Глава 6
Системный звуковой микшер

Глава 6
Системный звуковой микшер....1
Системный звуковой микшер....2
Структурная схема микшера....3
Структуры и функции микшера....6
Сбор данных для линии....8
Структура MIXERLINE....10
Функция mixerGetLineInfo....13
Сбор данных об органах управления линии....16
Структура MIXERLINECONTROLS....16
Структура MIXERCONTROL....17
Классы (типы) органов управления микшером....20
Функция mixerGetLineControls....25
Структура MIXERCONTROLDETAILS....27
Установка свойств и значений органов управления....30
Структуры установки свойств и значений органов управления....30
Функция mixerSetControlDetails....33
Функция mixerGetControlDetails....35
Структуры и функции для общего управления микшером....39
Структура MIXERCAPS....39
Функция mixerGetDevCaps....40
Функция mixerGetNumDevs....40
Функция mixerClose....40
Функция mixerOpen....41
Функция mixerGetID....43
Сообщения микшера....46
Пользовательское сообщение для заказного микшера....47
Классы микшера....47
Структура классов звукового микшера....48
Базовый класс CMixer....50
Организация данных....59
Классы органов управления....61
Класс CFader....62
Класс CSwitch....69
Класс мультиплексора CMux....73
Производный класс микшера CLineMixer....87
Методы для создания органов управления....90

Установка параметров органов управления....93
Удаление органов управления....94
Обработка сообщений при изменении параметров микшера другим приложением....95
Примечание к проекту 97

Системный звуковой микшер

Звуковой микшер является частью мультимедийной библиотеки Windows и описывает входные и выходные линии звуковой карты, а также органы управления размещенные на них. Библиотека размещается в двух файлах winmm.dll и в ее шестнадцатиразрядной версии mmsystem.dll. Она связывает вызовы мультимедийного пакета API (mixerXXX) с драйвером WDMAud System Driver (wdmaud.drv). Библиотека является нижним слоем, который позволяет программисту управлять звуковым драйвером. Все остальные программные компоненты, в том числе и DirectSound, используют эту библиотеку.

Полное управление звуковым микшером обеспечивает системная программа sndvol32.exe, которая представлена иконкой громкоговорителя в "system tray", . Многие широко применяемые программы, например проигрыватель Windows Media, Winamp не предоставляют прямого управления микшером. В этом легко убедиться, одновременно запустив sndvol32.exe и проигрыватель. Например, при регулировке громкости в проигрывателе громкость, установленная в звуковом микшере не изменяется. Проигрыватель только уменьшает уровень громкости, установленный в звуковом микшере, но не управляет непосредственно микшером. Дополнительно следует отметить, что интерфейсы DirectShow, которые управляют драйверами звукового устройства, не реализуют полное управление звуковым микшером и имеют ограниченные функции.

Зачастую это приводит к неудобству при работе с программой, поскольку для регулировки громкости, кроме регулятора громкости проигрывателя Windows Media приходится вызывать sndvol32.exe для дополнительного управления громкостью и включением звука. Неопытный пользователь может вообще не знать о существовании sndvol32.exe. В этом случае, если звук будет выключен с ее помощью, он не сможет вообще ничего прослушать с проигрывателем Windows Media. Та же ситуация наблюдается и с другим популярным проигрывателем Winamp. Особенно этот недостаток проявляется, если источники звука имеют большой разброс по уровню громкости, что заставляет часто использовать регулировки звука с помощью sndvol32.exe. Например, телевизионные программы очень часто имеют большой разброс по уровню звука.

При записи звука может возникнуть другая неприятная ситуация, связанная с особенностями аппаратной реализации микшера. Практически все

современные звуковые карты предоставляют два пути записи звука. Первый путь прямая запись от одного источника звука через канал мультиплексора, который производит независимое переключение источников звука. Второй канал записи образуется при использовании смесителя сигналов и подключения выхода смесителя на вход мультиплексора записи, что позволяет использовать эффекты наложения звука. Наличие двух путей прохождения сигналов может привести к появлению эха в конечном канале записи при определенном сочетании подключенных линий.

Подобные неприятности могут поставить разработчика перед выбором либо использовать совместно со своей программой системную программу sndvol32.exe, что крайне неудобно и снижает потребительские свойства разрабатываемой программы, либо внедрять в свою разработку управление звуковым микшером.

При внедрении в разработку управления системным микшером, в большинстве случаев, если только не пишется программа для специализированной обработки звука, достаточно общего управления включением и выключением сигнала, и управления громкостью в трактах записи и воспроизведения. Обычно нет необходимости обеспечивать полную функциональность микшера. Достаточно предоставить пользователю необходимые для функционирования программы регулировки звука. Остальные органы управления должны быть установлены по умолчанию или переключаться в соответствии с режимами программы таким образом, чтобы не возникало необходимости в использовании sndvol32.exe. Для этих целей желательно иметь простой в применении класс микшера, и в тоже время позволяющий наращивать его возможности без особых дополнительных усилий. Прежде чем перейти к проектированию такого класса рассмотрим общие принципы построения API, управляющего микшером.

Структурная схема микшера.

Основным элементом микшера является линия. Все линии можно разделить на две большие группы: входные линии и выходные линии. Входные линии подключаются к источнику звука с одной стороны и к выходной линии с другой стороны. Выходные линии подключаются к входной линии с одной стороны и к потребителю звукового сигнала с другой стороны. Каждая линия может иметь свой набор органов управления, которые в общем случае представляют собой регуляторы различного назначения, переключатели режимов работы и один или несколько каналов прохождения сигнала.

Кроме деления на входные и выходные, все линии еще делятся по типу применения. Например, может быть выходная линия для громкоговорителя и выходная линия для наушников, входная линия для микрофона и входная линия для линейного входа. Таким образом, линия описывается и типом компонента, который подключается к ней.

Потребители звукового сигнала также разделяются на две групп: потребители, воспроизводящие звук, и потребители, записывающие звук. Следует отметить, что прямого соответствия между источниками сигнала и физическими входами звуковой карты нет.

На рисунке 1 приводится упрощенная структура микшера интегрированной звуковой карты. Такой микшер имеет обычно две выходные линии, одна предназначена для записи звука, а вторая для его воспроизведения.

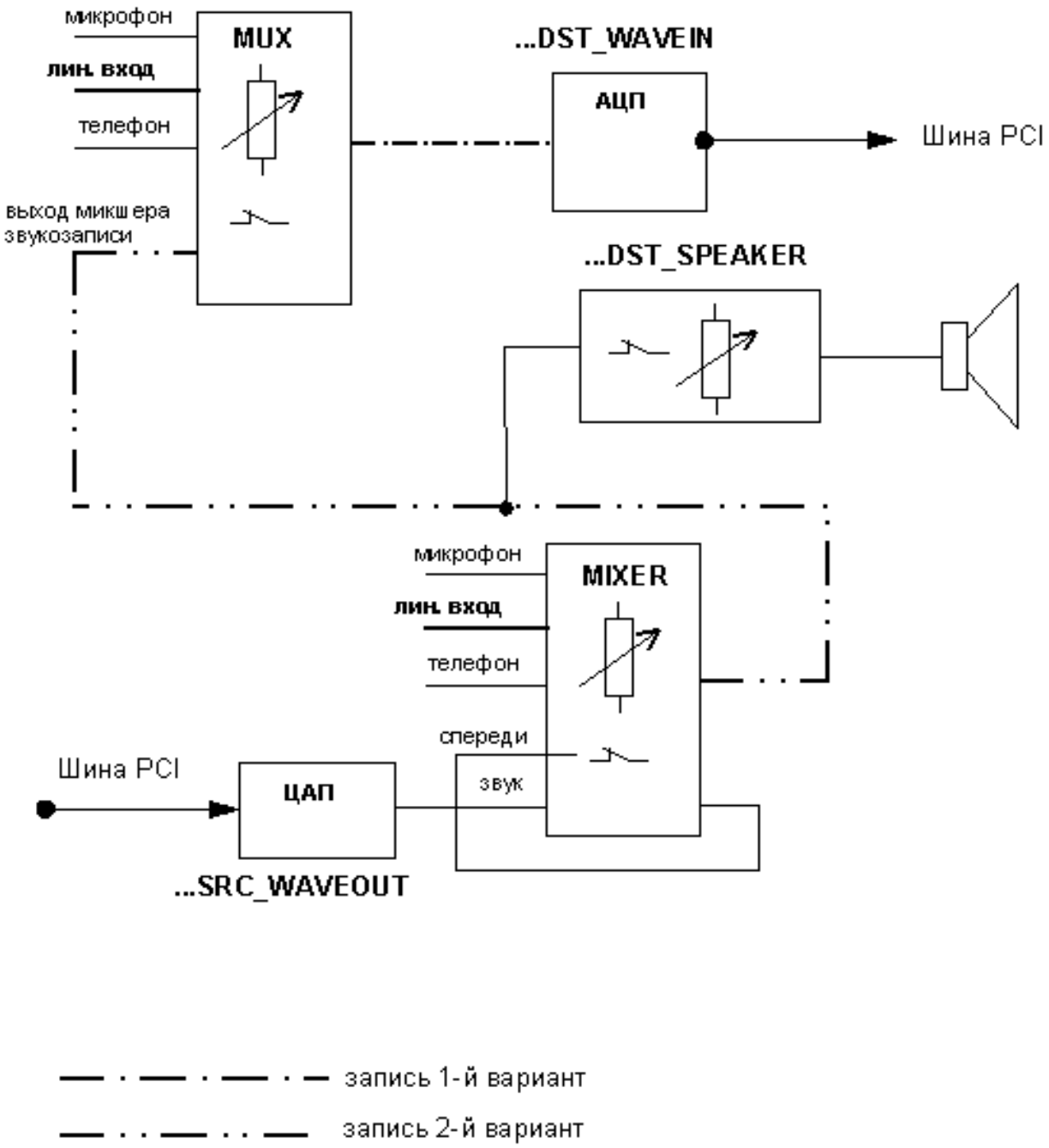

Рис.6. 1. Упрощенная структура микшера.

Линии определены константами типов компонентов MIXERLINE_COMPONENTTYPE_DST_WAVEIN (запись) и MIXERLINE_COMPONENTTYPE_DST_SPEAKERS (воспроизведение). Эти две линии являются типовыми линиями и присутствуют во всех звуковых картах. Линия записи (…DST_WAVEIN) физически является входом аналого-цифрового преобразователя.

К линии записи подключается входная линия, обычно являющаяся мультиплексором, который может подключить к линии записи не более одной входной сигнальной линии. Таким образом, мультиплексор можно рассматривать как сложную входную линию, содержащую N переключаемых входных линий. На рисунке мультиплексор обозначен как элемент MUX.

К линии воспроизведения (…DST_SPEAKERS) подключается смеситель (сумматор) сигналов. Часто он не представляется сложной линией как мультиплексор, а выглядит как простая совокупность входных линий, которые могут подключаться группой из N линий к выходной линии. Выход смесителя может подключаться к входу мультиплексора, образуя второй канал записи сигнала с одновременным его прослушиванием. При таком подключении появляется возможность производить запись от нескольких источников одновременно. Из всех входных линий следует отметить типовую линию, которая присуща всем звуковым картам и маркируется константой MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT. Эта линия является выходом цифро-аналогового преобразователя.

Вышеупомянутые константы определены в файле mmsystem.h. Среди них есть две константы (…UNDEFINED) для маркировки линий специфических для звуковой карты. Линии, промаркированные этими константами, изначально могут быть, как отключены, так и подключены. Часто они имеют имена, не отражающие их действительного назначения или отражающие их очень и очень условно. В качестве примера можно привести линию … SRC_WAVEOUT, которая имеет имя "Звук" и является выходом цифро-аналогового преобразователя. Слово "Звук" с позиции пользователя совершенно не отражает назначения линии. Также не ясно назначение специфической линии (…UNDEFINED) "Спереди". Можно предположить, что это подключение фронтальных громкоговорителей, точно определить это пользователь может только экспериментально. Подобные имена при проектировании класса микшера следует заменять собственными именами, более точно отражающими назначение органа управления или оставлять имена в английской транскрипции без перевода. На рисунке 1 показана экспериментально определенная цепь выключателя "Спереди". Как видно из рисунка он включен последовательно с выключателем "Звук".

Эта пара выключателей важна еще и по другой причине. Допустим, что производится запись с линейного входа звуковой карты. Для этого подключается линейный вход мультиплексора, а для прослушивания записываемого сигнала следует подключить линию "Звук" и линию "Спереди", что совершенно не очевидно для обычного пользователя. Допускаем при этом, что

пользователь знает, что нужно включить линию "Главный регулятор" и сделал это. Если необходимо произвести запись с суммированием источников тогда нужно на мультиплексоре записи включить "Выход микшера звукозаписи", а на смесителе воспроизведения "Лин. вход" и например "Микрофон". При этом необходимо отключить либо линию "Звук" либо линию "Спереди" или обе сразу. В противном случае создается канал передачи сигнала записи с выхода на вход через шину PCI и образование многократного эха. Задача усложняется тем обстоятельством, что производитель не считает нужным предоставить необходимую информацию в поставляемой документации. Кроме того, каждый производитель вправе по собственному усмотрению создавать специфические линии для своего устройства и, как правило, не объясняет их назначения.

Хорошего решения для выхода из этой ситуации нет. Точнее хорошим решением является плохое решение, оставить пользователя один на один с системным микшером, но решение плохо еще и тем, что окно sndvol32.exe слишком большое и неудобное. Можно попытаться несколько исправить положение, если создать класс для управления микшером, который позволял бы разделить управление микшером по разным диалогам и создавать органы управления раздельно для каждой линии. Тогда можно было бы отсеять часть заведомо ненужных органов управления и сгруппировать необходимые в соответствии с режимами работы программы. Далее сделаем попытку создать подобный класс, а пока рассмотрим подробно средства управления микшером.

Структуры и функции микшера

Линии и органы управления ими описываются иерархической системой структур и функций, которые позволяют оперировать конкретным органом управления на заданной линии. Иерархия структур их взаимосвязи и краткая характеристика показаны на рисунке 6.2.

Общий принцип управления определенным органом заключается в заполнении данными структуры MIXERLINE для определенной линии и передаче этих данных в нижестоящие структуры, которые использует их как исходные данные. Нижестоящие структуры, используя входную информацию, заполняются дополнительными данными вплоть до конкретного органа управления. Управляют этим процессом несколько функций. Каждая из функций заполняет данными определенную структуру и обеспечивает данными подчиненные структуры.

Все функции и структуры микшера делятся на две большие группы. Первая группа обеспечивает открытие микшера и получение общих данных. Вторая группа обеспечивает получение данных для определенной линии и взаимодействие с органами управления, размещенными на этой линии.

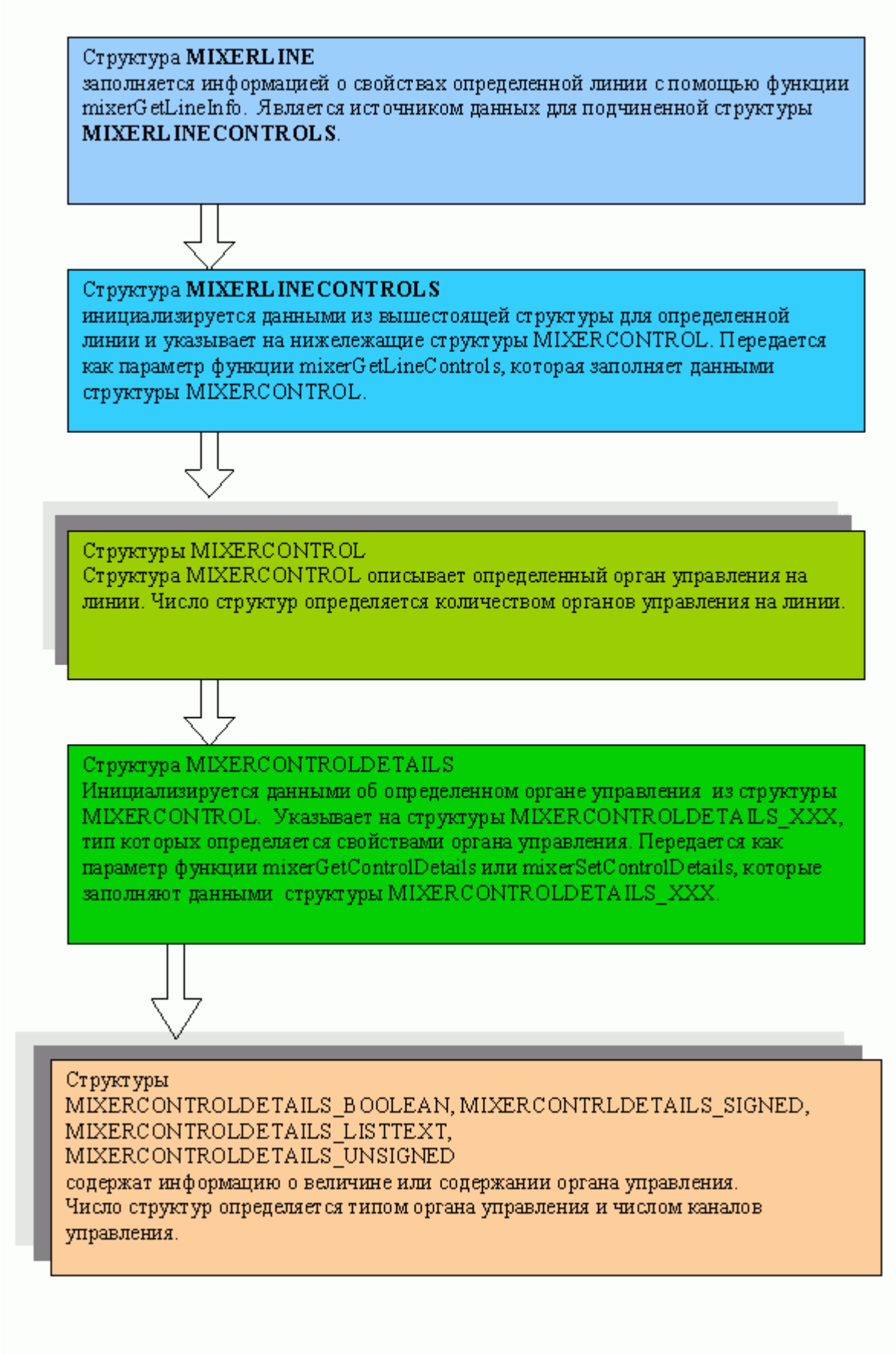

Рис.6. 2. Система структур, описывающая линии и органы управления.

Сбор данных для линии.

Сбор данных для линии производит многорежимная функция mixerGetLineInfo, которая заполняет данными структуру MIXERLINE. Режим работы функции определяется флагом, который передается в третьем параметре. Смысловое значение возвращаемой информации о линии определяется ее назначением, которое маркируется константами типа компонентов, подключаемых к линии. Другими словами назначение линии определяется типом источника или потребителя сигнала. Типы компонентов указываются в структуре MIXERLINE, но, учитывая их важность для понимания назначения линии, следует предварительно ознакомиться с ними.

Константы потребителей сигнала.

1. MIXERLINE_COMPONENTTYPE_DST_SPEAKERS.
   Линия для подключения громкоговорителей с регулировкой усиления. Типовой компонент для звуковых карт.
2. MIXERLINE_COMPONENTTYPE_DST_WAVEIN.
   Линия записи сигнала, которая подключается к входу АЦП и обеспечивает регулировку усиления. Типовой компонент для звуковых карт.
3. MIXERLINE_COMPONENTTYPE_DST_DIGITAL.
   Линия для подключения устройства с цифровым входом.
4. MIXERLINE_COMPONENTTYPE_DST_HEADPHONES.
   Линия для подключения головных телефонов с регулировкой усиления. В большинстве случаев используется та же линия, что и для громкоговорителя. В этом случае используется константа …DST_SPEAKERS.
5. MIXERLINE_COMPONENTTYPE_DST_LINE.
   Линия линейного выхода. Подключается к входу АЦП (аналого-цифровой преобразователь). Для большинства звуковых карт микшер использует тип …DST_WAVEIN.
6. MIXERLINE_COMPONENTTYPE_DST_MONITOR.
   Линия контрольного выхода.
7. MIXERLINE_COMPONENTTYPE_DST_TELEPHONE.
   Линия, которая используется для подключения к телефонной линии.
8. MIXERLINE_COMPONENTTYPE_DST_UNDEFINED.
   Все линии, не подпадающие под классификацию Microsoft.
9. MIXERLINE_COMPONENTTYPE_DST_VOICEIN.
   Линия для записи голоса. Соответствует линии …DST_WAVEIN. Линия выделена специально для записи и распознавания голоса. Микшер может не поддерживать эту линию.

Если звуковая карта имеет две выходные линии, то это будет линия громкоговорителя (…DST_SPEAKERS) и линия записи сигнала (… DST_WAVEIN).

Константы источников сигнала.

1. MIXERLINE_COMPONENTTYPE_SRC_ANALOG
   Линия для подключения аналогового источника сигнала, например магнитофона. Может быть использована и в других целях. Например, этой константой определяется линия фильтра звуковой карты, осуществляющего регулировку нижних частот.
2. MIXERLINE_COMPONENTTYPE_SRC_AUXILIARY
   Линия вспомогательного аналогового входа. Она может использоваться как линии для регулировки усиления при подключении к выходным линиям записи или воспроизведения.
3. MIXERLINE_COMPONENTTYPE_SRC_COMPACTDISC
   Линия для подключения аналогового выхода устройства для воспроизведения звуковых компакт-дисков.
4. MIXERLINE_COMPONENTTYPE_SRC_DIGITAL
   Линия для подключения цифровых источников звука.
5. MIXERLINE_COMPONENTTYPE_SRC_LINE
   Линия линейного аналогового входа. Может использоваться для записи и воспроизведения аналоговых источников звука. Эта линия аналогична линии определяемой константой вспомогательного источника MIXERLINE_COMPONENTTYPE_SRC_AUXILIARY.
6. MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE
   Линия для подключения микрофона.
7. MIXERLINE_COMPONENTTYPE_SRC_PCSPEAKER
   Линия источника звука для внутреннего громкоговорителя компьютера. Некоторые звуковые карты обеспечивают вывод звука для внутреннего громкоговорителя на свои выходные линии для записи или воспроизведения.
8. MIXERLINE_COMPONENTTYPE_SRC_SYNTHESIZER
   Линия звукового источника от внутреннего синтезатора.
9. MIXERLINE_COMPONENTTYPE_SRC_TELEPHONE
   Линия для подключения телефонной линии.
10. MIXERLINE_COMPONENTTYPE_SRC_UNDEFINED
    Все линии не подпадающие под классификацию Microsoft.
11. MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT
    Линия выхода цифро-аналогового преобразователя. Для большинства звуковых карт это линия для подключения к выходной линии MIXERLINE_COMPONENTTYPE_DST_SPEAKERS. Некоторые звуковые карты позволяют подключать эту линию к выходной линии записи MIXERLINE_COMPONENTTYPE_DST_WAVEIN.

Если число источников или потребителей сигнала для звуковой карты превышает число определенных констант, они маркируются константой xxxUNDEFINED и их следует идентифицировать по другим параметрам.

Структура MIXERLINE

Структура заполняется данными для определенной линии. Неиспользуемые члены структуры должны быть установлены в нуль. Перед использованием структуры целесообразно ее полностью заполнить нулями (очистить).

```
typedef struct {
    DWORD cbStruct;
    DWORD dwDestination;
    DWORD dwSource;
    DWORD dwLineID;
    DWORD fdwLine;
    MIXERLINE_LINEF_ACTIVE
    MIXERLINE_LINEF_DISCONNECTED
    MIXERLINE_LINEF_SOURCE
    DWORD dwUser;
    DWORD dwComponentType;
    DWORD cChannels;
    DWORD cConnections;
    DWORD cControls;
    CHAR  szShortName[MIXER_SHORT_NAME_CHARS];
    CHAR  szName[MIXER_LONG_NAME_CHARS];

   struct {
        DWORD     dwType;
        DWORD     dwDeviceID;
        WORD      wMid;
        WORD      wPid;
        MMVERSION vDriverVersion;

     CHAR      szPname[MAXPNAMELEN];
    } Target;
} MIXERLINE;
```

Члены структуры.

DWORD cbStruct;
Размер структуры, установить перед использованием.

DWORD dwDestination;
Индекс потребителя данной линии. Диапазон значений этого члена определяется функцией mixerGetDevCaps. Для обычной звуковой карты число потребителей равно двум (громкоговоритель и устройство записи), иногда бывает три потребителя. Применение этого члена определяется режимом вызова функции mixerGetLineInfo.

DWORD dwSource;
Индекс источника сигнала для данной линии. Диапазон значений этого члена и применение определяется функцией mixerGetLineInfo.

DWORD dwLineID;
Уникальный идентификатор линии, определяемый устройством.

DWORD fdwLine;
Флаг состояния и поддержки линии. Может иметь следующие значения:

- MIXERLINE_LINEF_ACTIVE
  Линия активна. Означает, что сигнал, возможно, проходит через линию. Если выходное звуковое устройство открыто приложением без посылки данных оно считается активным. Если линия не может быть точно определена как активная, флаг всегда устанавливается в состояние MIXERLINE_LINEF_ACTIVE.

- MIXERLINE_LINEF_DISCONNECTED
  Линия отключена. Состояние органов управления можно изменять, но влияние на сигнал при этом не оказывается.

- MIXERLINE_LINEF_SOURCE
  Указывает на входную линию, ассоциированную с выходной линией. Если флаг не установлен, то это выходная линия, ассоциированная с входной линией.

DWORD dwUser;
Данные, определяемые звуковым устройством для заказного микшера. Этот член не должен использоваться в других случаях, кроме заказного микшера.

DWORD dwComponentType;
Тип компонента, который подключается к линии. Для входных линий это тип источника сигнала, а для выходных это тип потребителя сигнала, подключенного к линии. Может иметь следующие значения:

- MIXERLINE_COMPONENTTYPE_DST_FIRST
- MIXERLINE_COMPONENTTYPE_DST_UNDEFINED
- MIXERLINE_COMPONENTTYPE_DST_DIGITAL
- MIXERLINE_COMPONENTTYPE_DST_LINE
- MIXERLINE_COMPONENTTYPE_DST_MONITOR
- MIXERLINE_COMPONENTTYPE_DST_SPEAKERS
- MIXERLINE_COMPONENTTYPE_DST_HEADPHONES
- MIXERLINE_COMPONENTTYPE_DST_TELEPHONE
- MIXERLINE_COMPONENTTYPE_DST_WAVEIN
- MIXERLINE_COMPONENTTYPE_DST_VOICEIN
- MIXERLINE_COMPONENTTYPE_DST_LAST

- MIXERLINE_COMPONENTTYPE_SRC_FIRST

- MIXERLINE_COMPONENTTYPE_SRC_UNDEFINED
- MIXERLINE_COMPONENTTYPE_SRC_DIGITAL
- MIXERLINE_COMPONENTTYPE_SRC_LINE
- MIXERLINE_COMPONENTTYPE_SRC_MICROPHONE
- MIXERLINE_COMPONENTTYPE_SRC_SYNTHESIZER
- MIXERLINE_COMPONENTTYPE_SRC_COMPACTDISC
- MIXERLINE_COMPONENTTYPE_SRC_TELEPHONE
- MIXERLINE_COMPONENTTYPE_SRC_PCSPEAKER
- MIXERLINE_COMPONENTTYPE_SRC_WAVEOUT
- MIXERLINE_COMPONENTTYPE_SRC_AUXILIARY
- MIXERLINE_COMPONENTTYPE_SRC_ANALOG
- MIXERLINE_COMPONENTTYPE_SRC_LAST

DWORD cChannels;
Число раздельных каналов управления. Для стерео сигнала равно двум.

DWORD cConnections;
Число подключенных входных линий к выходной линии. Используется только для выходных линий.

DWORD cControls;
Число органов управления линией.

CHAR szShortName[MIXER_SHORT_NAME_CHARS];
Короткое имя линии.

CHAR szName[MIXER_LONG_NAME_CHARS];
Полное имя линии.

Структура Target описывает звуковое устройство, относящееся к линии, данные которой содержаться в структуре MIXERLINE.

DWORD dwType;
Тип звукового устройства ассоциированного с линией. Определены следующие типы:

- MIXERLINE_TARGETTYPE_AUX;
  Дополнительное звуковое устройство.

- MIXERLINE_TARGETTYPE_MIDIIN;
  Входное MIDI устройство.

- MIXERLINE_TARGETTYPE_MIDIOUT;
  Выходное MIDI устройство.

- MIXERLINE_TARGETTYPE_UNDEFINED;
  Неопределенное устройство. Все члены структуры Target должны игнорироваться. Этот тип нельзя использовать при вызове

функции mixerGetLineInfo с флагом MIXER_GETLINEINFOF_TARGETTYPE.

- MIXERLINE_TARGETTYPE_WAVEIN;
  Входное волновое звуковое устройство.

- MIXERLINE_TARGETTYPE_WAVEOUT;
  Выходное волновое звуковое устройство.

DWORD dwDeviceID;
Идентификатор устройства при условии, что член dwType не равен константе MIXERLINE_TARGETTYPE_UNDEFINED. Он идентичен индексу звукового устройства.

WORD wMid;
Идентификатор производителя при условии, что член dwType не равен константе MIXERLINE_TARGETTYPE_UNDEFINED. Константы идентификаторов производителя находятся в файле MMReg.h.

WORD wPid;
Идентификатор изделия при условии, что член dwType не равен константе MIXERLINE_TARGETTYPE_UNDEFINED. Константы идентификаторов изделия находятся в файле MMReg.h.

MMVERSION vDriverVersion;
Версия драйвера устройства при условии, что член dwType не равен константе MIXERLINE_TARGETTYPE_UNDEFINED.

CHAR szPname[MAXPNAMELEN];
Имя изделия при условии, что член dwType не равен константе MIXERLINE_TARGETTYPE_UNDEFINED.

Функция mixerGetLineInfo

Функция mixerGetLineInfo возвращает данные для линии и заполняет ими структуру MIXERLINE.

```
MMRESULT mixerGetLineInfo(
HMIXEROBJ hmxobj,
LPMIXERLINE pmxl,
DWORD fdwInfo
);
```

Параметры функции.

hmxobj
Данный параметр может интерпретироваться в зависимости от значения флага в третьем параметре и в общем случае представляет собой дескриптор объекта, который управляет указанной линией.

pmxl

Указатель на структуру MIXERLINE. Размер структуры cbStruct всегда должен быть инициализирован перед вызовом функции.

fdwInfo
Флаг определяет источник возвращаемой информации и ее состав.

Флаг может иметь следующие значения:

- MIXER_GETLINEINFOF_COMPONENTTYPE;
  Возвращаются данные для первой линии, указанной типом компонента dwComponentType. Остальные члены структуры не требуют инициализации, кроме cbStruct.

- MIXER_GETLINEINFOF_DESTINATION;
  Возвращаются данные для выходной линии, указанной членом структуры dwDestination. Остальные члены структуры не требуют инициализации, кроме cbStruct.

- MIXER_GETLINEINFOF_LINEID;
  Возвращаются данные для линии, указанной своим идентификатором dwLineID. Обычно используется для обновления информации о состоянии линии. Остальные члены структуры не требуют инициализации, кроме cbStruct.

- MIXER_GETLINEINFOF_SOURCE;
  Возвращаются данные для входной линии, которая указана своим источником сигнала (членом структуры dwSource) и выходной линией (членом структуры dwDestination). Общее число подключенных источников (dwSource) определяется при вызове функции с флагом MIXER_GETLINEINFOF_DESTINATION. Остальные члены структуры не требуют инициализации, кроме cbStruct.

- MIXER_GETLINEINFOF_TARGETTYPE;
  Возврат информации о линии определяемой по типу звукового устройства. Члены dwType, wMid, wPid, vDriverVersion и szPname структуры Target должны быть установлены перед вызовом функции. Все эти величины можно получить из структур свойств для медиа устройств. Остальные члены структуры не требуют инициализации, кроме cbStruct.

Флаги функции, перечисленные ниже, указывают, какому объекту принадлежит дескриптор.

- MIXER_OBJECTF_AUX
  Флаг указывает, что параметр hmxobj является идентификатором дополнительного звукового устройства. Значение hmxobj может быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией auxGetNumDevs.

- MIXER_OBJECTF_HMIDIIN
  Флаг указывает, что параметр hmxobj является дескриптором входного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIDIOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIXER
  Флаг указывает, что параметр hmxobj является дескриптором микшера. Дескриптор возвращается функцией mixerOpen. Флаг не обязателен.

- MIXER_OBJECTF_HWAVEIN
  Флаг указывает, что параметр hmxobj является дескриптором входного звукового волнового устройства. Дескриптор возвращается функцией waveInOpen.

- MIXER_OBJECTF_HWAVEOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного звукового волнового устройства. Дескриптор возвращается функцией waveOutOpen.

- MIXER_OBJECTF_MIDIIN
  Флаг указывает, что параметр hmxobj является идентификатором входного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiInGetNumDevs.

- MIXER_OBJECTF_MIDIOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiOutGetNumDevs.

- MIXER_OBJECTF_MIXER
  Флаг указывает, что параметр hmxobj является идентификатором микшера. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией mixerGetNumDevs. Флаг не обязателен.

- MIXER_OBJECTF_WAVEIN
  Флаг указывает, что параметр hmxobj является идентификатором входного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveInGetNumDevs.

- MIXER_OBJECTF_WAVEOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveOutGetNumDevs.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание
MIXERR_INVALCONTROL	Ссылка на орган управления неправильна.
MIXERR_INVALLINE	Ссылка на линию управления неправильна.
MMSYSERR_BADDEVICEID	Параметр hmxobj указывает неправильный идентификатор устройства.
MMSYSERR_INVALFLAG	Флаги, один или более, неправильные.
MMSYSERR_INVALHANDLE	Дескриптор неправильный.
MMSYSERR_INVALPARAM	Параметры, один или более, неправильные.
MMSYSERR_NODRIVER	Нет доступного микшера с параметром hmxobj.

Сбор данных об органах управления линии

После сбора данных для линии производится сбор данных об органах управления для этой линии. Сбор данных выполняет функция mixerGetLineControls, которая использует структуру MIXERLINECONTROLS как входной параметр, описывающий линию. Эта же структура используется как выходной параметр, который заполняется данными на определенный орган управления или на все органы управления линией сразу.

Структура MIXERLINECONTROLS.

Структура MIXERLINECONTROLS содержит данные для всех органов управления линии. Член структуры pamxctrl указывает на структуры MIXERCONTROL, каждая из которых содержит данные для определенного органа управления. Перед использованием, MIXERLINECONTROLS инициализируется данными общими для всех органов управления из структуры MIXERLINE. Заполняется структура при вызове функции mixerGetLineControls.

```
typedef struct {
    DWORD cbStruct;
    DWORD dwLineID;
    union {
        DWORD dwControlID;
        DWORD dwControlType;
    };
    DWORD cControls;
```

```
    DWORD cbmxctrl;
    LPMIXERCONTROL pamxctrl;
} MIXERLINECONTROLS;
```

Члены структуры.

cbStruct;
Размер структуры.

dwLineID;
Идентификатор линии.

dwControlID;
Идентификатор органа управления. Указывается для возврата данных функцией mixerGetLineControls по идентификатору органа управления.

dwControlType;
Тип органа управления. Указывается для возврата данных функцией mixerGetLineControls по типу органа управления.

cControls;
Общее количество органов управления на линии. Определяет число структур MIXERCONTROL. Инициализируется перед вызовом функции mixerGetLineControls. Не может быть нулем. Может быть единицей, в том случае, если параметры органа управления определяются по идентификатору или по типу. Если линия не имеет органа управления, функция не вызывается.

cbmxctrl;
Размер структуры MIXERCONTROL.

pamxctrl;
Указатель на массив структур MIXERCONTROL. Не может быть нулем. Инициализируется перед вызовом функции mixerGetLineControls. Инициализация структур MIXERCONTROL не требуется.

Структура MIXERCONTROL

Структура MIXERCONTROL описывает состояние и параметры органа управления для линии. Ни один член структуры не требует инициализации перед использованием.

```
typedef struct {
    DWORD cbStruct;
    DWORD dwControlID;
    DWORD dwControlType;
    DWORD fdwControl;
    DWORD cMultipleItems;
    CHAR  szShortName[MIXER_SHORT_NAME_CHARS];
```

```
    CHAR  szName[MIXER_LONG_NAME_CHARS];
    union {
        struct {
            LONG lMinimum;
            LONG lMaximum;
        };
        struct {
            DWORD dwMinimum;
            DWORD dwMaximum;
        };
        DWORD dwReserved[6];
    } Bounds;
    union {
        DWORD cSteps;
        DWORD cbCustomData;
        DWORD dwReserved[6];
    } Metrics;
} MIXERCONTROL, *PMIXERCONTROL, FAR *LPMIXERCONTROL;
```

Члены структуры.

cbStruct
Размер структуры в байтах.

dwControlID
Уникальный идентификатор органа управления в пределах микшера. Для одного микшера не может быть двух органов управления с одинаковым идентификатором. Может иметь любой формат. Приложение должно использовать этот идентификатор только как абстрактный дескриптор.

dwControlType
Класс (тип) органа управления. Всего имеется восемь классов органов управления, определяемые следующими константами.

1. MIXERCONTROL_CT_CLASS_CUSTOM
   Заказной класс органа управления.

2. MIXERCONTROL_CT_CLASS_FADER
   Класс регулятора. Под регулятором понимается орган управления, изменяющий значение определенной величины, например громкость. Располагается вертикально.

3. MIXERCONTROL_CT_CLASS_LIST
   Класс списка. Например, может представлять список входов микшера с возможностью выбора.

4. MIXERCONTROL_CT_CLASS_METER
   Класс измерителя. Например, может измеряться уровень сигнала, проходящий через линию.

5. MIXERCONTROL_CT_CLASS_NUMBER
   Класс числовых органов управления, которые позволяют вводить и отображать данные в числовом виде.

6. MIXERCONTROL_CT_CLASS_SLIDER
   Класс ползунков (слайдер). По принципу применения это тот же регулятор, расположенный горизонтально.

7. MIXERCONTROL_CT_CLASS_SWITCH
   Класс выключателей.

8. MIXERCONTROL_CT_CLASS_TIME
   Класс управления временем, например для установки задержки при использовании реверберации.

Подробное описание классов органов управления приводится далее.

fdwControl
Состояние и флаг поддержки для органа управления. Определены следующие величины.

- MIXERCONTROL_CONTROLF_DISABLED
  Орган управления запрещен и не может быть использован. Приложение может только читать установки, но не может устанавливать значение органа управления.

- MIXERCONTROL_CONTROLF_MULTIPLE
  Орган управления имеет два или более установочных параметра на канал. Например, эквалайзер.

- MIXERCONTROL_CONTROLF_UNIFORM
  Орган управления осуществляет одновременное и одинаковое воздействие на все каналы.

cMultipleItems
Число параметров управления на канал для органа управления, у которого член fdwControl равен MIXERCONTROL_CONTROLF_MULTIPLE. В остальных случаях этот член будет равным нулю.

szShortName
Короткое имя органа управления.

szName
Полное имя органа управления.

Bounds
Пределы изменения величины для органа управления.

- lMinimum
  Минимальная знаковая величина для органа управления, использующего знаковые (положительные и отрицательные) величины для установки состояния параметра.
- lMaximum
  Максимальная знаковая величина для органа управления, использующего знаковые (положительные и отрицательные) величины для установки состояния параметра.

- dwMinimum
  Минимальная положительная величина для органа управления, использующего только положительные величины для установки состояния параметра.
- dwMaximum
  Максимальная положительная величина для органа управления, использующего только положительные величины для установки состояния параметра.

Metrics
Параметры, определяющие пределы величин.

- cSteps
  Шаг изменения величины параметра органа управления.

- cbCustomData
  Размер в байтах, требуемый для хранения состояния заказного органа управления, относящегося к классу MIXERCONTROL_CONTROLTYPE_CUSTOM.

dwReserved
Резерв.

Классы (типы) органов управления микшером

Под классами органов управления следует понимать классификацию по типу применения органа управления. Эти классы не имеет ничего общего с классами языка C++. Терминология наследована из оригинальной документации.

Микшер включает следующие классы органов управления.

- Заказные (не типовые) органы управления.
  Заказные органы управления наиболее универсальны. Их характеристики и косвенно их визуальное представление определяет драйвер. Относятся к классу MIXERCONTROL_CT_CLASS_CUSTOM.

- Микшеры или регуляторы (Fader).
  Относятся к классу MIXERCONTROL_CT_CLASS_FADER.
  Имеют вертикальное расположение и равномерную шкалу. Использу-

ют структуру MIXERCONTROLDETAILS_UNSIGNED для возврата и установки значений. Диапазон изменения значений лежит в пределах от 0 до 65535.
В таблице описываются типы регуляторов этого класса.

Тип	Описание	Константа
Fader	Главный регулятор.	MIXERCONTROL_ CONTROLTYPE_ FADER
Volume	Главный регулятор. Диапазон значений может зависеть от драйвера устройства.	MIXERCONTROL_ CONTROLTYPE_ VOLUME
Bass	Регулятор нижних частот. Частотные характеристики определяются устройством.	MIXERCONTROL_ CONTROLTYPE_ BASS
Treble	Регулятор верхних частот. Частотные характеристики определяются устройством.	MIXERCONTROL_ CONTROLTYPE_ TREBLE
Equalizer	Графический эквалайзер. Число полос и другие характеристики определяются устройством. Для возврата текстовых меток эквалайзера используется структура MIXER-CONTROLDETAILS_ LISTTEXT.	MIXERCONTROL_ CONTROLTYPE_ EQUALIZER

Таблица 1. Регуляторы класса Fader.

- Списки (List).
  Обеспечивают одиночный или множественный выбор связанных звуковых линий. Эти органы управления используют структуру MIXERCONTROLDETAILS_BOOLEAN для установки и возврата значений. Для возврата текстовых описаний используется структура MIXERCONTROLDETAILS_LISTTEXT.
  Относятся к классу MIXERCONTROL_CT_CLASS_LIST. В таблице описываются типы органов управления этого класса.

Тип	Описание	Константа типа
Одиночный выбор	Список с одиночной селекцией. Может использоваться не только для выбора источников сигнала, но и для выбора других компонентов, например фильтров.	MIXERCONTROL_ CONTROLTYPE_ SINGLESELECT
Мультиплексор	Одиночный выбор линий.	MIXERCONTROL_

(MUX)		CONTROLTYPE_ SINGLESELECT
Множественная селекция	Список с множественной селекцией. Может использоваться не только для выбора источников сигнала, но и для выбора других компонентов.	MIXERCONTROL_ CONTROLTYPE_ MULTIPLESELECT
Микшер	Позволяет выбирать несколько линий одновременно.	MIXERCONTROL_ CONTROLTYPE_ MIXER

Таблица 2. Органы управления класса списков (List).

- Измерители (Meter).
  Относятся к классу MIXERCONTROL_CT_CLASS_METER. Определяют значения данных, проходящих через звуковую линию. Эти органы управления используют структуры MIXERCONTROLDETAILS_BOOLEAN, MIXERCONTROLDETAILS_SIGNED, и MIXERCONTROLDETAILS_UNSIGNED для возврата и установки данных. В таблице описываются типы органов управления этого класса.

Тип	Описание	Константа
Boolean	Указывает, что величина имеет значение 0 (FALSE) или не нуль (TRUE). Величина представляется знаковым целым (integer).	MIXERCONTROL_ CONTROLTYPE_ BOOLEANMETER
Peak	Измеряет величины в пределах от -32768 до 32767.	MIXERCONTROL_ CONTROLTYPE_ PEAKMETER
Signed	Измеряет целые знаковые величины (integer) в пределах от - 231 до (231 - 1). Пределы определяются драйвером устройства.	MIXERCONTROL_ CONTROLTYPE_ SIGNEDMETER
Unsigned	Измеряет целые беззнаковые величины в пределах от 0 до (232 - 1). Пределы определяются драйвером устройства.	MIXERCONTROL_ CONTROLTYPE_ UNSIGNEDMETER

Таблица 3. Органы управления класса измерителей (Meter).

- Числовые органы управления.
  Относятся к классу MIXERCONTROL_CT_CLASS_NUMBER. Предназначаются для ввода и вывода числовых данных. Эти органы управления используют структуры MIXERCONTROLDETAILS_SIGNED и MIXERCONTROLDETAILS_ UNSIGNED для возврата и установки данных. В таблице описываются типы органов управления этого класса.

Тип	Описание	Константа
Signed	Разрешен ввод данных от - 231 до (231 - 1).	MIXERCONTROL_ CONTROLTYPE_ SIGNED
Unsigned	Разрешен ввод данных от 0 до (232 - 1).	MIXERCONTROL_ CONTROLTYPE_ UNSIGNED
Decibel	Позволяет вводить данные в децибелах с десятыми долями. Диапазон величин от - 32768 до 32767.	MIXERCONTROL_ CONTROLTYPE_ DECIBELS
Percent	Ввод величин в процентах.	MIXERCONTROL_ CONTROLTYPE_ PERCENT

Таблица 4. Числовые органы управления.

- Ползунки (Sliders).
  Относятся к классу MIXERCONTROL_CT_CLASS_SLIDER. Ползунки располагаются горизонтально. Эти органы управления используют структуры MIXERCONTROLDETAILS_SIGNED для возврата и установки данных. В таблице описываются типы органов управления этого класса.

Тип	Описание	Константа
Slider	Имеет диапазон от - 32768 до 32767. Драйвер устройства определяет пределы регулирования.	MIXERCONTROL_ CONTROLTYPE_ SLIDER
Pan	Имеет диапазон от -32768 до 32767 с нулем посередине. Используется для регулировки стерео баланса. Драйвер устройства определяет пределы регулирования.	MIXERCONTROL_ CONTROLTYPE_ PAN
QSound Pan	Обеспечивает расширенное управление с использованием QSound. Диапазон от -15 до 15.	MIXERCONTROL_ CONTROLTYPE_ QSOUNDPAN

Таблица 5. Регуляторы класса Slider.

- Выключатели (Switches).
  Относятся к классу MIXERCONTROL_CT_CLASS_SWITCH. Имеют два состояния. Эти органы управления используют структуру MIXER-CONTROLDETAILS_BOOLEAN для возврата и установки данных. В таблице описываются типы органов управления этого класса.

Тип	Описание	Константа
Boolean	Имеет два значения TRUE или FALSE.	MIXERCONTROL_ CONTROLTYPE_ BOOLEAN
Button	При нажатии кнопки устанавливается в значение TRUE.	MIXERCONTROL_ CONTROLTYPE_ BUTTON
On/Off	Альтернативный выключатель, который графически отображается иначе, чем выключатель Boolean. Устанавливается в состояние ON или OFF.	MIXERCONTROL_ CONTROLTYPE_ ONOFF
Mute	Подавляет или разрешает прохождение данных в линию. Используется как вспомогательный выключатель для управления подачей данных в микшер.	MIXERCONTROL_ CONTROLTYPE_MUTE
Mono	Переключатель режимов моно или стерео.	MIXERCONTROL_ CONTROLTYPE_ MONO
Loudness	В положении ON увеличивается подъем низких частот. В положении OFF устанавливается нормальный уровень низких частот. Значение подъема низких частот определяется драйвером устройства.	MIXERCONTROL_ CONTROLTYPE_ LOUDNESS
Stereo Enhanced	Увеличение разделения стерео каналов. Установка в положение ON увеличивает разделение каналов.	MIXERCONTROL_ CONTROLTYPE_ STEREOENH

Таблица 6. Органы управления класса выключателей (Slider).

- Управление временем (Time Controls).
  Относятся к классу MIXERCONTROL_CT_CLASS_TIME . Предназначается для введения временной задержки, например при реверберации. В таблице описываются типы органов управления этого класса.

Тип	Описание	Константа
Microsecond	Данные вводятся в микросекундах в	MIXERCONTROL_

	диапазоне от 0 до (232 -1).	CONTROLTYPE_ MICROTIME
Millisecond	Данные вводятся в миллисекундах в диапазоне от 0 до (232 -1).	MIXERCONTROL_ CONTROLTYPE_ MILLITIME

Таблица 7. Органы управления временем.

Функция mixerGetLineControls

Функция mixerGetLineControls возвращает данные для одного или всех органов управления линией. Заполняет массив структур MIXERCONTROL.

```
MMRESULT mixerGetLineControls(
  HMIXEROBJ hmxobj,
  LPMIXERLINECONTROLS pmxlc,
  DWORD fdwControls
);
```

Параметры.

hmxobj
Дескриптор объекта микшера.

pmxlc
Указатель на структуру MIXERLINECONTROLS.

fdwControls
Значения флага, определяющие тип возвращаемой информации. Возможны следующие значения.

- MIXER_GETLINECONTROLSF_ALL
  Функция возвращает данные на все органы управления линией, заполняя массив структур MIXERCONTROL. Число элементов массива равно числу органов управления. В структуре MIXERLINECONTROLS должны быть предварительно инициализированы все члены, кроме dwControlID и dwControlType, которые игнорируются при этом запросе.

- MIXER_GETLINECONTROLSF_ONEBYID
  Функция возвращает данные на один орган управления, который указан своим идентификатором dwControlID. Заполняется одна структура MIXERCONTROL. В структуре MIXERLINECONTROLS должны быть предварительно инициализированы все члены, кроме dwLineID и dwControlType, которые игнорируются при этом запросе. Член структуры MIXERLINECONTROLS cControls должен быть установлен в единицу.

- MIXER_GETLINECONTROLSF_ONEBYTYPE
  Функция возвращает данные на первый орган управления лини-

ей, который указан своим типом dwControlType. Заполняется одна структура MIXERCONTROL. В структуре MIXERLINECONTROLS должны быть предварительно инициализированы все члены, кроме члена dwControlID, который игнорируется при этом запросе. Член структуры MIXERLINECONTROLS cControls должен быть установлен в единицу.

Флаги функции, перечисленные ниже, указывают, какому объекту принадлежит дескриптор.

- MIXER_OBJECTF_AUX
  Флаг указывает, что параметр hmxobj является идентификатором дополнительного звукового устройства. Значение hmxobj может быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией auxGetNumDevs.

- MIXER_OBJECTF_HMIDIIN
  Флаг указывает, что параметр hmxobj является дескриптором входного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIDIOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIXER
  Флаг указывает, что параметр hmxobj является дескриптором микшера. Дескриптор возвращается функцией mixerOpen. Флаг не обязателен.

- MIXER_OBJECTF_HWAVEIN
  Флаг указывает, что параметр hmxobj является дескриптором входного звукового волнового устройства. Дескриптор возвращается функцией waveInOpen.

- MIXER_OBJECTF_HWAVEOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного звукового волнового устройства. Дескриптор возвращается функцией waveOutOpen.

- MIXER_OBJECTF_MIDIIN
  Флаг указывает, что параметр hmxobj является идентификатором входного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiInGetNumDevs.

- MIXER_OBJECTF_MIDIOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного устройства MIDI. Идентификатор должен быть в

диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiOutGetNumDevs.

- MIXER_OBJECTF_MIXER
  Флаг указывает, что параметр hmxobj является идентификатором микшера. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией mixerGetNumDevs. Флаг не обязателен.

- MIXER_OBJECTF_WAVEIN
  Флаг указывает, что параметр hmxobj является идентификатором входного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveInGetNumDevs.

- MIXER_OBJECTF_WAVEOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveOutGetNumDevs.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание
MIXERR_INVALCONTROL	Ссылка на орган управления неправильна.
MIXERR_INVALLINE	Ссылка на линию управления неправильна.
MMSYSERR_BADDEVICEID	Параметр hmxobj указывает неправильный идентификатор устройства.
MMSYSERR_INVALFLAG	Флаги, один или более, неправильные
MMSYSERR_INVALHANDLE	Дескриптор неправильный.
MMSYSERR_INVALPARAM	Параметры, один или более, неправильные
MMSYSERR_NODRIVER	Нет доступного микшера с указанным дескриптором hmxobj.

Структура MIXERCONTROLDETAILS

Структура MIXERCONTROLDETAILS указывает на структуры, содержащие значения величин органов управления или их свойства. Все члены этой структуры должны быть инициализированы до вызова функции mixerGetControlDetails или mixerSetControlDetails.

```
typedef struct {
    DWORD cbStruct;
```

```
    DWORD dwControlID;
    DWORD cChannels;
    union {
        HWND  hwndOwner;
        DWORD cMultipleItems;
    };
    DWORD  cbDetails;
    LPVOID paDetails;
} MIXERCONTROLDETAILS;
```

Члены структуры.
cbStruct
Размер структуры.

dwControlID
Идентификатор органа управления, для которого устанавливается или возвращается значение.

cChannels
Число каналов управления. Определены следующие величины.

- 0
  Используется для органа управления с установленным флагом MIXERCONTROL_CONTROLTYPE_CUSTOM.

- 1
  Это значение устанавливается если орган управление имеет установленный флаг MIXERCONTROL_CONTROLF_UNIFORM, или если приложению требуется одновременное управление всеми каналами.

- MIXERLINE.cChannels
  Число каналов на линии. Это значение устанавливается при раздельном управлении каналами. Приложение не может устанавливать промежуточные значения между 1 и числом каналов на линии. Это значение не может быть 0 для всех органов управления кроме заказного органа управления (MIXERCONTROL_CONTROLTYPE_CUSTOM).

hwndOwner
Дескриптор окна, в котором отображается диалоговое окно микшера. Этот параметр используется при указании флага MIXER_SETCONTROL-DETAILSF_CUSTOM в функции mixerSetControlDetails.

cMultipleItems
Число элементов в канале органа управления, для которых производится установка или возврат значений. Определены следующие значения:

- 0
  Используется для всех типов органов управления исключая

органы управления типа MIXERCONTROL_CONTROLF_MULTIPLE или MIXERCONTROL_CONTROLTYPE_CUSTOM.

- Значение члена cMultipleItems структуры MIXERCONTROL. Используется для органов управления класса MIXERCONTROL_CONTROLF_MULTIPLE.

- Значение члена hwndOwner структуры MIXERCONTROLDETAILS. Используется для заказных органов управления MIXERCONTROL_CONTROLTYPE_CUSTOM и при указании флага MIXER_SETCONTROLDETAILSF_CUSTOM в функции mixerSetControlDetails. В этом случае производится перегрузка значения hwndOwner, указанного в cMultipleItems. При использовании заказного органа управления (MIXERCONTROL_CONTROLTYPE_CUSTOM) без флага MIXERCONTROL_CONTROLTYPE_CUSTOM член hwndOwner устанавливается в 0. Приложение не может указывать никаких других значений для органа управления MIXERCONTROL_CONTROLF_MULTIPLE кроме величин указанных в члене cMultipleItems структуры MIXERCONTROL.

cbDetails
Размер в байтах одной из следующих структур для установки свойств органа управления или его величины:
MIXERCONTROLDETAILS_BOOLEAN,
MIXERCONTROLDETAILS_LISTTEXT,
MIXERCONTROLDETAILS_SIGNED,
MIXERCONTROLDETAILS_UNSIGNED.
Если орган управления заказной (MIXERCONTROL_CONTROLTYPE_CUSTOM), то этот член должен быть равный члену cbCustomData структуры MIXERCONTROL.

paDetails
Указатель на массив структур установки или возврата свойств органа управления.

Примечание.
Для органа управления типа MIXERCONTROL_CONTROLF_MULTIPLE размер буфера вычисляется как произведение членов структуры MIXERCONTROLDETAILS cChannels, cMultipleItems и cbDetails.
Для остальных органов управления размер буфера определяется как произведение членов структуры MIXERCONTROLDETAILS cChannels и cbDetails.
Для органа управления типа MIXERCONTROL_CONTROLF_MULTIPLE массив может обрабатываться как двухразмерный для левого и правого каналов.
Для органов управления других типов индекс элемента эквивалентен каналу, paDetails[0] левый канал и paDetails[1] правый канал.
Для заказного органа управления этот член должен указывать на буфер с размером не менее чем, указанный в члене cbCustomData структуры MIXERCONTROL.

Установка свойств и значений органов управления

После сбора данных об определенной линии и органах управления на ней можно возвращать состояние определенного органа управления или устанавливать его. Эту задачу выполняет пара функций mixerSetControlDetails и mixerGetControlDetails. В качестве входного и выходного параметра функции используют структуру MIXERCONTROLDETAILS, которая описывает тип органа управления. В зависимости от типа органа управления функции используют структуру специфическую для данного органа управления для установки или возврата его состояния. Всего существует четыре типа структур, которые описываются ниже.

Структуры установки свойств и значений органов управления

Структура MIXERCONTROLDETAILS_BOOLEAN

Структура возвращает и устанавливает логические значения органа управления.

```
typedef struct {
    LONG fValue;
} MIXERCONTROLDETAILS_BOOLEAN;

Члены структуры.

fValue
Значение нуль означает FALSE и не нуль означает TRUE.
```

Следующие стандартные органы управления используют эту структуру.

Измерители (meter):
MIXERCONTROL_CONTROLTYPE_BOOLEANMETER

Выключатели (Switch):
MIXERCONTROL_CONTROLTYPE_BOOLEAN
MIXERCONTROL_CONTROLTYPE_BUTTON
MIXERCONTROL_CONTROLTYPE_LOUDNESS
MIXERCONTROL_CONTROLTYPE_MONO
MIXERCONTROL_CONTROLTYPE_MUTE
MIXERCONTROL_CONTROLTYPE_ONOFF
MIXERCONTROL_CONTROLTYPE_STEREOENH

Списки (List):

```
MIXERCONTROL_CONTROLTYPE_MIXER
MIXERCONTROL_CONTROLTYPE_MULTIPLESELECT
MIXERCONTROL_CONTROLTYPE_MUX
MIXERCONTROL_CONTROLTYPE_SINGLESELECT
```

Структура MIXERCONTROLDETAILS_LISTTEXT

Структура возвращает текстовый список, текстовые метки, и (или) информацию о диапазоне (полосе) для органов управления, имеющих несколько полос управления.

```
typedef struct {
    DWORD dwParam1;
    DWORD dwParam2;
    CHAR  szName[MIXER_LONG_NAME_CHARS];
} MIXERCONTROLDETAILS_LISTTEXT;
```

Члены структуры.

dwParam1
Класс органа управления. Определены следующие типы:

- EQUALIZER
  Член Bounds dwMinimum структуры MIXERCONTROL.

- MIXER и MUX
  Член dwLineID структуры MIXERLINE.

- MULTIPLESELECT и SINGLESELECT
  Не определен, должен быть равен нулю.

dwParam2
То же, что и dwParam1.

szName
Имя одного пункта списка. Этот текст может использоваться как метка класса органа управления.

Следующие стандартные органы управления используют эту структуру.

Регуляторы (Fader):
MIXERCONTROL_CONTROLTYPE_EQUALIZER

Списки:
MIXERCONTROL_CONTROLTYPE_MIXER
MIXERCONTROL_CONTROLTYPE_MULTIPLESELECT
MIXERCONTROL_CONTROLTYPE_MUX
MIXERCONTROL_CONTROLTYPE_SINGLESELECT

Структура MIXERCONTROLDETAILS_SIGNED

Структура возвращает и устанавливает положительные и отрицательные значения органа управления.

```
typedef struct {
    LONG lValue;
} MIXERCONTROLDETAILS_SIGNED;
```

Члены структуры.

fValue
Величина типа LONG для установки и возврата значений органа управления.

Следующие стандартные органы управления используют эту структуру.

Измерители (meter):
MIXERCONTROL_CONTROLTYPE_PEAKMETER
MIXERCONTROL_CONTROLTYPE_SIGNEDMETER

Числовые (Number):
MIXERCONTROL_CONTROLTYPE_SIGNED
MIXERCONTROL_CONTROLTYPE_DECIBELS

Ползунки (Slider):
MIXERCONTROL_CONTROLTYPE_PAN
MIXERCONTROL_CONTROLTYPE_QSOUNDPAN
MIXERCONTROL_CONTROLTYPE_SLIDER
MIXERCONTROLDETAILS_UNSIGNED

Структура MIXERCONTROLDETAILS_UNSIGNED

Структура возвращает и устанавливает положительные значения органа управления.

```
typedef struct {
    DWORD dwValue;
} MIXERCONTROLDETAILS_UNSIGNED;
```

Члены структуры.

dwValue
Положительная величина для одиночного пункта или канала управления.

Следующие стандартные органы управления используют эту структуру.

Измерители (meter):

MIXERCONTROL_CONTROLTYPE_UNSIGNEDMETER

Числовые (Number):
MIXERCONTROL_CONTROLTYPE_UNSIGNED

Регуляторы (Fader):
MIXERCONTROL_CONTROLTYPE_BASS
MIXERCONTROL_CONTROLTYPE_EQUALIZER
MIXERCONTROL_CONTROLTYPE_FADER
MIXERCONTROL_CONTROLTYPE_TREBLE
MIXERCONTROL_CONTROLTYPE_VOLUME

Установка времени (Time):
MIXERCONTROL_CONTROLTYPE_MICROTIME
MIXERCONTROL_CONTROLTYPE_MILLITIME
MIXERCONTROL_CONTROLTYPE_PERCENT

Функция mixerSetControlDetails

Функция mixerSetControlDetails устанавливает величину или свойство одного органа управления на линии.

```
MMRESULT mixerSetControlDetails(
  HMIXEROBJ hmxobj,
  LPMIXERCONTROLDETAILS pmxcd,
  DWORD fdwDetails
);
```

Параметры.

hmxobj
Дескриптор объекта микшера.

pmxcd
Указатель на структуру MIXERCONTROLDETAILS.

fdwDetails
Флаг, определяющий режим установки свойств органа управления или величины.

- MIXER_SETCONTROLDETAILSF_CUSTOM
  Отображается заказной диалог для заказного микшера. Микшер производит сбор информации от пользователя и возвращает данные в указанный буфер. Дескриптор окна указывается в структуре MIXERCONTROLDETAILS. Он может быть равным нулю. Приложение может сохранять данные из диалога и восстанавливать настройки, используя флаг MIXER_SETCONTROLDETAILSF_VALUE.

- MIXER_SETCONTROLDETAILSF_VALUE
  Текущая устанавливаемая величина (величины) для органа управления. Указатель paDetails (структура MIXERCONTROLDETAILS) указывает на одну или более структур MIXERCONTROLDETAILS_XXX, которые содержат величины соответствующие классу органа управления.

Флаги функции, перечисленные ниже, указывают, какому объекту принадлежит дескриптор.

- MIXER_OBJECTF_AUX
  Флаг указывает, что параметр hmxobj является идентификатором дополнительного звукового устройства. Значение hmxobj может быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией auxGetNumDevs.

- MIXER_OBJECTF_HMIDIIN
  Флаг указывает, что параметр hmxobj является дескриптором входного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIDIOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIXER
  Флаг указывает, что параметр hmxobj является дескриптором микшера. Дескриптор возвращается функцией mixerOpen. Флаг не обязателен.

- MIXER_OBJECTF_HWAVEIN
  Флаг указывает, что параметр hmxobj является дескриптором входного звукового волнового устройства. Дескриптор возвращается функцией waveInOpen.

- MIXER_OBJECTF_HWAVEOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного звукового волнового устройства. Дескриптор возвращается функцией waveOutOpen.

- MIXER_OBJECTF_MIDIIN
  Флаг указывает, что параметр hmxobj является идентификатором входного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiInGetNumDevs.

- MIXER_OBJECTF_MIDIOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного устройства MIDI. Идентификатор должен быть в

диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiOutGetNumDevs.

- MIXER_OBJECTF_MIXER
  Флаг указывает, что параметр hmxobj является идентификатором микшера. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией mixerGetNumDevs. Флаг не обязателен.

- MIXER_OBJECTF_WAVEIN
  Флаг указывает, что параметр hmxobj является идентификатором входного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveInGetNumDevs.

- MIXER_OBJECTF_WAVEOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveOutGetNumDevs.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание
MIXERR_INVALCONTROL	Ссылка на орган управления неправильная.
MMSYSERR_BADDEVICEID	Параметр hmxobj указывает неправильный идентификатор устройства.
MMSYSERR_INVALFLAG	Флаги, один или более, неправильные
MMSYSERR_INVALHANDLE	Дескриптор неправильный.
MMSYSERR_INVALPARAM	Параметры, один или более, неправильные
MMSYSERR_NODRIVER	Нет доступного микшера с указанным дескриптором hmxobj.

Функция mixerGetControlDetails

Функция mixerGetControlDetails возвращает величину или свойство одного органа управления на линии.

```
MMRESULT mixerGetControlDetails(
  HMIXEROBJ hmxobj,
  LPMIXERCONTROLDETAILS pmxcd,
  DWORD fdwDetails
);
```

Параметры.

hmxobj
Дескриптор объекта микшера.

pmxcd
Указатель на структуру MIXERCONTROLDETAILS.

fdwDetails
Флаг, определяющий режим возврата свойств органа управления или величины.

- MIXER_GETCONTROLDETAILSF_LISTTEXT
  Возвращаются списки текстовых меток. Указатель paDetails (структура MIXERCONTROLDETAILS) указывает на одну или более структур MIXERCONTROLDETAILS_LISTTEXT, которые содержат текстовые метки для пунктов органа управления. Приложение должно возвратить все списки текстовых меток одновременно. Этот флаг не может использоваться с флагом MIXERCONTROL_CONTROLTYPE_CUSTOM.

- MIXER_GETCONTROLDETAILSF_VALUE
  Возвращается текущая величина (величины) для органа управления. Указатель paDetails (структура MIXERCONTROLDETAILS) указывает на одну или более структур MIXERCONTROLDETAILS_XXX, которые содержат величины соответствующие классу органа управления.

Флаги функции, перечисленные ниже, указывают, какому объекту принадлежит дескриптор.

- MIXER_OBJECTF_AUX
  Флаг указывает, что параметр hmxobj является идентификатором дополнительного звукового устройства. Значение hmxobj может быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией auxGetNumDevs.

- MIXER_OBJECTF_HMIDIIN
  Флаг указывает, что параметр hmxobj является дескриптором входного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIDIOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIXER
  Флаг указывает, что параметр hmxobj является дескриптором

микшера. Дескриптор возвращается функцией mixerOpen. Флаг не обязателен.

- MIXER_OBJECTF_HWAVEIN
  Флаг указывает, что параметр hmxobj является дескриптором входного звукового волнового устройства. Дескриптор возвращается функцией waveInOpen.

- MIXER_OBJECTF_HWAVEOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного звукового волнового устройства. Дескриптор возвращается функцией waveOutOpen.

- MIXER_OBJECTF_MIDIIN
  Флаг указывает, что параметр hmxobj является идентификатором входного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiInGetNumDevs.

- MIXER_OBJECTF_MIDIOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiOutGetNumDevs.

- MIXER_OBJECTF_MIXER
  Флаг указывает, что параметр hmxobj является идентификатором микшера. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией mixerGetNumDevs. Флаг не обязателен.

- MIXER_OBJECTF_WAVEIN
  Флаг указывает, что параметр hmxobj является идентификатором входного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveInGetNumDevs.

- MIXER_OBJECTF_WAVEOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveOutGetNumDevs.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание

MIXERR_INVALCONTROL	Ссылка на орган управления неправильна.
MMSYSERR_BADDEVICEID	Параметр hmxobj указывает неправильный идентификатор устройства.
MMSYSERR_INVALFLAG	Флаги, один или более, неправильные
MMSYSERR_INVALHANDLE	Дескриптор неправильный.
MMSYSERR_INVALPARAM	Параметры, один или более, неправильные
MMSYSERR_NODRIVER	Нет доступного микшера с указанным дескриптором hmxobj.

Структуры и функции для общего управления микшером

Общее управление микшером осуществляют пять функций mixerGetNumDevs, mixerGetDevCaps, mixerOpen, mixerClose, mixerGetID и структуру MIXERCAPS.

Они выполняют следующие задачи:

1. Определение числа микшеров в системе.
2. Определение общих свойств микшера.
3. Открытие микшера.
4. Закрытие микшера.
5. Определение идентификатора микшера.

Структура MIXERCAPS

Структура MIXERCAPS описывает общие свойства микшера.

```
typedef struct {
    WORD    wMid;
    WORD    wPid;
    MMVERSION vDriverVersion;
    CHAR    szPname[MAXPNAMELEN];
    DWORD   fdwSupport;
    DWORD   cDestinations;
} MIXERCAPS;
```

Члены структуры.
wMid
Идентификатор производителя. Он определен в файле MMReg.h, или в документации в параграфе Manufacturer and Product Identifiers.

wPid

Идентификатор изделия. Он определен в файле MMReg.h, или в документации в параграфе Manufacturer and Product Identifiers.

vDriverVersion
Версия драйвера. В старшем байте старший номер версии, в младшем младшая часть номера версии.

szPname
Имя изделия (звуковой карты). Если драйвер поддерживает много карт, строка должна предоставлять имя используемой карты.

fdwSupport
Информация поддержки для микшера. Не определена..

cDestinations
Число выходных линий микшера. Не может быть нулем, так как микшер должен иметь хотя бы одну выходную линию.

Функция mixerGetDevCaps

Заполняет структуру свойств микшера.

```
MMRESULT mixerGetDevCaps(
  UINT_PTR    uMxId,
  LPMIXERCAPS pmxcaps,
  UINT        cbmxcaps
);
```

Параметры.
uMxId
Идентификатор или дескриптор открытого микшера. Дескриптор микшера должен быть преобразован приложением в тип UINT_PTR.

pmxcaps
Указатель на заполняемую структуру MIXERCAPS.

cbmxcaps
Размер структуры MIXERCAPS в байтах.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание
MMSYSERR_BADDEVICEID	Параметр hmxobj указывает неправильный идентификатор устройства.
MMSYSERR_INVALHANDLE	Дескриптор неправильный.
MMSYSERR_INVALPARAM	Параметры, один или более, неправильные

Функция mixerGetNumDevs

Возвращает число устройств, представленных в системе или нуль при их отсутствии или недоступности.

UINT mixerGetNumDevs(VOID);

Функция mixerClose

Закрывает микшер.

```
MMRESULT mixerClose(
  HMIXER hmx
);
```

Параметр.

hmx
Дескриптор микшера, возвращенный функцией mixerOpen. После закрытия микшера дескриптор становится недействительным.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание
MMSYSERR_INVALHANDLE	Дескриптор неправильный.

Функция mixerOpen

Открывает микшер и устанавливает дескриптор микшера.

```
MMRESULT mixerOpen(
  LPHMIXER  phmx,
  UINT      uMxId,
  DWORD_PTR dwCallback,
  DWORD_PTR dwInstance,
  DWORD     fdwOpen
);
```

Параметры.

phmx
Указатель на переменную дескриптора, после безошибочного выполнения функции переменная не должна быть равна нулю.

uMxId

Идентификатор открываемого микшера. Идентификатор микшера можно получить, вызвав функцию mixerGetID для открытого звукового объекта или указать номер микшера, считая от нуля. Номер ноль принадлежит основной звуковой карте в системе. Максимальное число микшеров в системе можно определить, вызвав функцию mixerGetNumDevs. Карты (mapper) микшеров не существует в системе и идентификатор -1 использовать нельзя.

dwCallback
Дескриптор окна, в котором будут обрабатываться сообщения микшера. Механизм функции callback для микшера не используется, сообщения микшера обрабатываются в оконной процедуре или в обработчиках сообщений MFC. Если сообщения не будут обрабатываться параметр должен быть установлен в нуль.

dwInstance
Не используется.

fdwOpen
Флаги режима открывания микшера.

- CALLBACK_WINDOW
  Флаг указывает, что параметр dwCallback является дескриптором окна.

  Флаги функции, перечисленные ниже, указывают, какому объекту принадлежит дескриптор.

- MIXER_OBJECTF_AUX
  Флаг указывает, что параметр hmxobj является идентификатором дополнительного звукового устройства. Значение hmxobj может быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией auxGetNumDevs.

- MIXER_OBJECTF_HMIDIIN
  Флаг указывает, что параметр hmxobj является дескриптором входного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIDIOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIXER
  Флаг указывает, что параметр hmxobj является дескриптором микшера. Дескриптор возвращается функцией mixerOpen. Флаг не обязателен.

- MIXER_OBJECTF_HWAVEIN
  Флаг указывает, что параметр hmxobj является дескриптором

входного звукового волнового устройства. Дескриптор возвращается функцией waveInOpen.

- MIXER_OBJECTF_HWAVEOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного звукового волнового устройства. Дескриптор возвращается функцией waveOutOpen.

- MIXER_OBJECTF_MIDIIN
  Флаг указывает, что параметр hmxobj является идентификатором входного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiInGetNumDevs.

- MIXER_OBJECTF_MIDIOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiOutGetNumDevs.

- MIXER_OBJECTF_MIXER
  Флаг указывает, что параметр hmxobj является идентификатором микшера. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией mixerGetNumDevs. Флаг не обязателен.

- MIXER_OBJECTF_WAVEIN
  Флаг указывает, что параметр hmxobj является идентификатором входного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveInGetNumDevs.

- MIXER_OBJECTF_WAVEOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveOutGetNumDevs.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание
MMSYSERR_ALLOCATED	Ресурс уже используется максимально возможным количеством клиентов.
MMSYSERR_BADDEVICEID	Параметр hmxobj указывает неправильный идентификатор устройства.
MMSYSERR_INVALFLAG	Флаги, один или более, неправильные.

MMSYSERR_INVALHANDLE	Дескриптор неправильный.
MMSYSERR_INVALPARAM	Параметры, один или более, неправильные.
MMSYSERR_NODRIVER	Нет доступного микшера с указанным дескриптором hmxobj.
MMSYSERR_NOMEM	Ресурс недоступен для распределения.

Функция mixerGetID

Возвращает идентификатор микшера, который ассоциируется с указанным дескриптором устройства.

```
MMRESULT mixerGetID(
  HMIXEROBJ  hmxobj,
  UINT FAR * puMxId,
  DWORD      fdwId
);
```

Параметры.

hmxobj
Дескриптор объекта микшера указывает на связь с идентификатором устройства.

puMxId
Указатель на переменную, принимающую идентификатор микшера. Если произошла ошибка, идентификатору присваивается значение -1 и код возврата функции равен MMSYSERR_NODRIVER.

fdwId
Флаги функции, перечисленные ниже, указывают, какому объекту принадлежит дескриптор.

- MIXER_OBJECTF_AUX
  Флаг указывает, что параметр hmxobj является идентификатором дополнительного звукового устройства. Значение hmxobj может быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией auxGetNumDevs.

- MIXER_OBJECTF_HMIDIIN
  Флаг указывает, что параметр hmxobj является дескриптором входного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIDIOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного устройства MIDI. Дескриптор возвращается функцией midiInOpen.

- MIXER_OBJECTF_HMIXER
  Флаг указывает, что параметр hmxobj является дескриптором микшера. Дескриптор возвращается функцией mixerOpen. Флаг не обязателен.

- MIXER_OBJECTF_HWAVEIN
  Флаг указывает, что параметр hmxobj является дескриптором входного звукового волнового устройства. Дескриптор возвращается функцией waveInOpen.

- MIXER_OBJECTF_HWAVEOUT
  Флаг указывает, что параметр hmxobj является дескриптором выходного звукового волнового устройства. Дескриптор возвращается функцией waveOutOpen.

- MIXER_OBJECTF_MIDIIN
  Флаг указывает, что параметр hmxobj является идентификатором входного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiInGetNumDevs.

- MIXER_OBJECTF_MIDIOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного устройства MIDI. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией midiOutGetNumDevs.

- MIXER_OBJECTF_MIXER
  Флаг указывает, что параметр hmxobj является идентификатором микшера. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией mixerGetNumDevs. Флаг не обязателен.

- MIXER_OBJECTF_WAVEIN
  Флаг указывает, что параметр hmxobj является идентификатором входного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveInGetNumDevs.

- MIXER_OBJECTF_WAVEOUT
  Флаг указывает, что параметр hmxobj является идентификатором выходного волнового звукового устройства. Идентификатор должен быть в диапазоне от нуля, до величины на единицу меньшей, чем величина, возвращаемая функцией waveOutGetNumDevs.

Возвращаемые значения.
Возврат MMSYSERR_NOERROR означает выполнение без ошибки.

Величина	Описание
MMSYSERR_BADDEVICEID	Параметр hmxobj указывает неправильный идентификатор устройства.
MMSYSERR_INVALFLAG	Один или более флагов неправильные.
MMSYSERR_INVALHANDLE	Дескриптор неправильный.
MMSYSERR_INVALPARAM	Один или более параметров неправильные.
MMSYSERR_NODRIVER	Нет доступного микшера с указанным дескриптором hmxobj. Идентификатор устанавливается в -1.

Сообщения микшера

Микшер посылает приложению два сообщения MM_MIXM_CONTROL_CHANGE и MM_MIXM_LINE_CHANGE, которые могут быть обработаны в оконной процедуре или в обработчике сообщения MFC. Дескриптор окна указывается при открытии микшера. Механизм функции callback для микшера не используется.

Сообщение MM_MIXM_CONTROL_CHANGE посылается микшером для уведомления приложения о том, что состояние органа управления изменилось. Приложение должно отобразить изменение состояния своего органа управления. В результате при изменении значения органа управления другим приложением, например системной программой, орган управления микшером вашей программы также будет устанавливаться в соответствующее положение. Следует отметить, что при изменении состояния органа управления своим приложением будет получено сообщение микшера, что следует учитывать при написании обработчика сообщения. Обработчик сообщения не должен изменять состояние органа управления микшера, он должен только отобразить в интерфейсе пользователя новое состояние.

```
MM_MIXM_CONTROL_CHANGE
wParam = (WPARAM) hMixer
lParam = (LPARAM) dwControlID

Параметры сообщения.
hMixer
Дескриптор  микшера, который послал сообщение.

dwControlID
Идентификатор органа управления, состояние которого изменилось.
Этот идентификатор соответствует члену dwControlID структуры
MIXERCONTROL.
```

Сообщение MM_MIXM_LINE_CHANGE посылается микшером для уведомления приложения о том, что состояние линии изменилось. Приложение должно отобразить это изменение состояния линии.

```
MM_MIXM_LINE_CHANGE
wParam = (WPARAM) hMixer
lParam = (LPARAM) dwLineID

Параметры сообщения.
hMixer
Дескриптор  микшера, который послал сообщение.

dwLineID
Идентификатор линии, состояние которого изменилось. Этот идентификатор соответствует члену dwLineID структуры MIXERLINE.
```

Пользовательское сообщение для заказного микшера

Для заказного микшера существует функция, которая посылает пользовательское сообщение драйверу заказного микшера. Сообщение может посылаться только драйверу микшера, который поддерживает сообщения. Приложение должно предварительно проверить драйвер на соответствие, проверив члены wMid, wPid, vDriverVersion, и szPname структуры MIXERCAPS.

```
DWORD mixerMessage(
  HMIXER    driverID,
  UINT      uMsg,
  DWORD_PTR dwParam1,
  DWORD_PTR dwParam2
);

Параметры.
driverID
Идентификатор микшера, который примет сообщение. Он является
идентификатором устройства звуковой карты преобразованным в тип
HMIXER. Нельзя использовать дескриптор микшера, функция будет
возвращать ошибку MMSYSERR_NOSUPPORT.

uMsg
Сообщение, посылаемое заказному микшеру. Значение сообщения
должно быть равным или большим константы MXDM_USER.

dwParam1
Параметр сообщения.

dwParam2
Параметр сообщения.
```

Возвращаемые величины.

Величина	Описание
MMSYSERR_INVALHANDLE	Неправильный дескриптор устройства.
MMSYSERR_INVALPARAM	Параметр сообщения неправильный.
MMSYSERR_NOSUPPORT	Параметр deviceID должен быть идентификатором устройства.
MMSYSERR_NOTSUPPORTED	Микшер не обработал сообщение.

Классы микшера

В приложении, для управления параметрами звука, чаще всего требуются отдельные органы управления микшером. Даже в приложении, предназначенном для обработки звука, не всегда требуется вся совокупность органов управления микшером. Поэтому основной задачей разрабатываемых классов микшера будет создание органов управления для одной линии. Каждый объект класса управляет только своей линией.

Создание отдельного органа управления линии микшера постараемся максимально упростить. Он должен устанавливаться динамически в окне класса CStatic, что позволить легко управлять видом диалога и уменьшит число параметров при создании органа управления. Часто, для упрощения, в диалоге устанавливают шаблоны всех органов управления. Затем при инициализации отсутствующие органы управления делают недоступными. В результате диалог приобретает неряшливый вид, путает пользователя. При динамической установке органа управления, если он отсутствует для линии, орган управления в окно класса CStatic не устанавливается. Окно можно либо удалить, сделать невидимым или управлять иным способом. Если устанавливается несколько органов управления и промежуточный в последовательности орган управления отсутствует, то в его окно можно установить следующий по номеру орган управления. Последние ненужные окна просто скрываются или удаляются.

Сообщения от органов управления должны обрабатываться в собственных классах, а не в классах приложения. Это позволит упростить использование классов микшера и сосредоточить обработку сообщений от органов управления в одном месте.

Большое количество структур микшера и их сложность определяет достаточно сложную реализацию методов классов для управления элементами микшера. С целью упрощения реализации классов и придания им максимально простой логики построения было решено все данные для используемых органов управления сохранять в векторе, а для операций обработки данных использовать алгоритмы стандартной библиотеки STL. Классы микшера будут разрабатываться с определенными ограничениями по функциональности,

только для обеспечения потребностей проекта DirectShow. При необходимости их можно легко расширить и придать дополнительные возможности.

Для демонстрации проектирования классов микшера и их использования создадим отдельный проект на основе диалогового окна под именем AudioMixer.

Структура классов звукового микшера.

Логику реализации классов микшера диктует логика управления микшером с помощью рассмотренных выше структур и функций API. Все структуры и функции API явно делятся на три основные группы.

1. Функции и структуры, предназначенные для открытия микшера и получения общей информации.
2. Функции и структуры, предназначенные для управления органами микшера.
3. Обработка сообщений.

В соответствии с этой логикой классы управления микшером можно организовать следующим образом.

- Базовый класс, который отвечает за открытие микшера и создание фиктивного (скрытого) окна для обработки сообщений микшера. В базовом классе также производится сбор данных обо всех органах управления микшера.
- Производный класс, который отвечает за установку всех органов управления для одной линии. Линия определяется по типу компонента, подключенного к ней. Например, линия микрофона.
- Классы органов управления. Микшер предусматривает достаточно много разнообразных типов органов управления. Наиболее часто для управления линией используется регулятор усиления и выключатель сигнала. Поэтому все разнообразие регуляторов можно описать в классе регуляторов, а различные типы выключателей и переключателей в классе выключателей. Эти два типа органов управления являются основой для реализации более сложных органов управления, например мультиплексора.

Обработка сообщений микшера будет производиться в классе фиктивного окна, поскольку класс микшера не имеет своего окна. Указатель на фиктивное окно является членом базового класса CMixer. Такое решение вызвано тем, что дескриптор окна, в котором происходит обработка сообщений, должен быть передан в микшер при его открытии. Следовательно, окно должно быть создано и существовать до открытия микшера. Использование других окон для обработки сообщений, не связанных с классом микшера, повлечет за собой потерю автономности классов микшера.

Взаимодействие классов в общем виде в процессе создания органа управления можно представить следующим образом:

1. Метод базового класса CMixer открывает микшер и создает фиктивное окно для обработки сообщений микшера. Если параметр микшера изменяется посторонней программой, например системной, то от него приходит сообщение, которое обрабатывается в фиктивном окне. При этом вызывается метод, отображающий изменение положения регулятора. После открытия микшера производится сбор данных обо всех органах управления микшером. Данные сохраняются в векторе члене класса и используются для создания вектора данных установленных органов управления.
2. Метод производного класса CLineMixer использует вектор данных базового класса для установки органа управления. Данные на каждый установленный орган управления сохраняются в векторе члене класса. После установки всех необходимых органов управления вектор базового класса может быть очищен для освобождения памяти. Орган управления устанавливается во внешнем окне, которое должен быть создано в классе приложения. Это может быть элемент GroupBox в окне диалога или любое другое дочернее окно. Метод класса CLineMixer производит поиск в векторе данных базового класса необходимых органов управления для линии, создает их динамически и, используя дескриптор внешнего окна, устанавливает их в это окно.
3. При манипуляции органом управления, сообщение от него обрабатывается в классе органа управления. В обработчике сообщения вызывается метод из класса, который использует орган управления, для установки значения параметра микшера.

На рисунке 6.3 показана схема взаимодействия классов микшера.

Базовый класс CMixer

Основные задачи, возлагаемые на базовый класс, это:
открытие указанного микшера в режиме обработки сообщений, инициализация и сбор данных обо всех органах управления микшера, создание фиктивного окна и обработка сообщений микшера. С целью упрощения использования класса CMixer, открытие микшера и инициализация и сбор данных производится неявно при создании объекта производного класса. Производный класс отвечает за установку органа управления для определенной линии. Метод класса OpenMixer, а также конструктор являются закрытыми членами класса. В классе определим чистую виртуальную функцию, которая будет вызываться в производном классе при обработке сообщения микшера. Таким образом, класс становится абстрактным и не может быть использован прямо.

Остальные члены класса, предоставляющие общую информацию, являются открытыми.

```
class CMixer : public CObject
{
  DECLARE_DYNAMIC(CMixer)
protected:
  CMixer(void);

public:
  virtual~CMixer(void);

  //дескриптор открытого микшера
  HMIXER m_hMix;

  //Вызывается для отображения состояния органа
  //управления при получении сообщения от системного //микшера
  virtual void OnSetControl(HMIXER hMix
                          ,DWORD ctrlID)=0;

  //Возвращает информацию об источниках сигнала
  //для выходной линии
  BOOL GetInfoSourceComponent(DWORD dest);

  //Возвращает число микшеров в системе
  UINT GetNumMix(void);

  //Возвращает свойства указанного микшера
  BOOL GetMixCaps(UINT mixID
                ,MIXERCAPS & mixCaps/*out*/);

  //Вектор данных для всех органов управления микшера
  vector<DATA_CTRL> m_vCtrlAll;

  //Заполняет вектор данных для всех органов управления
  //выходной линии, определенной компонентом, и для
  //входных линий, которые связаны с выходной.
  BOOL GetAllControlsForComponentDST (
                           DWORD compTypeDST);

protected:
  //Идентификатор используемого микшера
  DWORD m_mixID;

  //Инициализация микшера
  BOOL OpenMixer(HWND m_hwndMsg);

  //Указатель на окно принимающее сообщение
  CMsgWnd* m_pMsgWnd;

  //Число выходных линий
  DWORD m_cDestinations;
```

```
  //Добавление в вектор данных для органов управления
  //линии
  BOOL AddVectorCtrlLine(MIXERLINE& mixLine);

public:
  //Очистка вектора данных органов управления
  void Clear(void);
};
```

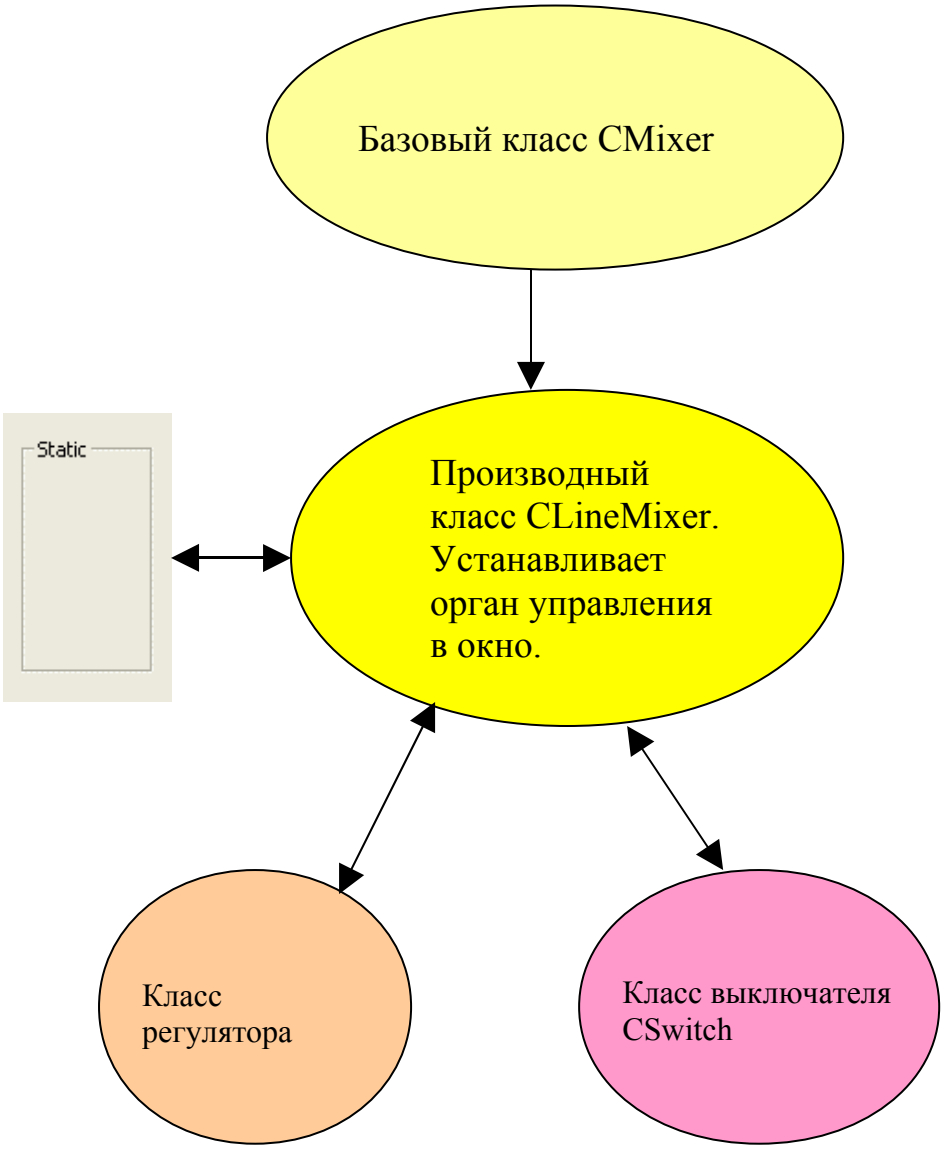

Рис.6. 3 Взаимодействие классов микшера.

Фиктивное окно для обработки сообщения микшера MM_MIXM_CONTROL_CHANGE создается на куче в методе CMixer::OpenMixer. Идентификатор окна, при его создании, указывается произвольный. Класс фиктивного окна объявляется в файле базового класса как "друг", для доступа к закрытым членам базового класса.

```
class CMsgWnd : public CWnd
{
friend CMixer;

  DECLARE_DYNAMIC(CMsgWnd)

protected:
  CMsgWnd(CMixer* pMix);
  virtual ~CMsgWnd();

  //Указатель на базовый класс
  CMixer* m_pMix;

protected:
  DECLARE_MESSAGE_MAP()

  //Обработка сообщений микшера
  afx_msg LRESULT OnCtrlChange(WPARAM nID
                              , LPARAM lParam);
};
```

Реализация класса.

Обработка сообщения микшера об изменении состояния параметра системного микшера другим приложением, например, изменилась громкость на выходной линии, производится в классе фиктивного окна. В обработчике вызывается виртуальная функция OnSetControl, которая перегружена в производном классе CLineMixer. Из данной функции вызываются методы для отображения состояния органов управления в классах владельцах этих органов управления. В результате в приложении движок регулятора передвигается в нужное положение.

Обработка сообщений о смене состояния органа управления своим приложением производится в классах органов управления. Передача состояния органа управления производится путем вызова метода из объекта класса, который владеет регулятором. В результате в классах приложения нет никаких обработчиков сообщений от органов управления. Это существенно упрощает использование классов микшера. На рисунке 6.4 показана схема взаимодействия классов при обработке сообщений.

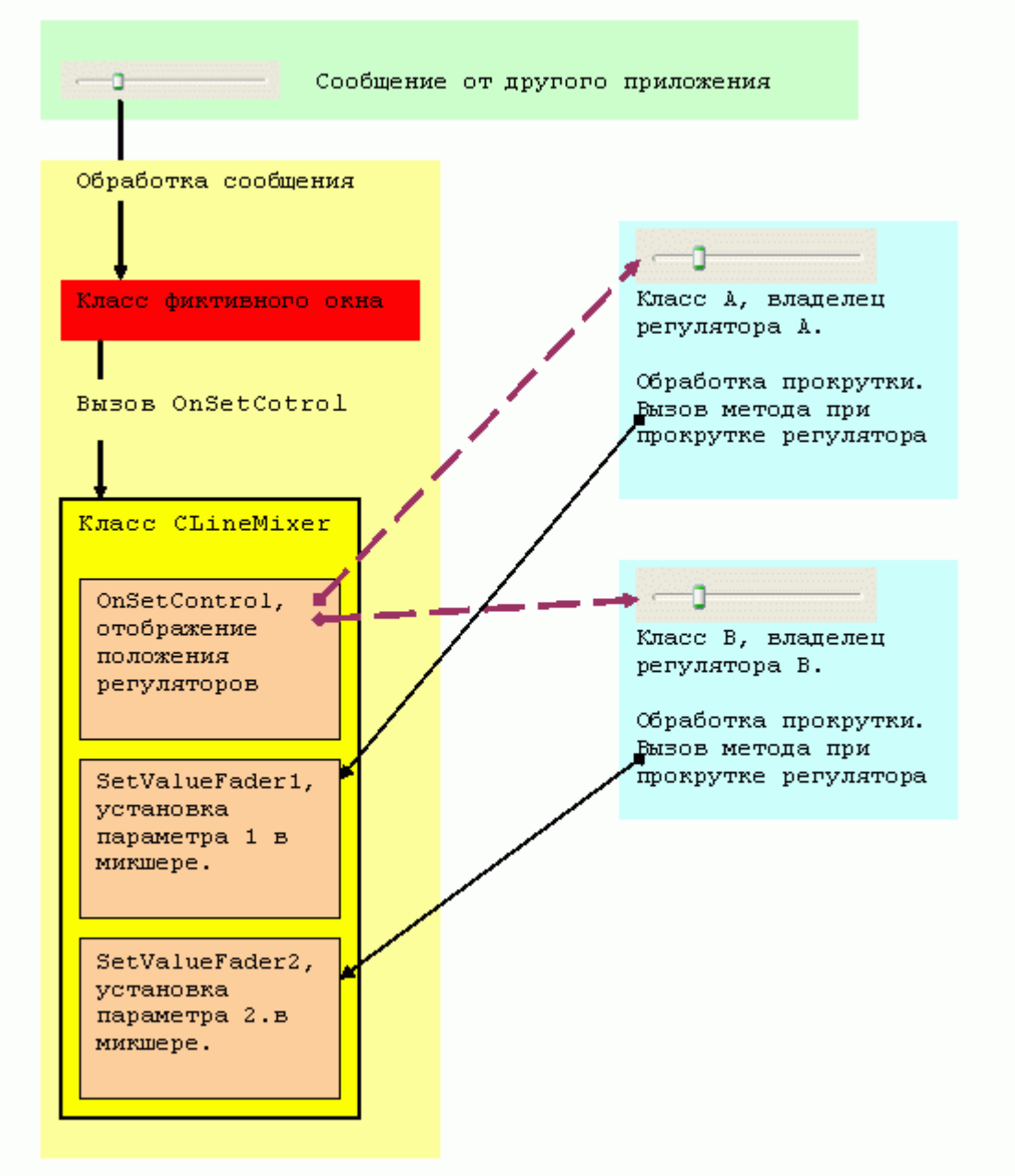

Рис.6. 4. Обработка сообщений микшера

```
IMPLEMENT_DYNAMIC(CMsgWnd, CWnd)
CMsgWnd::CMsgWnd(CMixer* pMix):m_pMix(NULL)
{
  //Указатель на базовый класс
  m_pMix = pMix;
}

CMsgWnd::~CMsgWnd()
{
}

BEGIN_MESSAGE_MAP(CMsgWnd, CWnd)
```

```
  //Обработка сообщения микшера
  ON_MESSAGE(MM_MIXM_CONTROL_CHANGE,OnCtrlChange)
END_MESSAGE_MAP()


//Обработка сообщение MM_MIXM_CONTROL_CHANGE
LRESULT CMsgWnd::OnCtrlChange(WPARAM wp,LPARAM lp)
{
  HMIXER hmix = reinterpret_cast<HMIXER>(wp);
  DWORD ctrlID = static_cast<DWORD>(lp);

  //Отображает изменение состояния органа управления
  //микшера, когда микшер управляется другим
  //приложением, не своим
  //Если установлены два органа для одной линии на
  //разных вкладках, то для управления от своего
  //приложения проверку следует убрать

  const MSG* pmsg = CWnd::GetCurrentMessage();

  CWnd* pwnd = WindowFromPoint(pmsg->pt);
  if(pwnd && !(pwnd->IsKindOf(RUNTIME_CLASS(CSwitch))
        || pwnd->IsKindOf(RUNTIME_CLASS(CFader))))
  {
    m_pMix->OnSetControl(hmix,ctrlID);
  }

  return 0;
}

BOOL CMixer::OpenMixer(HWND hwndMsg)
{
  //Проверка идентификатора микшера
  if(m_mixID > (mixerGetNumDevs() -1))
  {
    char buf[15];
    itoa(m_mixID,buf,10);
    string str =
  "В системе нет звукового микшера с номером ";
    str += buf;
    ::MessageBox(NULL, str.c_str()
                      ,"Error Mixer",MB_OK);
    return 0;
  }

  //Фиктивное окно для обработки сообщения микшера
  m_pMsgWnd = new CMsgWnd(this);
  ASSERT(m_pMsgWnd);

  m_pMsgWnd->Create(_T("STATIC"), NULL, WS_CHILD,CRect(0,0,5,5)
              , CWnd::FromHandle(hwndMsg), 0x777);
```

```
  //Открываем микшер
  if(::mixerOpen(&m_hMix,m_mixID,
        reinterpret_cast<DWORD_PTR>(m_pMsgWnd->m_hWnd)
        ,NULL, CALLBACK_WINDOW)  !=  MMSYSERR_NOERROR)
  {
    ::MessageBox(NULL,_T("Микшер не доступен")
            ,_T("Error Mixer"),MB_OK);
    return 0;
  }

  //Определяем число выходных линий
  MIXERCAPS mixCaps = {0};
  ::mixerGetDevCaps(m_mixID,&mixCaps
                          ,sizeof(MIXERCAPS));
  m_cDestinations = mixCaps.cDestinations;

  //Заполняем вектор органов управления для линии с
  //данным компонентом и для всех линий, которые могут
  //быть подключены к ней.
  if(mixCaps.cDestinations)
  {
    GetAllControlsForComponentDST(
                MIXERLINE_COMPONENTTYPE_DST_SPEAKERS);
  }

  //Заполняем вектор органов управления для линии с
  //данным компонентом и для всех линий, которые могут
  //быть подключены к ней
  if(mixCaps.cDestinations > 1)
  {
    GetAllControlsForComponentDST(
                  MIXERLINE_COMPONENTTYPE_DST_WAVEIN);
  }

  return 1;
}
```

Метод GetAllControlsForComponentDST предназначен для сбора данных обо всех органах управления одной выходной линии и всех входных линиях, которые могут подключаться к указанной выходной линии. Предварительное определение общих свойств микшера позволяет узнать общее число выходных линий. Для выполнения большинства задач достаточно двух типовых линий MIXERLINE_COMPONENTTYPE_DST_WAVEIN и MIXERLINE_COMPONENTTYPE_DST_SPEAKERS.

```
BOOL CMixer::GetAllControlsForComponentDST(
                          DWORD compTypeDST)
{
  MIXERLINE mixLine = {0};

  //Инициализация структуры MIXERLINE
```

```
  mixLine.cbStruct = sizeof(MIXERLINE);
  mixLine.dwComponentType = compTypeDST;

  //Инициализация структуры MIXERLINE
  // запрашиваем информацию для выходной линии
  if(MMSYSERR_NOERROR != ::mixerGetLineInfo(
          reinterpret_cast<HMIXEROBJ>(m_hMix)
          ,&mixLine,MIXER_GETLINEINFOF_COMPONENTTYPE))
    return FALSE;

  //Записать данные об органах управления для выходной //линии
  if(!AddVectorCtrlLine(mixLine))
  {
    //Нет органов управления на выходной линии
  }

  //Общее число подключений к выходной линии
  DWORD j = mixLine.cConnections;

  //вернуть все данные о линиях подключаемых к выходной
  //линии и найти все их органы управления.
  for(DWORD i= 0; i < j; ++i)
  {
    mixLine.dwSource = i;

    if(MMSYSERR_NOERROR == ::mixerGetLineInfo(
              reinterpret_cast<HMIXEROBJ>(m_hMix)
              ,&mixLine,MIXER_GETLINEINFOF_SOURCE))
    {
      AddVectorCtrlLine(mixLine);
    }
  }

  return TRUE;
}
```

Во вспомогательном методе AddVectorCtrlLine производится возврат данных для каждой линии и запись данных в формате класса данных в вектор.

```
BOOL CMixer::AddVectorCtrlLine(MIXERLINE& mixLine)
{
  //Нет органов управления
  if(!mixLine.cControls)
    return 0;

  MIXERCONTROL mixCtrl = {0};
  MIXERLINECONTROLS mixCtrls = {0};
  DATA_CTRL dctrl;
  //Временный  вектор для хранения структур MIXERCONTROL
  vector<MIXERCONTROL> vtemp;

  //Заполняем данными линии объект класса
```

```
  dctrl.cChannals     = mixLine.cChannels;
  dctrl.dwComponentType = mixLine.dwComponentType;
  dctrl.dwLineID      = mixLine.dwLineID;
  dctrl.dwDestination   = mixLine.dwDestination;
  dctrl.rPlace      = CRect(0,0,0,0);
  strcpy(dctrl.szShortName,mixLine.szShortName);
  dctrl.pCtrl = NULL;

  //Получить данные об органах управления на  линии.
  //Заполняем вектор пустыми структурами по
  //числу органов управления
  vtemp.assign(mixLine.cControls,mixCtrl);

  //Инициализируем управляющую структуру  //MIXERLINECONTROLS
  mixCtrls.cbStruct = sizeof(MIXERLINECONTROLS);
  mixCtrls.cbmxctrl = sizeof(MIXERCONTROL);
  mixCtrls.dwLineID = mixLine.dwLineID;
  mixCtrls.cControls = mixLine.cControls;

  //Указатель на вектор структур MIXERCONTROL
  mixCtrls.pamxctrl = &vtemp[0];

  //Заполняем вектор данными обо всех органах
  //управления на данной линии
  if(MMSYSERR_NOERROR != ::mixerGetLineControls(
             reinterpret_cast<HMIXEROBJ>(m_hMix)
             ,&mixCtrls,MIXER_GETLINECONTROLSF_ALL))
    return 0;

  //формируем вектор данных
  size_t sz = vtemp.size();

  for(size_t i=0; i< sz; ++i)
  {
    //заполняем структуру MIXERCONTROL
    dctrl.sMixCtrl = vtemp[i];

    //запись элемента в вектор
    m_vCtrlAll.push_back(dctrl);
  }

  return TRUE;
}
```

Для предоставления информации о микшере пользователю или разработчику, дополнительно создан метод GetInfoSourceComponent, который выводит данные структуры MIXERLINE в сообщение. Реализация остальных методов базового класса особенностей не имеет и ознакомится с ней, можно в исходных текстах проекта.

Организация данных

Для установки органа управления в окно диалога требуются данные в основном из двух структур MIXERLINE и MIXERCONTROL. Из первой структуры необходимы не все данные, достаточно только идентификатора линии, число каналов управления и имя линии. Вторая структура, содержащая данные на орган управления может понадобиться полностью. Некоторые дополнительные данные, которые могут потребоваться показаны в объявлении класса. Данные организуем в виде класса, обеспечивающего полноценное применение контейнеров и алгоритмов стандартной библиотеки STL.

```
class DATA_CTRL
{
public:
  //Конструктор
  DATA_CTRL():pCtrl(0)
  { };

  ~DATA_CTRL() {};

//Конструктор копировщик
  DATA_CTRL(const DATA_CTRL & data)
  { *this = data; };

//Данные
  //Указатель на отображаемый орган управления
  void* pCtrl;

  //Координаты органа управления в окне диалога
  CRect  rPlace;

  //Тип компонента линии
  DWORD dwComponentType;

  //Индекс выходной линии
  DWORD dwDestination;

  //Идентификатор линии
  DWORD dwLineID;

  //Число каналов управления
  DWORD cChannals;

  //Короткое имя линии
  CHAR  szShortName[MIXER_SHORT_NAME_CHARS];

  //Структура MIXERCONTROL
  MIXERCONTROL sMixCtrl;

//Оператор присваивания
  const DATA_CTRL & operator = (const DATA_CTRL & data)
```

```
  {
    if(this != &data)
    {
      pCtrl = data.pCtrl;
      rPlace = data.rPlace;
      dwComponentType = data.dwComponentType;
      dwLineID = data.dwLineID;
      cChannals = data.cChannals;
      ::CopyMemory(&sMixCtrl,&data.sMixCtrl
                        ,sizeof(MIXERCONTROL));
      ::CopyMemory(szShortName,data.szShortName
                        ,MIXER_SHORT_NAME_CHARS);
    }
    return *this;
  };

  //Оператор сравнения по ID линии
  bool operator == (const DWORD& lineID) const
  {
    return dwLineID == lineID;
  };
};
```

Для сбора и хранения данных, установленных органов управления используем контейнер STL типа вектор.

```
  vector<DATA_CTRL> m_vCtrl;
```

Данные обо всех органах управления линии будут добавляться в вектор каждый раз при установке нового органа управления линии. Данные сохраняются на протяжении всего периода жизни микшера. Подобная организация данных позволяет использовать алгоритмы STL для их обработки, что существенно упрощает и уменьшает объем кода программы. Кроме того, эффективность обработки должна быть выше, чем при повторных поисках данных в микшере с использованием первичных структур и функций микшера, поскольку объем данных значительно уменьшен за счет предварительного отбора для определенной линии. Главной причиной выбора такого построения данных является резкое упрощение логики написания кода для работы с органом управления, его нужно просто отыскать в векторе и установить в нужное положение или запросить данные. Кроме того, для алгоритмов STL обычно пишутся короткие и простые объекты функций, определяющие критерии обработки данных, что позволяет значительно ускорить и упростить работу программиста. Все объекты функций, используемые в алгоритме поиска данных, находятся в том же файле, что класс данных. Для примера ниже приводится текст одного объекта функции для поиска в векторе регулятора с указанным компонентом для определенной линии. Идентификатор линии указывается во втором параметре dwData.

```
struct FindCtrlTypeFaderForLine :public
        std::binary_function<DATA_CTRL,DWORD,bool>
{
  bool operator () (const DATA_CTRL& data
                         ,const DWORD dwLineID) const
  {
    return (((data.sMixCtrl.dwControlType
        & MIXERCONTROL_CT_CLASS_FADER)
        == MIXERCONTROL_CT_CLASS_FADER)
        && !data.sMixCtrl.cMultipleItems
        && (data.dwLineID == dwLineID));
  }
};
```

Классы органов управления

Выходная линия воспроизведения MIXERLINE_COMPONENT-TYPE_DST_SPEAKERS обычно содержит регулятор уровня сигнала и один или более выключателей, которые управляют прохождением сигнала. К выходной линии подключаются несколько входных линий. Входные линии чаще всего подключаются к выходной линии воспроизведения через смеситель и также имеют органы управления, регуляторы усиления и выключатели. В результате, в канале воспроизведения от одного источника звука может два комплекта органов управления.

Выходная линия записи …_DST_SPEAKERS чаще всего имеет на входе переключатель входных линий (мультиплексор), который может подключать только один источник сигнала к линии записи. Выходная линия записи обычно не имеет других органов управления, кроме мультиплексора, и при записи используются только органы управления входной линии. Если при записи звуковая карта позволяет использовать смеситель канала воспроизведения для наложения звука от нескольких источников, то тогда число органов управления увеличивается пропорционально числу источников звукового сигнала.

Регуляторы, используемые на линии для регулировки уровня сигнала, могут принадлежать разным классам (типам). Наиболее часто используются регуляторы класса Fader (тип Fader и Volume). Класс регуляторов Slider типа Pan также часто используется для регулировки стерео баланса, но в большинстве звуковых карт он реализован не аппаратно или программно на уровне драйвера, а реализуется приложением с использованием двухканальной регулировки на входной линии. То есть используется один и тот же регулятор в двух представлениях, как регулятор уровня сигнала и как регулятор стерео баланса. В приложениях, не предназначенных для специализированной записи звука, регулировка стерео баланса практически не нужна, и можно ограничиться только классом регуляторов Fader.

Для управления прохождением сигнала через линию используются чаще всего выключатели класса Switch типа Mute. Класс Switch также включает еще несколько типов переключателей, но все они булевского типа и могут быть представлены одним классом.

Для решения поставленной задачи создадим два класса CFader и CSwitch, которые будут представлять обобщенный регулятор уровня сигнала и обобщенный выключатель. Классы органов управления будем создавать с таким расчетом, чтобы обеспечить их максимальную автономность и независимость от класса объекта, который их использует. Тогда классы органов управления можно будет использовать не только со звуковым микшером, но и с другими устройствами.

Класс CFader

Класс CFader создается производным от класса MFC CSliderCtrl. Он может отображать вертикальный, горизонтальный тип регулятора и разновидность горизонтального с нулем по середине для отображения регулятора стерео баланса. Регулятор стерео баланса для решаемых задач не нужен и поэтому в данной версии класса не реализован. Показания регулятора в процентах от полной величины отображаются в дополнительном окне класса CWndBuddy, производного от класса MFC CStatic. Координаты этого окна связаны с координатами регулятора и не требуется дополнительных действий по его установке при создании регулятора. Любой регулятор устанавливает и возвращает свое положение в процентах от полной шкалы. Шкала принимает значения в диапазоне от 0 до 100% для обычных регуляторов. Для регулятора стерео баланса диапазон лежит от -100% до +100% с нулем посредине. Весь расчет действительных значений параметров микшера производится в классе, который использует регулятор.

Обработка сообщений о смене положения движка регулятора производится в его собственном классе. Из обработчика перемещения движка регулятора вызывается метод, того объекта, который использует данный регулятор. Для различения объектов разных классов, которые имеют регуляторы, производится проверка времени выполнения на принадлежность органа управления классу, который его создал. Для этого проверяется указатель на класс, создавший регулятор. Для дополнительной идентификации регулятор передает указатель на себя (this) объекту, который владеет регулятором. Таким образом, достигается максимальная автономность класса регулятора и независимость от пределов регулируемых параметров.

Для облегчения отладки все окна в режиме Debug имеют рамки. Регулятор размещается внутри ограничивающего окна класса CStatic, создание которого возлагается на программиста. Это может быть шаблон (group box) установленный в редакторе, как в проекте, или дочернее окно диалога созданное другим способом. Окну присваивается имя линии или другое произвольное имя.

Заголовочный файл класса также включает и класса окна CWndBuddy, который служит для вывода показаний регулятора в процентах от максимального значения.

```
// CWndBuddy
//Класс окна для вывода показаний регулятора

class CWndBuddy : public CStatic
{
public:
  CWndBuddy() {};
  virtual ~CWndBuddy(){};

};
```

Объекты класса CFader создаются динамически, однако в классе с целью его упрощения не приняты меры по запрету создания объектов в стеке. При необходимости класс можно легко модернизировать, используя методику, использованную в классе CLineMixer.

```
//Ширина регулятора при вертикальном размещении
#define CTRL_W  35
//Высота регулятора при горизонтальном размещении
#define CTRL_H  20

//Стили регулятора
#define STYLE     (WS_CHILD|WS_VISIBLE|TBS_NOTICKS)

//Без рамок
#ifndef _DEBUG
#define STYLE_FADER_VERT  (STYLE|TBS_VERT|TBS_BOTH)
#define STYLE_FADER_HORZ  (STYLE|TBS_HORZ|TBS_BOTH)
#define STYLE_PAN   (STYLE|TBS_HORZ|TBS_BOTTOM)
#endif //_DEBUG

//Рамки для отладки
#ifdef _DEBUG
#define STYLE_FADER_VERT
                      (STYLE|WS_BORDER|TBS_VERT|TBS_BOTH)
#define STYLE_FADER_HORZ
                      (STYLE|WS_BORDER|TBS_HORZ|TBS_BOTH)
#define STYLE_PAN
                      (STYLE|WS_BORDER|TBS_HORZ|TBS_BOTTOM)
#endif //_DEBUG

// CFader

class CFader : public CSliderCtrl
{
  DECLARE_DYNAMIC(CFader)
```

```
public:
//Указатель на ограничивающее окно
  CFader(
        CWnd* pWndLimit
        ,DWORD style      //Стиль регулятора
        ,void* pOwner     //Указатель на объект создавший
                            // регулятор (владелец)
  );

  virtual ~CFader();

private:
//Закрытый конструктор копировщик запрещает копирование
  CFader(const CFader & ) { };

//Закрытый оператор присваивания запрещает присваивания
  const CFader & operator = (const CFader & ) {};
public:

  //Возвращает показания регулятора в процентах
  int myGetFaderValueProc(void);

  //Устанавливает положение регулятора по данным
  //в процентах
  void mySetFaderValueProc(int nProc);

  //Возвращает прямоугольник окна регулятора
  CRect myGetPlace(void);

protected:

  //Координаты установленного регулятора
  CRect m_rPlace;

  //Указатель на окно вывода показаний регулятора
  const auto_ptr<CWndBuddy> m_pWndBuddy;

  //Стиль регулятора
  DWORD m_style;

  //Счетчик числа объектов и формирование ID
  static UINT m_count;

  //Указатель на объект владеющий регулятором
  void* m_pOwner;

  //Размещение регулятора в ограничивающем окне
  BOOL myToPlaceFader(CWnd* pWndLimit, DWORD style
                                      , void* pOwner);

  //Создание окна показаний регулятора
  void myMakeWndBuddy(CRect& rect,BOOL bSliderOrient);
```

```
  //Устанавливает в окне показания регулятора
  //в процентах
  void mySetWndBuddyText(int nProc);


  DECLARE_MESSAGE_MAP()
  //Отраженные сообщения о прокрутке регулятора
  afx_msg void VScroll(UINT /*nSBCode*/,UINT /*nPos*/);
  afx_msg void HScroll(UINT /*nSBCode*/,UINT /*nPos*/);

};
```

В конструкторе класса производится вызов метода myToPlaceFader, который создает регулятор, устанавливает пределы изменения показаний, а также вызывается метод myMakeWndBuddy, создающий окно для отображения показаний регулятора. При создании каждого регулятора ему присваивается собственный идентификатор, который формируется с помощью счетчика объектов m_count. Базовый идентификатор IDC_FADER назначаем произвольно, с помощью редактора ресурсов. Это, при необходимости, поможет идентифицировать регулятор по идентификатору или определить число установленных регуляторов. В данном проекте эти возможности не используются.

```
  CFader::CFader(
  CWnd* pWndLimit,DWORD style,void* pOwner):
                   m_pOwner(NULL)
                   , m_style(0)
                   ,m_pWndBuddy(new CWndBuddy)
  {
    if(myToPlaceFader(pWndLimit,style,pOwner))
    {
      //Инициализируем указатель на объект собственник
      m_pOwner = pOwner;

      //Считаем объекты
      ++m_count;
    }
  }
```

Установка пределов изменения показаний для регуляторов имеет одну особенность. Дело в том, что базовый класс CSliderCtrl при вертикальной установке регулятора по умолчанию располагает отметку с минимальным значением вверху, а максимальным значением внизу. Такое расположение шкалы не соответствует общепринятому расположению, для которого показания увеличиваются при движении снизу вверх. Для установки минимальной и максимальной отметки в правильное положение служит константа стиля TBS_DOWNISLEFT, которая просто не оказывает никакого влияния, несмотря на то, что версии всех средств разработки соответствуют требованиям

использования этой константы. Поэтому пришлось диапазон изменения показаний вертикального регулятора сместить в область отрицательных чисел и дополнительно учитывать положение регулятора для корректировки его показаний.

```
BOOL CFader::myToPlaceFader(CWnd* pWndLimit,DWORD style
                            , void* pOwner)
{
  //Высоту окна регулятора определяем по размерам
  //используемого шрифта диалога
  //Определяем высоту текущего шрифта в пикселях
  //для ограничивающего окна
  CFont* pfnt = pWndLimit->GetFont();
  ASSERT(pfnt);
  LOGFONT lf;
  pfnt->GetLogFont(&lf);

  //Число пикселей на дюйм экрана
  int pixinch = ::GetDeviceCaps(
           pWndLimit->GetDC()->m_hDC ,LOGPIXELSY);


  //высота шрифта в пикселях
  int fntsz =
  static_cast<int>(::MulDiv(abs(lf.lfHeight)
                              ,pixinch,72) );

  CRect rect;
  //Координаты экрана
  pWndLimit->GetWindowRect(&rect);

  //  Координаты ограничивающего окна указываем в
  // в клиентских координатах основного окна
  // диалога
  pWndLimit->GetParent()->ScreenToClient(&rect);

  if(style == STYLE_FADER_VERT)
  {
    //Вертикальный регулятор
    int left = rect.left+5;
    m_rPlace.SetRect(left,rect.top+ fntsz* 5/2
                    ,left+CTRL_W,rect.bottom-5);
  }

  if(style == STYLE_FADER_HORZ || style == STYLE_PAN)
  {
    //горизонтальный
    m_rPlace.SetRect(rect.left+ CTRL_W,rect.top+20
                ,rect.right-5,rect.top +20 + CTRL_H);
  }
```

```
  //При создании регулятора указываем основной диалог //в каче-
  стве родительского окно для упрощения расчета //координат
  BOOL bSliderOrient = 1;
  if(Create(style, m_rPlace, pWndLimit->GetParent()
                                ,IDC_FADER+m_count))
  {
    m_style = style;

    //Установка пределов для регуляторов
    if(style == STYLE_FADER_VERT)
    {
      bSliderOrient = 0;
      SetRange(-100,0,1);
    }

    if(style == STYLE_FADER_HORZ)
      SetRange(0,100,1);

    if(style == STYLE_PAN)
      SetRange(-100,100,1);

    //Создание окна для вывода позиции регулятора
    myMakeWndBuddy(m_rPlace, bSliderOrient);

    //Установка показаний регулятора
    mySetWndBuddyText(myGetFaderValueProc());
    return 1;
  }

  return 0;
}
```

Положение окна для вывода показаний регулятора привязано к прямоугольнику регулятора, а размеры окна определяются характеристиками шрифта.

```
void CFader::myMakeWndBuddy(CRect& rect,BOOL
                         bSliderOrient)
{
  ASSERT(m_pWndBuddy.get());

  CRect rB(rect);
  DWORD style=WS_CHILD|WS_VISIBLE;

//Отладочная рамка
#ifdef DEBUG
    style |= WS_BORDER;
#endif //DEBUG

  //Создаем в координатах родительского окна
  //(ограничивающего)
  if(m_pWndBuddy->Create("",style,rB
```

```
                        ,this->GetParent()))
  {
    m_pWndBuddy->mySetFont();

    //Определяем максимальные размеры строки текста
    CDC* pDC = m_pWndBuddy->GetDC();
    ASSERT(pDC);
    CSize szText = pDC->GetOutputTextExtent("100%",4);

    //Корректировка положения окна
    if(!bSliderOrient)
    {//vert
      rB.top -= szText.cy;
      rB.bottom = rect.top;
      rB.right = rB.left + szText.cx;
    }
    else
    {//horz
      rB.right = rect.left;
      rB.left = rB.right- szText.cx +1;
    }

    m_pWndBuddy->MoveWindow(&rB);
  }
}
```

Обработка сообщения от регулятора производится в классе CFader, с использованием отраженных сообщений о прокрутке. Для класса CSliderCtrl, который является базовым для класса CFader, в документации предлагается использовать обработчики событий WM_HSCROLL или WM_VSCROLL, в зависимости от ориентации регулятора. Располагаются они в классе родительского окна, которое и так зачастую перегружено различными обработчиками событий. Значительно удобней иметь обработчики событий в своем классе, это скрывает детали использования класса и облегчает его переносимость. Отраженные сообщения WM_HSCROLL_REFLECT( ) и WM_VSCROLL_REFLECT( ) плохо документированы и в документации можно найти их описание только в разделе TN062: Message Reflection for Windows Controls. Их установка в карту сообщений производится вручную.

```
BEGIN_MESSAGE_MAP(CFader, CSliderCtrl)

  //Отраженные сообщения о прокрутке регулятора
  ON_WM_VSCROLL_REFLECT()
  ON_WM_HSCROLL_REFLECT()

END_MESSAGE_MAP()
```

В обработчиках этих сообщений производится вызов метода, устанавливающего значение величины в микшере. Определение положения движка регулятора производит метод myGetFaderValueProc, что на первый взгляд кажется избыточным, так как второй параметр обработчика указывает позицию регулятора, но позиция движка регулятора передается только при его перетаскивании. Если же производится щелчок по шкале регулятора, то движок регулятора устанавливается в новое положение, а значение позиции, передаваемой во втором параметре, становится равным нулю, что не соответствует действительности.

```
void CFader::VScroll(UINT /*nSBCode*/, UINT nPos)
{
  //установка значения величины в микшере
  ASSERT(m_pOwner);

  CLineMixer* pmix = static_cast<CLineMixer*>(m_pOwner);

  //Проверка указателя на принадлежность классу
  if(pmix && pmix->IsKindOf(RUNTIME_CLASS(CLineMixer)))
  {
    //Установить значение параметра микшера
    pmix->SetValueFader(this,myGetFaderValueProc());

    //Отобразить значение параметра микшера
    mySetWndBuddyText(pmix->GetValueFader(this));
  }
}

void CFader::HScroll(UINT /*nSBCode*/, UINT nPos)
{
  //Установка значения величины в микшере
  ASSERT(m_pOwner);

  //Проверка указателя на принадлежность классу
  CLineMixer* pmix =  static_cast<CLineMixer*>(m_pOwner);

  if(pmix && pmix->IsKindOf(RUNTIME_CLASS(CLineMixer)))
  {
    //Установить значение параметра микшера
    pmix->SetValueFader(this,myGetFaderValueProc());

    //Отобразить значение параметра микшера
    mySetWndBuddyText(pmix->GetValueFader(this));
  }
}
```

Остальные детали реализации класса не имеют особенностей и с ними можно ознакомиться в проекте.

Класс CSwitch

Класс CSwitch создается производным от класса MFC CButton. Класс реализован подобно классу CFader, его реализации несколько проще, он имеет только два метода. Установка выключателей производится относительно точки, которая передается как параметр по ссылке и является левым верхним углом окна выключателя. После установки первого выключателя координаты точки рассчитываются для установки второго выключателя с учетом направления установки. Предусмотрено два направления установки, справа от регулятора и вниз от регулятора. Расчет точки смещения предусмотрен только для выключателей располагаемых горизонтально в одну линию. При необходимости установить несколько выключателей под регулятором вертикального расположения следует произвести доработку класса CFader и CSwitch. Количество выключателей для разных линий может быть различным, что повлечет за собой установку вертикальных регуляторов разного размера. Диалог в таком случае будет иметь неряшливый вид. Лучшим решением является установка дополнительной кнопки, которая будет открывать отдельное окно с дополнительными выключателями. Чаще всего используются выключатели типа mute, которые блокируют прохождение сигнала, остальные требуются редко их лучше вынести в отдельный диалог.

```
class CSwitch : public CButton
{
  DECLARE_DYNAMIC(CSwitch)

public:

  CSwitch(
    //Указатель на  ограничивающее окно
    CWnd* pWndLimit,

    //Точка, относительно которой производится установка   //вы-
ключателя
    CPoint& ptOffSet,

    DWORD style,//стиль выключателя

    //Расположение выключателей относительно окна
    //установки, 0 вниз, 1 вправо
    DWORD dwOrient,

    //Указатель на объект создавший орган управления
    void* pOwner,

    //Имя выключателя
    CString nameWnd
    );
```

```
  virtual ~CSwitch();

private:
//Закрытый конструктор копировщик запрещает копирование
  CSwitch(const CSwitch & ) { };

//Закрытый оператор присваивания запрещает присваивания
  const CSwitch & operator = (const CSwitch & ) {};

protected:
  //Координаты выключателя
  CRect m_rPlace;

  DECLARE_MESSAGE_MAP()
  //Обработка click
  afx_msg void OnBnClicked();

  //Счетчик числа объектов и формирование ID
  static UINT m_count;

  //Указатель на объект создавший переключатель
  void* m_pOwner;

  //установка выключателя относительно точки ptOffSet. //Возвра-
щается с новыми значениями для установки
  //следующего выключателя или ptOffSet == CPoint(0,0)
  BOOL myToPlaceSwitch(CWnd* pWndLimit
                 ,CPoint& ptOffSet,DWORD style,
                 DWORD dwOrient,CString nameWnd);

public:

  // возвращает прямоугольник своего окна
  CRect myGetPlace(void);
};
```

Установка выключателя производится в методе myToPlaceSwitch. Для определения размеров окна выключателя используется длина строки имени. Поскольку, в отличие от вывода показаний регулятора, размер выводимого текста переключателя неизвестен, то имя переключателя передается как параметр в конструкторе.

```
BOOL CSwitch::myToPlaceSwitch(CWnd* pWndLimit,CPoint&
ptOffSet,DWORD style
              , DWORD dwOrient,CString nameWnd)
{
  if(!pWndLimit)
      return FALSE;

  CWnd* pParent = pWndLimit->GetParent();
```

```
  CRect rLimit;
  pWndLimit->GetWindowRect(&rLimit);
  pParent->ScreenToClient(&rLimit);

  //Не устанавливать если нет места в ограничивающем //окне
  if(ptOffSet.x > rLimit.right - 50)
  {
    ptOffSet.SetPoint(0,0);
    return 0;
  }

  //Формируем окно для установки выключателя
  CRect rWnd(ptOffSet,CPoint(ptOffSet.x +
            SW_WH,ptOffSet.y + SW_WH));

  if(Create("",style,rWnd,pParent
                   ,IDC_SWITCH + m_count))
  {
    //Устанавливаем шрифт
    SetFont(pParent->GetFont());

    //Корректировка размеров окна выключателя по длине   //его
имени
    CDC* pDC = GetDC();
    ASSERT(pDC);
    CSize szText = pDC->GetOutputTextExtent(nameWnd);

    m_rPlace.SetRect(rWnd.left,rWnd.top
                   ,rWnd.left+ szText.cx + 10
                   ,rWnd.top + SW_WH);

    if((m_rPlace.right <= rLimit.right - 50)
                   && dwOrient == ORIENT_RIGHT)
    {
      //Возвращается следующая точка для установки
      // выключателя
      ptOffSet.Offset(m_rPlace.Width()+1,0);
    }
    else //Корректировка правой границы переключателя
    {
      m_rPlace.right = rLimit.right -1;
      ptOffSet.SetPoint(0,0);
    }

    MoveWindow(m_rPlace);
    SetWindowText(nameWnd);

    return TRUE;
  }

  return FALSE;
}
```

Обработка сообщений от выключателя также производится в своем классе, аналогично обработке сообщений в классе регулятора.

```
void CSwitch::OnBnClicked()
{
  //проверка указателя на принадлежность классу
  CLineMixer* pmix =  static_cast<CLineMixer*>(m_pOwner);

  if(pmix && pmix->IsKindOf(RUNTIME_CLASS(CLineMixer)))
  {
    pmix->SetStateSwitch(this);
  }

  //проверка указателя на принадлежность классу
  CMux* pmux = static_cast<CMux*>(m_pOwner);
  if(pmux && pmux->IsKindOf(RUNTIME_CLASS(CMux)))
  {
    //Переключение линии мультиплексора
    pmux->myToggleLineMux(this);
  }
}
```

Класс мультиплексора CMux

Линия записи MIXERLINE_COMPONENTTYPE_DST_WAVEIN обычно не имеет собственного регулятора и выключателя, она имеет в качестве органа управления мультиплексор, который подключает одну входную линию из нескольких возможных. Входная линия является источником записываемого сигнала со своими органами управления. Входная линия всегда имеет регулятор, который используется как регулятор уровня записи. Линия микрофона кроме регулятора может иметь один или несколько переключателей для коммутации дополнительного микрофона и включения предварительного усилителя.

Системная программа sndvol32.exe при отображении окна канала записи показывает регуляторы всех линий, совершенно напрасно занимая площадь экрана, поскольку рабочими органами являются только органы управления одной линии, подключенной мультиплексором к выходной линии записи. Для уменьшения окна мультиплексора достаточно отобразить один регулятор для всех линий и кнопки переключения мультиплексора. Если линия имеет выключатели, то их следует отображать только при ее подключении. Мультиплексор можно сделать еще компактней, если кнопки переключения заменить элементом CComboBox или CComboBoxEx.

Все данные для линий мультиплексора записываются при инициализации в вектор установленных элементов, также как для остальных органов

управления, но без их реальной установки и с нулевыми указателями на органы управления.

Мультиплексор будет иметь один регулятор класса CFader, который управляет линией, а его показания отображают состояние подключенной линии. При инициализации определяется подключенная линия, и параметры регулятора устанавливаются в соответствии с текущими показаниями регулируемой величины. Значение указателя на объект регулятора и его координаты также записываются в данные. В процессе переключении линии данные регулятора (указатель и прямоугольник размещения) очищаются в векторе установленных элементов для старой линии. Эти данные присваиваются новой подключенной линии. В результате в векторе всегда будет находиться один регулятор подключенной, остальные входные линии мультиплексора имеют нулевые данные для регулятора.

Несколько иначе подойдем к созданию выключателей на линии. Они будут устанавливаться в зарезервированное место между регулятором и переключателями мультиплексора. Если выключатели есть на линии, они будут создаваться, а их данные записываться в вектор. При подключении линии, которая не имеет выключателей, выключатели предыдущей линии уничтожаются, а их данные очищаются.

Установка параметров мультиплексора разделена на две части, регулятор и выключатели используют общие для всех линий методы, описанные ниже в классе CLineMixer, а переключение входов мультиплексора является задачей класса CMux.

Таким образом, сохраняется единый подход к управлению всеми органами управления микшера, в векторе установленных элементов управления всегда будет одна подключенная линия мультиплексора, которой можно управлять также как и другими линиями.

Мультиплексор относится к классу MIXERCONTROL_CT_CLASS_LIST. Он управляется структурами MIXERCONTROLDETAILS_LISTTEXT и MIXERCONTROLDETAILS_BOOLEAN. Переключение линий можно отобразить разными способами. Например, в виде списка класса CCheckListBox или списка другого класса. Можно имитировать список кнопками класса CButton. Выбран второй вариант, имитация кнопок, поскольку можно использовать готовый класс CSwitch, а для того, чтобы по внешнему виду переключатели мультиплексора отличались от выключателей линии и переключались как зависимый переключатель, кнопки устанавливаются со стилем BS_AUTORADIOBUTTON | BS_PUSHLIKE. Следует отметить, что в этом случае возникает задача различения кнопок мультиплексора и выключателей линии, так как используется один обработчик нажатия кнопки в классе CSwitch, но это решается намного проще, по сравнению с созданием нового класса для списка. Еще одно замечание относится к ширине кнопок, которая определяется длиной строки имени переключателя. Кнопки будут иметь разную ширину. Если это неприемлемо, тогда придется доработать класс мультиплексора.

Для реализации класса мультиплексора необходимы исходные данные линии, на которой он находится, для этого конструктор класса получает в качестве параметра итератор вектора, с помощью которого класс получает данные линии. Данные линии копируются и сохраняются постоянно в виде члена класса. Такое хранение данных избыточно и можно модернизировать класс, сохранив только необходимые данные.

Мультиплексор, как упоминалось ранее, управляется с помощью двух структур, это неудобно и может привести к дополнительным ошибкам. Поэтому данные этих структур сведены в одну структуру и дополнены указателем на переключатель мультиплексора, что позволяет упростить обработку переключения линий. Состояние линий мультиплексора сохраняется в векторе члене класса. Обработка вектора состояний в основном заключается в поиске линии соответствующей нажатой кнопке переключателя мультиплексора. Для поиска линии используется объект функция FindLinePtrSwitch, которая также является членом класса. Структура состояний простая и не содержит сложных данных, поэтому в ней определен только оператор сравнения, операции копирования и присваивания выполняются по умолчанию.

```
class CMux : public CWnd
{
  DECLARE_DYNAMIC(CMux)

  public:

  CMux(
    //Указатель на ограничивающее окно
    CWnd* pWndLimit

    //Указатель на объект создавший орган управления
    ,void* pOwner

    //Итератор вектора, указывающий на данные линии
  //мультиплексора
    ,vector<DATA_CTRL>::iterator posDST
    );

  virtual ~CMux();

  //Указатель на объект создавший переключатель
  void* m_pOwner;

private:

  //Закрытый конструктор копировщик запрещает
  //копирование
  CMux(const CMux & ) { };

  //Закрытый оператор присваивания запрещает
```

```
  //присваивание
  const CMux & operator = (const CMux & ) {};

protected:

  //состояние одной линии, подключаемой к
  //мультиплексору
  struct STATE_SRC_MUX
  {
    //указатель на переключатель мультиплексора
    CSwitch* pSw;
    DWORD idLine; //ID линии
    BOOL  bState; //состояние линии
    //Имя линии
    CHAR nameLine[MIXER_LONG_NAME_CHARS];

    //Оператор == для поиска включенной линии
    bool operator == (const STATE_SRC_MUX& st) const
    {
      return st.bState == bState;
    };
  };

  //Вектор состояния мультиплексора
  vector<STATE_SRC_MUX> m_vStMux;

  //Поиск линии мультиплексора по указателю кнопки
  struct FindLinePtrSwitch :public
          std::binary_function<CMux::STATE_SRC_MUX
                              , CSwitch*, bool>
  {
    bool operator ()
    (const CMux::STATE_SRC_MUX& data1,const CSwitch*
                                            pSw) const
    {
      return  data1.pSw == pSw;
    }
  };

  //Координаты мультиплексора
  CRect m_rPlace;

  //Счетчик числа объектов и формирование ID
  static UINT m_count;

  //Указатель на данные для линии мультиплексора
  auto_ptr<DATA_CTRL> m_pdt;

  //Указатель на ограничивающее окно мультиплексора
  CWnd* m_pWndLimit;

  //дескриптор ограничивающего окна
  HWND m_hwndLimit;
```

```
  //Указатель на регулятор уровня записи
  CFader* m_pFd;

  DECLARE_MESSAGE_MAP()
  public:

  //Производит переключение линии мультиплексора
  void myToggleLineMux(CSwitch* pSw);

  //Отображение состояния микшера после изменения
  //другим приложением
  void myDisplayStateMux(void);

  protected:
  //Возвращает прямоугольник своего окна
  CRect myGetPlace(void);

  //Заполняет вектор состояния линий
  BOOL myGetStateMux();

  //Очистка и установка состояния линии мультиплексора BOOL
mySetStateMux(DWORD idLine);

  //Создание и инициализация мультиплексора
  BOOL myMakeMux();

  //Формирование органов управления мультиплексора для
  //выбранной линии
  void myMakeCtrlMuxForLine(DWORD idLine);

  //Очистка данных для предыдущей линии
  void myClearOldLine(void);

};
```

Создание мультиплексора и инициализация его состояния производится в методе myMakeMux. Создаваемые переключатели мультиплексора не являются органами управления, какой либо линии. Они создаются как члены класса мультиплексора и по завершению работы удаляются в деструкторе класса. Указатели на кнопки переключателей сохраняются в векторе состояния линий мультиплексора, запись данных переключателей в вектор установленных органов управления не производится.

```
BOOL CMux::myMakeMux()
{
  if(!m_pWndLimit)
    return FALSE;

  CLineMixer* pLineMix = static_cast<CLineMixer*>(m_pOwner);
```

```
//Установка имени мультиплексора
CString sname(m_pdt->szShortName);
sname.Insert(0,"Мультиплексор   ");
m_pWndLimit->SetWindowText(sname);

//дескриптор ограничивающего окна необходим для
//возврата указателя на окно после его прокрутки
//(FromHandle(m_hwndLimit);), поскольку m_pWndLimit
//становится недействительным и в тоже время он
//сохраняет старое значение.

m_hwndLimit = m_pWndLimit->m_hWnd;

CWnd* pParent = m_pWndLimit->GetParent();

//Возвращаем прямоугольник окна, относительно
//которого производим установку
CRect rLimit;
m_pWndLimit->GetWindowRect(&rLimit);
pParent->ScreenToClient(&rLimit);

//Инициализируем вектор состояний линий
STATE_SRC_MUX stMux = {0};
m_vStMux.assign(
        m_pdt->sMixCtrl.cMultipleItems,stMux);

//Возвращаем состояние мультиплексора
myGetStateMux();

//Найти подключенную линию мультиплексора
vector<STATE_SRC_MUX>::iterator posStTrue;
stMux.bState = TRUE;
posStTrue =
        find(m_vStMux.begin(),m_vStMux.end(),stMux);

if(posStTrue == m_vStMux.end())
  return FALSE;

//Создаем один регулятор для всех линий
//мультиплексора
m_pFd =
        new CFader(m_pWndLimit,STYLE_FADER_HORZ,this);
if(!m_pFd)
  return FALSE;


//Найти все регуляторы и выключатели для линий
//мультиплексора  и переписать их в вектор
//установленных элементов
size_t sz = m_vStMux.size();
vector<DATA_CTRL>::iterator pos;

for(size_t i = 0; i < sz; ++i)
```

```
{
  //Найти линию
  pos = find(pLineMix->m_vCtrlAll.begin()
  ,pLineMix->m_vCtrlAll.end(),m_vStMux[i].idLine);

  if(pos == pLineMix->m_vCtrlAll.end())
    continue;

  DWORD lineID = (*pos).dwLineID;


  //Найти регулятор для этой линии
  pos = find_if(pLineMix->m_vCtrlAll.begin()
          ,pLineMix->m_vCtrlAll.end()
          ,bind2nd(FindCtrlTypeFaderForLine(),lineID));

  //Записать в вектор установленных органов управления
  if(pos != pLineMix->m_vCtrlAll.end())
    pLineMix->m_vCtrl.push_back(*pos);

  //Найти первый выключатель для этой линии
  pos = find_if(pLineMix->m_vCtrlAll.begin()
        ,pLineMix->m_vCtrlAll.end()
        ,bind2nd(FindCtrlTypeSwitchForLine(),lineID));

  //Найти остальные выключатели для этой линии
  while(pos != pLineMix->m_vCtrlAll.end())
  {
    //Записать в вектор установленных органов
    //управления
    pLineMix->m_vCtrl.push_back(*pos);

    //Ищем остальные выключатели со следующей позиции
    pos = find_if(++pos,pLineMix->m_vCtrlAll.end()
        ,bind2nd(FindCtrlTypeSwitchForLine(),lineID));
  }

}

//Создание органов управления мультиплексора для
//выбранной линии
myMakeCtrlMuxForLine((*posStTrue).idLine);

//Создаем  переключатели мультиплексора
//Стиль
DWORD style = STYLE_RADIOBUTTON_PUSHLIKE;

//Точка начала
CPoint pt(rLimit.left + 1,m_pFd->myGetPlace().bottom
                                                + 25);

//Найти и создать переключатели для всех линий
//мультиплексора
```

```
  CSwitch* pSw = NULL;

  for(DWORD i=0; i m_pdt->sMixCtrl.cMultipleItems; ++i)
  {
    CString str = m_vStMux[i].nameLine;
    if(str.GetLength() > 10)
      str = str.Left(10);

    //Создаем переключатель и записываем его данные
    pSw = new CSwitch(m_pWndLimit,pt,style
                              ,ORIENT_RIGHT,this,str);
    if(pSw)
    {
      m_vStMux[i].pSw = pSw;

      //Отобразить состояние линии
      if(m_vStMux[i].bState)
        pSw->SetCheck(BST_CHECKED);
    }

    //Второй и последующий ряд переключателей
    //мультиплексора
    if(pt == CPoint(0,0) && pSw)
    {
      pt.SetPoint(rLimit.left+1
                  , pSw->myGetPlace().bottom +3);
    }
  } //for

  //Корректировка окна мультиплексора
  rLimit.bottom = pSw->myGetPlace().bottom +3;
  m_pWndLimit->MoveWindow(&rLimit);
  m_rPlace = rLimit;

  return TRUE;
}
```

Возврат состояния мультиплексора и заполнение вектора состояний производится в методе myGetStateMux. В методе производится установка соответствия между структурами MIXERCONTROLDETAILS_LISTTEXT и MIXERCONTROLDETAILS_BOOLEAN, а также формирование структуры состояния и запись состояния линий в вектор. Подключенная линия отображается как нажатая кнопка переключателя.

```
BOOL CMux::myGetStateMux()
{
  CLineMixer* pLineMix = static_cast<CLineMixer*>(m_pOwner);

  //Инициализация структуры MIXERCONTROLDETAILS
  MIXERCONTROLDETAILS mixDets = {0};
  mixDets.cbStruct = sizeof(MIXERCONTROLDETAILS);
```

```
mixDets.cChannels = 1;// управление одноканальное
mixDets.dwControlID = m_pdt->sMixCtrl.dwControlID;
mixDets.cMultipleItems =
            m_pdt->sMixCtrl.cMultipleItems;

//Инициализация структуры для списка
mixDets.cbDetails =
          sizeof(MIXERCONTROLDETAILS_LISTTEXT);

//Структура для списка
MIXERCONTROLDETAILS_LISTTEXT mixDetList = {0};

// вектор для хранения структур
//MIXERCONTROLDETAILS_LISTTEXT
vector<MIXERCONTROLDETAILS_LISTTEXT> vList;

//Заполняем вектор пустыми структурами по
//числу органов управления
vList.assign(mixDets.cMultipleItems *
                mixDets.cChannels,mixDetList);

//Указатель на первую структуру в массиве
mixDets.paDetails = &vList[0];

//Заполнение списка
MMRESULT res = mixerGetControlDetails(
        reinterpret_cast<HMIXEROBJ>(pLineMix->m_hMix)
        ,&mixDets,MIXER_GETCONTROLDETAILSF_LISTTEXT);

if(res != MMSYSERR_NOERROR)
  return FALSE;

//Инициализация структуры для возврата состояния
//переключателя
MIXERCONTROLDETAILS_BOOLEAN mixDetBool = {0};
mixDets.cbDetails =
              sizeof(MIXERCONTROLDETAILS_BOOLEAN);

// вектор для хранения структур //MIXERCONTROLDETAILS_BOOLEN
vector<MIXERCONTROLDETAILS_BOOLEAN> vBool;

//Заполняем вектор пустыми структурами по
//числу органов управления
vBool.assign(mixDets.cMultipleItems *
                mixDets.cChannels,mixDetBool);

//Указатель на первую структуру в массиве
mixDets.paDetails = &vBool[0];

//Возврат данных
res = mixerGetControlDetails(
        reinterpret_cast<HMIXEROBJ>(pLineMix->m_hMix)
```

```
          ,&mixDets,MIXER_GETCONTROLDETAILSF_VALUE);

  if(res != MMSYSERR_NOERROR)
    return FALSE;

  //Переписываем данные в вектор состояния
  size_t sz = vList.size();

  if(m_vStMux.size() < sz)
    return FALSE;

  for(size_t i=0; i<sz; ++i)
  {
    m_vStMux[i].idLine = vList[i].dwParam1;
    strcpy(m_vStMux[i].nameLine,vList[i].szName);
    m_vStMux[i].bState = vBool[i].fValue;

    //отобразить состояние
    if(m_vStMux[i].pSw && m_vStMux[i].bState)
      m_vStMux[i].pSw->SetCheck(BST_CHECKED);
  }

  return TRUE;
}
```

Метод myMakeCtrlMuxForLine производит установку данных для регулятора и создание выключателей для линии, если таковые имеются.

```
void CMux::myMakeCtrlMuxForLine(DWORD idLine)
{
  //Указатель на класс владелец мультиплексора
  CLineMixer* pLineMix =
                 static_cast<CLineMixer*>(m_pOwner);

  //обновляем указатель на ограничивающее окно
  m_pWndLimit = FromHandle(m_hwndLimit);

  //возвращаем прямоугольник окна, относительно //которого
  производим установку
  m_pWndLimit->GetWindowRect(&m_rPlace);

  //обновляем координаты в клиентской области в случае //про-
  крутки
  CWnd* pParent = m_pWndLimit->GetParent();
  ASSERT(pParent);
  pParent->ScreenToClient(&m_rPlace);

  //Найти линию, произвести запись данных
  //и установить показания органов управления
  vector<DATA_CTRL>::iterator pos;
  pos = find_if(pLineMix->m_vCtrl.begin()
```

```
        ,pLineMix->m_vCtrl.end()
        ,bind2nd(FindCtrlTypeFaderForLine(),idLine));

  //Обновляем координаты регулятора
  CRect r;
  m_pFd->GetWindowRect(&r);
  pParent->ScreenToClient(&r);

  //Точка установки выключателей
  CPoint ptOffSet(m_rPlace.left+1,r.bottom+3);

  if(pos != pLineMix->m_vCtrl.end())
  {
    //Записать указатель и позицию регулятора
    (*pos).pCtrl = static_cast<void*>(m_pFd);
    (*pos).rPlace = r;

    //Начальная установка  значения регулятора
    m_pFd->mySetFaderValueProc(
              pLineMix->GetValueFader(m_pFd));
  }

  //Найти выключатели для этой линии и установить
  pos = find_if(pLineMix->m_vCtrl.begin()
        ,pLineMix->m_vCtrl.end()
        ,bind2nd(FindCtrlTypeSwitchForLine(),idLine));

  if(pos == pLineMix->m_vCtrl.end())
      return;

  //Найти остальные выключатели для этой линии
  while(pos != pLineMix->m_vCtrl.end())
  {
    CSwitch* pSw = new CSwitch(m_pWndLimit
                    , ptOffSet,STYLE_CHECK_BOX,1
                    ,this,(*pos).sMixCtrl.szShortName);
    (*pos).pCtrl = pSw;
    (*pos).rPlace = r;

    if(pSw)
        pSw->SetCheck(pLineMix->GetStateSwitch(*pos));

    //Ищем остальные выключатели со следующей позиции
    pos = find_if(++pos,pLineMix->m_vCtrl.end()
            ,bind2nd(FindCtrlTypeSwitchForLine()
            ,(*pos).dwLineID));
  }
}
```

Переключение линий мультиплексора производится в методе myToggleLineMux. Вызов метода выполняется в обработчике CSwitch::OnBnClicked(), поскольку переключатели являются объектами этого

класса. Для определения принадлежности указателя переключателю мультиплексора производится дополнительная проверка. Если это выключатель линии, то производится его обычная установка с помощью метода CLineMixer::SetStateSwitch.

```
void CMux::myToggleLineMux(CSwitch* pSw)
{
  //Указатель на класс владелец мультиплексора
  CLineMixer* pLineMix =  static_cast<CLineMixer*>(m_pOwner);

  //Найти  новую линию
  vector<STATE_SRC_MUX>::iterator posStNew;
  posStNew = find_if(m_vStMux.begin(),m_vStMux.end()
                 ,bind2nd(FindLinePtrSwitch(),pSw));

  if(posStNew == m_vStMux.end())
    return;

  //Это выключатель на линии или переключатель
  //мультиплексора
  if(pSw != (*posStNew).pSw)
  {
    //Это переключатель на линии
    pLineMix->SetStateSwitch(pSw);
    return;
  }

  //Очистить старую линию
  myClearOldLine();

  //Устанавливаем новое состояние линий
  mySetStateMux((*posStNew).idLine);

  // установка органов управления мультиплексора
  //для выбранной линии
  myMakeCtrlMuxForLine((*posStNew).idLine);

  //Обновляем вектор состояния линий
  myGetStateMux();
}
```

Очистка состояния линии производится перед установкой нового состояния линии при переключении мультиплексора и при смене состояния мультиплексора другим приложением. Эту задачу выполняет метод myClearOldLine.

```
void CMux::myClearOldLine(void)
{
  //Указатель на класс владелец мультиплексора
  CLineMixer* pLineMix =  static_cast<CLineMixer*>(m_pOwner);
```

```
//Найти старую подключенную линию мультиплексора
vector<STATE_SRC_MUX>::iterator posStOld;
STATE_SRC_MUX stMux;
stMux.bState = TRUE;

posStOld =  find(m_vStMux.begin(),m_vStMux.end(),stMux);

if(posStOld == m_vStMux.end())
  return;

ASSERT((*posStOld).pSw);

(*posStOld).pSw->SetCheck(BST_UNCHECKED);

//Найти выключатели для старой линии
vector<DATA_CTRL>::iterator pos;

pos = find_if(pLineMix->m_vCtrl.begin()
          ,pLineMix->m_vCtrl.end()
          ,bind2nd(FindCtrlTypeSwitchForLine()
          ,(*posStOld).idLine));

//и удалить их
while(pos != pLineMix->m_vCtrl.end())
{
  if((*pos).pCtrl)
  {
    //Удалить выключатели
    CSwitch* pSw =
            static_cast<CSwitch*>((*pos).pCtrl);
    delete pSw;

    (*pos).pCtrl = NULL;
    (*pos).rPlace = CRect(0,0,0,0);
  }

  //Ищем остальные выключатели со следующей позиции
  pos = find_if(++pos,pLineMix->m_vCtrl.end()
              ,bind2nd(FindCtrlTypeSwitchForLine()
              ,(*posStOld).idLine));
}

//Очистить данные регулятора на старой линии
pos = find_if(pLineMix->m_vCtrl.begin()
                ,pLineMix->m_vCtrl.end()
                ,bind2nd(FindCtrlTypeFaderForLine()
                ,(*posStOld).idLine));

if(pos != pLineMix->m_vCtrl.end())
{
  (*pos).pCtrl = NULL;
  (*pos).rPlace = CRect(0,0,0,0);
}
```

```
}
```

Установка состояния линии мультиплексора производится в методе mySetStateMux.

```
BOOL CMux::mySetStateMux(DWORD idLine)
{
  //Инициализация структуры MIXERCONTROLDETAILS
  MIXERCONTROLDETAILS mixDets = {0};
  mixDets.cbStruct = sizeof(MIXERCONTROLDETAILS);

  //Общая инициализация для списка и данных
  mixDets.cChannels = 1;  //Управление одноканальное
  mixDets.dwControlID = m_pdt->sMixCtrl.dwControlID;
  mixDets.cMultipleItems =
          m_pdt->sMixCtrl.cMultipleItems;

  //Инициализация структуры для списка
  mixDets.cbDetails =
         sizeof(MIXERCONTROLDETAILS_LISTTEXT);

  //Структура для списка
  MIXERCONTROLDETAILS_LISTTEXT mixDetList = {0};

  //Вектор для хранения структур //MIXERCONTROLDETAILS_LISTTEXT
  vector<MIXERCONTROLDETAILS_LISTTEXT> vList;

  //Заполняем вектор пустыми структурами по
  //числу органов управления
  vList.assign(mixDets.cMultipleItems
                       * mixDets.cChannels,mixDetList);

  //Указатель на первую структуру в массиве
  mixDets.paDetails = &vList[0];

  //Заполнение списка
  CLineMixer* pLineMix =
               static_cast<CLineMixer*>(m_pOwner);
  MMRESULT res = mixerGetControlDetails(
          reinterpret_cast<HMIXEROBJ>(pLineMix->m_hMix)
          ,&mixDets,MIXER_GETCONTROLDETAILSF_LISTTEXT);

  if(res != MMSYSERR_NOERROR)
    return FALSE;


  //Найти индекс вектора для линии с указанным ID
  size_t sz = vList.size();
  size_t ind = -1;
  for(size_t i=0; i < sz; ++i)
  {
    if(vList[i].dwParam1 == idLine)
```

```
    {
      ind = i;
      break;
    }
  }

  if(ind == -1)
    return FALSE;

  //Инициализация структуры для установки положения
  //переключателя
  MIXERCONTROLDETAILS_BOOLEAN mixDetBool = {0};
  mixDets.cbDetails =
            sizeof(MIXERCONTROLDETAILS_BOOLEAN);

  //Вектор для хранения структур
  //MIXERCONTROLDETAILS_BOOLEN
  vector<MIXERCONTROLDETAILS_BOOLEAN> vBool;

  //Заполняем вектор пустыми структурами по
  //числу органов управления
  vBool.assign(sz,mixDetBool);

  //Указатель на первую структуру в массиве
  mixDets.paDetails = &vBool[0];

  //Установить значение
  vBool[ind].fValue = 1;

  //Установка данных
  res = mixerSetControlDetails(
          reinterpret_cast<HMIXEROBJ>(pLineMix->m_hMix)
          ,&mixDets,MIXER_SETCONTROLDETAILSF_VALUE);

  if(res != MMSYSERR_NOERROR)
    return FALSE;

  return TRUE;
}
```

Производный класс микшера CLineMixer

На производный класс возлагается выполнение всей основной работы по созданию органа управления, установке его в диалоге и управление им. Следуя стремлению создать классы максимально простые в применении, создадим класс для использования его с интеллектуальным указателем (smart pointer) библиотеки STL. Зададимся целью реализовать класс таким образом, чтобы при вызове конструктора объекта класса происходило открытие микшера и первоначальная инициализация. Применение "интеллектуального"

указателя позволяет избавиться от забот о распределении памяти и предотвращает утечку памяти при возникновении исключения, но налагает ряд условий на класс, с которым он будет использоваться.

"Интеллектуальный" указатель предполагает, что объект будет создаваться в куче, поэтому для предупреждения попыток создания объекта в стеке следует заблокировать такую возможность. Второе весьма значительное ограничение, которое существует в библиотеке STL, это невозможность использования "интеллектуального" указателя в качестве элемента контейнера. Точнее, можно создать контейнер "интеллектуальных" указателей, но это грозит большими неприятностями, поскольку при копировании объект "интеллектуального" указателя изменяется. Возможность копирования и присваивания для класса по этой причине также следует запретить.

Реализация этих условий не сложна и не требует большого объема работы. Следует конструктор по умолчанию сделать закрытым. Также определить закрытые конструктор копирования и оператор присваивания, их реализацию можно не делать. Вместо конструктора используется статический псевдо конструктор, который является методом класса. Псевдо конструктор вызывает закрытый конструктор и создает объект на куче. Описанная часть класса показана ниже.

```
class CLineMixer :public CMixer
{
protected:
  //Конструктор
  CLineMixer(DWORD mixID,HWND hwndMsg);

public:
  ~CLineMixer(void);

  //Псевдоконструктор
  static CLineMixer* MakeMixer(DWORD mixID
                          , HWND hwndMsg)
  {
      return new CLineMixer(mixID,hwndMsg);
  };

private:
//Закрытый конструктор копировщик запрещает копирование
  CLineMixer(const CLineMixer & )
    { };

//Закрытый оператор присваивания запрещает присваивания
  const CLineMixer & operator = (const CLineMixer & )     {};

.................................................................................................

}
```

При вызове конструктора производится инициализация номера микшера в базовом классе, а затем вызывается метод базового класса, открывающий микшер.

```
CLineMixer::CLineMixer(DWORD mixID,HWND hwndMsg)
{
  CMixer::m_mixID = mixID;
  CMixer::OpenMixer(hwndMsg);
}
```

В результате можно объединить одной строкой кода создание объекта в куче и открытие микшера, причем отпадает необходимость в корректном удалении объекта.

```
  auto_ptr<CLineMixer>  pM(CLineMixer::MakeMixer(0,m_hWnd));
```

В этой строке используется локальный "интеллектуальный" указатель pM, в результате объект будет удален при выходе из области видимости. Такой результат не решает поставленной задачи, микшер должен быть создан как объект в классе диалога, то есть указатель на микшер должен быть переменной класса диалога. Использовать уже объявленный указатель член класса вместо указателя pM нельзя, здесь проявляется другая тонкость использования "интеллектуальных" указателей, указатель можно инициализировать только прямо по значению, а не обычным присваиванием. При присваивании старый указатель становится недействительным и равным нулю, объект, на который он указывал, теперь будет принадлежать новому указателю. Таким образом, в данном случае нарушается общее, и казалось незыблемое, правило присваивания, правая сторона операции присваивания изменилась, старый указатель стал равен нулю.

Для "интеллектуальных" указателей такое поведение называется передачей прав владения, которое гласит, что два "интеллектуальных" указателя не могут одновременно указывать на один и тот же объект. Поэтому для "интеллектуального" указателя члена класса приходится написать еще одну строку для передачи права владения.

```
m_pMixer = pM;
```

Нужно помнить, что указатель pM после передачи прав владения не может больше использоваться. Существует возможность избежать передачи прав владения. Для этого нужно провести инициализацию "интеллектуального" указателя класса в списке инициализации конструктора диалога.

```
CAudioMixerDlg::CAudioMixerDlg(CWnd* pParent /*=NULL*/)
  : CDialog(CAudioMixerDlg::IDD, pParent)
  ,m_pMixer(CLineMixer::MakeMixer(0,m_hWnd))
{
```

```
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

Все было бы хорошо, если бы не дескриптор окна для обработки сообщений. В момент инициализации указателя он равен нулю, и обработка сообщений производиться не будет. В данном случае такой вариант инициализации указателя оказался неприемлем.

Существует еще один способ передачи прав владения, который позволяет избавиться от ввода промежуточного локального "интеллектуального" указателя. Для этого используется функция reset класса "интеллектуального" указателя. Она производит удаление объекта, которым владеет указатель и инициализацию указателя новым объектом. Поскольку указатель член класса первоначально равен нулю и его удаление безопасно, то можно использовать эту функцию для первичной инициализации.

```
m_pMixer.reset(CLineMixer::MakeMixer(0,m_hWnd));
```

В результате код существенно упрощается и отпадает необходимость ввода промежуточного указателя.

Методы для создания органов управления

Для создания органов управления линии напишем два метода. Первый метод будет создавать органы управления для выходной линии, возвращать ее индекс и тип установленного органа управления (компонента). Второй метод будет создавать органы управления для входной линии, которая связана с выходной линией. Разделение на два метода позволяет определить присутствие запрашиваемой выходной линии, тип установленного органа управления, и только после этого вызывать метод для установки органов управления входной линией.

Метод MakeLineCtrlDST создает на выходной линии два типа органов управления, регулятор с выключателями или мультиплексор. Метод легко расширяется для новых органов управления. Метод возвращает тип установленного органа управления и индекс выходной линии dwDestination.

```
DWORD CLineMixer::MakeLineCtrlDST(CWnd* pwnd
              ,BOOL bSliderOrient,DWORD compTypeDST
              ,DWORD & dwDestination)
{
  DWORD bret = 0;
  if(!m_hMix)
    return bret;

  if(compTypeDST < MIXERLINE_COMPONENTTYPE_DST_FIRST
```

```
      || compTypeDST > MIXERLINE_COMPONENTTYPE_DST_LAST)
    return bret;

  ASSERT(pwnd);

  vector<DATA_CTRL>::iterator posDST;

  DATA_CTRL dt;

  //Найти  линию с компонентом DST
  posDST = find_if(m_vCtrlAll.begin(),m_vCtrlAll.end()
          ,bind2nd(FindTypeComponentDST(),compTypeDST));

  if(posDST == m_vCtrlAll.end())
  {
    dwDestination = -1;
    return bret;
  }

  //Возвращаем индекс выходной линии
  dwDestination = (*posDST).dwDestination;

  //Создаем регулятор и выключатели на линии DST
  if(((*posDST).sMixCtrl.dwControlType
                & MIXERCONTROL_CONTROLTYPE_FADER)
                == MIXERCONTROL_CONTROLTYPE_FADER
                && (*posDST).sMixCtrl.cMultipleItems == 0)
  {
    //Создаем регулятор для выходной линии с данным
    //компонентом
    CFader* pFd = MakeFader(pwnd,posDST,bSliderOrient);

    //Создаем выключатели для данного регулятора
    BOOL ret =
    MakeSwitch(pwnd,pFd,STYLE_CHECK_BOX,bSliderOrient);

    if(ret || pFd)
      bret = CTRL_FADER;
  }

  //Создаем мультиплексор на линии DST
  if(((*posDST).sMixCtrl.dwControlType
              & MIXERCONTROL_CONTROLTYPE_MUX)
              == MIXERCONTROL_CONTROLTYPE_MUX)
  {
    if(MakeMux(pwnd,posDST))
      bret = CTRL_MUX;
  }
  return bret;
}
```

Второй метод MakeLineCtrlSRC создает органы управления на входной линии связанной с выходной линией. На входной линии обычно имеются только регуляторы и выключатели, поэтому метод устанавливает только эти органы управления.

```
BOOL CLineMixer::MakeLineCtrlSRC(CWnd* pwnd
              ,BOOL bSliderOrient,DWORD compTypeSRC
              ,DWORD dwDestination)
{
  BOOL bret =FALSE;
  if(!m_hMix)
    return bret;

  //Только для линий источников
  if(compTypeSRC < MIXERLINE_COMPONENTTYPE_SRC_FIRST
      || compTypeSRC > MIXERLINE_COMPONENTTYPE_SRC_LAST)
    return bret;

  ASSERT(pwnd);

  vector<DATA_CTRL>::iterator posSRC;

  DATA_CTRL dt;
  dt.dwComponentType = compTypeSRC;
  dt.dwDestination = dwDestination;

  //Найти  линию с компонентом SRC
  posSRC = find_if(m_vCtrlAll.begin(),m_vCtrlAll.end()
  ,bind2nd(FindTypeComponentAndDestinasion(),dt));

  if(posSRC== m_vCtrlAll.end())
    return bret;


  //Найти регулятор для этой линии
  dt = (*posSRC);
  posSRC = find_if(m_vCtrlAll.begin(),m_vCtrlAll.end()
  ,bind2nd(FindCtrlTypeFaderForLine(),dt.dwLineID));

  if(posSRC != m_vCtrlAll.end())
  {
    //Создаем регулятор для выходной линии с данным компонентом
    CFader* pFd = MakeFader(pwnd,posSRC,bSliderOrient);
    if(pFd)
      bret = TRUE;
    //Создаем выключатели для данного регулятора
    if(MakeSwitch(pwnd,pFd,STYLE_CHECK_BOX,bSliderOrient))
      bret = TRUE;
  }

  return bret;
}
```

Для создания органа управления определенного типа пишется собственный метод, который подчиняется общему условию, создается орган управления и его данные записываются в вектор установленных элементов. В качестве примера ниже приводится текст метода для создания регулятора. С остальными методами можно ознакомиться в проекте.

```
CFader* CLineMixer::MakeFader(CWnd* pwnd
              , vector<DATA_CTRL>::iterator pos, BOOL bSliderOri-
            ent )
{
  CFader* pFd=0;

  //Если регулятор найден, производим установку
  //Стиль для регулятора баланса не используется
  //при необходимости следует учесть
  DWORD style = STYLE_FADER_HORZ;
  if(!bSliderOrient)
    style = STYLE_FADER_VERT;

  //Только не эквалайзер
  if((*pos).sMixCtrl.cMultipleItems)
    return NULL;

  if(pFd = new CFader(pwnd,style,this))
  {
    //Установка заголовка ограничивающего окна
    //по имени регулятора
    string name((*pos).sMixCtrl.szShortName);

    //Или по имени линии
    //string name((*pos).szShortName);

    name.insert(0," ");
    pwnd->SetWindowText(name.c_str());

    //Запись указателя и позиции регулятора в вектор
    DATA_CTRL dt = *pos;
    dt.pCtrl = static_cast<void*>(pFd);
    dt.rPlace = pFd->myGetPlace();

    //Записать в вектор установленных органов управления
    m_vCtrl.push_back(dt);

    //Начальная установка значения регулятора
    pFd->mySetFaderValueProc(GetValueFader(pFd));

  }
  return pFd;
}
```

Установка параметров органов управления

Для установки и возврата параметров органов управления напишем по два метода для каждого типа органа управления, регулятора и выключателя. Установка параметров мультиплексора разделена на две части, регулятор и выключатели используют методы, описанные ниже, а переключение входов мультиплексора является задачей класса CMux. Методы для управления параметрами органов управления создаются в классе микшера CLineMixer. Таким образом, достигается автономность классов органов управления, из которых вызываются специфические методы, устанавливающие параметры. В самом классе органа управления его показания определяются только в относительных величинах и не имеют отношения к действительным значениям параметра, которым он управляет. Принцип установки и возврата значений параметров одинаков как для регуляторов, так и для выключателей. Поэтому, в качестве примера, рассмотрим пару методов для управления параметрами регулятора.

```
void CLineMixer::SetValueFader(CFader* pFader
                                    , int nProc)
{
  //Найти орган управления по указателю
  vector<DATA_CTRL>::iterator pos;
  pos = find_if(m_vCtrl.begin(),m_vCtrl.end()
                   ,bind2nd(FindCtrlPtr(),pFader));

  if(pos == m_vCtrl.end())
    return;

//Инициализация структуры MIXERCONTROLDETAILS
  MIXERCONTROLDETAILS mixDets = {0};
  mixDets.cbStruct = sizeof(MIXERCONTROLDETAILS);
  mixDets.cbDetails =
              sizeof(MIXERCONTROLDETAILS_UNSIGNED);

  //Управление одноканальное
  mixDets.cChannels = 1; //(*pos).cChannals
  mixDets.dwControlID = (*pos).sMixCtrl.dwControlID;

  //Проверка класса (типа) регулятора
  if(((*pos).sMixCtrl.dwControlType &
         MIXERCONTROL_CT_CLASS_MASK )
               != MIXERCONTROL_CT_CLASS_FADER )
    return;

  //Структура для установки значения
  MIXERCONTROLDETAILS_UNSIGNED mixDetUn = {0};
  mixDets.paDetails = &mixDetUn;

  //Устанавливаем значение величины
```

```
    mixDetUn.dwValue = ((*pos).sMixCtrl.Bounds.dwMaximum
                                    * nProc) / 100;

    MMRESULT ret = mixerSetControlDetails(
            reinterpret_cast<HMIXEROBJ>(m_hMix)
            ,&mixDets,MIXER_SETCONTROLDETAILSF_VALUE);

    ASSERT(ret == MMSYSERR_NOERROR);
  }
```

Установка и возврат состояния выключателей производится аналогично установке других параметров органов управления. Для этого в классе имеются два метода CLineMixer::SetStateSwitch и CLineMixer::GetStateSwitch, ознакомиться с ними можно в проекте.

Удаление органов управления

Указатели на органы управления сохраняются в векторе как указатели на тип void. Во избежание утечки памяти, объекты органов управления расположенные на куче должны быть удалены до автоматического удаления вектора. В методе CLineMixer::DeleteControls просматривается весь вектор данных, и удаляются объекты органов управления. При удалении производится проверка принадлежности указателя к классу.

```
void CLineMixer::DeleteControls(void)
{
  size_t sz = m_vCtrl.size();

  //Удаляем объекты регуляторов
  for(size_t i=0; i< sz; ++i)
  {
    if(m_vCtrl[i].pCtrl)
    {
      CSwitch* pSw =
              static_cast<CSwitch*>(m_vCtrl[i].pCtrl);
      if(!pSw)
          continue;

      if(pSw->IsKindOf(RUNTIME_CLASS(CSwitch)))
      {
        delete pSw;
        m_vCtrl[i].pCtrl = 0;
      }

      CFader* pFd = static_cast<CFader*>(m_vCtrl[i].pCtrl);

      if(!pFd)
          continue;

      if(pFd->IsKindOf(RUNTIME_CLASS(CFader)))
      {
```

```
        delete pFd;
        m_vCtrl[i].pCtrl = 0;
      }

      CMux* pMux = static_cast<CMux*>(m_vCtrl[i].pCtrl);
      if(!pMux)
        continue;

      if(pMux->IsKindOf(RUNTIME_CLASS(CMux)))
      {
        delete pMux;
        m_vCtrl[i].pCtrl = 0;
      }
    }
  }
}
```

Обработка сообщений при изменении параметров микшера другим приложением

Если другое приложение изменяет параметры микшера, то в классе CLineMixer вызывается перегруженная функция OnSetControl. В этой функции в векторе установленных элементов отыскивается орган управления по идентификатору, и производятся вызовы методов для отображения состояния органов управления в собственном приложении.

```
void CLineMixer::OnSetControl(HMIXER hMix,DWORD ctrlID)
{
  if(hMix != m_hMix)
    return;

  //найти орган управления по идентификатору
  vector<DATA_CTRL>::iterator pos;

  pos = find_if(m_vCtrl.begin(),m_vCtrl.end()
                 ,bind2nd(FindCtrlID(),ctrlID));

  //Просматриваем весь вектор для всех органов
  //управления. Если в разных местах приложения
  //установлены несколько экземпляров органов
  //управления для одной и той же линии,
  //то будут отображаться изменения во всех экземплярах
  //органа управления, при условии, что вызов метода не //блоки-
  руется в обработчике сообщения

  while(pos!= m_vCtrl.end() && (*pos).pCtrl)
  {
    CFader* pFd = static_cast<CFader*>((*pos).pCtrl);
    if(pFd && pFd->IsKindOf(RUNTIME_CLASS(CFader)))
    {
      //Значение величины в процентах
```

```
        int nProc = GetValueFader(pFd);
        //Установка положения движка
        pFd->mySetFaderValueProc(nProc);
      }

      CSwitch* pSw = static_cast<CSwitch*>((*pos).pCtrl);
      if(pSw && pSw->IsKindOf(RUNTIME_CLASS(CSwitch)))
      {
        //Установка переключателя
        pSw->SetCheck(GetStateSwitch(*pos));
      }

      CMux* pMux = static_cast<CMux*>((*pos).pCtrl);
      if(pMux && pMux->IsKindOf(RUNTIME_CLASS(CMux)))
      {
        pMux->myDisplayStateMux();
      }

      //Поиск следующего органа управления со следующей
      //позиции
      pos = find_if(++pos,m_vCtrl.end()
                        ,bind2nd(FindCtrlID(),ctrlID));

    }
}
```

Результат работы программы показан на рисунке 5.

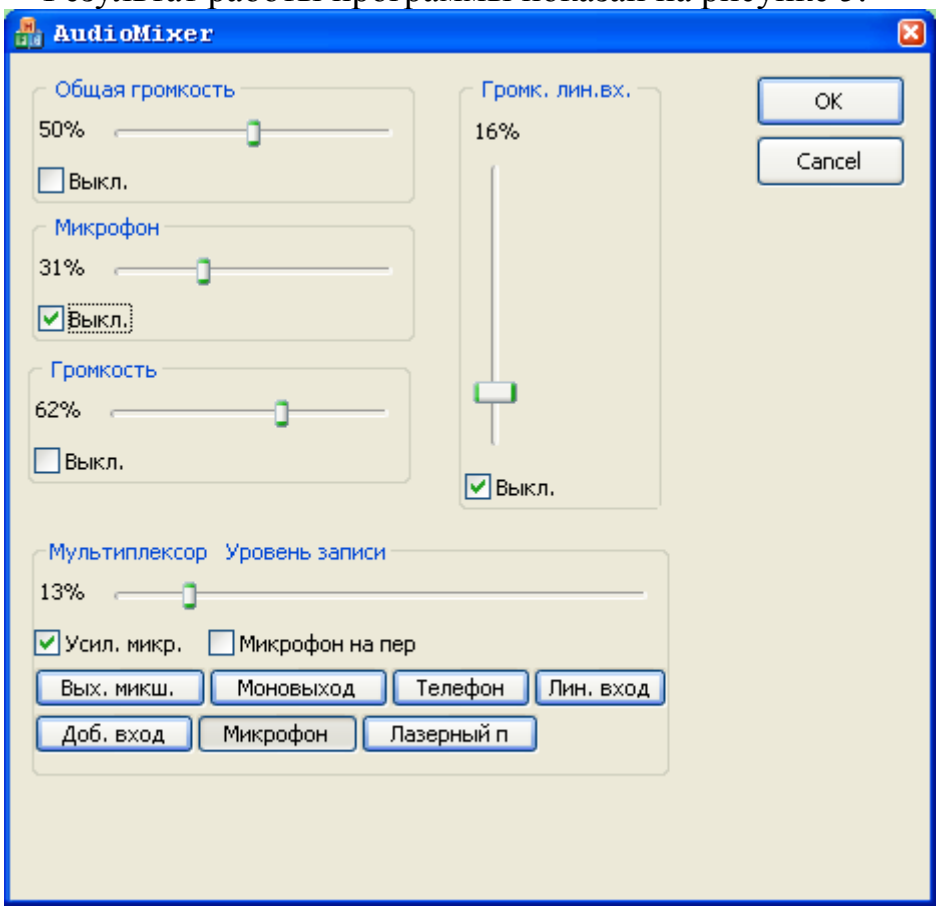

Рис.6. 5. Линии системного микшера.

Примечание к проекту

Для использования стилей Windows XP, среда разработки создает файл под именем проекта с расширением manifest. Этот файл следует поместить в папку ресурсов. В файле ресурсов AudioMixer.rc, в конце дописать строку CREATEPROCESS_MANIFEST_RESOURCE_ID RT_MANIFEST ".\\res\\AudioMixer.manifest", в которой указан путь к файлу манифеста. Более подробно ознакомиться с применением файла манифеста можно в разделе Using Windows XP Visual Styles документации MSDN.