudio::UpdateMeter(CDC *pDC, BOOL fForceUpdate)
{
    ASSERT(pDC);

    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_SIGNED      mxValue;

    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwMeterControlID;
    mxDet.cChannels = 1;
    mxDet.cMultipleItems = 0;
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_SIGNED);
    mxDet.paDetails = &mxValue;
    if ((!mixerGetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_GETCONTROLDETAILSF_
VALUE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR))
        return;

    if (mxValue.lValue < 0)
        return;

    if (!fForceUpdate && mxValue.lValue == m_lLastMeter)
        return;

    m_lLastMeter = mxValue.lValue;
    RECT    rect = m_MeterRect;
```

```
    pDC->DrawEdge(&rect, BDR_SUNKENOUTER | BDR_SUNKENINNER, BF_RECT);

    rect.bottom -= 2;
    rect.left += 2;
    rect.right -= 2;
    rect.top += 2;
    LONG    lLED = (rect.bottom - rect.top) / 20;

    /*
     * fudge the value by 8% to match MS meter output
     */
    mxValue.lValue = (long)((float)mxValue.lValue * 1.08);

    int i;
    /*
     * Draw the lighted rectangles
     */
    for (i = 0; i < 20 && mxValue.lValue > 0; i++, mxValue.lValue -= 1638)
    {
        rect.top = rect.bottom - (lLED + 1);
        switch (i)
        {
            case 0:
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
                pDC->SelectObject(&brushDkGreen);
                break;
            case 8:
            case 9:
            case 10:
            case 11:
                pDC->SelectObject(&brushLtGreen);
                break;
            case 12:
            case 13:
            case 14:
            case 15:
                pDC->SelectObject(&brushYellow);
                break;
            default:
                pDC->SelectObject(&brushRed);
        }
        pDC->Rectangle(&rect);
        rect.bottom = rect.top + 1;
    }
//  pDC->SetBkColor(GetSysColor(COLOR_BTNFACE));
    pDC->SelectObject(brushGray);
    /*
     * Draw the dimmed rectangles
     */
    for (; i < 20; i++)
    {
        rect.top = rect.bottom - (lLED + 1);
        pDC->SelectObject(&penBlack);
```

```
        pDC->Rectangle(&rect);
        switch (i)
        {
            case 0:
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
                pDC->SelectObject(&penDkGreen);
                break;
            case 8:
            case 9:
            case 10:
            case 11:
                pDC->SelectObject(&penLtGreen);
                break;
            case 12:
            case 13:
            case 14:
            case 15:
                pDC->SelectObject(&penYellow);
                break;
            default:
                pDC->SelectObject(&penRed);
        }
        /*
         * Draw a few color pixels, (drawing with a pattern brush doesn't work
         * with the Diamond video driver
         */
        pDC->MoveTo(rect.left + 4, rect.top + 1);
        pDC->LineTo(rect.left + 5, rect.top + 1);
        pDC->MoveTo(rect.left + 8, rect.top + 1);
        pDC->LineTo(rect.left + 9, rect.top + 1);
        pDC->MoveTo(rect.left + 2, rect.top + 2);
        pDC->LineTo(rect.left + 3, rect.top + 2);
        pDC->MoveTo(rect.left + 6, rect.top + 2);
        pDC->LineTo(rect.left + 7, rect.top + 2);
        pDC->MoveTo(rect.left + 10, rect.top + 2);
        pDC->LineTo(rect.left + 11, rect.top + 2);
        pDC->MoveTo(rect.left + 4, rect.top + 3);
        pDC->LineTo(rect.left + 5, rect.top + 3);
        pDC->MoveTo(rect.left + 8, rect.top + 3);
        pDC->LineTo(rect.left + 9, rect.top + 3);

        rect.bottom = rect.top + 1;
    }
}

/* FUNCTION ************************************************************
 *
 * Name:     InitializeAGC
 * Notes:    Initialize the Auto Gain Control
 *
 ***********************************************************************/
oid CAudio::InitializeAGC()
{
    if (!m_pAGC)
```

```
        return;

    MIXERLINECONTROLS   mxLc;
    MIXERCONTROL        mxCt;

    mxLc.cbStruct = sizeof(MIXERLINECONTROLS);
    mxLc.dwLineID = m_dwLineID;
    mxLc.cControls = 1;
    mxLc.cbmxctrl = sizeof(MIXERCONTROL);
    mxLc.pamxctrl = &mxCt;
    mxLc.dwControlType = MIXERCONTROL_CONTROLTYPE_ONOFF;
    if (mixerGetLineControls((HMIXEROBJ) FIRST_CARD, &mxLc
            , MIXER_GETLINECONTROLSF_ONEBYTYPE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
    {
        if (strstr(mxCt.szName, "AGC") || strstr(mxCt.szName, "Gain"))
        {
            m_dwAGCControlID = mxCt.dwControlID;

//          m_pAGC->SetWindowText(mxCt.szName);
            m_pAGC->ShowWindow(SW_SHOW);
            m_pAGC->EnableWindow(TRUE);
            GetAGC();
        }
    }
}
.
/* FUNCTION *****************************************************************
 *
 * Name:    SetNewAGC
 * Notes:   Send the value from the dialog control to the driver
 *
 ****************************************************************************/
void CAudio::SetNewAGC()
{
    if (!m_pAGC)
        return;

    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_BOOLEAN     mxAGC;


    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwAGCControlID;
    mxDet.cChannels = 1;
    mxDet.cMultipleItems = m_cMultipleItems;
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_BOOLEAN);
    mxDet.paDetails = &mxAGC;
    mxAGC.fValue = m_pAGC->GetCheck();
    mixerSetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_SETCONTROLDETAILSF_VALUE
| MIXER_OBJECTF_MIXER);
}
.
/* FUNCTION *****************************************************************
 *
 * Name:    GetAGC
 * Notes:   Get the current value from the driver
 *
 ****************************************************************************/
BOOL CAudio::GetAGC()
{
```

```
    BOOL    fValue = 0;
    MIXERCONTROLDETAILS             mxDet;
    MIXERCONTROLDETAILS_BOOLEAN     mxAGC;

    mxDet.cbStruct = sizeof(MIXERCONTROLDETAILS);
    mxDet.dwControlID = m_dwAGCControlID;
    mxDet.cChannels = 1;
    mxDet.cMultipleItems = 0;
    mxDet.cbDetails = sizeof(MIXERCONTROLDETAILS_BOOLEAN);
    mxDet.paDetails = &mxAGC;
    if (mixerGetControlDetails((HMIXEROBJ) FIRST_CARD, &mxDet, MIXER_GETCONTROLDETAILSF_VA
LUE | MIXER_OBJECTF_MIXER) == MMSYSERR_NOERROR)
    {
        fValue = mxAGC.fValue;
        if (m_pAGC)
        {
            m_pAGC->SetCheck(fValue);
        }
    }
    return fValue;
}

/* FUNCTION ****************************************************************
 *
 * Name:    FloatToWord
 * Notes:   Convert a floating point number to a WORD with rounding
 *
 ***************************************************************************/
WORD CAudio::FloatToWord(FLOAT fIn)
{
    WORD    wOut = (WORD)fIn;
    if (fIn - (FLOAT)wOut >= .5)
        wOut++;
    return (wOut);
}
```

```
/* SOURCE_FILE ************************************************************************
 *
 * File:     HiddenWindow.h
 *
 * Owner:    Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ------------------------------------------------------------------------------------
 *  Leonard 13Nov96      Creation
 *
 *END*********************************************************************************/

class   CMicPropSheet;
class   CMicCtrlDlg;
class   CIODlg;
class   CIODlg;
class   COptionsDlg;
class   CAudio;

//////////////////////////////////////////////////////////////////////////////////////
// CHiddenWindow window

class CHiddenWindow : public CWnd
{
// Construction
public:
    CHiddenWindow();

// Attributes
private:
    CMicPropSheet*  m_pPropSheet;
    CMicCtrlDlg*    m_pMicDlg;
    CIODlg*         m_pInputsDlg;
    CIODlg*         m_pOutputsDlg;
//  CIODlg*         m_pVoiceDlg;
    COptionsDlg*    m_pOptionsDlg;
    HICON           m_hIconTray;
    HICON           m_hIconTray2;
    HICON           m_hIconTrayCur;
    CPtrList        m_InputList;
    CPtrList        m_OutputList;
    HMIXER          m_hMixer;
    BOOL            m_fOnTray;

// Operations
public:
    void    WindowInit();
    void    PropSheetDestroyed();
    void    UpdateTray();

private:
    void    ShowPropSheet();
    void    SetupTray(DWORD dwMessage);
    void    InitializeAudioObjects();
    CAudio* CreateAudioObject(  int         iType
                              , MIXERLINE   *pmxLine
                              , CAudio      *pMasterControl);
    CAudio* GetAudioObjectFromID(int iType, LONG lID);
```

```
    BOOL    CheckDefaults();
    void    ModifyTrayIcon(BOOL fWarn);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CHiddenWindow)
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CHiddenWindow();

    // Generated message map functions
protected:
    //{{AFX_MSG(CHiddenWindow)
    afx_msg void OnOpen();
    afx_msg void OnClose();
    afx_msg void OnAbout();
    //}}AFX_MSG
    afx_msg LRESULT OnNotifyIcon(WPARAM, LPARAM);
    afx_msg LRESULT OnShowYourself(WPARAM, LPARAM);
    afx_msg LRESULT OnControlChange(WPARAM, LPARAM);
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
/* SOURCE_FILE ************************************************************
 *
 * File:     HiddenWindow.cpp
 *
 * Title:
 *
 * Owner:    Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ------------------------------------------------------------------------
 *  Leonard 06Nov96      Creation
 *
 *END*********************************************************************/

/*========================== INCLUDES & DEFINES ============================*/

#include "stdafx.h"
#include "MicCtrl.h"
#include "MicPropSheet.h"
#include "HiddenWindow.h"
#include "IODlg.h"
#include "MicCtrlDlg.h"
#include "OptionsDlg.h"
#include "CAudio.h"

#define MYWM_NOTIFYICON          (WM_APP+100)

// static const char szWarnKey[]            = "WarnOnStartup";
// static const char szTrayKey[]            = "Taskbar";

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/* FUNCTION ***************************************************************
 *
 * Name:     CHiddenWindow - Constructor
 *
 **************************************************************************/
CHiddenWindow::CHiddenWindow()
{
    m_pPropSheet = 0;
    m_pMicDlg = 0;
    m_pInputsDlg = 0;
    m_pOutputsDlg = 0;
    m_pOptionsDlg = 0;
    m_hMixer = 0;
    m_fOnTray = FALSE;
}


/* FUNCTION ***************************************************************
 *
 * Name:         CHiddenWindow - Destructor
 *
 **************************************************************************/
```

```
CHiddenWindow::~CHiddenWindow()
{

    if (m_pPropSheet)
        delete m_pPropSheet;
    if (m_pMicDlg)
        delete m_pMicDlg;
    if (m_pInputsDlg)
        delete m_pInputsDlg;
    if (m_pOutputsDlg)
        delete m_pOutputsDlg;
    if (m_pOptionsDlg)
    {
        if (m_pOptionsDlg->m_fTray)
            SetupTray(NIM_DELETE);
        delete m_pOptionsDlg;
    }

    while (!m_InputList.IsEmpty ())
    {
        delete (CAudio*)m_InputList.RemoveHead ();
    }
    while (!m_OutputList.IsEmpty ())
    {
        delete (CAudio*)m_OutputList.RemoveHead ();
    }

    if (m_hMixer)
        mixerClose(m_hMixer);
}


BEGIN_MESSAGE_MAP(CHiddenWindow, CWnd)
    //{{AFX_MSG_MAP(CHiddenWindow)
    ON_COMMAND(IDM_OPEN, OnOpen)
    ON_COMMAND(IDM_CLOSE, OnClose)
    ON_COMMAND(IDM_ABOUT, OnAbout)
    //}}AFX_MSG_MAP
    ON_MESSAGE(MYWM_NOTIFYICON, OnNotifyIcon)
    ON_MESSAGE(MYWM_SHOWYOURSELF, OnShowYourself)
    ON_MESSAGE(MM_MIXM_CONTROL_CHANGE, OnControlChange)
END_MESSAGE_MAP()

.
/* FUNCTION ****************************************************************
 *
 * Name:    WindowInit
 * Notes:
 *
 ****************************************************************************/
void CHiddenWindow::WindowInit()
{
    m_hIconTrayCur = m_hIconTray = AfxGetApp()->LoadIcon(IDI_TRAY);
    m_hIconTray2 = AfxGetApp()->LoadIcon(IDI_TRAY2);

    /*
     * Create Property Sheet & Pages
     */
    CString sString;
    sString.LoadString(IDS_TITLE);
```

```
    m_pPropSheet = new CMicPropSheet(sString);
    m_pMicDlg = new CMicCtrlDlg;
    m_pInputsDlg = new CIODlg;
    m_pOutputsDlg = new CIODlg;
//  m_pVoiceDlg = new CIODlg;
    m_pOptionsDlg = new COptionsDlg;

    /*
     * Initialize dialog member variables
     */
    m_pPropSheet->m_pMicDlg = m_pMicDlg;
    m_pPropSheet->m_pOptionsDlg = m_pOptionsDlg;
    m_pPropSheet->m_pPlaybackDlg = m_pOutputsDlg;
    m_pPropSheet->m_pRecordingDlg = m_pInputsDlg;
//  m_pPropSheet->m_pVoiceDlg = m_pVoiceDlg;
    m_pPropSheet->m_pHiddenWindow = this;

    m_pMicDlg->m_pOptionsDlg = m_pOptionsDlg;

    m_pInputsDlg->m_iLineType = INPUT_LINE;
    m_pOutputsDlg->m_iLineType = OUTPUT_LINE;
//  m_pVoiceDlg->m_iLineType = VOICE_LINE;
    m_pInputsDlg->m_pOptionsDlg = m_pOptionsDlg;
    m_pOutputsDlg->m_pOptionsDlg = m_pOptionsDlg;
//  m_pVoiceDlg->m_pOptionsDlg = m_pOptionsDlg;

    m_pOptionsDlg->m_pPropSheet = m_pPropSheet;
    m_pOptionsDlg->m_pHiddenWindow = this;

    /*
     * Add pages and display property sheet
     */
    int iPage = 1;

    m_pPropSheet->AddPage(m_pMicDlg);
    if (m_pOptionsDlg->m_fRecording)
    {
        m_pPropSheet->AddPage(m_pInputsDlg);
        m_pPropSheet->m_iRecordingPage = iPage++;
    }
    else
    {
        m_pPropSheet->m_iRecordingPage = -1;
    }
    if (m_pOptionsDlg->m_fPlayback)
    {
        m_pPropSheet->AddPage(m_pOutputsDlg);
        m_pPropSheet->m_iPlaybackPage = iPage++;
    }
    else
    {
        m_pPropSheet->m_iPlaybackPage = -1;
    }
//  if (m_pOptionsDlg->m_fVoice)
//  {
//      m_pPropSheet->AddPage(m_pVoiceDlg);
//      m_pPropSheet->m_iVoicePage = iPage;
//  }
//  else
//  {
```

```
//      m_pPropSheet->m_iVoicePage = -1;
//  }
    m_pPropSheet->AddPage(m_pOptionsDlg);

    InitializeAudioObjects();

    /*
     * Setup system tray
     */
    CMicCtrlApp*    pApp = (CMicCtrlApp*)AfxGetApp();
    if (pApp->m_fTraySwitch)
    {
        m_fOnTray = TRUE;
        SetupTray(NIM_ADD);
    }
    else
    {
        ShowPropSheet();
    }

    return;
}

/* FUNCTION ****************************************************************
 *
 * Name:    ShowPropSheet
 * Notes:   If the property sheet is already created, bring it to the front
 *          Otherwise, create it
 *
 ***************************************************************************/
void CHiddenWindow::ShowPropSheet()
{
    if (m_pPropSheet->m_hWnd)
    {
        m_pPropSheet->ShowWindow(SW_RESTORE);
        m_pPropSheet->SetForegroundWindow();
    }
    else
    {
        m_pPropSheet->Create();
    }
}

/* FUNCTION ****************************************************************
 *
 * Name:    SetupTray
 * Notes:   Add/Modify/Remove the icon on the system tray
 *
 ***************************************************************************/
void CHiddenWindow::SetupTray(DWORD dwMessage)
{
    NOTIFYICONDATA  notify;
    CString         sString;

    notify.cbSize           = sizeof(NOTIFYICONDATA);
    notify.hWnd             = m_hWnd;
    notify.uID              = 1;
    notify.uFlags           = NIF_ICON | NIF_MESSAGE | NIF_TIP;
    notify.uCallbackMessage = MYWM_NOTIFYICON;
    notify.hIcon            = m_hIconTrayCur;
```

```
    sString.LoadString(IDS_TRAY_TIP);
    strcpy(notify.szTip, (LPSTR)(LPCSTR)sString);

    Shell_NotifyIcon(dwMessage, &notify);
}
.
/* FUNCTION ****************************************************************
 *
 * Name:    UpdateTray
 * Notes:   Add/remove the icon from the system tray, based on the options dlg
 *
 ***************************************************************************/
void CHiddenWindow::UpdateTray()
{
    if (m_pOptionsDlg->m_fTray)
    {
        SetupTray(NIM_ADD);
        CheckDefaults();
        m_fOnTray = TRUE;
    }
    else
    {
        SetupTray(NIM_DELETE);
        m_fOnTray = FALSE;
    }
}
.
/* FUNCTION ****************************************************************
 *
 * Name:    OnNotifyIcon
 * Notes:   Process mouse messages from the system tray
 *
 ***************************************************************************/
LRESULT CHiddenWindow::OnNotifyIcon(WPARAM wParam, LPARAM lParam)
{
    switch (lParam)
    {
    case WM_RBUTTONDOWN:
    {
        /*
         * Non-default click, popup the context menu
         */
        CMenu   menu;
        if (menu.LoadMenu(IDR_TRAY_MENU))
        {
            CMenu* pPopup = menu.GetSubMenu(0);

            if (pPopup)
            {
                POINT   point;
                GetCursorPos(&point);
                pPopup->TrackPopupMenu(TPM_LEFTALIGN|TPM_RIGHTBUTTON, point.x, point.y, th
is);
            }
        }
        break;
    }

    case WM_LBUTTONDOWN:
        /*
```

```
        * default click, display the property sheet
        */
       ShowPropSheet();
       break;

   default:
       break;
   }

   return(TRUE);
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnDefaults
 * Notes:   "Set Defaults" menu handler
 *
 ***************************************************************************/
//void CHiddenWindow::OnDefaults()
//{
//  m_pMicDlg->SetDefaults();
//}

/* FUNCTION ****************************************************************
 *
 * Name:    OnOpen
 * Notes:   "Open" menu handler
 *
 ***************************************************************************/
void CHiddenWindow::OnOpen()
{
    ShowPropSheet();
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnClose
 * Notes:   "Close" menu handler
 *
 ***************************************************************************/
void CHiddenWindow::OnClose()
{
     AfxGetMainWnd()->PostMessage(WM_QUIT);
}
/* FUNCTION ****************************************************************
 *
 * Name:    PropSheetDestroyed
 * Notes:   If the user closed the property sheet, and the option is not set
 *          not set to run from the system tray, shut down the application.
 ***************************************************************************/
void CHiddenWindow::PropSheetDestroyed()
{
//  if (!m_pOptionsDlg->m_fTray)
    if (!m_fOnTray)
    {
         AfxGetMainWnd()->PostMessage(WM_QUIT);
```

```
    }
}
.
/* FUNCTION ****************************************************************
 *
 * Name:     InitializeAudioObjects
 * Notes:    Find the audio objects we care about, so we can see if they are
 *           setup properly.
 *
 ***************************************************************************/
void CHiddenWindow::InitializeAudioObjects()
{
    if (!mixerGetNumDevs())
    {
        return;
    }

    MIXERCAPS   mxCaps;
    if (mixerGetDevCaps(FIRST_CARD, &mxCaps, sizeof(MIXERCAPS)) != MMSYSERR_NOERROR)
    {
        ASSERT(TRUE);
        return;
    }

    MIXERLINE   mxLine;
    CAudio      *pMasterAudio;
    int         iCon, iNumCon;

    /*
     * Get the master output control
     */
    mxLine.cbStruct = sizeof(MIXERLINE);
    mxLine.dwDestination = OUTPUT_LINE;
    mxLine.dwSource = 0;
    if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_DESTINATION |
MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
    {
        ASSERT(TRUE);
        return;
    }
    pMasterAudio = CreateAudioObject(OUTPUT_LINE, &mxLine, NULL);

    int iMute = 0;

    /*
     * Look through the rest of the output controls for the Wave control and
     * the Microphone control.
     */
    for (iCon = 0, iNumCon = (int)mxLine.cConnections; iCon < iNumCon; iCon++)
    {
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination  = OUTPUT_LINE;
        mxLine.dwSource = iCon;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE | M
IXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
            return;
        }
        if ((mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT)
```

```
            || (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE))
        {
            CreateAudioObject(OUTPUT_LINE, &mxLine, pMasterAudio);
        }
    }

    iMute = 0;

    if (mxCaps.cDestinations > 1)
    {
        /*
         * Get the master input control
         */
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination = INPUT_LINE;
        mxLine.dwSource = 0;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_DESTINATIO
N | MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
            return;
        }
        pMasterAudio = CreateAudioObject(INPUT_LINE, &mxLine, NULL);

        /*
         * At least one of the drivers () falsely reports that 3 of
         * of its four controls are microphones.  This kludge is to look for
         * the one true microphone.
         */
        int iMic = -1;
        iNumCon = (int)mxLine.cConnections;
        for (iCon = 0; iCon < iNumCon; iCon++)
        {
            mxLine.cbStruct = sizeof(MIXERLINE);
            mxLine.dwDestination  = INPUT_LINE;
            mxLine.dwSource = iCon;
            if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE
 | MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
            {
                ASSERT(TRUE);
                return;
            }
            if (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
            {
                if (iMic == -1)
                {
                    /*
                     * default to the first one if all of the string compares fail
                     */
                    iMic = iCon;
                }
                if (strstr(mxLine.szName, "Microphone"))
                {
                    iMic = iCon;
                    break;
                }
            }
        }
        /*
         * Look through the input controls for the Microphone
```

```
         * control.
         */
        for (iCon = 0; iCon < iNumCon; iCon++)
        {
            mxLine.cbStruct = sizeof(MIXERLINE);
            mxLine.dwDestination  = INPUT_LINE;
            mxLine.dwSource = iCon;
            if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE
 | MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
            {
                ASSERT(TRUE);
                return;
            }
            if (iCon == iMic)
            {
                CreateAudioObject(INPUT_LINE, &mxLine, pMasterAudio);
                break;
            }
        }
    }

    mixerOpen(&m_hMixer, FIRST_CARD, (DWORD)m_hWnd, 0, CALLBACK_WINDOW | MIXER_OBJECTF_MIX
ER);

    CheckDefaults();
}

.
/* FUNCTION ****************************************************************
 *
            CreateAudioObject
 *
 ***************************************************************************/
CAudio* CHiddenWindow::CreateAudioObject(  int  iType
                                , MIXERLINE     *pmxLine
                                , CAudio        *pMasterControl)
{
    CAudio *    pAudio = new CAudio;

    pAudio->m_iType             = iType;
    pAudio->m_dwSource          = pmxLine->dwSource;
    pAudio->m_dwLineID          = pmxLine->dwLineID;
    pAudio->m_dwComponentType   = pmxLine->dwComponentType;
    pAudio->m_cChannels         = pmxLine->cChannels;
    pAudio->m_sShortName        = pmxLine->szShortName;
    pAudio->m_sName             = pmxLine->szName;
    pAudio->m_pMasterControl    = pMasterControl;
    pAudio->m_fCheckForInvertMute = FALSE;

    pAudio->InitControls();

    if (iType == OUTPUT_LINE)
        m_OutputList.AddTail((CObject*)pAudio);
    else
        m_InputList.AddTail((CObject*)pAudio);

    return(pAudio);
}
.
/* FUNCTION ****************************************************************
```

```
 *
 * Name:     GetAudioObjectFromID
 * Notes:    Find the Audio Object that matches the supplied ID
 *
 ********************************************************************************/
CAudio* CHiddenWindow::GetAudioObjectFromID(int iType, LONG lID)
{
    BOOL        fMatch = FALSE;
    CAudio*     pAudio;
    POSITION    position = m_OutputList.GetHeadPosition();

    /*
     * loop through each output device
     */
    while(position && !fMatch)
    {
        pAudio = (CAudio*)m_OutputList.GetNext(position);
        switch (iType)
        {
        case VOLUME_CONTROL_ID:
            if (pAudio->m_dwVolControlID == (DWORD)lID)
                fMatch = TRUE;
            break;
        case MUTE_CONTROL_ID:
            if (pAudio->m_dwMuteControlID == (DWORD)lID)
                fMatch = TRUE;
            break;
        }
    }
    if (!fMatch)
    {
        position = m_InputList.GetHeadPosition();

        /*
         * loop through each input device
         */
        while(position && !fMatch)
        {
            pAudio = (CAudio*)m_InputList.GetNext(position);
            switch (iType)
            {
            case VOLUME_CONTROL_ID:
                if (pAudio->m_dwVolControlID == (DWORD)lID)
                    fMatch = TRUE;
                break;
            case MUTE_CONTROL_ID:
                if (pAudio->m_dwMuteControlID == (DWORD)lID)
                    fMatch = TRUE;
                break;
            }
        }
    }
    return(fMatch ? pAudio : NULL);
}

/* FUNCTION *********************************************************************
 *
 * Name:     OnControlChange
 * Notes:    This is called by the driver whenever any control has changed
 *
```

```
 ****************************************************************************/
LRESULT CHiddenWindow::OnControlChange(WPARAM wParam, LPARAM lParam)
{

    /*
     * Is this one of the volume controls we care about?
     */
    CAudio* pAudio = GetAudioObjectFromID(VOLUME_CONTROL_ID, lParam);
    if (pAudio)
    {
        CheckDefaults();
    }
    else
    {
        /*
         * Is this one of the mute controls we care about?
         */
        pAudio = GetAudioObjectFromID(MUTE_CONTROL_ID, lParam);
        if (pAudio)
        {
            CheckDefaults();
        }
    }
    return(TRUE);
}

/* FUNCTION ****************************************************************
 *
 * Name:    CheckDefaults
 * Notes:   Check to see if the controls we care about are setup to work
 * with the microphone.
 *
 ****************************************************************************/
BOOL CHiddenWindow::CheckDefaults()
{
    CAudio*     pAudio;
    POSITION    position = m_OutputList.GetHeadPosition();
    BOOL        fDefault = TRUE;

    /*
     * loop through each output device
     */
    while(position && fDefault)
    {
        pAudio = (CAudio*)m_OutputList.GetNext(position);
        if (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
        {
            if (!pAudio->CheckDefaultMute(TRUE))
                fDefault = FALSE;
        }
        else
        {
            if (!pAudio->CheckDefaultMute(FALSE))
                fDefault = FALSE;
            if (!pAudio->CheckDefaultVolume())
                fDefault = FALSE;
        }
    }
    position = m_InputList.GetHeadPosition();
```

```
    /*
     * loop through each input device
     */
    while(position && fDefault)
    {
        pAudio = (CAudio*)m_InputList.GetNext(position);

        if (!pAudio->CheckDefaultMute(FALSE))
            fDefault = FALSE;
        if (!pAudio->CheckDefaultVolume())
            fDefault = FALSE;
    }
    /*
     * Modify the tray icon to indicate good or bad microphone setup
     */
    if (fDefault)
    {
        ModifyTrayIcon(FALSE);
    }
    else
    {
        ModifyTrayIcon(TRUE);
    }
    return(fDefault);
}

/* FUNCTION ****************************************************************
 *
 * Name:    ModifyTrayIcon
 * Notes:   If fWarn, change the tray icon to indicate a problem for using
 *          the microphone
 *
 ****************************************************************************/
void CHiddenWindow::ModifyTrayIcon(BOOL fWarn)
{
    if (m_pOptionsDlg->m_fTray)
    {
        m_hIconTrayCur = fWarn ? m_hIconTray2 : m_hIconTray;
        SetupTray(NIM_MODIFY);
    }
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnShowYourself
 * Notes:
 *
 ****************************************************************************/
LRESULT CHiddenWindow::OnShowYourself(WPARAM wParam, LPARAM lParam)
{
    ShowPropSheet();
    return (TRUE);
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnAbout
 * Notes:
 *
```

```
 ***************************************************************************/
void CHiddenWindow::OnAbout()
{
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();
}
```

```
/* SOURCE_FILE ********************************************************************
 *
 * File:     IODlg.h
 *
 * Owner:    Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ------------------------------------------------------------------------------
 *  Leonard 13Nov96      Creation
 *
 *END*****************************************************************************/

#include "CAudio.h"

class   COptionsDlg;

class CIODlg : public CPropertyPage
{
    DECLARE_DYNCREATE(CIODlg)

// Construction
public:
    CIODlg();
    ~CIODlg();

// Dialog Data
    //{{AFX_DATA(CIODlg)
    enum { IDD = IDD_IO_DIALOG };
    CStatic m_VolWarn6;
    CStatic m_VolWarn7;
    CStatic m_VolWarn5;
    CStatic m_VolWarn4;
    CStatic m_VolWarn3;
    CStatic m_VolWarn2;
    CStatic m_VolWarn1;
    CStatic m_MuteWarn7;
    CStatic m_MuteWarn6;
    CStatic m_MuteWarn5;
    CStatic m_MuteWarn4;
    CStatic m_MuteWarn3;
    CStatic m_MuteWarn2;
    CStatic m_MuteWarn1;
    CStatic m_MuteWarnText2;
    CStatic m_MuteWarnText;
    CStatic m_MuteWarn;
    CStatic m_BalText5;
    CSliderCtrl m_Volume7;
    CSliderCtrl m_Volume6;
    CSliderCtrl m_Volume5;
    CSliderCtrl m_Volume4;
    CSliderCtrl m_Volume3;
    CSliderCtrl m_Volume2;
    CSliderCtrl m_Volume1;
    CStatic m_VolText7;
    CStatic m_VolText6;
    CStatic m_VolText5;
    CStatic m_VolText4;
    CStatic m_VolText3;
    CStatic m_VolText2;
```

```
    CStatic m_VolText1;
    CStatic m_Title7;
    CStatic m_Title6;
    CStatic m_Title5;
    CStatic m_Title4;
    CStatic m_Title3;
    CStatic m_Title2;
    CStatic m_Title1;
    CButton m_Mute7;
    CButton m_Mute6;
    CButton m_Mute5;
    CButton m_Mute4;
    CButton m_Mute3;
    CButton m_Mute2;
    CButton m_Mute1;
    CButton m_Line7;
    CButton m_Line6;
    CButton m_Line5;
    CButton m_Line4;
    CButton m_Line3;
    CButton m_Line2;
    CStatic m_Icon9;
    CStatic m_Icon8;
    CStatic m_Icon7;
    CStatic m_Icon6;
    CStatic m_Icon5;
    CStatic m_Icon4;
    CStatic m_Icon3;
    CStatic m_Icon2;
    CStatic m_Icon14;
    CStatic m_Icon13;
    CStatic m_Icon12;
    CStatic m_Icon11;
    CStatic m_Icon10;
    CStatic m_Icon1;
    CSliderCtrl m_Balance7;
    CSliderCtrl m_Balance6;
    CSliderCtrl m_Balance5;
    CSliderCtrl m_Balance4;
    CSliderCtrl m_Balance3;
    CSliderCtrl m_Balance2;
    CSliderCtrl m_Balance1;
    CStatic m_BalText7;
    CStatic m_BalText6;
    CStatic m_BalText4;
    CStatic m_BalText3;
    CStatic m_BalText2;
    CStatic m_BalText1;
    //}}AFX_DATA

public:
    int     m_iLineType;
    COptionsDlg *m_pOptionsDlg;

    CAudio* CreateAudioObject(  int iType
                              , MIXERLINE     *pmxLine
                              , CSliderCtrl   *pBalance
                              , CSliderCtrl   *pVolume
                              , CButton       *pMute
                              , CStatic       *pTitle
```

```
                                , CStatic       *pBalText
                                , CStatic       *pVolText
                                , CStatic       *pIcon1
                                , CStatic       *pIcon2
                                , CButton       *pLine
                                , CStatic       *pMuteWarn
                                , CStatic       *pVolWarn
                                , CAudio        *pMasterControl);
    CAudio* GetAudioObjectFromID(int iType, LONG lID);
    BOOL    CheckDefaults();
    void    FindMeterControls();
    void    UpdateMeter(CDC *pDC = 0, BOOL fForceUpdate = FALSE);

protected:
    CPtrList    m_AudioList;
    CPtrList    m_MeterList;
    HMIXER      m_hMixer;
    UINT        m_uMeterTimerId;

// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CIODlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CIODlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    afx_msg void OnDestroy();
    //}}AFX_MSG
    afx_msg LRESULT OnControlChange(WPARAM, LPARAM);
    afx_msg LRESULT OnLineChange(WPARAM, LPARAM);
    DECLARE_MESSAGE_MAP()

};
```

```
/* SOURCE_FILE ************************************************************
 *
 * File:     IODlg.cpp
 *
 * Owner:    Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ------------------------------------------------------------------------
 *  Leonard 25Oct96      Creation
 *
 *END*********************************************************************/

/*========================== INCLUDES & DEFINES ============================*/

#include "stdafx.h"
#include "MicCtrl.h"
#include "IODlg.h"
#include "caudio.h"
#include "OptionsDlg.h"

#define TIMER_METER     0x5678

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CIODlg, CPropertyPage)

/* FUNCTION ***************************************************************
 *
 * Name:     CIODlg
 *
 *************************************************************************/
CIODlg::CIODlg() : CPropertyPage(CIODlg::IDD)
{
    //{{AFX_DATA_INIT(CIODlg)
    //}}AFX_DATA_INIT

    m_iLineType = -1;
}

CIODlg::~CIODlg()
{
}

void CIODlg::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CIODlg)
    DDX_Control(pDX, IDC_VOL_WARN6, m_VolWarn6);
    DDX_Control(pDX, IDC_VOL_WARN7, m_VolWarn7);
    DDX_Control(pDX, IDC_VOL_WARN5, m_VolWarn5);
    DDX_Control(pDX, IDC_VOL_WARN4, m_VolWarn4);
    DDX_Control(pDX, IDC_VOL_WARN3, m_VolWarn3);
    DDX_Control(pDX, IDC_VOL_WARN2, m_VolWarn2);
    DDX_Control(pDX, IDC_VOL_WARN1, m_VolWarn1);
```

```
DDX_Control(pDX, IDC_MUTE_WARN7, m_MuteWarn7);
DDX_Control(pDX, IDC_MUTE_WARN6, m_MuteWarn6);
DDX_Control(pDX, IDC_MUTE_WARN5, m_MuteWarn5);
DDX_Control(pDX, IDC_MUTE_WARN4, m_MuteWarn4);
DDX_Control(pDX, IDC_MUTE_WARN3, m_MuteWarn3);
DDX_Control(pDX, IDC_MUTE_WARN2, m_MuteWarn2);
DDX_Control(pDX, IDC_MUTE_WARN1, m_MuteWarn1);
DDX_Control(pDX, IDC_MUTE_WARN_TEXT2, m_MuteWarnText2);
DDX_Control(pDX, IDC_MUTE_WARN_TEXT, m_MuteWarnText);
DDX_Control(pDX, IDC_MUTE_WARN, m_MuteWarn);
DDX_Control(pDX, IDC_BAL_TEXT_5, m_BalText5);
DDX_Control(pDX, IDC_VOLUME_7, m_Volume7);
DDX_Control(pDX, IDC_VOLUME_6, m_Volume6);
DDX_Control(pDX, IDC_VOLUME_5, m_Volume5);
DDX_Control(pDX, IDC_VOLUME_4, m_Volume4);
DDX_Control(pDX, IDC_VOLUME_3, m_Volume3);
DDX_Control(pDX, IDC_VOLUME_2, m_Volume2);
DDX_Control(pDX, IDC_VOLUME_1, m_Volume1);
DDX_Control(pDX, IDC_VOL_TEXT_7, m_VolText7);
DDX_Control(pDX, IDC_VOL_TEXT_6, m_VolText6);
DDX_Control(pDX, IDC_VOL_TEXT_5, m_VolText5);
DDX_Control(pDX, IDC_VOL_TEXT_4, m_VolText4);
DDX_Control(pDX, IDC_VOL_TEXT_3, m_VolText3);
DDX_Control(pDX, IDC_VOL_TEXT_2, m_VolText2);
DDX_Control(pDX, IDC_VOL_TEXT_1, m_VolText1);
DDX_Control(pDX, IDC_TITLE_7, m_Title7);
DDX_Control(pDX, IDC_TITLE_6, m_Title6);
DDX_Control(pDX, IDC_TITLE_5, m_Title5);
DDX_Control(pDX, IDC_TITLE_4, m_Title4);
DDX_Control(pDX, IDC_TITLE_3, m_Title3);
DDX_Control(pDX, IDC_TITLE_2, m_Title2);
DDX_Control(pDX, IDC_TITLE_1, m_Title1);
DDX_Control(pDX, IDC_MUTE_7, m_Mute7);
DDX_Control(pDX, IDC_MUTE_6, m_Mute6);
DDX_Control(pDX, IDC_MUTE_5, m_Mute5);
DDX_Control(pDX, IDC_MUTE_4, m_Mute4);
DDX_Control(pDX, IDC_MUTE_3, m_Mute3);
DDX_Control(pDX, IDC_MUTE_2, m_Mute2);
DDX_Control(pDX, IDC_MUTE_1, m_Mute1);
DDX_Control(pDX, IDC_LINE_7, m_Line7);
DDX_Control(pDX, IDC_LINE_6, m_Line6);
DDX_Control(pDX, IDC_LINE_5, m_Line5);
DDX_Control(pDX, IDC_LINE_4, m_Line4);
DDX_Control(pDX, IDC_LINE_3, m_Line3);
DDX_Control(pDX, IDC_LINE_2, m_Line2);
DDX_Control(pDX, IDC_ICON_9, m_Icon9);
DDX_Control(pDX, IDC_ICON_8, m_Icon8);
DDX_Control(pDX, IDC_ICON_7, m_Icon7);
DDX_Control(pDX, IDC_ICON_6, m_Icon6);
DDX_Control(pDX, IDC_ICON_5, m_Icon5);
DDX_Control(pDX, IDC_ICON_4, m_Icon4);
DDX_Control(pDX, IDC_ICON_3, m_Icon3);
DDX_Control(pDX, IDC_ICON_2, m_Icon2);
DDX_Control(pDX, IDC_ICON_14, m_Icon14);
DDX_Control(pDX, IDC_ICON_13, m_Icon13);
DDX_Control(pDX, IDC_ICON_12, m_Icon12);
DDX_Control(pDX, IDC_ICON_11, m_Icon11);
DDX_Control(pDX, IDC_ICON_10, m_Icon10);
DDX_Control(pDX, IDC_ICON_1, m_Icon1);
DDX_Control(pDX, IDC_BALANCE_7, m_Balance7);
```

```
    DDX_Control(pDX, IDC_BALANCE_6, m_Balance6);
    DDX_Control(pDX, IDC_BALANCE_5, m_Balance5);
    DDX_Control(pDX, IDC_BALANCE_4, m_Balance4);
    DDX_Control(pDX, IDC_BALANCE_3, m_Balance3);
    DDX_Control(pDX, IDC_BALANCE_2, m_Balance2);
    DDX_Control(pDX, IDC_BALANCE_1, m_Balance1);
    DDX_Control(pDX, IDC_BAL_TEXT_7, m_BalText7);
    DDX_Control(pDX, IDC_BAL_TEXT_6, m_BalText6);
    DDX_Control(pDX, IDC_BAL_TEXT_4, m_BalText4);
    DDX_Control(pDX, IDC_BAL_TEXT_3, m_BalText3);
    DDX_Control(pDX, IDC_BAL_TEXT_2, m_BalText2);
    DDX_Control(pDX, IDC_BAL_TEXT_1, m_BalText1);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CIODlg, CPropertyPage)
    //{{AFX_MSG_MAP(CIODlg)
    ON_WM_HSCROLL()
    ON_WM_VSCROLL()
    ON_WM_TIMER()
    ON_WM_ERASEBKGND()
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
    ON_MESSAGE(MM_MIXM_CONTROL_CHANGE, OnControlChange)
    ON_MESSAGE(MM_MIXM_LINE_CHANGE, OnLineChange)
END_MESSAGE_MAP()

/* FUNCTION ****************************************************************
 *
 * Name:    OnInitDialog
 *
 ****************************************************************************/
BOOL CIODlg::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    m_hMixer = 0;
    m_uMeterTimerId = 0;

    if (!mixerGetNumDevs())
    {
        return(TRUE);
    }

    MIXERCAPS   mxCaps;
    if (mixerGetDevCaps(FIRST_CARD, &mxCaps, sizeof(MIXERCAPS)) != MMSYSERR_NOERROR)
    {
        ASSERT(TRUE);
        return(TRUE);
    }

    CAudio      *pAudio;
    CAudio      *pMasterAudio;
    MIXERLINE   mxLine;
    int         iCon, iNumCon;

    /*
     * Get all controls of m_iLineType (Record or Playback)
```

```
     */
    mxLine.cbStruct = sizeof(MIXERLINE);
    mxLine.dwDestination = m_iLineType;
    mxLine.dwSource = 0;
    if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_DESTINATION |
MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
    {
        ASSERT(TRUE);
        return(TRUE);
    }

    pMasterAudio = CreateAudioObject(m_iLineType
                                , &mxLine
                                , &m_Balance1
                                , &m_Volume1
                                , &m_Mute1
                                , &m_Title1
                                , &m_BalText1
                                , &m_VolText1
                                , &m_Icon1
                                , &m_Icon2
                                , &m_Line2
                                , &m_MuteWarn1
                                , &m_VolWarn1
                                , NULL);

    m_AudioList.AddTail((CObject*)pMasterAudio);

    /*
     * At least one of the drivers () falsely reports that 3 of
     * of its four controls are microphones.  This kludge is to look for
     * the one true microphone.
     */
    int iMic = -1;
    iNumCon = (int)mxLine.cConnections;
    for (iCon = 0; iCon < iNumCon; iCon++)
    {
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination  = m_iLineType;
        mxLine.dwSource = iCon;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE | M
IXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
            return(TRUE);
        }
        if (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
        {
            if (iMic == -1)
            {
                /*
                 * default to the first one if all of the string compares fail
                 */
                iMic = iCon;
            }
            if (strstr(mxLine.szName, "Microphone"))
            {
                iMic = iCon;
                break;
            }
```

```
        }
    }

    int iMute = 0;

    for (iCon = 0; iCon < iNumCon; iCon++)
    {
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination  = m_iLineType;
        mxLine.dwSource = iCon;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE | M
IXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
            return(TRUE);
        }
        /*
         * if this is not really a microphone, reset the component type
         * (see kludge above)
         */
        if (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE
                && iCon != iMic)
            mxLine.dwComponentType = 0;
        pAudio = 0;
        switch (++iMute)
        {
        case 1:
            pAudio = CreateAudioObject(m_iLineType
                                , &mxLine
                                , &m_Balance2
                                , &m_Volume2
                                , &m_Mute2
                                , &m_Title2
                                , &m_BalText2
                                , &m_VolText2
                                , &m_Icon3
                                , &m_Icon4
                                , &m_Line3
                                , &m_MuteWarn2
                                , &m_VolWarn2
                                , pMasterAudio);
            break;
        case 2:
            pAudio = CreateAudioObject(m_iLineType
                                , &mxLine
                                , &m_Balance3
                                , &m_Volume3
                                , &m_Mute3
                                , &m_Title3
                                , &m_BalText3
                                , &m_VolText3
                                , &m_Icon5
                                , &m_Icon6
                                , &m_Line4
                                , &m_MuteWarn3
                                , &m_VolWarn3
                                , pMasterAudio);
            break;
        case 3:
            pAudio = CreateAudioObject(m_iLineType
```

```
                    , &mxLine
                    , &m_Balance4
                    , &m_Volume4
                    , &m_Mute4
                    , &m_Title4
                    , &m_BalText4
                    , &m_VolText4
                    , &m_Icon7
                    , &m_Icon8
                    , &m_Line5
                    , &m_MuteWarn4
                    , &m_VolWarn4
                    , pMasterAudio);
    break;
case 4:
    pAudio = CreateAudioObject(m_iLineType
                    , &mxLine
                    , &m_Balance5
                    , &m_Volume5
                    , &m_Mute5
                    , &m_Title5
                    , &m_BalText5
                    , &m_VolText5
                    , &m_Icon9
                    , &m_Icon10
                    , &m_Line6
                    , &m_MuteWarn5
                    , &m_VolWarn5
                    , pMasterAudio);
    break;
case 5:
    pAudio = CreateAudioObject(m_iLineType
                    , &mxLine
                    , &m_Balance6
                    , &m_Volume6
                    , &m_Mute6
                    , &m_Title6
                    , &m_BalText6
                    , &m_VolText6
                    , &m_Icon11
                    , &m_Icon12
                    , &m_Line7
                    , &m_MuteWarn6
                    , &m_VolWarn6
                    , pMasterAudio);
    break;
case 6:
    pAudio = CreateAudioObject(m_iLineType
                    , &mxLine
                    , &m_Balance7
                    , &m_Volume7
                    , &m_Mute7
                    , &m_Title7
                    , &m_BalText7
                    , &m_VolText7
                    , &m_Icon13
                    , &m_Icon14
                    , NULL
                    , &m_MuteWarn7
                    , &m_VolWarn7
```

```
                            , pMasterAudio);
            break;
        }
        if (pAudio)
            m_AudioList.AddTail((CObject*)pAudio);
    }


    /*
     * Open the mixer so we can get notifications of changes
     */
    mixerOpen(&m_hMixer, FIRST_CARD, (DWORD)m_hWnd, 0, CALLBACK_WINDOW | MIXER_OBJECTF_MIX
ER);

    CheckDefaults();

    /*
     * Find any VU meters
     */
    FindMeterControls();
    /*
     * Check to see if the meters should be activated
     */
    OnLineChange(0, 0);

    return TRUE;
}

/* FUNCTION ************************************************************************
 *
 * Name:    CreateAudioObject
 *
 ************************************************************************************/
CAudio* CIODlg::CreateAudioObject(    int           iType
                                    , MIXERLINE     *pmxLine
                                    , CSliderCtrl   *pBalance
                                    , CSliderCtrl   *pVolume
                                    , CButton       *pMute
                                    , CStatic       *pTitle
                                    , CStatic       *pBalText
                                    , CStatic       *pVolText
                                    , CStatic       *pIcon1
                                    , CStatic       *pIcon2
                                    , CButton       *pLine
                                    , CStatic       *pMuteWarn
                                    , CStatic       *pVolWarn
                                    , CAudio        *pMasterControl)
{
    CAudio *    pAudio = new CAudio;

    pAudio->m_iType             = iType;
    pAudio->m_dwLineID          = pmxLine->dwLineID;
    pAudio->m_dwComponentType   = pmxLine->dwComponentType;
    pAudio->m_cChannels         = pmxLine->cChannels;
    pAudio->m_sShortName        = pmxLine->szShortName;
    pAudio->m_sName             = pmxLine->szName;
    pAudio->m_pBalance          = pBalance;
    pAudio->m_pVolume           = pVolume;
    pAudio->m_pMute             = pMute;
    pAudio->m_pTitle            = pTitle;
```

```
    pAudio->m_pBalText          = pBalText;
    pAudio->m_pVolText          = pVolText;
    pAudio->m_pIcon1            = pIcon1;
    pAudio->m_pIcon2            = pIcon2;
    pAudio->m_pLine             = pLine;
    pAudio->m_pMuteWarn         = pMuteWarn;
    pAudio->m_pVolWarn          = pVolWarn;
    pAudio->m_pMasterControl    = pMasterControl;

    if (pBalance)
        pAudio->m_lBalanceID = pBalance->GetDlgCtrlID();
    if (pVolume)
        pAudio->m_lVolumeID = pVolume->GetDlgCtrlID();
    if (pMute)
        pAudio->m_lMuteID = pMute->GetDlgCtrlID();

    pAudio->InitControls();

    return(pAudio);
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnHScroll
 * Notes:   Called when one of the Balance controls changes
 *
 ***************************************************************************/
void CIODlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CPropertyPage::OnHScroll(nSBCode, nPos, pScrollBar);

    LONG    lID = GetWindowLong(pScrollBar->m_hWnd, GWL_ID);
    CAudio* pAudio = GetAudioObjectFromID(BALANCE_ID, lID);
    if (pAudio)
    {
        pAudio->SetNewVolume();
    }
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnVScroll
 * Notes:   Called when one of the Volume controls changes
 *
 ***************************************************************************/
void CIODlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

    LONG    lID = GetWindowLong(pScrollBar->m_hWnd, GWL_ID);
    CAudio* pAudio = GetAudioObjectFromID(VOLUME_ID, lID);
    if (pAudio)
    {
        pAudio->SetNewVolume();
        CheckDefaults();
    }
}
```

```
/* FUNCTION ****************************************************************
 *
 * Name:     OnCommand
 * Notes:    Called when one of the Mute controls changes
 *
 ***************************************************************************/
BOOL CIODlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    if (HIWORD(wParam) == BN_CLICKED)
    {
        LONG    lID = (LONG)LOWORD(wParam);
        CAudio* pAudio = GetAudioObjectFromID(MUTE_ID, lID);
        if (pAudio)
        {
            pAudio->SetNewMute();
            CheckDefaults();
        }
    }
    return CPropertyPage::OnCommand(wParam, lParam);
}

/* FUNCTION ****************************************************************
 *
 * Name:     GetAudioObjectFromID
 *           Find the Audio object, given a ID
 *
 ***************************************************************************/
CAudio* CIODlg::GetAudioObjectFromID(int iType, LONG lID)
{
    BOOL        fMatch = FALSE;
    CAudio*     pAudio;
    POSITION    position = m_AudioList.GetHeadPosition();

    /*
     * loop through each output device
     */
    while(position && !fMatch)
    {
        pAudio = (CAudio*)m_AudioList.GetNext(position);
        switch (iType)
        {
        case BALANCE_ID:
            if (pAudio->m_lBalanceID == lID)
                fMatch = TRUE;
            break;
        case VOLUME_ID:
            if (pAudio->m_lVolumeID == lID)
                fMatch = TRUE;
            break;
        case MUTE_ID:
            if (pAudio->m_lMuteID == lID)
                fMatch = TRUE;
            break;
        case VOLUME_CONTROL_ID:
            if (pAudio->m_dwVolControlID == (DWORD)lID)
                fMatch = TRUE;
            break;
        case MUTE_CONTROL_ID:
            if (pAudio->m_dwMuteControlID == (DWORD)lID)
```

```
                fMatch = TRUE;
                break;
            }
        }
        return(fMatch ? pAudio : NULL);
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnControlChange
 * Notes:   Called by the driver when any control changes
 *
 ***************************************************************************/
LRESULT CIODlg::OnControlChange(WPARAM wParam, LPARAM lParam)
{

    /*
     * Find the control that changed and update the dialog control
     */
    CAudio* pAudio = GetAudioObjectFromID(VOLUME_CONTROL_ID, lParam);
    if (pAudio)
    {
        pAudio->GetVolume();
        CheckDefaults();
    }
    else
    {
        pAudio = GetAudioObjectFromID(MUTE_CONTROL_ID, lParam);
        if (pAudio)
        {
            POSITION    position = m_AudioList.GetHeadPosition();

            /*
             * Because of multiple-select objects there may be more than one
             * mute object affected by this change, so look at all of them
             */
            while(position)
            {
                pAudio = (CAudio*)m_AudioList.GetNext(position);
                if (pAudio->m_dwMuteControlID == (DWORD)lParam)
                    pAudio->GetMute();
            }
        }
        CheckDefaults();
    }
    return(TRUE);
}

/* FUNCTION ****************************************************************
 *
 * Name:    CheckDefaults
 * Notes:   Check the controls we care about to see if they will work with
 *          the microphone
 *
 ***************************************************************************/
BOOL CIODlg::CheckDefaults()

    if (m_iLineType == VOICE_LINE)
        return(TRUE);
```

```
    CAudio*     pAudio;
    POSITION    position = m_AudioList.GetHeadPosition();
    int         nNonDefault = 0;

    /*
     * loop through each device
     */
    while(position)
    {
        pAudio = (CAudio*)m_AudioList.GetNext(position);
        if (m_iLineType == OUTPUT_LINE)
        {
            if (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT
                    || pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_DST_SPEAKERS)
            {
                if (!pAudio->CheckDefaultMute(FALSE))
                    nNonDefault++;
                if (!pAudio->CheckDefaultVolume())
                    nNonDefault++;
            }
            else if (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
            {
                if (!pAudio->CheckDefaultMute(TRUE))
                    nNonDefault++;
            }
        }
        else
        {
            if ((pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
                        || (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_DST_WAVEI
V))
            {
                if (!pAudio->CheckDefaultMute(FALSE))
                    nNonDefault++;
                if (!pAudio->CheckDefaultVolume())
                    nNonDefault++;
            }
        }
    }
    /*
     * Show/hide the controls that warn of problems
     */
    if (nNonDefault)
    {
        m_MuteWarn.ShowWindow(TRUE);
        if (nNonDefault > 1)
        {
            m_MuteWarnText.ShowWindow(FALSE);
            m_MuteWarnText2.ShowWindow(TRUE);
        }
        else
        {
            m_MuteWarnText2.ShowWindow(FALSE);
            m_MuteWarnText.ShowWindow(TRUE);
        }
        return(FALSE);
    }
    else
    {
        m_MuteWarn.ShowWindow(FALSE);
```

```
        m_MuteWarnText.ShowWindow(FALSE);
        m_MuteWarnText2.ShowWindow(FALSE);
        return(TRUE);
    }
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnLineChange
 * Notes:   Called by the driver when the line changes, the only condition
 *          we need to check for is if a meter has stopped or started.
 *
 ***************************************************************************/
LRESULT CIODlg::OnLineChange(WPARAM wParam, LPARAM lParam)
{
    BOOL        fActive = FALSE;
    CAudio*     pAudio;
    POSITION    position = m_MeterList.GetHeadPosition();
    /*
     * loop through each meter device
     */
    while(position)
    {
        pAudio = (CAudio*)m_MeterList.GetNext(position);
        if (pAudio->IsLineActive())
        {
            fActive = TRUE;
        }
    }
    if (fActive)
    {
        /*
         * start the meter timer
         */
        if (!m_uMeterTimerId)
        {
            m_uMeterTimerId = SetTimer(TIMER_METER, VU_METER_TIME, NULL);
            UpdateMeter();
        }
    }
    else
    {
        /*
         * kill the meter timer
         */
        if (m_uMeterTimerId)
        {
            KillTimer(m_uMeterTimerId);
            m_uMeterTimerId = 0;
            UpdateMeter();
        }
    }
    return(TRUE);
}

/* FUNCTION ****************************************************************
 *
 * Name:    FindMeterControls
 * Notes:   Create a list of all VU meter controls
 *
```

```
 ****************************************************************************/
void CIODlg::FindMeterControls()
{
    CAudio*     pAudio;
    POSITION    position = m_AudioList.GetHeadPosition();

    /*
     * loop through each device
     */
    while(position)
    {
        pAudio = (CAudio*)m_AudioList.GetNext(position);
        if (pAudio->m_dwMeterControlID != 0xffffffff)
        {
            m_MeterList.AddTail((CObject*)pAudio);
        }
    }
}
.
/* FUNCTION ******************************************************************
 *
 * Name:    UpdateMeter
 * Notes:   Update all of the meters
 *
 ****************************************************************************/
void CIODlg::UpdateMeter(CDC *pDC, BOOL fForceUpdate)
{
    CDC *pDC2 = (pDC ? pDC : GetDC());
    CAudio*     pAudio;
    POSITION    position = m_MeterList.GetHeadPosition();
    /*
     * loop through each meter device
     */
    while(position)
    {
        pAudio = (CAudio*)m_MeterList.GetNext(position);
        pAudio->UpdateMeter(pDC2, fForceUpdate);
    }
    if (!pDC && pDC2)
        ReleaseDC(pDC2);
}

.
/* FUNCTION ******************************************************************
 *
 * Name:    OnTimer
 * Notes:   The timer went off, time to update the meters
 *
 ****************************************************************************/
void CIODlg::OnTimer(UINT nIDEvent)
{
    if (nIDEvent == m_uMeterTimerId)
    {
        UpdateMeter();
    }

    CPropertyPage::OnTimer(nIDEvent);
}
.
/* FUNCTION ******************************************************************
```

```
 *
 * Name:     OnEraseBkgnd
 * Notes:
 *
 ********************************************************************************/
BOOL CIODlg::OnEraseBkgnd(CDC* pDC)
{
    BOOL    fReturn;
    fReturn = CPropertyPage::OnEraseBkgnd(pDC);
    UpdateMeter(pDC, TRUE);
    return(fReturn);
}

.
/* FUNCTION ************************************************************************
 *
 * Name:     OnDestroy
 * Notes:
 *
 ********************************************************************************/
void CIODlg::OnDestroy()
{
    CPropertyPage::OnDestroy();

    while (!m_AudioList.IsEmpty ())
    {
        delete (CAudio*)m_AudioList.RemoveHead ();
    }
    while (!m_MeterList.IsEmpty ())
    {
        m_MeterList.RemoveHead ();
    }

    if (m_hMixer)
        mixerClose(m_hMixer);
}
```

```
/* SOURCE_FILE ************************************************************
 *
 * File:    MicCtrl.h
 *
 * Owner:   Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ------------------------------------------------------------------------
 *  Leonard 13Nov96     Creation
 *
 *END*********************************************************************/

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

#define MYWM_SHOWYOURSELF   (WM_APP+101)

#define OUTPUT_LINE         0
#define INPUT_LINE          1
#define VOICE_LINE          2

#define FIRST_CARD          0

#define BALANCE_ID          0
#define VOLUME_ID           1
#define MUTE_ID             2
#define VOLUME_CONTROL_ID   3
#define MUTE_CONTROL_ID     4
#define AGC_ID              5
#define AGC_CONTROL_ID      6

/////////////////////////////////////////////////////////////////////////////
// CMicCtrlApp:
// See MicCtrl.cpp for the implementation of this class
//

class CMicCtrlApp : public CWinApp
{
public:
    CMicCtrlApp();

    BOOL    m_fTraySwitch;
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMicCtrlApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation
private:

    //{{AFX_MSG(CMicCtrlApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////
```

```
/* SOURCE_FILE ********************************************************************
 *
 * File:    MicCtrl.cpp
 *
 * Owner:   Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ------------------------------------------------------------------------------
 *  Leonard 25Oct96     Creation
 *
 *END*****************************************************************************/

/*========================== INCLUDES & DEFINES ============================*/

#include "stdafx.h"
#include "MicCtrl.h"
#include "HiddenWindow.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


BEGIN_MESSAGE_MAP(CMicCtrlApp, CWinApp)
    //{{AFX_MSG_MAP(CMicCtrlApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()


/* FUNCTION ***********************************************************************
 *
 * Name:        CMicCtrlApp
 * Title:
 * References:
 * Return:
 * Notes:
 *
 *********************************************************************************/
CMicCtrlApp::CMicCtrlApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CMicCtrlApp object

CMicCtrlApp     theApp;
CHiddenWindow   main;


/* FUNCTION ***********************************************************************
 *
 * Name:        InitInstance
```

```
 * Title:
 * References:
 * Return:
 * Notes:
 *
 ********************************************************************************/
BOOL CMicCtrlApp::InitInstance()
{
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();         // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();   // Call this when linking to MFC statically
#endif
    m_pMainWnd = &main;

    CString sString;
    sString.LoadString(IDS_REG_KEY);
    SetRegistryKey (sString);

    /*
     * Check the command line for the tray switch
     */
    m_fTraySwitch = FALSE;
    if (m_lpCmdLine[0] != '\0')
    {
        CString sString;
        sString.LoadString(IDS_TRAYSWITCH);
        if (sString.CompareNoCase(m_lpCmdLine) == 0)
            m_fTraySwitch = TRUE;
    }

    LPCTSTR lpClass = AfxRegisterWndClass( 0);
    if (!lpClass)
    {
        ASSERT(TRUE);
        return(FALSE);
    }

    HWND hWnd = FindWindow(lpClass, "AudioCOMMANDER");

    if (hWnd)
    {
        /*
         * Notify running application and shutdown
         */

        PostMessage(hWnd, MYWM_SHOWYOURSELF, 0, 0);

        return FALSE;
    }
    else
    {
        /*
         * Create the main window, which is a hidden window
         */
```

```
        if (!main.CreateEx(0, lpClass, "AudioCOMMANDER", WS_BORDER | WS_DLGFRAME, 0,0,0,0,
NULL, NULL))
        {
            ASSERT(TRUE);
            return(FALSE);
        }

        main.WindowInit();

        return TRUE;
    }
}
```

```
// MicCtrlDlg.h : header file
//

class   CAudio;
:lass   COptionsDlg;
class   CMicPropSheet;

/////////////////////////////////////////////////////////////////////////////
// CMicCtrlDlg dialog

class CMicCtrlDlg : public CPropertyPage
{
    DECLARE_DYNCREATE(CMicCtrlDlg)

// Construction
public:
    CMicCtrlDlg();  // standard constructor
    virtual ~CMicCtrlDlg();

// Dialog Data
    //{{AFX_DATA(CMicCtrlDlg)
    enum { IDD = IDD_MICCTRL_DIALOG };
    CStatic m_MuteWarnText2;
    CButton m_AGC;
    CStatic m_VolWarn4;
    CSliderCtrl m_RecVolume;
    CStatic m_RecVolText;
    CStatic m_RecTitle;
    CButton m_RecSelect;
    CStatic m_RecIcon2;
    CStatic m_RecIcon1;
    CSliderCtrl m_RecBalance;
    CStatic m_RecBalText;
    CStatic m_MuteWarn2;
    CStatic m_VolWarn3;
    CStatic m_VolWarn2;
    CStatic m_VolWarn1;
    CStatic m_MuteWarn9;
    CStatic m_MuteWarn8;
    CStatic m_MuteWarn10;
    CStatic m_MuteWarn1;
    CStatic m_MuteWarnText;
    CStatic m_MuteWarn;
    CStatic m_MicIcon2;
    CStatic m_MicIcon1;
    CStatic m_MicVolText;
    CStatic m_MicTitle;
    CStatic m_MicBalText;
    CStatic m_MasterVolText;
    CStatic m_MasterTitle;
    CStatic m_MasterIcon2;
    CStatic m_MasterIcon1;
    CStatic m_MasterBalText;
    CStatic m_WaveVolText;
    CStatic m_WaveTitle;
    CStatic m_WaveIcon2;
    CStatic m_WaveIcon1;
    CStatic m_WaveBalText;
    CButton m_Mute_1;
    CButton m_MicSelect;
```

```
    CButton m_WaveMute;
    CButton m_MasterMute;
    CSliderCtrl m_MasterVolume;
    CSliderCtrl m_MasterBalance;
    CSliderCtrl m_WaveVolume;
    CSliderCtrl m_WaveBalance;
    CSliderCtrl m_MicVolume;
    CSliderCtrl m_MicBalance;
    //}}AFX_DATA

    COptionsDlg *m_pOptionsDlg;

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMicCtrlDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
    //}}AFX_VIRTUAL

// Implementation
private:
    CAudio* CreateAudioObject(  int             iType
                              , MIXERLINE       *pmxLine
                              , CSliderCtrl     *pBalance
                              , CSliderCtrl     *pVolume
                              , CButton         *pMute
                              , CStatic         *pTitle
                              , CStatic         *pBalText
                              , CStatic         *pVolText
                              , CStatic         *pIcon1
                              , CStatic         *pIcon2
                              , CStatic         *pMuteWarn
                              , CStatic         *pVolWarn
                              , CButton         *pAGC
                              , CAudio          *pMasterControl);
    CAudio* GetAudioObjectFromID(int iType, LONG lID);
    BOOL    CheckDefaults();
    void    ErrorMessage(UINT nID, UINT nType);
    void    FindMeterControls();
    void    UpdateMeter(CDC *pDC = 0, BOOL fForceUpdate = FALSE);
public:
    void SetDefaults();

protected:
    CPtrList    m_InputList;
    CPtrList    m_OutputList;
    CPtrList    m_MeterList;
    HMIXER      m_hMixer;
    UINT        m_uMeterTimerId;

    // Generated message map functions
    //{{AFX_MSG(CMicCtrlDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    afx_msg void OnDefaults();
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    afx_msg void OnDestroy();
    //}}AFX_MSG
```

```
    afx_msg LRESULT OnControlChange(WPARAM, LPARAM);
    afx_msg LRESULT OnLineChange(WPARAM, LPARAM);
    DECLARE_MESSAGE_MAP()
};
```

```
/* SOURCE_FILE ********************************************************************
 *
 * File:    MicCtrlDlg.cpp
 *
 * Owner:   Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 * ------------------------------------------------------------------------------
 *  Leonard 21Oct96     Creation
 *
 *END*****************************************************************************/

/*============================ INCLUDES & DEFINES ============================*/

#include "stdafx.h"
#include "MicCtrl.h"
#include "MicCtrlDlg.h"
#include "OptionsDlg.h"
#include "CAudio.h"

#define TIMER_METER     0x4567

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/* FUNCTION ***********************************************************************
 *
 * Name:        CMicCtrlDlg
 * Title:
 * References:
 * Return:
 * Notes:
 *
 *********************************************************************************/
IMPLEMENT_DYNCREATE(CMicCtrlDlg, CPropertyPage)

CMicCtrlDlg::CMicCtrlDlg()
    : CPropertyPage(CMicCtrlDlg::IDD)
{
    //{{AFX_DATA_INIT(CMicCtrlDlg)
    //}}AFX_DATA_INIT
}

CMicCtrlDlg::~CMicCtrlDlg()
{
}

void CMicCtrlDlg::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMicCtrlDlg)
    DDX_Control(pDX, IDC_MUTE_WARN_TEXT2, m_MuteWarnText2);
    DDX_Control(pDX, IDC_AGC, m_AGC);
    DDX_Control(pDX, IDC_VOL_WARN4, m_VolWarn4);
    DDX_Control(pDX, IDC_REC_VOLUME, m_RecVolume);
```

```
    DDX_Control(pDX, IDC_REC_VOL_TEXT, m_RecVolText);
    DDX_Control(pDX, IDC_REC_TITLE, m_RecTitle);
    DDX_Control(pDX, IDC_REC_SELECT, m_RecSelect);
    DDX_Control(pDX, IDC_REC_ICON2, m_RecIcon2);
    DDX_Control(pDX, IDC_REC_ICON1, m_RecIcon1);
    DDX_Control(pDX, IDC_REC_BALANCE, m_RecBalance);
    DDX_Control(pDX, IDC_REC_BAL_TEXT, m_RecBalText);
    DDX_Control(pDX, IDC_MUTE_WARN2, m_MuteWarn2);
    DDX_Control(pDX, IDC_VOL_WARN3, m_VolWarn3);
    DDX_Control(pDX, IDC_VOL_WARN2, m_VolWarn2);
    DDX_Control(pDX, IDC_VOL_WARN1, m_VolWarn1);
    DDX_Control(pDX, IDC_MUTE_WARN9, m_MuteWarn9);
    DDX_Control(pDX, IDC_MUTE_WARN8, m_MuteWarn8);
    DDX_Control(pDX, IDC_MUTE_WARN10, m_MuteWarn10);
    DDX_Control(pDX, IDC_MUTE_WARN1, m_MuteWarn1);
    DDX_Control(pDX, IDC_MUTE_WARN_TEXT, m_MuteWarnText);
    DDX_Control(pDX, IDC_MUTE_WARN, m_MuteWarn);
    DDX_Control(pDX, IDC_MIC_ICON2, m_MicIcon2);
    DDX_Control(pDX, IDC_MIC_ICON1, m_MicIcon1);
    DDX_Control(pDX, IDC_MIC_VOL_TEXT, m_MicVolText);
    DDX_Control(pDX, IDC_MIC_TITLE, m_MicTitle);
    DDX_Control(pDX, IDC_MIC_BAL_TEXT, m_MicBalText);
    DDX_Control(pDX, IDC_MASTER_VOL_TEXT, m_MasterVolText);
    DDX_Control(pDX, IDC_MASTER_TITLE, m_MasterTitle);
    DDX_Control(pDX, IDC_MASTER_ICON2, m_MasterIcon2);
    DDX_Control(pDX, IDC_MASTER_ICON1, m_MasterIcon1);
    DDX_Control(pDX, IDC_MASTER_BAL_TEXT, m_MasterBalText);
    DDX_Control(pDX, IDC_WAVE_VOL_TEXT, m_WaveVolText);
    DDX_Control(pDX, IDC_WAVE_TITLE, m_WaveTitle);
    DDX_Control(pDX, IDC_WAVE_ICON2, m_WaveIcon2);
    DDX_Control(pDX, IDC_WAVE_ICON1, m_WaveIcon1);
    DDX_Control(pDX, IDC_WAVE_BAL_TEXT, m_WaveBalText);
    DDX_Control(pDX, IDC_MUTE_1, m_Mute_1);
    DDX_Control(pDX, IDC_MIC_SELECT, m_MicSelect);
    DDX_Control(pDX, IDC_WAVE_MUTE, m_WaveMute);
    DDX_Control(pDX, IDC_MASTER_MUTE, m_MasterMute);
    DDX_Control(pDX, IDC_MASTER_VOLUME, m_MasterVolume);
    DDX_Control(pDX, IDC_MASTER_BALANCE, m_MasterBalance);
    DDX_Control(pDX, IDC_WAVE_VOLUME, m_WaveVolume);
    DDX_Control(pDX, IDC_WAVE_BALANCE, m_WaveBalance);
    DDX_Control(pDX, IDC_MIC_VOLUME, m_MicVolume);
    DDX_Control(pDX, IDC_MIC_BALANCE, m_MicBalance);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMicCtrlDlg, CPropertyPage)
    //{{AFX_MSG_MAP(CMicCtrlDlg)
    ON_WM_VSCROLL()
    ON_WM_HSCROLL()
    ON_BN_CLICKED(IDC_DEFAULTS, OnDefaults)
    ON_WM_TIMER()
    ON_WM_ERASEBKGND()
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
    ON_MESSAGE(MM_MIXM_CONTROL_CHANGE, OnControlChange)
    ON_MESSAGE(MM_MIXM_LINE_CHANGE, OnLineChange)
END_MESSAGE_MAP()
```

```
/* FUNCTION ****************************************************************
 *
 * Name:    OnInitDialog
 *
 ****************************************************************************/
BOOL CMicCtrlDlg::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    m_hMixer = 0;
    m_uMeterTimerId = 0;

    if (!mixerGetNumDevs())
    {
        ErrorMessage(IDS_NO_CARD, MB_ICONSTOP);
        return(TRUE);
    }

    MIXERCAPS   mxCaps;
    if (mixerGetDevCaps(FIRST_CARD, &mxCaps, sizeof(MIXERCAPS)) != MMSYSERR_NOERROR)
    {
        ASSERT(TRUE);
        return(TRUE);
    }

    MIXERLINE   mxLine;
    CAudio      *pMasterAudio;
    int         iCon, iNumCon;

    /*
     * Get the output (playback) controls that we are interested in
     */
    mxLine.cbStruct = sizeof(MIXERLINE);
    mxLine.dwDestination = OUTPUT_LINE;
    mxLine.dwSource = 0;
    if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_DESTINATION |
MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
    {
        ASSERT(TRUE);
        return(TRUE);
    }
    pMasterAudio = CreateAudioObject(OUTPUT_LINE
                    , &mxLine
                    , &m_MasterBalance
                    , &m_MasterVolume
                    , &m_MasterMute
                    , &m_MasterTitle
                    , &m_MasterBalText
                    , &m_MasterVolText
                    , &m_MasterIcon1
                    , &m_MasterIcon2
                    , &m_MuteWarn8
                    , &m_VolWarn2
                    , NULL
                    , NULL);

    int iMute = 0;

    for (iCon = 0, iNumCon = (int)mxLine.cConnections; iCon < iNumCon; iCon++)
```

```
    {
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination  = OUTPUT_LINE;
        mxLine.dwSource = iCon;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE | M
IXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
            return(TRUE);
        }
        if (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT)
        {
            CreateAudioObject(OUTPUT_LINE
                              , &mxLine
                              , &m_WaveBalance
                              , &m_WaveVolume
                              , &m_WaveMute
                              , &m_WaveTitle
                              , &m_WaveBalText
                              , &m_WaveVolText
                              , &m_WaveIcon1
                              , &m_WaveIcon2
                              , &m_MuteWarn9
                              , &m_VolWarn3
                              , NULL
                              , pMasterAudio);
        }
        else if (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
        {
            CreateAudioObject(OUTPUT_LINE
                              , &mxLine
                              , NULL
                              , NULL
                              , &m_Mute_1
                              , NULL
                              , NULL
                              , NULL
                              , NULL
                              , NULL
                              , &m_MuteWarn10
                              , NULL
                              , NULL
                              , pMasterAudio);
        }
    }

    iMute = 0;

    /*
     * Get the input (recording) controls we are interested in
     */
    if (mxCaps.cDestinations > 1)
    {
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination = INPUT_LINE;
        mxLine.dwSource = 0;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_DESTINATIO
N | MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
```

```
        return(TRUE);
    }
    pMasterAudio = CreateAudioObject(INPUT_LINE
                    , &mxLine
                    , &m_RecBalance
                    , &m_RecVolume
                    , &m_RecSelect
                    , &m_RecTitle
                    , &m_RecBalText
                    , &m_RecVolText
                    , &m_RecIcon1
                    , &m_RecIcon2
                    , &m_MuteWarn2
                    , &m_VolWarn4
                    , NULL
                    , NULL);

    /*
     * At least one of the drivers () falsely reports that 3 of
     * of its four controls are microphones.  This kludge is to look for
     * the one true microphone.
     */
    int iMic = -1;
    iNumCon = (int)mxLine.cConnections;
    for (iCon = 0; iCon < iNumCon; iCon++)
    {
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination  = INPUT_LINE;
        mxLine.dwSource = iCon;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE
| MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
            return(TRUE);
        }
        if (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
        {
            if (iMic == -1)
            {
                /*
                 * default to the first one if all of the string compares fail
                 */
                iMic = iCon;
            }
            if (strstr(mxLine.szName, "Microphone"))
            {
                iMic = iCon;
                break;
            }
        }
    }
    for (iCon = 0; iCon < iNumCon; iCon++)
    {
        mxLine.cbStruct = sizeof(MIXERLINE);
        mxLine.dwDestination  = INPUT_LINE;
        mxLine.dwSource = iCon;
        if (mixerGetLineInfo((HMIXEROBJ)FIRST_CARD, &mxLine, MIXER_GETLINEINFOF_SOURCE
| MIXER_OBJECTF_MIXER) != MMSYSERR_NOERROR)
        {
            ASSERT(TRUE);
```

```
                return(TRUE);
            }
//          if (mxLine.dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
            if (iCon == iMic)
            {
                CreateAudioObject(INPUT_LINE
                                  , &mxLine
                                  , &m_MicBalance
                                  , &m_MicVolume
                                  , &m_MicSelect
                                  , &m_MicTitle
                                  , &m_MicBalText
                                  , &m_MicVolText
                                  , &m_MicIcon1
                                  , &m_MicIcon2
                                  , &m_MuteWarn1
                                  , &m_VolWarn1
                                  , &m_AGC
                                  , pMasterAudio);
            }
        }
    }
    else
    {
        ErrorMessage(IDS_NO_INPUT, MB_ICONSTOP);
    }

    /*
     * Open the mixer so we can get notifications of changes
     */
    mixerOpen(&m_hMixer, FIRST_CARD, (DWORD)m_hWnd, 0, CALLBACK_WINDOW | MIXER_OBJECTF_MIX
ER);

    /*
     * Check for proper control settings, warn if necessary
     */
//  if (!CheckDefaults()
//          && m_pOptionsDlg->m_fWarnStartup)
//      ErrorMessage(IDS_BAD_DEFAULTS, MB_ICONEXCLAMATION);
    CheckDefaults();

    /*
     * Find any VU meters
     */
    FindMeterControls();
    /*
     * Check to see if the meters should be activated
     */
    OnLineChange(0, 0);

    return TRUE;
}


/* FUNCTION ***************************************************************
 *
 * Name:    CreateAudioObject
 *
 ***************************************************************************/
```

```
CAudio* CMicCtrlDlg::CreateAudioObject(  int     iType
                                , MIXERLINE     *pmxLine
                                , CSliderCtrl   *pBalance
                                , CSliderCtrl   *pVolume
                                , CButton       *pMute
                                , CStatic       *pTitle
                                , CStatic       *pBalText
                                , CStatic       *pVolText
                                , CStatic       *pIcon1
                                , CStatic       *pIcon2
                                , CStatic       *pMuteWarn
                                , CStatic       *pVolWarn
                                , CButton       *pAGC
                                , CAudio        *pMasterControl)
{
    CAudio *    pAudio = new CAudio;

    pAudio->m_iType             = iType;
    pAudio->m_dwSource          = pmxLine->dwSource;
    pAudio->m_dwLineID          = pmxLine->dwLineID;
    pAudio->m_dwComponentType   = pmxLine->dwComponentType;
    pAudio->m_cChannels         = pmxLine->cChannels;
    pAudio->m_sShortName        = pmxLine->szShortName;
    pAudio->m_sName             = pmxLine->szName;
    pAudio->m_pBalance          = pBalance;
    pAudio->m_pVolume           = pVolume;
    pAudio->m_pMute             = pMute;
    pAudio->m_pTitle            = pTitle;
    pAudio->m_pBalText          = pBalText;
    pAudio->m_pVolText          = pVolText;
    pAudio->m_pIcon1            = pIcon1;
    pAudio->m_pIcon2            = pIcon2;
    pAudio->m_pMuteWarn         = pMuteWarn;
    pAudio->m_pVolWarn          = pVolWarn;
    pAudio->m_pAGC              = pAGC;
    pAudio->m_pMasterControl    = pMasterControl;

    if (pBalance)
        pAudio->m_lBalanceID = pBalance->GetDlgCtrlID();
    if (pVolume)
        pAudio->m_lVolumeID = pVolume->GetDlgCtrlID();
    if (pMute)
        pAudio->m_lMuteID = pMute->GetDlgCtrlID();
    if (pAGC)
        pAudio->m_lAGCID = pAGC->GetDlgCtrlID();

    pAudio->InitControls();

    if (iType == OUTPUT_LINE)
        m_OutputList.AddTail((CObject*)pAudio);
    else
        m_InputList.AddTail((CObject*)pAudio);

    return(pAudio);
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnVScroll
 * Notes:   Called when one of the Volume controls changes
```

```
 *
 ****************************************************************************/
void CMicCtrlDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CPropertyPage::OnVScroll(nSBCode, nPos, pScrollBar);

    LONG    lID = GetWindowLong(pScrollBar->m_hWnd, GWL_ID);
    CAudio* pAudio = GetAudioObjectFromID(VOLUME_ID, lID);
    if (pAudio)
    {
        pAudio->SetNewVolume();
        CheckDefaults();
    }
}

/* FUNCTION ****************************************************************
 *
 * Name:    GetAudioObjectFromID
 * Notes:   Find the Audio object, given a ID
 *
 ****************************************************************************/
CAudio* CMicCtrlDlg::GetAudioObjectFromID(int iType, LONG lID)
{
    BOOL        fMatch = FALSE;
    CAudio*     pAudio;
    POSITION    position = m_OutputList.GetHeadPosition();

    /*
     * loop through each output device
     */
    while(position && !fMatch)
    {
        pAudio = (CAudio*)m_OutputList.GetNext(position);
        switch (iType)
        {
        case BALANCE_ID:
            if (pAudio->m_lBalanceID == lID)
                fMatch = TRUE;
            break;
        case VOLUME_ID:
            if (pAudio->m_lVolumeID == lID)
                fMatch = TRUE;
            break;
        case MUTE_ID:
            if (pAudio->m_lMuteID == lID)
                fMatch = TRUE;
            break;
        case AGC_ID:
            if (pAudio->m_lAGCID == lID)
                fMatch = TRUE;
            break;
        case VOLUME_CONTROL_ID:
            if (pAudio->m_dwVolControlID == (DWORD)lID)
                fMatch = TRUE;
            break;
        case MUTE_CONTROL_ID:
            if (pAudio->m_dwMuteControlID == (DWORD)lID)
                fMatch = TRUE;
            break;
```

```
        case AGC_CONTROL_ID:
            if (pAudio->m_dwAGCControlID == (DWORD)lID)
                fMatch = TRUE;
            break;
        }
    }
    if (!fMatch)
    {
        position = m_InputList.GetHeadPosition();

        /*
         * loop through each input device
         */
        while(position && !fMatch)
        {
            pAudio = (CAudio*)m_InputList.GetNext(position);
            switch (iType)
            {
            case BALANCE_ID:
                if (pAudio->m_lBalanceID == lID)
                    fMatch = TRUE;
                break;
            case VOLUME_ID:
                if (pAudio->m_lVolumeID == lID)
                    fMatch = TRUE;
                break;
            case MUTE_ID:
                if (pAudio->m_lMuteID == lID)
                    fMatch = TRUE;
                break;
            case AGC_ID:
                if (pAudio->m_lAGCID == lID)
                    fMatch = TRUE;
                break;
            case VOLUME_CONTROL_ID:
                if (pAudio->m_dwVolControlID == (DWORD)lID)
                    fMatch = TRUE;
                break;
            case MUTE_CONTROL_ID:
                if (pAudio->m_dwMuteControlID == (DWORD)lID)
                    fMatch = TRUE;
                break;
            case AGC_CONTROL_ID:
                if (pAudio->m_dwAGCControlID == (DWORD)lID)
                    fMatch = TRUE;
                break;
            }
        }
    }
    return(fMatch ? pAudio : NULL);
}

/* FUNCTION ********************************************************************
 *
 * Name:    OnHScroll
 * Notes:   Called when one of the Balance controls changes
 *
 ********************************************************************************/
void CMicCtrlDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
```

```
    CPropertyPage::OnHScroll(nSBCode, nPos, pScrollBar);

    LONG    lID = GetWindowLong(pScrollBar->m_hWnd, GWL_ID);
    CAudio* pAudio = GetAudioObjectFromID(BALANCE_ID, lID);
    if (pAudio)
    {
        pAudio->SetNewVolume();
    }
}

.
/* FUNCTION ****************************************************************
 *
 * Name:    OnCommand
 * Notes:   Called when one of the Mute controls changes
 *
 ***************************************************************************/
BOOL CMicCtrlDlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    if (HIWORD(wParam) == BN_CLICKED)
    {
        LONG    lID = (LONG)LOWORD(wParam);
        CAudio* pAudio = GetAudioObjectFromID(MUTE_ID, lID);
        if (pAudio)
        {
            pAudio->SetNewMute();
            CheckDefaults();
        }
        else
        {
            pAudio = GetAudioObjectFromID(AGC_ID, lID);
            if (pAudio)
            {
                pAudio->SetNewAGC();
            }
        }
    }
    return CPropertyPage::OnCommand(wParam, lParam);
}
.
/* FUNCTION ****************************************************************
 *
 * Name:    OnControlChange
 * Notes:   Called by the driver when any control changes
 *
 ***************************************************************************/
LRESULT CMicCtrlDlg::OnControlChange(WPARAM wParam, LPARAM lParam)
{

    /*
     * Find the control that changed and update the dialog control
     */
    CAudio* pAudio = GetAudioObjectFromID(VOLUME_CONTROL_ID, lParam);
    if (pAudio)
    {
        pAudio->GetVolume();
        CheckDefaults();
    }
    else
    {
```

```
        pAudio = GetAudioObjectFromID(MUTE_CONTROL_ID, lParam);
        if (pAudio)
        {
            POSITION    position = m_OutputList.GetHeadPosition();

            /*
             * Because of multiple-select objects there may be more than one
             * mute object affected by this change, so look at all of them
             */
            while(position)
            {
                pAudio = (CAudio*)m_OutputList.GetNext(position);
                if (pAudio->m_dwMuteControlID == (DWORD)lParam)
                    pAudio->GetMute();
            }

            position = m_InputList.GetHeadPosition();

            /*
             * Because of multiple-select objects there may be more than one
             * mute object affected by this change, so look at all of them
             */
            while(position)
            {
                pAudio = (CAudio*)m_InputList.GetNext(position);
                if (pAudio->m_dwMuteControlID == (DWORD)lParam)
                    pAudio->GetMute();
            }
            CheckDefaults();
        }
        else
        {
            pAudio = GetAudioObjectFromID(AGC_CONTROL_ID, lParam);
            if (pAudio)
            {
                pAudio->GetAGC();
                CheckDefaults();
            }
        }
    }
    return(TRUE);
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnLineChange
 * Notes:   Called by the driver when the line changes, the only condition
 *          we need to check for is if a meter has stopped or started.
 *
 ***************************************************************************/
LRESULT CMicCtrlDlg::OnLineChange(WPARAM wParam, LPARAM lParam)
{
    BOOL        fActive = FALSE;
    CAudio*     pAudio;
    POSITION    position = m_MeterList.GetHeadPosition();
    /*
     * loop through each meter device
     */
    while(position)
    {
```

```
        pAudio = (CAudio*)m_MeterList.GetNext(position);
        if (pAudio->IsLineActive())
        {
            fActive = TRUE;
        }
    }
    if (fActive)
    {
        /*
         * start the meter timer
         */
        if (!m_uMeterTimerId)
        {
            m_uMeterTimerId = SetTimer(TIMER_METER, VU_METER_TIME, NULL);
            UpdateMeter();
        }
    }
    else
    {
        /*
         * kill the meter timer
         */
        if (m_uMeterTimerId)
        {
            KillTimer(m_uMeterTimerId);
            m_uMeterTimerId = 0;
            UpdateMeter();
        }
    }
    return(TRUE);
}

/* FUNCTION ********************************************************************
 *
 * Name:    OnDefaults
 * Notes:   Set the defaults on the controls
 *
 ********************************************************************************/
void CMicCtrlDlg::OnDefaults()
{
    CAudio*     pAudio;
    POSITION    position = m_OutputList.GetHeadPosition();

    /*
     * loop through each output device
     */
    while(position)
    {
        pAudio = (CAudio*)m_OutputList.GetNext(position);
        pAudio->SetDefaultVolume();
        if (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT
                || pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_DST_SPEAKERS)
        {
            pAudio->SetDefaultMute(FALSE);
        }
        else if (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
        {
            pAudio->SetDefaultMute(TRUE);
        }
```

```
    }
    position = m_InputList.GetHeadPosition();

    /*
     * loop through each input device
     */
    while(position)
    {
        pAudio = (CAudio*)m_InputList.GetNext(position);
        pAudio->SetDefaultVolume();
        if ((pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
                || (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_DST_WAVEIN))
        {
            pAudio->SetDefaultMute(FALSE);
        }
    }
}

/* FUNCTION ************************************************************
 *
 * Name:    CheckDefaults
 * Notes:   Check the controls we care about to see if they will work with
 *          the microphone
 *
 ************************************************************************/
BOOL CMicCtrlDlg::CheckDefaults()
{
    CAudio*     pAudio;
    POSITION    position = m_OutputList.GetHeadPosition();
    int         nNonDefault = 0;

    /*
     * loop through each output device
     */
    while(position)
    {
        pAudio = (CAudio*)m_OutputList.GetNext(position);
        if (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT
                || pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_DST_SPEAKERS)
        {
            if (!pAudio->CheckDefaultMute(FALSE))
                nNonDefault++;
            if (!pAudio->CheckDefaultVolume())
                nNonDefault++;
        }
        else if (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
        {
            if (!pAudio->CheckDefaultMute(TRUE))
                nNonDefault++;
        }

    }
    position = m_InputList.GetHeadPosition();

    /*
     * loop through each input device
     */
    while(position)
    {
```

```
        pAudio = (CAudio*)m_InputList.GetNext(position);
        if ((pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE)
                    || (pAudio->m_dwComponentType == MIXERLINE_COMPONENTTYPE_DST_WAVEIN))
        {
            if (!pAudio->CheckDefaultMute(FALSE))
                nNonDefault++;
            if (!pAudio->CheckDefaultVolume())
                nNonDefault++;
        }
    }
    /*
     * Show/hide the controls that warn of problems
     */
    if (nNonDefault)
    {
        m_MuteWarn.ShowWindow(TRUE);
        if (nNonDefault > 1)
        {
            m_MuteWarnText.ShowWindow(FALSE);
            m_MuteWarnText2.ShowWindow(TRUE);
        }
        else
        {
            m_MuteWarnText2.ShowWindow(FALSE);
            m_MuteWarnText.ShowWindow(TRUE);
        }
        return(FALSE);
    }
    else
    {
        m_MuteWarn.ShowWindow(FALSE);
        m_MuteWarnText.ShowWindow(FALSE);
        m_MuteWarnText2.ShowWindow(FALSE);
        return(TRUE);

    }
}
.
/* FUNCTION ****************************************************************
 *
 * Name:    ErrorMessage
 *
 ****************************************************************************/
void CMicCtrlDlg::ErrorMessage(UINT nID, UINT nType)
{
        CString sMessage;
        sMessage.LoadString(nID);
        CString sCaption;
        sCaption.LoadString(IDS_ERROR_CAPTION);
        MessageBox(sMessage, sCaption, nType);

}
.
/* FUNCTION ****************************************************************
 *
 * Name:    SetDefaults
 *
 ****************************************************************************/
void CMicCtrlDlg::SetDefaults()
{
```

```
    OnDefaults();
    CheckDefaults();
}

/* FUNCTION ****************************************************************
 *
 * Name:     FindMeterControls
 * Notes:    Create a list of all VU meter controls
 *
 ***************************************************************************/
void CMicCtrlDlg::FindMeterControls()
{
    CAudio*     pAudio;
    POSITION    position = m_OutputList.GetHeadPosition();

    /*
     * loop through each output device
     */
    while(position)
    {
        pAudio = (CAudio*)m_OutputList.GetNext(position);
        if (pAudio->m_dwMeterControlID != 0xffffffff)
        {
            m_MeterList.AddTail((CObject*)pAudio);
        }
    }

    /*
     * loop through each input device
     */
    position = m_InputList.GetHeadPosition();
    while(position)
    {
        pAudio = (CAudio*)m_InputList.GetNext(position);
        if (pAudio->m_dwMeterControlID != 0xffffffff)
        {
            m_MeterList.AddTail((CObject*)pAudio);
        }
    }
}

/* FUNCTION ****************************************************************
 *
 * Name:     OnTimer
 * Notes:    The timer went off, time to update the meters
 *
 ***************************************************************************/
void CMicCtrlDlg::OnTimer(UINT nIDEvent)
{
    if (nIDEvent == m_uMeterTimerId)
    {
        UpdateMeter();
    }

    CPropertyPage::OnTimer(nIDEvent);
}

/* FUNCTION ****************************************************************
 *
```

```
 * Name:    UpdateMeter
 * Notes:   Update all of the meters
 *
 ********************************************************************************/
void CMicCtrlDlg::UpdateMeter(CDC *pDC, BOOL fForceUpdate)
{
    CDC *pDC2 = (pDC ? pDC : GetDC());
    CAudio*     pAudio;
    POSITION    position = m_MeterList.GetHeadPosition();
    /*
     * loop through each meter device
     */
    while(position)
    {
        pAudio = (CAudio*)m_MeterList.GetNext(position);
        pAudio->UpdateMeter(pDC2, fForceUpdate);
    }
    if (!pDC && pDC2)
        ReleaseDC(pDC2);
}


/* FUNCTION ********************************************************************
 *
 * Name:    OnEraseBkgnd
 * Notes:
 *
 ********************************************************************************/
BOOL CMicCtrlDlg::OnEraseBkgnd(CDC* pDC)
{
    BOOL    fReturn;
    fReturn = CPropertyPage::OnEraseBkgnd(pDC);
    UpdateMeter(pDC, TRUE);
    return(fReturn);
}


/* FUNCTION ********************************************************************
 *
 * Name:    OnDestroy
 * Notes:
 *
 ********************************************************************************/
void CMicCtrlDlg::OnDestroy()
{
    CPropertyPage::OnDestroy();

    while (!m_InputList.IsEmpty ())
    {
        delete (CAudio*)m_InputList.RemoveHead ();
    }
    while (!m_OutputList.IsEmpty ())
    {
        delete (CAudio*)m_OutputList.RemoveHead ();
    }
    while (!m_MeterList.IsEmpty ())
    {
        m_MeterList.RemoveHead ();
    }
```

```
    if (m_hMixer)
        mixerClose(m_hMixer);
}
```

```
// MicPropSheet.h : header file
//
class   CMicCtrlDlg;
class   COptionsDlg;
class   CIODlg;
class   CHiddenWindow;

/////////////////////////////////////////////////////////////////////////////
// CMicPropSheet

class CMicPropSheet : public CPropertySheet
{
    DECLARE_DYNAMIC(CMicPropSheet)

// Construction
public:
    CMicPropSheet(UINT nIDCaption, CWnd* pParentWnd = NULL, UINT iSelectPage = 0);
    CMicPropSheet(LPCTSTR pszCaption, CWnd* pParentWnd = NULL, UINT iSelectPage = 0);

// Attributes
private:
    CWnd        *m_pCardName;

public:
    CMicCtrlDlg *m_pMicDlg;
    CIODlg      *m_pRecordingDlg;
    CIODlg      *m_pPlaybackDlg;
//  CIODlg      *m_pVoiceDlg;
    COptionsDlg *m_pOptionsDlg;
    CHiddenWindow   *m_pHiddenWindow;
    int         m_iRecordingPage;
    int         m_iPlaybackPage;
//  int         m_iVoicePage;

// Operations
private:

public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMicPropSheet)
    public:
    virtual BOOL OnInitDialog();
    protected:
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMicPropSheet();
    void    UpdateTabs();

    // Generated message map functions
protected:
    //{{AFX_MSG(CMicPropSheet)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnDestroy();
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

.
class CAboutDlg : public CDialog
{
public:
    CAboutDlg(CWnd* pParent = NULL);

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    CListBox    m_PhoneList;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
////////////////////////////////////////////////////////////////////////////
```

```
/* SOURCE_FILE ************************************************..***************
 *
 * File:     MicPropSheet.cpp
 *
 * Owner:    Leonard
 *
 *  Copyright (C) 1996, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ---------------------------------------------------------------------------
 *  Leonard 26Oct96     Creation
 *
 *END*************************************************************************/

/*========================== INCLUDES & DEFINES ============================*/

#include "stdafx.h"
#include "MicCtrl.h"
#include "MicCtrlDlg.h"
#include "OptionsDlg.h"
#include "IODlg.h"
#include "MicPropSheet.h"
#include "HiddenWindow.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/* FUNCTION ***************************************************************
 *
 * Name:     CAboutDlg
 * Notes:
 *
 *************************************************************************/
CAboutDlg::CAboutDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CAboutDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    DDX_Control(pDX, IDC_PHONELIST, m_PhoneList);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/* FUNCTION ***************************************************************
 *
```

```
 * Name:    OnInitDialog
 * Notes:
 *
 ********************************************************************************/
BOOL CAboutDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    int     iPhoneID[] =
                        {
                            IDS_PHONE_1,
                            IDS_PHONE_2,
                            IDS_PHONE_3,
                            IDS_PHONE_4,
                            IDS_PHONE_5,
                            IDS_PHONE_6,
                            IDS_PHONE_7,
                            IDS_PHONE_8,
                            IDS_PHONE_9,
                            IDS_PHONE_10,
                            IDS_PHONE_11,
                            IDS_PHONE_12,
                            IDS_PHONE_13,
                            IDS_PHONE_14,
                            IDS_PHONE_15,
                            IDS_PHONE_16,
                            IDS_PHONE_17
                        };
    CString sPhone;
    m_PhoneList.SetTabStops(100);
    for (int i = 0; i < sizeof(iPhoneID)/sizeof(iPhoneID[0]); i++)
    {
        sPhone.LoadString(iPhoneID[i]);
        m_PhoneList.AddString(sPhone);
    }

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
.
/* FUNCTION ********************************************************************
 *
 * Name:    CMicPropSheet
 * Notes:
 *
 ********************************************************************************/
IMPLEMENT_DYNAMIC(CMicPropSheet, CPropertySheet)

CMicPropSheet::CMicPropSheet(UINT nIDCaption, CWnd* pParentWnd, UINT iSelectPage)
    :CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
{
}

CMicPropSheet::CMicPropSheet(LPCTSTR pszCaption, CWnd* pParentWnd, UINT iSelectPage)
    :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
{
    m_pCardName = 0;
    m_pHiddenWindow = 0;
}
```

```
CMicPropSheet::~CMicPropSheet()
{
}

BEGIN_MESSAGE_MAP(CMicPropSheet, CPropertySheet)
    //{{AFX_MSG_MAP(CMicPropSheet)
    ON_WM_CREATE()
    ON_WM_SYSCOMMAND()
    ON_WM_DESTROY()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

.
/* FUNCTION ****************************************************************
 *
 * Name:    OnCreate
 * Notes:
 *
 ****************************************************************************/
int CMicPropSheet::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CPropertySheet::OnCreate(lpCreateStruct) == -1)
        return -1;

    /*
     * Set the proper Icon
     */
    HICON   hIcon;
    hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    ::SetClassLong(m_hWnd, GCL_HICON, (LONG)hIcon);

    /*
     * Add the minimize button to the window
     */
    LONG    lStyle;
    lStyle = ::GetWindowLong(m_hWnd, GWL_STYLE);
    lStyle |= WS_MINIMIZEBOX | WS_SYSMENU;
    ::SetWindowLong(m_hWnd, GWL_STYLE, lStyle);

    /*****************************************************************/
    /*                                                               */
    /* Cleanup the system menu                                       */
    /*                                                               */
    /*****************************************************************/

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    CString strMenu;

    /*
     * Remove "Maximize from the system menu
     */
    pSysMenu->RemoveMenu(SC_MAXIMIZE, MF_BYCOMMAND);

    /*
     * Add "About" to the system menu
     */
    strMenu.LoadString(IDS_ABOUTBOX);
    if (!strMenu.IsEmpty())
    {
```

```
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strMenu);
    }

    return 0;
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnInitDialog
 * Notes:
 *
 ***************************************************************************/
BOOL CMicPropSheet::OnInitDialog()
{
    /*
     * turn of modeless flag so OnInitDialog will create the default buttons
     * at the bottom of the property sheet
     */
    BOOL    fModeless = m_bModeless;
    m_bModeless = FALSE;
    BOOL    fReturn = CPropertySheet::OnInitDialog();
    m_bModeless = fModeless;
    CString sString;

    /*
     * Change tab text
     */
    UpdateTabs();

    /*
     * Hide and disable the "Cancel" and "Apply" buttons.
     * Move the "OK" button to where the "Apply" button was and rename it
     * to "Close".
     */
    CWnd *  pItem;
    WINDOWPLACEMENT wp;
    BOOL    fMove = FALSE;

    pItem = GetDlgItem(ID_APPLY_NOW);
    if (pItem)
    {
        wp.length = sizeof(WINDOWPLACEMENT);
        if (pItem->GetWindowPlacement(&wp))
            fMove = TRUE;

        pItem->EnableWindow(FALSE);
        pItem->ShowWindow(SW_HIDE);
    }
    pItem = GetDlgItem(IDCANCEL);
    if (pItem)
    {
        pItem->EnableWindow(FALSE);
        pItem->ShowWindow(SW_HIDE);
    }
    pItem = GetDlgItem(IDOK);
    if (pItem)
    {
        if (fMove)
```

```
            pItem->MoveWindow(&wp.rcNormalPosition, FALSE);

        sString.LoadString(IDS_CLOSE_BUTTON);
        pItem->SetWindowText(sString);
    }
    pItem = GetDlgItem(IDHELP);
    if (pItem)
    {
        sString.LoadString(IDS_HELP_BUTTON);
        pItem->SetWindowText(sString);
    }

    /*
     * Add a new controls
     */
    if (fMove)
    {
        HWND    hWnd;
        CFont   *pFont = GetFont();

        /*
         * Add card name display area
         */
        hWnd = ::CreateWindow(_T("static"), "",
            WS_CHILD|WS_VISIBLE,
            0, 0, 0, 0, m_hWnd, (HMENU)IDC_CARD_NAME,
            AfxGetInstanceHandle(), NULL);

        m_pCardName = GetDlgItem(IDC_CARD_NAME);
        if (m_pCardName)
        {
            wp.rcNormalPosition.top += 4;
            wp.rcNormalPosition.bottom -= 4;
            wp.rcNormalPosition.right = wp.rcNormalPosition.left - 8;
            wp.rcNormalPosition.left = 20;
            m_pCardName->MoveWindow(&wp.rcNormalPosition, FALSE);
        }
        if (pFont)
            m_pCardName->SetFont(pFont, FALSE);
    }

    /*
     * Display the sound driver name
     */
    if (m_pCardName)
    {
        MIXERCAPS   mxCaps;
        if (mixerGetDevCaps(FIRST_CARD, &mxCaps, sizeof(MIXERCAPS)) == MMSYSERR_NOERROR)
        {
            m_pCardName->SetWindowText(mxCaps.szPname);
        }
    }

    return fReturn;
}

/* FUNCTION *************************************************************
 *
 * Name:    OnSysCommand
```

```
 * Notes:
 *
 ********************************************************************************/
void CMicPropSheet::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout(this);
        dlgAbout.DoModal();
    }
    else
    {
        CPropertySheet::OnSysCommand(nID, lParam);
    }
}

/* FUNCTION ************************************************************************
 *
 * Name:    UpdateTabs
 * Notes:
 *
 ********************************************************************************/
void CMicPropSheet::UpdateTabs()
{
    BOOL        fOption = TRUE;
    int         i;
    int         iPageCnt = GetPageCount();
    TC_ITEM     item;
    item.mask = TCIF_TEXT;
    CString     sString;

    CTabCtrl*   pTab = GetTabControl();

    /*
     * If any of the pages have been added or removed, remove all and start over
     */
    if ((m_pOptionsDlg->m_fRecording && m_iRecordingPage == -1)
            || (!m_pOptionsDlg->m_fRecording && m_iRecordingPage > 0)
            || (m_pOptionsDlg->m_fPlayback && m_iPlaybackPage == -1)
            || (!m_pOptionsDlg->m_fPlayback && m_iPlaybackPage > 0))
//          || (m_pOptionsDlg->m_fVoice && m_iVoicePage == -1)
//          || (!m_pOptionsDlg->m_fVoice && m_iVoicePage > 0))
    {
        for (i = 1; i < iPageCnt; i++)
        {
            RemovePage(1);
        }
        iPageCnt = 1;
        /*
         * Add Recording page
         */
        if (m_pOptionsDlg->m_fRecording)
        {
            AddPage(m_pRecordingDlg);
            m_iRecordingPage = iPageCnt++;
        }
        else
        {
            m_iRecordingPage = -1;
        }
```

```
        /*
         * Add Playback page
         */
        if (m_pOptionsDlg->m_fPlayback)
        {
            AddPage(m_pPlaybackDlg);
            m_iPlaybackPage = iPageCnt++;
        }
        else
        {
            m_iPlaybackPage = -1;
        }
//      /*
//       * Add Voice Commands page
//       */
//      if (m_pOptionsDlg->m_fVoice)
//      {
//          AddPage(m_pVoiceDlg);
//          m_iVoicePage = iPageCnt;
//      }
//      else
//      {
//          m_iVoicePage = -1;
//      }
        /*
         * Add options page
         */
        AddPage(m_pOptionsDlg);
        SetActivePage(GetPageCount()-1);
    }
    /*
     * Make sure the right text is on the tabs
     */
    if (pTab)
    {
        if (m_pOptionsDlg->m_fRecording)
        {
            sString.LoadString(IDS_RECORDING_TAB);
            item.pszText = (LPSTR)(LPCSTR)sString;
            pTab->SetItem(m_iRecordingPage, &item);
        }
        if (m_pOptionsDlg->m_fPlayback)
        {
            sString.LoadString(IDS_PLAYBACK_TAB);
            item.pszText = (LPSTR)(LPCSTR)sString;
            pTab->SetItem(m_iPlaybackPage, &item);
        }
//      if (m_pOptionsDlg->m_fVoice)
//      {
//          sString.LoadString(IDS_VOICE_TAB);
//          item.pszText = (LPSTR)(LPCSTR)sString;
//          pTab->SetItem(m_iVoicePage, &item);
//      }
    }

//  sString.LoadString(GetPageCount() > 2 ? IDS_TITLE2 : IDS_TITLE);
//  SetTitle(sString);
}
```

```
/* FUNCTION ****************************************************************
 *
 * Name:    OnDestroy
 * Notes:
 *
 ***************************************************************************/
void CMicPropSheet::OnDestroy()
{
    CPropertySheet::OnDestroy();

    m_pHiddenWindow->PropSheetDestroyed();
}

/* FUNCTION ****************************************************************
 *
 * Name:    OnCommand
 * Notes:
 *
 ***************************************************************************/
BOOL CMicPropSheet::OnCommand(WPARAM wParam, LPARAM lParam)
{

    if (LOWORD(wParam) == IDOK)
    {
        EndDialog(IDOK);
        return TRUE;
    }
    else if (LOWORD(wParam) == IDHELP)
    {
//      WinHelp(0, HELP_CONTENTS);
        /*
         * Show "About" until there is a real help file
         */
        CAboutDlg dlgAbout(this);
        dlgAbout.DoModal();
        return TRUE;
    }
    return CPropertySheet::OnCommand(wParam, lParam);
}
```

```
// OptionsDlg.h : header file
//

class   CMicPropSheet;
class   CHiddenWindow;

/////////////////////////////////////////////////////////////////////////////
// COptionsDlg dialog

class COptionsDlg : public CPropertyPage
{
    DECLARE_DYNCREATE(COptionsDlg)

// Construction
public:
    COptionsDlg();   // standard constructor
    virtual ~COptionsDlg();

    CMicPropSheet   *m_pPropSheet;
    CHiddenWindow   *m_pHiddenWindow;

// Dialog Data
    //{{AFX_DATA(COptionsDlg)
    enum { IDD = IDD_OPTIONS_DIALOG };
    BOOL    m_fTray;
    BOOL    m_fPlayback;
    BOOL    m_fRecording;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(COptionsDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(COptionsDlg)
    afx_msg void OnDestroy();
    afx_msg void OnTray();
    afx_msg void OnPlayback();
    afx_msg void OnRecord();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

```
/* SOURCE_FILE ********************************************************************
 *
 * File:    OptionsDlg.cpp
 *
 * Title:
 *
 * Owner:   Leonard
 *
 *  Copyright (C) 1996 Novell, Inc, All Rights Reserved
 *  COMPANY CONFIDENTIAL -- NOT FOR DISTRIBUTION
 * ------------------------------------------------------------------------------
 * <Description>
 *
 * Revision history (Most recent change first):
 *  $Log$
 *
 *  Leonard 01Nov96     Creation
 *
 *END*****************************************************************************/

/*========================== INCLUDES & DEFINES ============================*/

#include "stdafx.h"
#include "MicCtrl.h"
#include "OptionsDlg.h"
#include "MicPropSheet.h"
#include "HiddenWindow.h"

static const char szSettingsSection[]   = "Settings";
static const char szRecordingKey[]      = "ShowRecordingControls";
static const char szPlaybackKey[]       = "ShowPlaybackControls";
// static const char szVoiceKey[]           = "ShowVoiceControls";

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/* FUNCTION ***********************************************************************
 *
 * Name:        COptionsDlg
 * Title:
 * References:
 * Return:
 * Notes:
 *
 *********************************************************************************/
IMPLEMENT_DYNCREATE(COptionsDlg, CPropertyPage)

COptionsDlg::COptionsDlg()
    : CPropertyPage(COptionsDlg::IDD)
{
    //{{AFX_DATA_INIT(COptionsDlg)
    m_fTray = FALSE;
    m_fPlayback = FALSE;
    m_fRecording = FALSE;
    //}}AFX_DATA_INIT
```

```
    /*
     * Get registry information
     */
    CMicCtrlApp*    pApp = (CMicCtrlApp*)AfxGetApp();
    m_fTray = pApp->m_fTraySwitch;
    m_fRecording = pApp->GetProfileInt(szSettingsSection, szRecordingKey, FALSE);
    m_fPlayback = pApp->GetProfileInt(szSettingsSection, szPlaybackKey, FALSE);
//  m_fVoice = pApp->GetProfileInt(szSettingsSection, szVoiceKey, FALSE);
}

COptionsDlg::~COptionsDlg()
{

}

void COptionsDlg::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(COptionsDlg)
    DDX_Check(pDX, IDC_TRAY, m_fTray);
    DDX_Check(pDX, IDC_PLAYBACK, m_fPlayback);
    DDX_Check(pDX, IDC_RECORD, m_fRecording);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(COptionsDlg, CPropertyPage)
    //{{AFX_MSG_MAP(COptionsDlg)
    ON_WM_DESTROY()
    ON_BN_CLICKED(IDC_TRAY, OnTray)
    ON_BN_CLICKED(IDC_PLAYBACK, OnPlayback)
    ON_BN_CLICKED(IDC_RECORD, OnRecord)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

.
/* FUNCTION ****************************************************************
 *
 * Name:        OnDestroy
 * Title:
 * References:
 * Return:
 * Notes:
 *
 ****************************************************************************/
void COptionsDlg::OnDestroy()
{
    UpdateData(TRUE);

    CPropertyPage::OnDestroy();
}

.
/* FUNCTION ****************************************************************
 *
 * Name:        OnTray
 * Title:
 * References:
 * Return:
 * Notes:
```

```
 *
 ********************************************************************************/
void COptionsDlg::OnTray()
{
    UpdateData(TRUE);
    m_pHiddenWindow->UpdateTray();
    if (m_fTray)
    {
        /*
         * Add to the "Run" key in the registry
         */
        HKEY    hSubKey;
        if (RegOpenKeyEx(HKEY_LOCAL_MACHINE
                    , "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
                    , NULL
                    , KEY_WRITE
                    , &hSubKey) == ERROR_SUCCESS)
        {
            char    szBuf[_MAX_FNAME * 2];
            HMODULE hMod;

            hMod = GetModuleHandle(NULL);
            if (hMod)
            {
                if (GetModuleFileName(hMod, szBuf, sizeof(szBuf)))
                {
                    CString sString;
                    sString.LoadString(IDS_TRAYSWITCH);
                    strcat(szBuf, " ");
                    strcat(szBuf, (LPSTR)(LPCSTR)sString);
                    RegSetValueEx(hSubKey
                                , "AudioCOMMANDER"
                                , NULL
                                , REG_SZ
                                , (LPBYTE)szBuf
                                , strlen(szBuf));
                    RegCloseKey(hSubKey);
                }
            }
        }
    }
    else
    {
        /*
         * Remove from the "Run" key in the registry
         */
        HKEY    hSubKey;
        if (RegOpenKeyEx(HKEY_LOCAL_MACHINE
                    , "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
                    , NULL
                    , KEY_WRITE
                    , &hSubKey) == ERROR_SUCCESS)
        {
            RegDeleteValue(hSubKey
                        , "AudioCOMMANDER");
            RegCloseKey(hSubKey);
        }
    }
}
```

```
/* FUNCTION ****************************************************************
 *
 * Name:        OnPlayback
 * Title:
 * References:
 * Return:
 * Notes:
 *
 ***************************************************************************/
void COptionsDlg::OnPlayback()
{
    UpdateData(TRUE);
    m_pPropSheet->UpdateTabs();
    /*
     * Save options to registry
     */
    CWinApp*    pApp = AfxGetApp();
    pApp->WriteProfileInt(szSettingsSection, szPlaybackKey, m_fPlayback);
}

/* FUNCTION ****************************************************************
 *
 * Name:        OnRecord
 * Title:
 * References:
 * Return:
 * Notes:
 *
 ***************************************************************************/
void COptionsDlg::OnRecord()
{
    UpdateData(TRUE);
    m_pPropSheet->UpdateTabs();
    /*
     * Save options to registry
     */
    CWinApp*    pApp = AfxGetApp();
    pApp->WriteProfileInt(szSettingsSection, szRecordingKey, m_fRecording);
}

/* FUNCTION ****************************************************************
 *
 * Name:        OnVoice
 * Title:
 * References:
 * Return:
 * Notes:
 *
 ***************************************************************************/
// void COptionsDlg::OnVoice()
// {
//  UpdateData(TRUE);
//  m_pPropSheet->UpdateTabs();
//  /*
//   * Save options to registry
//   */
//  CWinApp*    pApp = AfxGetApp();
```

```
//  pApp->WriteProfileInt(szSettingsSection, szVoiceKey, m_fVoice);
// }
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC support for Windows 95 Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include "mmsystem.h"
```

```
// stdafx.cpp : source file that includes just the standard includes
//  MicCtrl.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

WHAT IS CLAIMED IS:

1. A method for configuring a WINDOWS computer for audio applications, comprising the steps of:

providing a WINDOWS computer having a WINDOWS 95 or WINDOWS NT operating system with means to install data stored in a medium;

installing the data stored in the medium to operate in a menu format with a taskbar tray;

querying the operating system to detect a computer sound card software driver;

querying the computer sound card driver to determine the type of computer sound card installed; and

inputting default settings pertaining to the operation of the computer sound card detected.

2. A method according to claim 1, wherein the menu format is dialog-based.

3. A method according to claim 2, wherein menu format includes computer sound card settings for microphone operations.

4. A method according to claim 3, wherein the computer sound card settings include a balance, volume, and mute settings.

5. A method according to claim 3, wherein the computer sound card settings include a balance, volume, and select settings.

6. A method according to claim 1, wherein the means to install data stored in a medium is a computer hard drive.

7. A method according to claim 1 comprising the additional step of:

establishing a smart icon on the taskbar tray by selecting a dialog-based option on the menu format to correct an improper setting of a microphone control.

8. A method according to claim 7 comprising the additional step of:

providing a set default means on the taskbar tray to allow the user to establish nominal settings

means for all microphone controls to ensure effective performance for microphone applications.

9. A computerized system for controlling the computer sound card audio functions in a WINDOWS computer with WINDOWS 95 or WINDOWS NT operating system, comprising:

a WINDOWS computer having a computer sound card means to provide a user the ability to perform certain audio applications;

a data means installed in the computer to determine computer sound card capabilities and appropriate settings of the computer sound card;

a plurality of tabbed means displayed on a computer screen that allow the user to select which audio settings to manipulate for the certain audio application;

an alert displayed on the tabbed means to notify the user of individual controls which are set improperly;

means to set all computer sound card microphone related settings to nominal default values; and

alert means displayed on the taskbar tray to notify the user of improper audio settings that effects microphone performance.

10. A system according to claim 9, wherein said audio functions comprise at least a balance controller and a volume slider controller displayed in tabbed dialogs.

11. A system according to claim 10, wherein the audio functions comprise a mute setting, select setting, or VU meter displayed in tabbed dialogs.

12. A system according to claim 11, wherein the tabbed dialogs comprise at least a microphone tabbed dialog and options tabbed dialog.

13. A system according to claim 12, wherein the tabbed means comprise at least a microphone tabbed dialog, a recording tabbed dialog, a playback tabbed dialog, and an options tabbed dialog.

14. A system according to claim 13, wherein the microphone tabbed dialog controls a gain of either a microphone input or other settings effecting microphone performance.

15. A system according to claim 13, wherein the recording tabbed dialog controls a plurality of computer sound card audio inputs settings.

16. A system according to claim 13, wherein the playback tabbed dialog controls a plurality of computer sound card output settings.

17. A system according to claim 13, wherein the options tabbed dialog controls the display of the recording tabbed dialog, playback tabbed dialog, and smart icon alert means displayed on the taskbar tray.

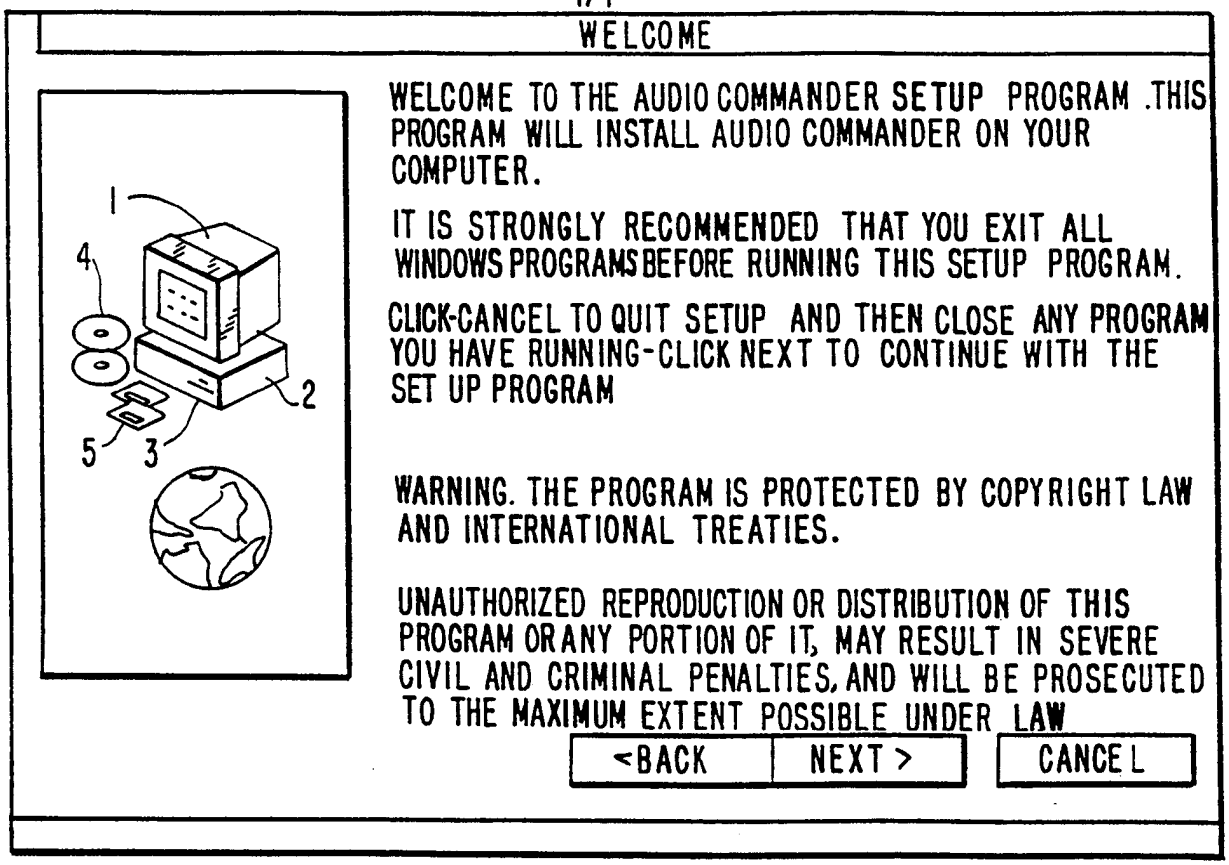

WELCOME
1
4
2
5
3
WELCOME TO THE AUDIO COMMANDER SETUP PROGRAM .THIS PROGRAM WILL INSTALL AUDIO COMMANDER ON YOUR COMPUTER.
IT IS STRONGLY RECOMMENDED THAT YOU EXIT ALL WINDOWS PROGRAMS BEFORE RUNNING THIS SETUP PROGRAM.
CLICK-CANCEL TO QUIT SETUP AND THEN CLOSE ANY PROGRAM YOU HAVE RUNNING-CLICK NEXT TO CONTINUE WITH THE SET UP PROGRAM
WARNING. THE PROGRAM IS PROTECTED BY COPYRIGHT LAW AND INTERNATIONAL TREATIES.
UNAUTHORIZED REPRODUCTION OR DISTRIBUTION OF THIS PROGRAM OR ANY PORTION OF IT, MAY RESULT IN SEVERE CIVIL AND CRIMINAL PENALTIES, AND WILL BE PROSECUTED TO THE MAXIMUM EXTENT POSSIBLE UNDER LAW
<BACK
NEXT >
CANCEL

FIG.1

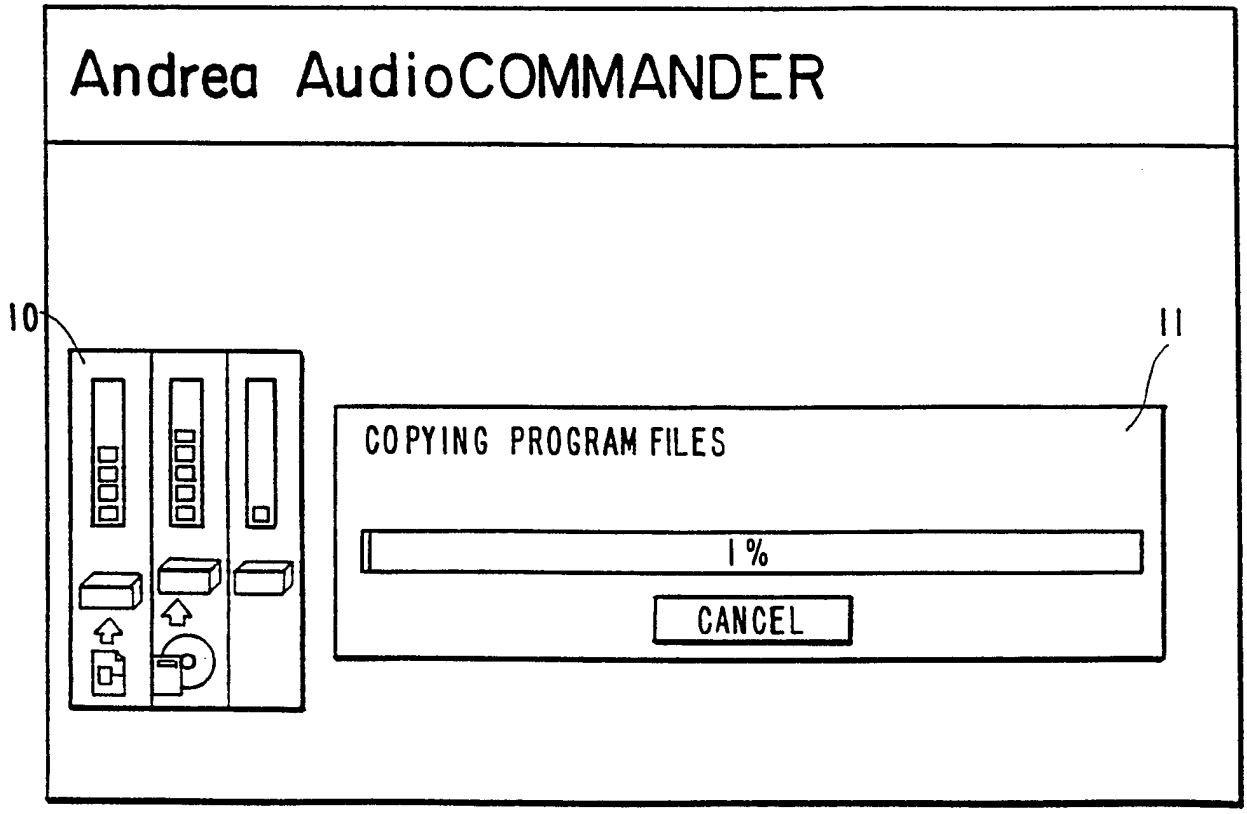
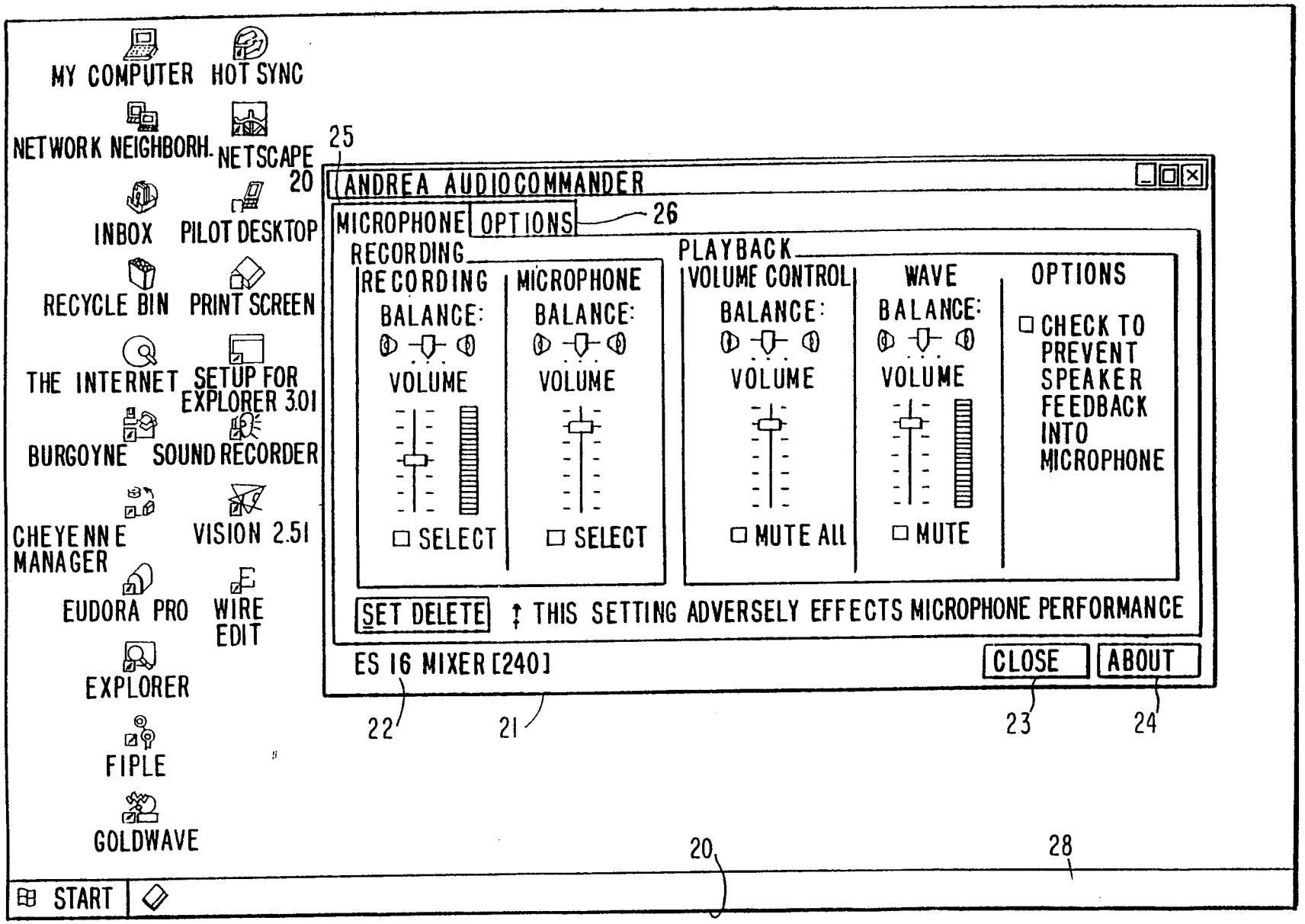

Andrea AudioCOMMANDER
10
11
COPYING PROGRAM FILES
1 %
CANCEL

FIG.2

FIG.3



SUBSTITUTE SHEET (RULE 26)

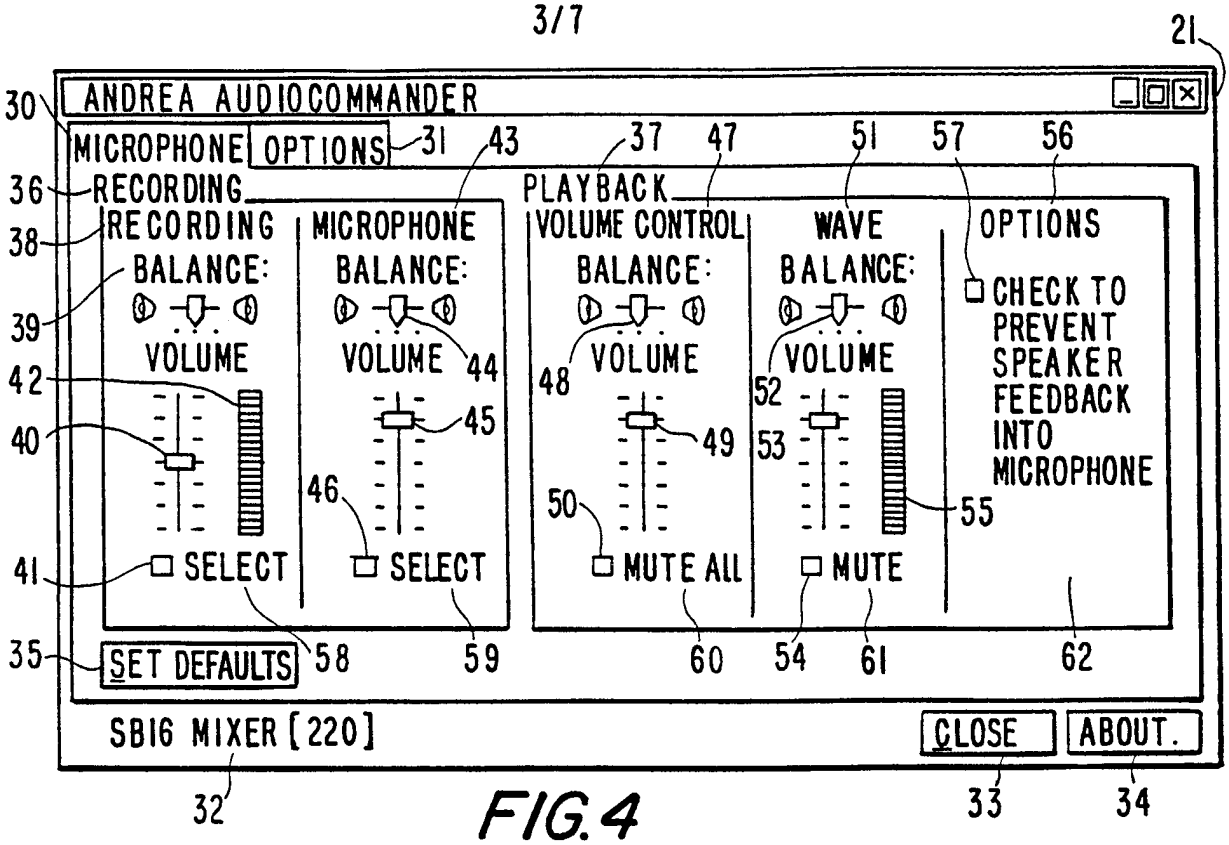

21
30
ANDREA AUDIOCOMMANDER
MICROPHONE
OPTIONS
31
43
37
47
51
57
56
36
RECORDING
PLAYBACK
38
RECORDING
BALANCE:
VOLUME
MICROPHONE
BALANCE:
VOLUME
VOLUME CONTROL
BALANCE:
VOLUME
WAVE
BALANCE:
VOLUME
OPTIONS
CHECK TO PREVENT SPEAKER FEEDBACK INTO MICROPHONE
39
42
40
44
45
46
48
49
50
52
53
55
41
SELECT
SELECT
MUTE ALL
MUTE
35
SET DEFAULTS
58
59
60
54
61
62
SB16 MIXER [220]
CLOSE
ABOUT.
32
33
34

FIG.4

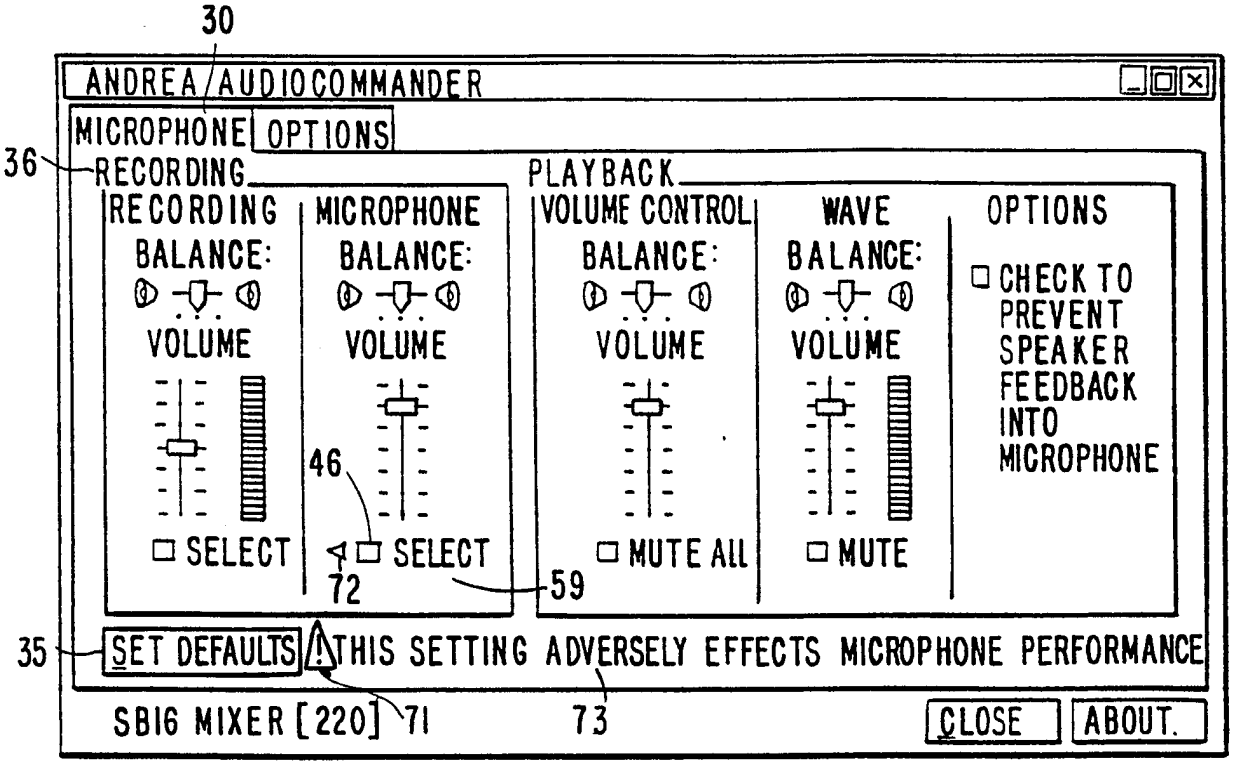

30
ANDREA AUDIOCOMMANDER
MICROPHONE
OPTIONS
36
RECORDING
PLAYBACK
RECORDING
BALANCE:
VOLUME
MICROPHONE
BALANCE:
VOLUME
VOLUME CONTROL
BALANCE:
VOLUME
WAVE
BALANCE:
VOLUME
OPTIONS
CHECK TO PREVENT SPEAKER FEEDBACK INTO MICROPHONE
46
SELECT
SELECT
MUTE ALL
MUTE
72
59
35
SET DEFAULTS
THIS SETTING ADVERSELY EFFECTS MICROPHONE PERFORMANCE
71
73
SB16 MIXER [220]
CLOSE
ABOUT.

FIG.5

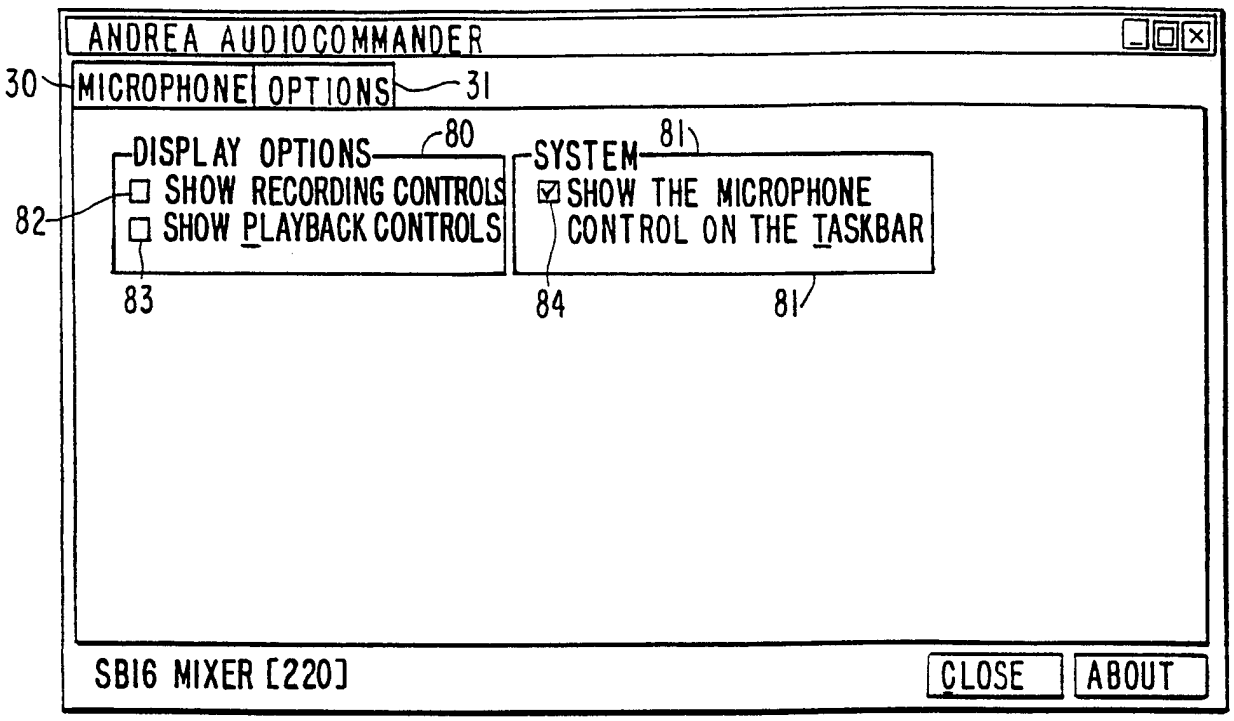

ANDREA AUDIOCOMMANDER
30
MICROPHONE OPTIONS
31
DISPLAY OPTIONS
80
SYSTEM
81
SHOW RECORDING CONTROLS
SHOW PLAYBACK CONTROLS
82
SHOW THE MICROPHONE CONTROL ON THE TASKBAR
83
84
81
SB16 MIXER [220]
CLOSE
ABOUT

FIG.6

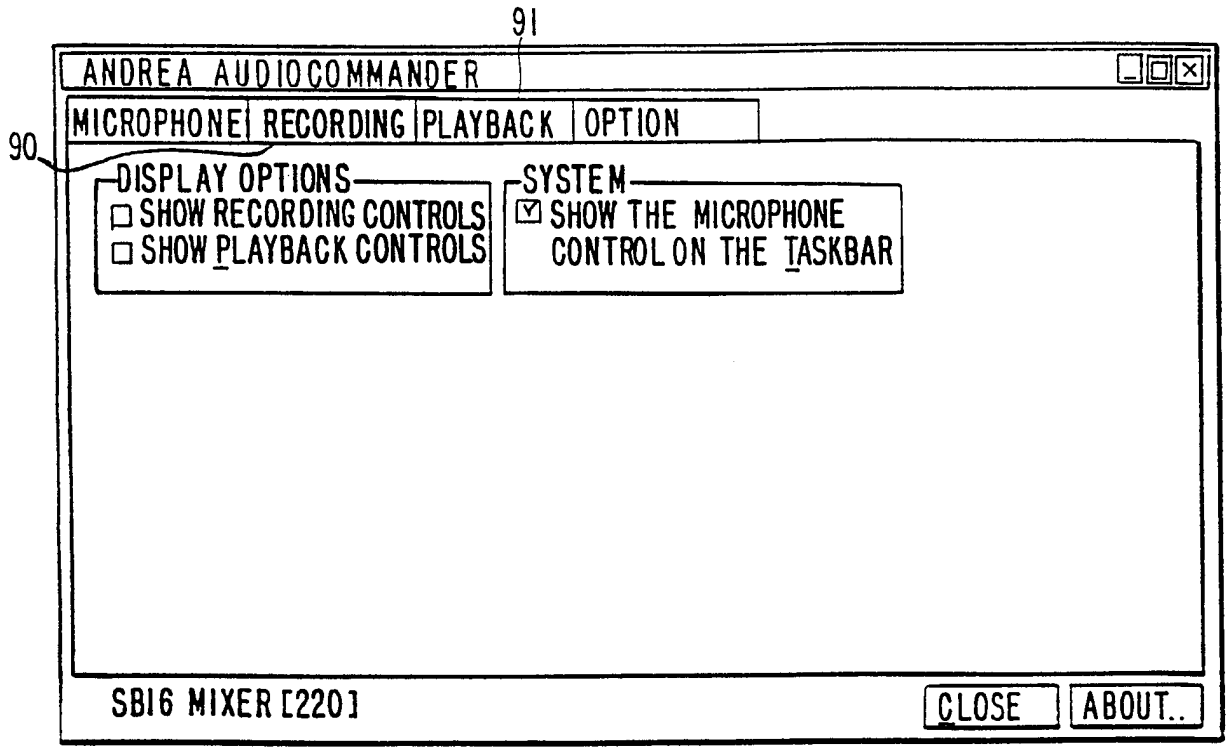

91
ANDREA AUDIOCOMMANDER
MICROPHONE RECORDING PLAYBACK OPTION
90
DISPLAY OPTIONS
SHOW RECORDING CONTROLS
SHOW PLAYBACK CONTROLS
SYSTEM
SHOW THE MICROPHONE CONTROL ON THE TASKBAR
SB16 MIXER [220]
CLOSE
ABOUT..

FIG.7

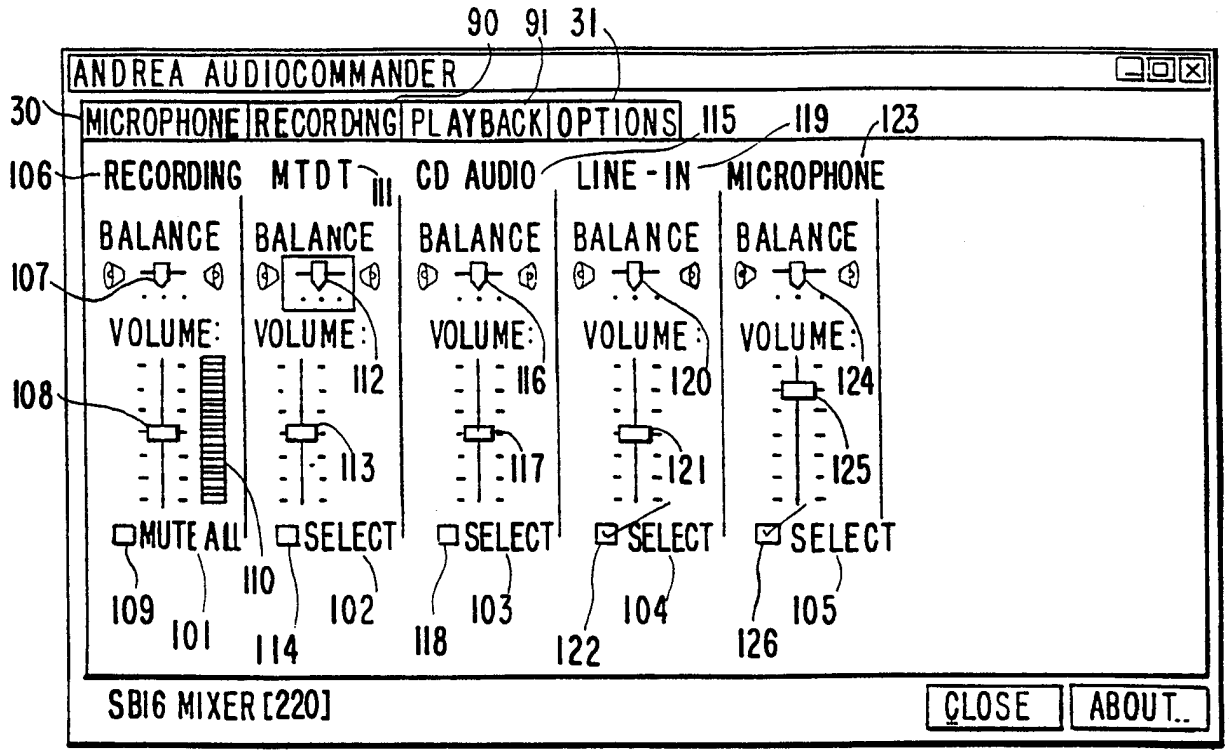

90
91
31
ANDREA AUDIOCOMMANDER
30
MICROPHONE
RECORDING
PLAYBACK
OPTIONS
115
119
123
106
RECORDING
MTDT
111
CD AUDIO
LINE-IN
MICROPHONE
BALANCE
107
VOLUME:
108
112
113
116
117
120
121
124
125
MUTE ALL
SELECT
109
101
110
114
102
118
103
122
104
126
105
SB16 MIXER [220]
CLOSE
ABOUT..

FIG.8

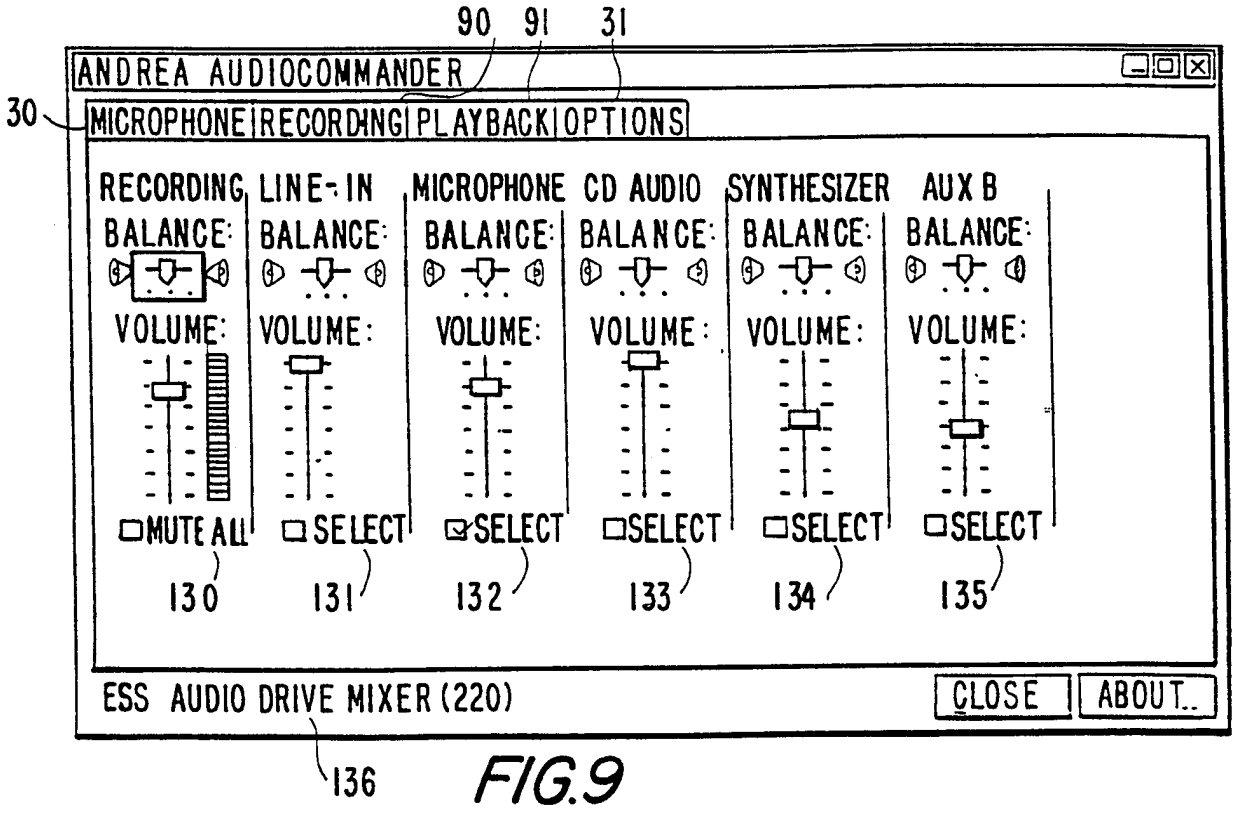

90
91
31
ANDREA AUDIOCOMMANDER
30
MICROPHONE
RECORDING
PLAYBACK
OPTIONS
RECORDING
LINE-IN
MICROPHONE
CD AUDIO
SYNTHESIZER
AUX B
BALANCE:
VOLUME:
MUTE ALL
SELECT
130
131
132
133
134
135
ESS AUDIO DRIVE MIXER (220)
CLOSE
ABOUT..
136

FIG.9

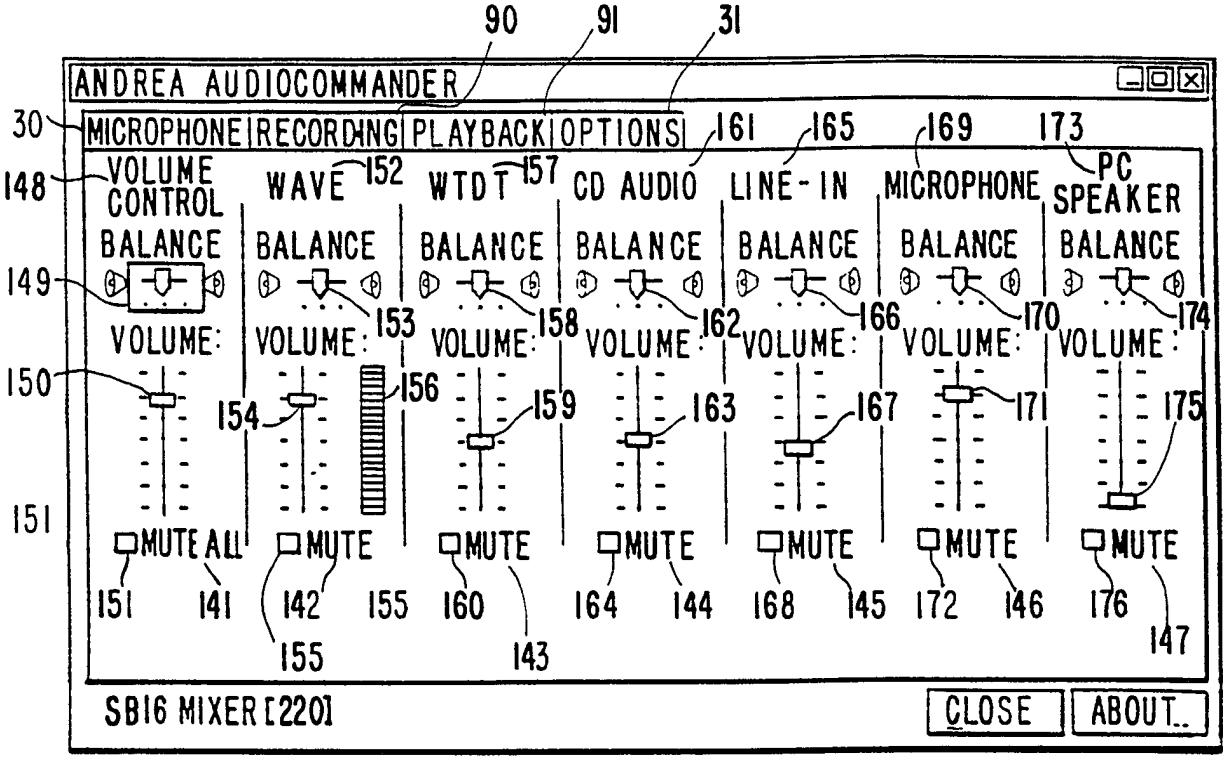

ANDREA AUDIOCOMMANDER
MICROPHONE
RECORDING
PLAYBACK
OPTIONS
VOLUME CONTROL
WAVE
WTDT
CD AUDIO
LINE-IN
MICROPHONE
PC SPEAKER
BALANCE
VOLUME:
MUTE ALL
MUTE
SB16 MIXER [220]
CLOSE
ABOUT..
30
31
90
91
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
141
142
143
144
145
146
147

FIG. 10

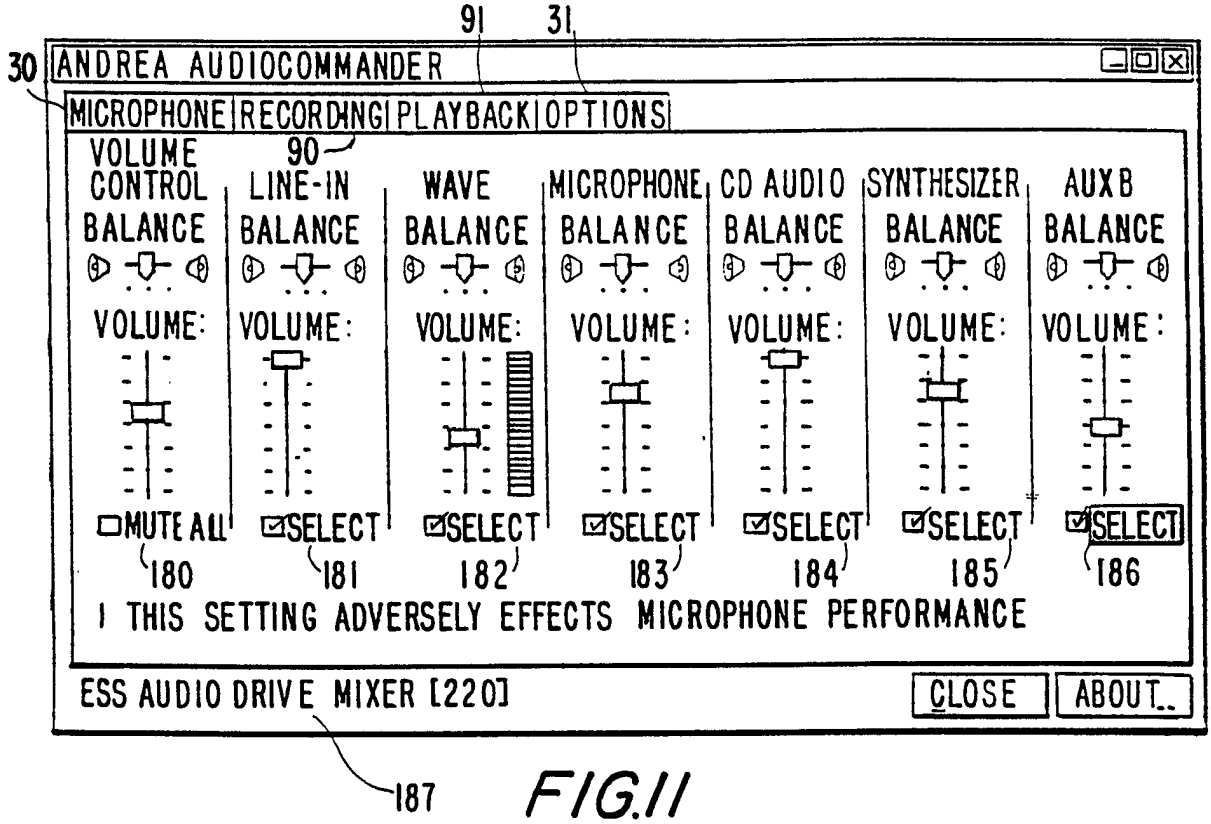

ANDREA AUDIOCOMMANDER
MICROPHONE
RECORDING
PLAYBACK
OPTIONS
VOLUME CONTROL
LINE-IN
WAVE
MICROPHONE
CD AUDIO
SYNTHESIZER
AUXB
BALANCE
VOLUME:
MUTE ALL
SELECT
1 THIS SETTING ADVERSELY EFFECTS MICROPHONE PERFORMANCE
ESS AUDIO DRIVE MIXER [220]
CLOSE
ABOUT..
30
31
90
91
180
181
182
183
184
185
186
187

FIG. 11

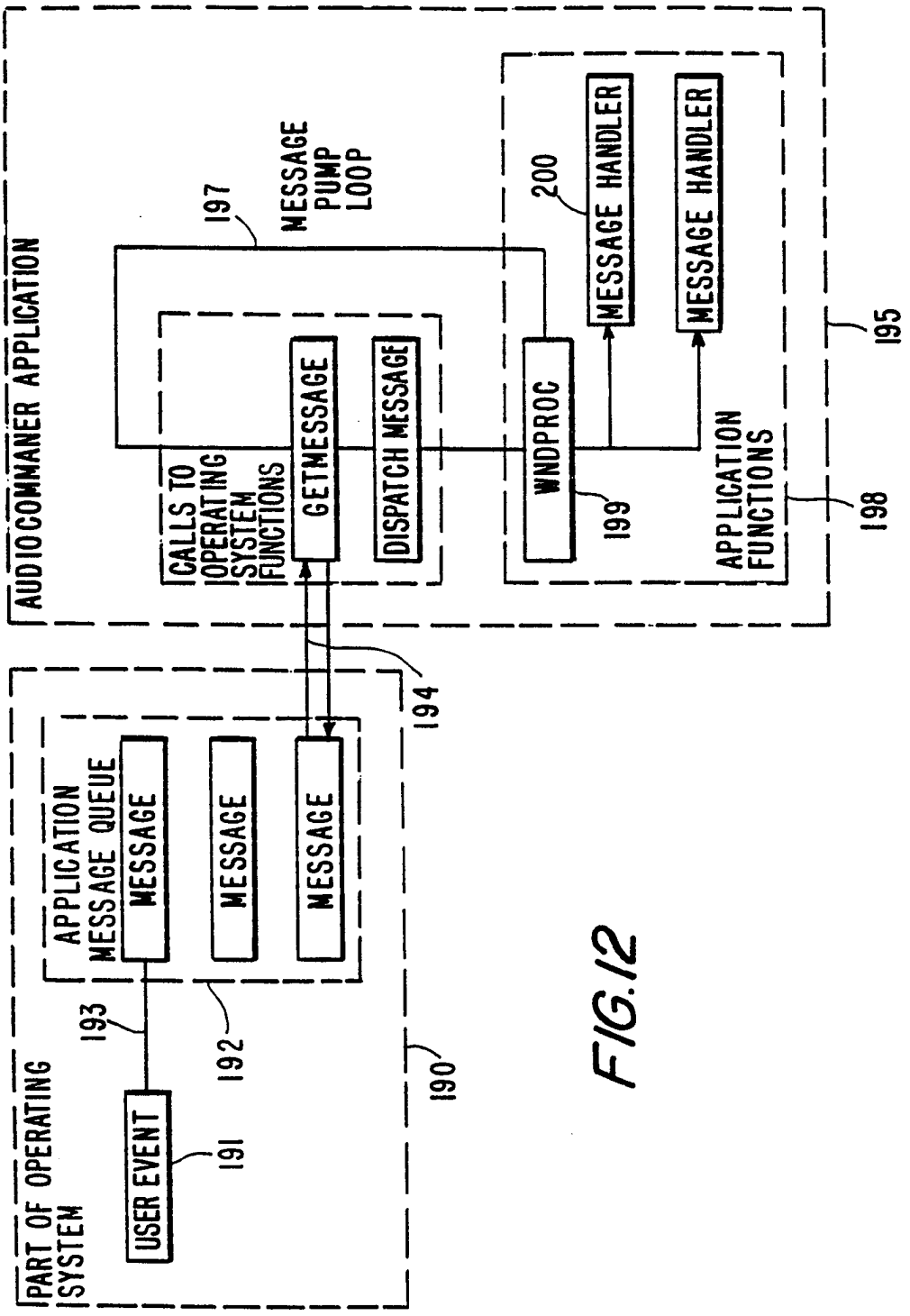

AUDIOCOMMANER APPLICATION
MESSAGE PUMP LOOP
197
200
MESSAGE HANDLER
MESSAGE HANDLER
195
CALLS TO OPERATING SYSTEM FUNCTIONS
GETMESSAGE
DISPATCH MESSAGE
WNDPROC
199
APPLICATION FUNCTIONS
198
194
APPLICATION MESSAGE QUEUE
MESSAGE
MESSAGE
MESSAGE
193
192
PART OF OPERATING SYSTEM
USER EVENT
191
190

FIG.12