Глава 5
Звуковой канал DirectShow

Глава 5
Звуковой канал DirectShow....1
Сбор и предварительное прослушивание звука....1
Интерфейсы звукового канала....5
Интерфейс IAMAudioInputMixer....7
Установка фильтра захвата звука....15
Установка медиа формата на звуковом фильтре....16
Интерфейс пользователя для управления звуковыми форматами....23
Управление параметрами звуковой карты....24
Определение свойств контактов звукового фильтра....26
Установка параметров органов управления....31
Результаты экспериментальной проверки....32
Реализация управления системным микшером....33

Сбор и предварительное прослушивание звука

Перед чтением этого раздела следует предварительно ознакомиться с главой "Системный звуковой микшер".

Для обработки звука DirectShow предоставляет фильтр захвата звука базирующийся на звуковой карте. Каждая карта установленная в системе представляет отдельный экземпляр фильтра. Кроме фильтра сбора звука DirectShow предоставляет еще два фильтра, фильтр Audio Renderer (WaveOut) и фильтр DirectSound Renderer. Фильтр Audio Renderer (WaveOut) использует waveOut* API для обработки звука. Фильтр DirectSound Renderer реализует те же самые возможности, что и фильтр Audio Renderer, но по умолчанию используется только фильтр DirectSound Renderer. При записи телепрограммы фильтр DirectSound Renderer обеспечивает канал прослушивания записываемого звука. В режиме предварительного просмотра в канале звука может использоваться фильтр DirectSound Renderer, но фильтр звуковой карты полностью обеспечивает прослушивание телепрограммы, и принципиальной необходимости в фильтре DirectSound Renderer нет. Возможно, в каких-то особых случаях могут понадобиться эффекты DirectSound при прослушивании телепрограммы, но для обычных задач звукового сопровождения телевизионного просмотра фильтр DirectSound Renderer не нужен. Данный фильтр необходим для контроля звукового канала, если производится запись телепрограммы. В

тоже время данный фильтр принимает участие в процессе построения звуковой ветви графа по умолчанию, то есть при использовании методов "интеллектуального" построения графа, что приводит к некоторым проблемам. Подробно этот вопрос рассматривается далее.

Если следовать рекомендациям документации DirectShow, то ветка графа для записи и прослушивания звука должна строиться так, как показано на рисунке 5.1. Построим граф в соответствии с рекомендациями и проверим его функционирование.

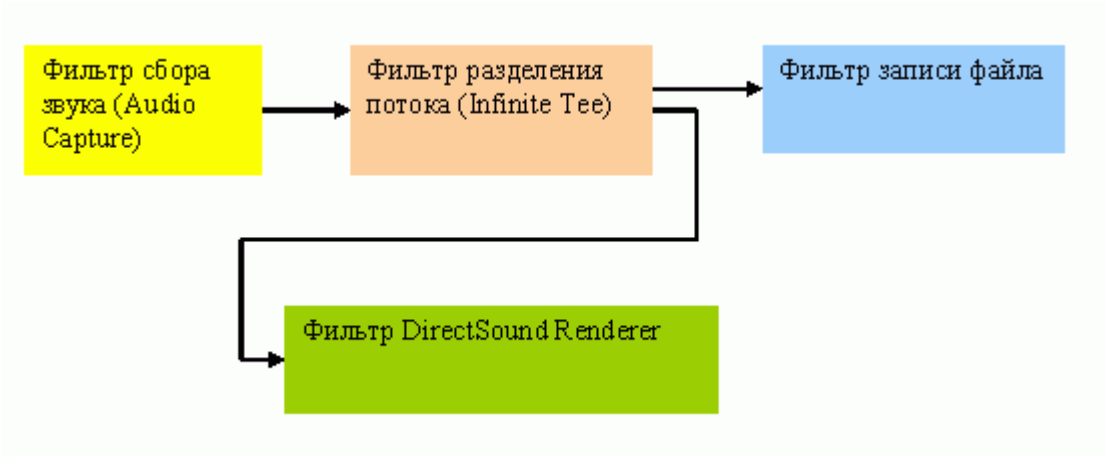

Рис.5. 1. Сбор и воспроизведения звука.

Для реализации ветви графа в таком виде, необходимо произвести установку фильтра сбора в граф по данным моникера, установку фильтра Infinite Pin Tee и достройку звуковой части графа до воспроизводящего фильтра звука включительно. Запись звука в файл будет рассматриваться отдельно в главе посвященной записи файлов. Фильтр Infinite Pin Tee устанавливается и подключается только вручную. С выхода фильтра можно Infinite Pin Tee можно использовать методы для автоматической достройки графа.

Метод CCapture::InitCaptureAudio строит граф обработки звука в соответствии с рекомендациями DirectShow.

```
BOOL CCapture::InitCaptureAudio(IMoniker* pMoniker)
{
  HRESULT hr=S_OK;
  BOOL bret = TRUE;
  string sErr = "Error InitCaptureAudio";

  //Указатель на фильтр Infinite Pin Tee
  IBaseFilter* pIT=0;

  //Указатели на контакты фильтров
  IPin* pOutCapA=0, *pInIT=0, *pOutIT=0;
```

```
try
{
  pMoniker->AddRef();

  //Установка фильтра звуковой карты
  if(S_OK != (hr=
  m_pGraph2->AddSourceFilterForMoniker(
        pMoniker ,NULL,NULL,&m_pCapA)))
    throw hr;

  //Если использовать RenderStream  установится
  //Smart Tee. Он будет работать, но рекомендуется
  //использовать фильтр Infinite Pin Tee

  //hr = m_pBuilder->RenderStream(
  //     &PIN_CATEGORY_CAPTURE,&MEDIATYPE_Audio
  //           , m_pCapA, NULL,NULL);

  //Установка фильтра разделителя данных
  //Infinite Pin Tee
  if(S_OK != (hr = CoCreateInstance(CLSID_InfTee
        ,NULL,  CLSCTX_INPROC,IID_IBaseFilter
        ,reinterpret_cast<void **>(&pIT))))
    throw hr;

  if(S_OK != (hr = m_pGraph->AddFilter(pIT
                          ,L"Infinite Pin Tee")))
    throw hr;

  //Находим выходной контакт звуковой карты
  if(S_OK != (hr =
      m_pBuilder->FindPin(m_pCapA,PINDIR_OUTPUT
                        ,&PIN_CATEGORY_CAPTURE
                        ,0,0,0,&pOutCapA)))
    throw hr;

  //Находим входной контакт фильтра Infinite Pin Tee
  if(S_OK != (hr =
            m_pBuilder->FindPin(pIT,PINDIR_INPUT
            , NULL,0,0,0,&pInIT)))
    throw hr;

  //Находим выходной контакт фильтра Infinite Pin Tee
  if(S_OK != (hr =
            m_pBuilder->FindPin(pIT,PINDIR_OUTPUT
            ,NULL,0,0,0,&pOutIT)))
    throw hr;

  //Подключаем Infinite Pin Tee к звуковой карте
  if(S_OK != (hr =
            m_pGraph2->Connect(pOutCapA,pInIT)))
    throw hr;
```

```
//Достраиваем граф от выхода Infinite Pin Tee, метод будет
//устанавливать воспроизводящий звуковой фильтр по умолчанию
    if(S_OK != (hr = m_pGraph2->Render(pOutIT)))
      throw hr;
  }
  catch(HRESULT h)
  {
    bret = 0;
    CTVshowApp::MessageError(h,sErr);
  }

  pMoniker->Release();

  if(pIT)   { pIT->Release(); pIT=0; }
  if(pOutCapA)  { pOutCapA->Release(); pOutCapA=0; }
  if(pInIT)   { pInIT->Release(); pInIT=0; }
  if(pOutIT)    { pOutIT->Release(); pOutIT=0; }

  return bret;
}
```

Результат работы метода можно увидеть в графическом редакторе. Часть картинки графического редактора показана ниже. Часть графа для записи не построена, но она пока и нужна.

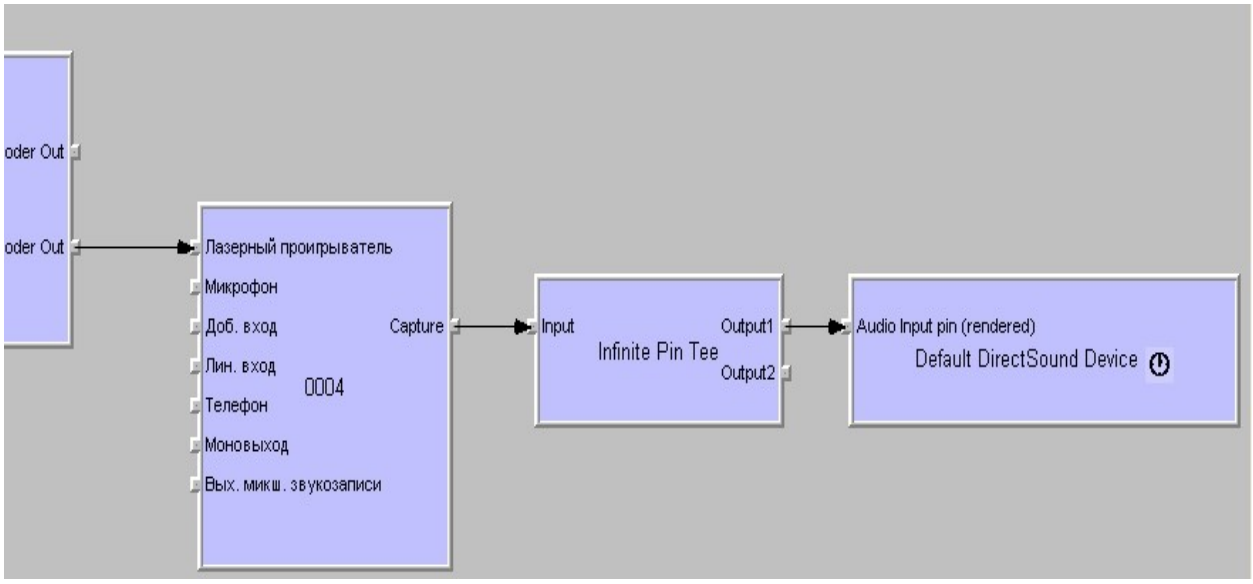

Рис.5. 2. Реализация воспроизведения звука.

Возможно, что такая схема построения звуковой ветви графа для определенных типов звуковых карт не доставит никаких неприятностей, и будет работать хорошо.

Проблема заключается в том, что ни фильтр DirectSound Renderer, ни фильтр звуковой карты не обеспечивают полноценного управления звуковым микшером, что может привести к образованию канала обратной связи через шину PCI или параллельного (двойного) канала звука. То есть предоставляе-

мые интерфейсы фильтра звуковой карты и (или) интерфейсы фильтра DirectSound Renderer обеспечивают только частичное управление микшером звуковой карты, не перекрывая возможностей управления даже интегрированной звуковой карты. Более подробно об этом рассказано в главе "Системный звуковой микшер".

Например, если звуковая карта допускает подключение линии …SRC_WAVEOUT к входу мультиплексора записи, а такая возможность есть у многих типов карт, то при записи сигнала через смеситель канала воспроизведения возможно появление цепи обратной связи. Вследствие этого возникает многократное эхо в структуре записываемого сигнала. Предупредить появление эха при таком режиме записи можно, отключив линию …SRC_WAVEOUT , но это невозможно средствами DirectShow и приходится привлекать системную программу sndvol32.exe. Если не использовать фильтр DirectSound Renderer, то удается избежать появления канала обратной связи, но при этом теряется возможность контролировать на слух уровень записываемого звукового сигнала. Прослушать сигнал можно по каналу воспроизведения звука, но это будет не записываемый сигнал, а сигнал с выхода воспроизведения звуковой карты. Если только просматривать телепередачи, то тогда этого достаточно, но в и этом случае не удается организовать управление каналом звука на входе (линейном) карты, поскольку интерфейс IAMAudioInputMixer управляет только линией записи. Таким образом, невозможность контролировать линию воспроизведения средствами DirectShow приводит к дополнительным проблемам.

Подводя итог сказанному выше, можно сделать вывод, что ни интерфейс IAMAudioInputMixer на фильтре звуковой карты ни фильтр DirectSound Renderer не обеспечивают полноценного управления. В результате приходится привлекать системную программу sndvol32.exe для предварительной настройки канала звука, какие-то входы отключать, другие подключать. Процесс требует определенных знаний и зачастую ставит в тупик пользователя. Дополнительно учитывая тот факт, что уровни звука разных телепрограмм могут значительно отличаться по уровню сигнала, можно констатировать прискорбный факт частого обращения к системной программе sndvol32.exe при просмотре телепередач. Некоторые источники предлагают создание дополнительного фильтра DirectShow, который в определенной степени решает эту проблему. Это не самое лучшее решение. Можно решить задачу проще, не создавая фильтра, а введя в программу управление системным микшером.

Интерфейсы звукового канала

Все фильтры, участвующие в решении задачи сбора звука предоставляют несколько интерфейсов, которые сведены таблицу 1.

Интерфейс	Описание
IAMAudioInputMixer	Управление звуковым потоком на входных контактах

	звукового фильтра сбора. Интерфейс реализуется на входных контактах, может быть реализован и на самом фильтре сбора звука.
IAMAudioRendererStats	Возвращает статистическую информацию о параметрах обработки звукового потока в воспроизводящем фильтре. Интерфейс реализован на фильтрах Audio Renderer (WaveOut) и DirectSound Renderer.
IAMBufferNegotiation	Управляет распределением буфера звукового фильтра сбора. Интерфейс должен поддерживаться фильтром сбора. Размер буфера определяет эффективность и качество воспроизведения звука. Предоставляется на выходном контакте фильтра сбора.
IAMClockSlave	Управляет допустимой ошибкой воспроизведения звука при синхронизации звукового потока от внешнего временного источника. Изменять установки данного интерфейса не рекомендуется за исключением особых случаев. Интерфейс реализован на фильтрах Audio Renderer (WaveOut) и DirectSound Renderer.
IAMDirectSound	Позволяет приложению указать на активное окно, управляющее воспроизведением с применением DirectSound. Интерфейс реализован на фильтрах Audio Renderer (WaveOut) и DirectSound Renderer. Остальные методы не реализованы.
IAMResourceControl	Позволяет приложению захватить и сохранить ресурсы звукового устройства.
IAMStreamConfig	Осуществляет управление медиа форматом звукового потока в фильтре сбора звука. Интерфейс реализован на выходном контакте фильтра.
IBasicAudio	Управляет громкостью и стереобалансом. Интерфейс реализован на фильтрах Audio Renderer (WaveOut) и DirectSound Renderer.Доступ к интерфейсу предоставляет менеджер графа.

Таблица 1. Интерфейсы сбора и воспроизведения звука.

Основными интерфейсами для управления сбором звука являются интерфейс фильтра звуковой карты, IAMAudioInputMixer, который управляет входным микшером фильтра сбора и интерфейс IAMStreamConfig, который устанавливает медиа формат звукового потока. Методы интерфейса IAMStreamConfig рассматривались ранее в главе посвященной предварительному просмотру. Остальные интерфейсы предоставляют редко требуемые

возможности по управлению звуковым потоком. Возможности интерфейса IBasicAudio ограничиваются простейшими регулировками громкости и стереобаланса и перекрываются интерфейсом IAMAudioInputMixer фильтра звуковой карты. Интерфейс IAMDirectSound, представляемый фильтром DirectSound Renderer, реализует только два метода из заявленных восьми, которые управляют фокусом окна при воспроизведении файла.

Таким образом, следует рассмотреть только возможности предоставляемые интерфейсом IAMAudioInputMixer.

Интерфейс IAMAudioInputMixer

Интерфейс IAMAudioInputMixer рассмотрим подробно, поскольку его можно использовать для управления параметрами звука при записи. Данный интерфейс управляет входами звуковой карты, которые представляются как контакты фильтра DirectShow. Он управляет параметрами звукового потока на входах фильтра, например уровнем сигнала, включением и отключением входов и некоторыми другими. Интерфейс использует все возможности управления, которые предоставляет линия записи …DST_WAVEIN и дополняет ее, например реализацией регулировки стереобаланса. Интерфейс может быть реализован и на самом фильтре сбора для управления общими параметрами. Интерфейс управляет только органами управления на линии записи звуковой карты …DST_WAVEIN, он не обеспечивает полного управления звуковым микшером. Реализация этого интерфейса определяется свойствами используемых устройств и зависит от производителя этих устройств.

Метод	Описание
put_Enable	Разрешает или запрещает прохождение сигнала через входной контакт.
get_Enable	Определяет состояние входного контакта.
put_Mono	Устанавливает режим моно.
get_Mono	Определяет состояние режима моно.
put_MixLevel	Устанавливает уровень записываемого сигнала на входе.
get_MixLevel	Возвращает уровень записываемого сигнала на входе.
put_Pan	Устанавливает уровень стереобаланса.
get_Pan	Возвращает уровень стереобаланса.
put_Loudness	Разрешает подъем нижних частот на малой громкости (Loudness).
get_Loudness	Возвращает состояние выключателя подъема нижних частот на малой громкости.
put_Treble	Устанавливает уровень высоких частот.
get_Treble	Возвращает уровень высоких частот.
get_TrebleRange	Возвращает диапазон регулировки высоких частот.

put_Bass	Устанавливает уровень низких частот.
get_Bass	Возвращает уровень низких частот.
get_BassRange	Возвращает диапазон регулировки низких частот.

Таблица 2. Интерфейс IAMAudioInputMixer.

Метод put_Enable

Управляет прохождением сигнала через входной контакт. Реализуется на входных контактах.

```
HRESULT put_Enable(
  BOOL fEnable
);

Параметры

fEnable
Указывает разрешить или запретить прохождение сигнала.
Значение TRUE разрешат прохождение сигнала.
Значение FALSE  запрещает прохождение сигнала.

Возвращаемые величины

Значение S_OK указывает на удачное завершение.
Значение E_NOTIMPL  указывает, что метод не реализован.
```

Метод get_Enable

Метод возвращает состояние входа, включен или отключен. Реализуется на входных контактах.

```
HRESULT get_Enable(
  BOOL *pfEnable
);

Параметры

pfEnable
Указатель на переменную, которая принимает состояние входа.

Значение TRUE означает, что вход  пропускает сигнал.
Значение FALSE  означает, что вход  не пропускает сигнал.

Возвращаемые величины
```

Код	Значение
S_OK	Выполнено.
E_NOTIMPL	Метод не реализован.
E_POINTER	Нулевой указатель.

Метод put_Mono

Метод устанавливает режим моно для входного контакта.

```
HRESULT put_Mono(
  BOOL fMono
);
```

Параметры

fMono
Значение TRUE устанавливает режим моно.
Значение FALSE устанавливает мультиканальный режим.

Возвращаемые величины

Код	Описание
E_FAIL	Ошибка.
S_OK	Выполнение успешное.

Метод get_Mono

Определяет, установлен ли режим моно.

```
HRESULT get_Mono(
  BOOL *pfMono
);
```

Параметры

pfMono
Указатель на переменную, принимающую результат запроса.
Значение TRUE указывает, что установлен режим моно.
Значение FALSE указывает, что установлен мультиканальный режим.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод put_MixLevel

Метод устанавливает уровень сигнала на входном контакте.

```
HRESULT put_MixLevel(
  double Level
```

```
);
```

Параметры

Level
Уровень сигнала. Возможны следующие значения.

Величина	Описание
От 0.0 до 1.0	Нуль указывает, что сигнал отсутствует. Единица указывает на установку полной громкости.
AMF_AUTOMATICGAIN	Разрешает автоматическую регулировку уровня.

Возвращаемые величины

Код	Описание
S_OK	Успешное выполнение.
E_NOTIMPL	Фильтр не поддерживает флаг AMF_AUTOMATICGAIN.
E_INVALIDARG	Аргумент неправильный.

Метод get_MixLevel

Метод возвращает уровень сигнала.

```
HRESULT get_MixLevel(
  double *pLevel
);
```

Параметры

pLevel
Указатель на переменную, принимающую результат выполнения метода.

Возможны следующие значения.

Величина	Описание
От 0.0 до 1.0	Нуль указывает, что сигнал отсутствует. Единица указывает на установку полной громкости.
AMF_AUTOMATICGAIN	Разрешает автоматическую регулировку уровня.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод put_Pan

Метод устанавливает уровень стереобаланса.

```
HRESULT put_Pan(
  double Pan
);
```

Параметры
Pan
Возможны значения в диапазоне от -1.0 до 1.0.

Величина	Описание
-1	Полностью только левый
0	Центр
1	Полностью только правый

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод get_Pan

Возвращает уровень стереобаланса.

```
HRESULT get_Pan(
  double *pPan
);
```

Параметры

pPan
Указатель на переменную принимающую значение величины. Возможны значения в диапазоне от -1.0 до 1.0.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод put_Loudness

Метод разрешает или запрещает подъем низких частот при пониженных уровнях громкости (тонкомпенсация).

```
HRESULT put_Loudness(
  BOOL fLoudness
);
```

Параметры

fLoudness
Значение TRUE включает тонкомпенсацию.
Значение FALSE выключает тонкомпенсацию.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод get_Loudness

Метод возвращает текущее состояние выключателя тонкомпенсации.

```
HRESULT get_Loudness(
  BOOL *pfLoudness
);
```

Параметры

pfLoudness

Указатель на переменную, принимающую результат выполнения метода.
Значение TRUE означает, что тонкомпенсация включена.
Значение FALSE означает, что тонкомпенсация выключена.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод put_Treble

Устанавливает уровень верхних частот.

```
HRESULT put_Treble(
  double Treble
);
```

Параметры

Treble
Указывает значение в децибелах.

Возвращаемые величины

Код	Описание

S_OK	Успешное выполнение.
E_INVALIDARG	Аргумент вышел из допустимого диапазона.

Метод get_Treble

Возвращает значение уровня верхних частот.

```
HRESULT get_Treble(
  double *pTreble
);
```

Параметры

pTreble
Переменная, принимающая результат выполнения метода в децибелах.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод get_TrebleRange

Метод возвращает допустимый диапазон установки уровня верхних частот.

```
HRESULT get_TrebleRange(
  double *pRange
);
```

Параметры

pRange
Указатель на переменную, принимающую результат выполнения метода.
Например, значение 6.0 означает диапазон величин от -6.0 до 6.0 децибел.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод put_Bass

Устанавливает уровень нижних частот.

```
HRESULT put_Bass(
  double Bass
);
```

Параметры

Bass
Указывает значение в децибелах.

Возвращаемые величины

Код	Описание
S_OK	Успешное выполнение.
E_INVALIDARG	Аргумент вышел из допустимого диапазона.

Метод get_Bass

Возвращает значение уровня нижних частот.

```
HRESULT get_Bass(
  double *pBass
);
```

Параметры

pBass
Переменная, принимающая результат выполнения метода в децибелах.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Метод get_BassRange

Метод возвращает допустимый диапазон установки уровня нижних частот.

```
HRESULT get_BassRange(
  double *pRange
);
```

Параметры

pRange
Указатель на переменную, принимающую результат выполнения метода.
Например, значение 6.0 означает диапазон величин от -6.0 до 6.0 децибел.

Возвращаемые величины

При успешном выполнении возвращает S_OK, в противном случае общий код ошибки HRESULT.

Установка фильтра захвата звука

Процесс установки фильтр захвата звука в граф ничем не отличается от аналогичного процесса для фильтра захвата видео, который был рассмотрен ранее. Для этого также используется системный перечислитель, который производит отбор моникеров всех звуковых устройств. Перечисление моникеров звуковых устройств выполняется в том же методе CTVshowApp::myEnumDevice, который перечисляет и видео устройства.
Фильтр захвата звука должен устанавливаться в граф до начала построения графа. Это связано с тем, что некоторые методы DirectShow могут требовать участия фильтра захвата звука при автоматическом построении графа.
В большинстве случаев звуковая карта имеет один выход, который обеспечивает запись звука и одновременно является выходом для прослушивания. В некоторых случаях, когда звуковая карта не позволяет произвести запись звука и его одновременное прослушивание с выхода записи карты, следует подключить фильтр для разделения потока звуковых данных на две части. Именно так рекомендует делать документация DirectShow. Для выполнения разделения потока данных рекомендуется использовать фильтр Infinite Pin Tee, который имеет один входной контакт и выходных столько, сколько потребуется. Он разделяет входной поток данных без копирования и принимает поток с любым медиа форматом. Создание и подключение фильтра Infinite Pin Tee к звуковому фильтру производится вручную. Подключение Infinite Pin Tee к воспроизводящему звуковому фильтру DirectSound Renderer можно производить автоматически с помощью метода IGraphBuilder::Render.
Принимать решение использовать фильтр DirectSound Renderer или нет, следует с учетом технических характеристик реальной звуковой карты. В общем случае наиболее удачным решением следует считать установку фильтра DirectSound Renderer и реализацию управления системным микшером. Такое решение позволит использовать все возможности, предоставляемые звуковой картой в рамках одного приложения.
Следует отметить, что использование метода RenderStream для построения графа, начиная с выхода звукового фильтра, вызовет подключение к выходному контакту фильтра Smart Tee, который будет вполне успешно работать, по крайней мере, в пределах проведенных испытаний. Фильтр Smart Tee обладает избыточной функциональностью и его не следует использовать для разделения звукового потока.

Построение ветви графа для воспроизведения и записи звука может отличаться от варианта, рассматриваемого здесь. Некоторые устройства могут не полностью соответствовать требованиям DirectShow или не реализовывать их в полном объеме, и тогда при построении канала звука следует учитывать их индивидуальные особенности.

Установка фильтра захвата звука производится в обработчике CTVshowView::OnNMClickListVideo сразу после инициализации построителя

графа и перед инициализацией фильтра захвата видео.

```
.....................................................  //Про-
веряем наличие звуковой карты
    if(m_arIMonDev_A.IsEmpty())
    {
      AfxMessageBox(
      _T("Не найдено звуковое устройство, дальнейшая
                работа невозможна"));
      return;
    }

    //Установить звуковой фильтр сбора по умолчанию,
    //основное устройство всегда имеет нулевой номер
    idx = m_listAudio.GetItemData(0);

    //Инициализируем фильтр сбора звука
    if(!pCap->InitCaptureAudio(m_arIMonDev_A[idx]))
    {
      AfxMessageBox(
            _T("Ошибка при инициализации фильтра
            захвата звука"));
      return;
    }
.....................................................
```

Установка медиа формата на звуковом фильтре

Основным параметром, определяющим качество записи и воспроизведения звука, является текущий медиа формат звукового фильтра. Установка медиа формата на звуковом фильтре несколько проще по сравнению с аналогичной установкой для видеоизображения. Для установки формата звука используется тот же самый интерфейс IAMStreamConfig и уже известная структура AM_MEDIA_TYPE, в которой член formattype установлен в значение FORMAT_WaveFormatEx. Структура AM_MEDIA_TYPE, описывающая звук, в качестве форматирующего блока содержит только одну структуру WAVEFORMATEX. Данные второй структуры AUDIO_STREAM_CONFIG_CAPS, которая описывает диапазон форматов, могут понадобиться только для первоначальных установок, да и то не всегда. В редких случаях может понадобиться представление пользователю параметров этих структур для управления. Ниже представлено описание структуры AUDIO_STREAM_CONFIG_CAPS.

Структура описывает диапазон звуковых форматов.

```
typedef struct _AUDIO_STREAM_CONFIG_CAPS {
    GUID guid;
    ULONG MinimumChannels;
```

```
    ULONG MaximumChannels;
    ULONG ChannelsGranularity;
    ULONG MinimumBitsPerSample;
    ULONG MaximumBitsPerSample;
    ULONG BitsPerSampleGranularity;
    ULONG MinimumSampleFrequency;
    ULONG MaximumSampleFrequency;
    ULONG SampleFrequencyGranularity;
} AUDIO_STREAM_CONFIG_CAPS;
```

Члены структуры.

guid
Должен быть установлен тип MEDIATYPE_Audio.

MinimumChannels
Минимальное число каналов.

MaximumChannels
Максимальное число каналов.

ChannelsGranularity
Шаг при установке числа каналов. Например, при четырех каналах фильтр может устанавливать их с шагом 2 начиная от минимального значения. Таким образом, допустимые значения 2 и 4.

MinimumBitsPerSample
Минимальное число бит на один отсчет сигнала.

MaximumBitsPerSample
Максимальное число бит на один отсчет сигнала.

BitsPerSampleGranularity
Шаг при установке числа бит на один отсчет сигнала

MinimumSampleFrequency
Минимальная частота оцифровки сигнала..

MaximumSampleFrequency
Максимальная частота оцифровки сигнала.

SampleFrequencyGranularity
Шаг при установке частоты оцифровки сигнала.

Для упрощения реализации данные структуры AUDIO_STREAM_CONFIG_CAPS использовать не будем, поскольку фильтр предоставляет определенное число типовых наборов параметров, которые возвращаются в нескольких структурах WAVEFORMATEX. Тогда появляется возможность значительно упростить процесс установки формата для фильтра захвата звука по сравнению с фильтром захвата видео. Установка формата звука возможна без разъединения фильтров, необходимо только

остановить граф, установить формат и опять запустить граф. Разъединения фильтров не требуется, поскольку по умолчанию воспроизводящий фильтр звука принимает все форматы поддерживаемые обычной звуковой картой. Дополнительное согласование форматов может понадобиться только в случае применения особой звуковой карты и (или) специальной обработки звука специфическим фильтром, что не входит в круг рассматриваемых задач.

Структура WAVEFORMATEX описывает формат звуковых данных. Структура содержит информацию, общую для всех звуковых данных многоканальной передачи звука. Члены структуры описывают только один или два канала звуковых данных. Если число каналов больше двух эта структура включается как первый член в структуру WAVEFORMATEXTENSIBLE, которая содержит дополнительную информацию. Структура WAVEFORMATEX объявлена в файле Mmreg.h

```
typedef struct {
  WORD  wFormatTag;
  WORD  nChannels;
  DWORD nSamplesPerSec;
  DWORD nAvgBytesPerSec;
  WORD  nBlockAlign;
  WORD  wBitsPerSample;
  WORD  cbSize;
} WAVEFORMATEX;
```

Члены структуры.

wFormatTag
Указывает на тип формата. Весь список форматов находится в файле Mmreg.h. Для одноканальных и двуканальных данных тип формата WAVE_FORMAT_PCM.

nChannels
Число каналов. Для моно данных один канал, для стерео данных два канала.

nSamplesPerSec
Частота оцифровки аналогового сигнала. Для формата WAVE_FORMAT_PCM этот член может иметь значения 8.0 kHz, 11.025 kHz, 22.05 kHz, и 44.1 kHz.

wBitsPerSample
Разрядность чисел при оцифровке аналогового сигнала. Для формата WAVE_FORMAT_PCM этот член может иметь значения 8 или 16 бит. Для формата WAVE_FORMAT_EXTENSIBLE это число может иметь любое значение кратное 8. В некоторых случаях при сжатии данных этот член может быть равен 0.

nBlockAlign
Размер минимальной порции данных (блок) в байтах. Определяется как произведение nChannels на wBitsPerSample и деленное на 8.

При обработке данных они всегда должны выравниваться на границу блока.

nAvgBytesPerSec
Требуемая скорость передачи данных в байтах за секунду. Этот член определяется как произведение nSamplesPerSec на nBlockAlign.

cbSize
Число байт дополнительных данных присоединенных к концу структуры, для форматов отличных от PCM форматов. Для формата WAVE_FORMAT_PCM этот член игнорируется. Если дополнительных данных нет, этот член должен быть равен нулю.

Для установки формата на звуковом фильтре следует написать методы аналогичные методам для установки формата на фильтре захвата видео и написать класс для хранения данных. Поскольку для установки формата достаточно только структуры WAVEFORMATEX, то можно было не создавать класс данных, но он может быть полезен для более сложных случаев обработки звуковых данных. Реализован класс данных аналогично классу данных для фильтра захвата видео и поэтому в особых комментариях не нуждается.

```
class CFmtAudio
{
public:
  CFmtAudio() //Конструктор
  {
  };
  ~CFmtAudio() {};

  //Конструктор копировщик
  CFmtAudio(const CFmtAudio & mediaFmt)
{
  *this = mediaFmt;
};

  //Структура формата
  WAVEFORMATEX wfmt;

  //Оператор присваивания
  const CFmtAudio & operator =
                      (const CFmtAudio & mediaFmt)
  {
    if(this != &mediaFmt)
    {
      ::CopyMemory(&wfmt,&mediaFmt
                           ,sizeof(WAVEFORMATEX));
    }
    return *this;
  };
```

```
  //Оператор <
  bool operator < (const CFmtAudio & mediaFmt) const
  {
    return
            wfmt.nChannels <  mediaFmt.wfmt.nChannels
            && wfmt.nSamplesPerSec
            <  mediaFmt.wfmt.nSamplesPerSec
            && wfmt.nAvgBytesPerSec
            <   mediaFmt.wfmt.nAvgBytesPerSec
            && wfmt.nBlockAlign <
            mediaFmt.wfmt.nBlockAlign;
  };

  bool operator == (const CFmtAudio& mediaFmt) const
  {
    return
    wfmt.wFormatTag  == mediaFmt.wfmt.wFormatTag
    && wfmt.nChannels ==  mediaFmt.wfmt.nChannels
    && wfmt.nSamplesPerSec ==
                        mediaFmt.wfmt.nSamplesPerSec
    && wfmt.nAvgBytesPerSec ==
                        mediaFmt.wfmt.nAvgBytesPerSec
    && wfmt.nBlockAlign ==  mediaFmt.wfmt.nBlockAlign;
  };
};
```

Для возврата свойств звукового фильтра пишем метод CCapture::GetAudioCaps аналогично методу для фильтра захвата видео, но значительно проще из-за меньшего объема данных. Метод заполняет данными вектор, который является членом класса. При возврате свойств возвращаются типовые наборы параметров звука, которые будут предоставляться пользователю для выбора. Также напишем еще два метода. Один для возврата текущего формата звука и второй для установки нового формата. Все методы используют уже знакомый интерфейс управления форматами IAMStreamConfig.

```
BOOL CCapture::GetAudioCaps()
{
  BOOL bret = TRUE;
  HRESULT hr = S_OK;
  CComPtr<IAMStreamConfig> pSC;
  AM_MEDIA_TYPE *pmt = 0;

  AUDIO_STREAM_CONFIG_CAPS ascc;
  int count =0; int size = 0;

  string sErr = "Ошибка CCapture::GetStreamCaps ";

  CFmtAudio fmtA;
```

```
  try
  {
    //Найти IAMStreamConfig для выходного контакта фильтра //зву-
    ка
    if(S_OK !=(hr = m_pBuilder->FindInterface(
          &PIN_CATEGORY_CAPTURE
                  ,&MEDIATYPE_Audio,m_pCapA
                  ,IID_IAMStreamConfig
                  ,reinterpret_cast<void**>(&pSC))))
      throw hr;

    //Определяем общее количество свойств
    if(S_OK != (hr =
          pSC->GetNumberOfCapabilities(&count,&size)))
      throw hr;

    if(!count)
      throw hr = E_FAIL;// нет свойств формата

    for(int i = 0; i< count; ++i)
    {
      //Заполняем структуры форматов
      if(S_OK !=(hr = pSC->GetStreamCaps(i,&pmt
                      ,reinterpret_cast<BYTE*>(&ascc))))
        throw hr;

      if(pmt->formattype == FORMAT_WaveFormatEx
          && pmt->subtype==MEDIASUBTYPE_PCM)
      {
        fmtA.wfmt =
        *reinterpret_cast<WAVEFORMATEX*>(pmt->pbFormat);

        //Добавить в вектор
        m_vFmtA.push_back(fmtA);
      }
    }
  }
  catch(HRESULT )
  {
    bret = 0;
    //CTVshowApp::myMessageError(h,sErr);
  }

  DeleteMediaTypeAndFormat(pmt);

  return bret;
}
```

Возврат текущего формата.

```
AM_MEDIA_TYPE* CCapture::GetFmtAudio(void)
{
  HRESULT hr = S_OK;
```

```
  CComPtr<IAMStreamConfig> pSC;

  try
  {
    //Найти IAMStreamConfig для выходного контакта
    if(S_OK != (hr = m_pBuilder->FindInterface(
                &PIN_CATEGORY_CAPTURE,
                        &MEDIATYPE_Audio
                        ,m_pCapA,IID_IAMStreamConfig
                        ,reinterpret_cast<void**>(&pSC))))
      throw hr;

    //Освободить текущую структуру со старыми данными
    if(m_pMediaType)
    {
      DeleteMediaTypeAndFormat(m_pMediaType);
      m_pMediaType = 0;
    }

    //Заполнить структуру новыми данными
    if(S_OK != (hr = pSC->GetFormat(&m_pMediaType)))
      throw hr;

  }
  catch(HRESULT)
  {
    //освободить структуру AM_MEDIA_TYPE только в случае   //ис-
ключения
    if(m_pMediaType)
    {
      DeleteMediaTypeAndFormat(m_pMediaType);
      m_pMediaType = 0;
    }
    //CTVshowApp::myMessageError(h,sErr);
  }

  return m_pMediaType;}
```

Установка звукового формата в методе SetFmtAudio имеет некоторую особенность, которая заключается в том, что перед вызовом метода, структура AM_MEDIA_TYPE (член класса CCapture) уже заполнена данными формата, который выбрал пользователь. Поэтому в методе при установке формата просто используется указатель на структуру без ее заполнения.

```
BOOL CCapture::SetFmtAudio(void)
{
  BOOL bret = TRUE;
  HRESULT hr = S_OK;
  CComPtr<IAMStreamConfig> pSC;

  try
```

```
  {
    // найти IAMStreamConfig для выходного контакта
    if(S_OK != (hr = m_pBuilder->FindInterface(
              &PIN_CATEGORY_CAPTURE,
                      &MEDIATYPE_Audio,m_pCapA
                      ,IID_IAMStreamConfig
                      ,reinterpret_cast<void**>(&pSC))))
      throw hr;

// Установить новый формат. Структура предварительно //заполня-
ется новыми данными при выборе
//формата пользователем.
    if(S_OK != (hr = pSC->SetFormat(m_pMediaType)))
      throw hr;

  }
  catch(HRESULT)
  {
    bret = 0;
    //освободить структуру AM_MEDIA_TYPE
    if(m_pMediaType)
    {
      DeleteMediaTypeAndFormat(m_pMediaType);
      m_pMediaType = 0;
    }
  }

  return bret;
}
```

Интерфейс пользователя для управления звуковыми форматами

Одного элемента CComboBox вполне достаточно для выбора и установки звукового формата, поскольку список форматов невелик, и его можно предоставить пользователю полностью в отличие от видео формата. Для обслуживания элемента необходимо написать метод заполняющий список. При выборе формата пользователем возникает задача точной идентификации выбранного формата, поскольку звуковые форматы не идентифицируются с помощью GUID. В результате в данных элемента интерфейса нельзя сохранить указатель на признак конкретного формата. Приходится в классе CCapture сохранять весь вектор форматов и производить выбор по индексу вектора. Остальные решения как, например, синтаксический анализ строк элемента интерфейса и поиск по результату анализа оказываются более сложными в реализации. Для выбора формата в элементе CComboBox достаточно представить пользователю число каналов воспроизведения звука, частоту дискретизации и скорость потока. Метод CFView1::myFillListFmtAudio производит заполнение списка форматов.

Установку формата звука будем производить в обработчике элемента управления при смене строки. С этой целью создадим, стандартный обработчик события CBN_SELCHANGE, и в нем устанавливаем новый формат звука. С текстом интерфейсных методов можно ознакомиться в архиве проектов.

Управление параметрами звуковой карты

Рассматривая проблему построения звукового тракта, следует учесть то обстоятельство, что карта телеприемника имеет внешний выход звука, который должен подключаться к линейному входу звуковой карты.
Для полного управления звуком в режиме предварительного просмотра пользователю достаточно предоставить в распоряжение органы управления звуковой карты расположенные на входе линии линейного входа и на выходе линии, подключенной к громкоговорителям. То есть управлять линией воспроизведения звуковой карты. Таким образом, можно было бы управлять уровнем входного сигнала, поступающего с выхода карты телеприемника и общей громкостью. Управлять линией записи можно и с помощью интерфейса IAMAudioInputMixer. Но, не все так просто. Решение не слишком эффективное, поскольку часть проблемы решается с помощью интерфейса IAMAudioInputMixer, а для решения второй части проблемы необходимо вводить прямое управление драйвером звуковой карты. Может проще вообще не использовать интерфейс IAMAudioInputMixer.

Существует еще и интерфейс IBasicAudio. Если установлен фильтр DirectSound, тогда можно управлять общей громкостью звука на этом фильтре, используя интерфейс IBasicAudio, но не более.

Пока проверим насколько интерфейс IAMAudioInputMixer, полезен и удобен в реальной работе со звуковой картой. Вариант проекта находится в папке DirectSound. Этот проект не будет иметь развития, он предназначен только для проверки решения по построению ветви графа для звукового канала в соответствии с рекомендациями DirectShow.

Для проверки необходимо установить на вкладке класса CFView1 два регулятора, один из которых будем использовать для управления громкостью, а второй стереобалансом. Используем элементы ActiveX, предоставляемые средой разработки, это ускорит работу с интерфейсом программы. На основе элемента ActiveX Microsoft Slider Control 6.0 реализуем класс CSlider. В классе добавим два обработчика прокрутки и два метода инициализации.

```
.....................................................................................
  DECLARE_EVENTSINK_MAP()
  void ScrollSliderPan();
  void ScrollSliderLevel();
```

```
  //Инициализация исходного состояния регуляторов
public:
  void InitSliderLevel(void);

//Установка исходного состояния регулятора
//стереобаланса
  void InitSliderPan(void);
```

Для определения свойств входных линий фильтра звука понадобится класс данных, который напишем в файле Capture.h по уже знакомой схеме.

```
class CInputAudio
{
public:
  CInputAudio(): pinCaps(CAPSNO)//конструктор
  {
    ::ZeroMemory(wpinID,sizeof(wpinID));
  };

  ~CInputAudio() {};

  //Конструктор копировщик
  CInputAudio(const CInputAudio & pinCap)
{
*this = pinCap;
};

  //Перечисление свойств входных контактов
  //фильтра звука

  typedef enum
  {
    CAPSNO,
    ENABLE,           //Вкл. или выключение контакта
    MONO,               //Объединение в моно канал
    MIXLEVEL = 0x4,   //Уровень смесителя
    PAN = 0x8,          //Баланс каналов
    LOUDNESS = 0x10,  //Вкл. и выкл. громкости
    TREBLE = 0x20,    //Регулировка верхних частот
    BASS = 0x40       //Регулировка нижних частот
  }MIXCAPS;

  //Данные
  WCHAR wpinID[32]; //Идентификатор контакта
  MIXCAPS pinCaps;  //Свойства контакта

  //Оператор присваивания
  const CInputAudio & operator =
                          (const CInputAudio & in)
  {
    if(this != &in)
    {
```

```
      wcscpy(wpinID,in.wpinID);
      pinCaps = in.pinCaps;
    }
    return *this;
  };

  //Оператор сравнения по идентификатору контакта
  bool operator == (const CInputAudio& pinCap) const
  {
    if(wcscmp(wpinID,pinCap.wpinID)< 0)
      return 1;
    return 0;
  };
};
```

Определение свойств контактов звукового фильтра

Для определения, какие органы управления предоставляются звуковым фильтром необходимо собрать сведения о свойствах входных контактов. Сохранять данные будем в векторе, из которого затем можно извлечь свойства любого входного контакта. Для проверки данного устройства ограничимся только линейным входом звукового фильтра, поскольку телевизионная карта выдает сигнал на линейный вход звуковой карты. Метод CCapture::GetCapsInputLine заполняет свойствами линейного входа объект данных CInputAudio.

```
BOOL CCapture::GetCapsInputLine(CInputAudio & inLine)
{
  //Заполнить вектор свойств входов

  vector<CInputAudio> vID;
  GetCapsInputAudio(vID);

  if(vID.empty()) return 0;

  //Ищем линейный вход

  vector<CInputAudio>::iterator pos;
  pos = find_if(vID.begin(),vID.end(),FindInputLine());

  //Возвращаем данные

  if(pos != vID.end())
  {
    inLine = *pos;
    return 1;
  }

  return 0;
```

```
}
```

Заполнение вектора свойств контактов производит метод GetCapsInputAudio. При заполнении вектора тестируются методы интерфейса IAMAudioInputMixer. Это необходимо для определения какой тип органа управления установлен на линии, поскольку иным способом это невозможно определить и следовательно сложно реализовать интерфейс управления микшером. Метод put_Enable позволяет определить тип переключателя входов мультиплексор это или сумматор или же просто одиночная линия входа. Если метод позволяет только включать вход, тогда это мультиплексор, если же можно вход включать и выключать тогда это сумматор.

```
void CCapture::GetCapsInputAudio(vector<CInputAudio>& vID)
{
  IEnumPins* pEnum =0;
  IPin* pPin = 0;
  LPWSTR pinID = L"";

  if(!m_pCapA) return;

  //Интерфейс на фильтре

  IAMAudioInputMixer* pFMix = 0;
  HRESULT hr = m_pCapA->QueryInterface(&pFMix);

  //Создать перечислитель контактов

    hr = m_pCapA->EnumPins(&pEnum);
    if (FAILED(hr)) return ;

  CInputAudio inPin;
  while (pEnum->Next(1, &pPin, NULL) == S_OK)
  {
     PIN_DIRECTION pinDir;

    //проверяем направление контакта
     pPin->QueryDirection(&pinDir);
     if (pinDir == PINDIR_INPUT)
     {
      inPin.pinCaps = inPin.CAPSNO;//Очистить

      //Идентификатор контакта
      pPin->QueryId(&pinID);
      wcscpy(inPin.wpinID,pinID);

      //Интерфейс на контактах
      IAMAudioInputMixer *pMix=0;
      hr = pPin->QueryInterface(&pMix);

      //проверка наличия свойств контакта
      if(S_OK== hr)
```

```
{
  BOOL bres =0;

  //Сохранить состояние переключателя
  hr = pMix->get_Enable(&bres);

  if((S_OK == pMix->put_Enable(0))
      && (S_OK == pMix->put_Enable(1)))
  {
    //если условие выполнено то это сумматор
    inPin.pinCaps =
    static_cast<CInputAudio::MIXCAPS>(
    inPin.pinCaps | inPin.ENABLE);

    //восстановить состояние
    pMix->put_Enable(bres);
  }
  else
  {
    hr = pMix->put_Enable(1);
    hr = pMix->put_Enable(0);

    //если put_Enable(1) == S_OK, а put_Enable(0)== FAILED
    //то это мультиплексор на линии записи
  }

  pMix->get_Mono(&bres);
  if((S_OK == pMix->put_Mono(0))
          && (S_OK == pMix->put_Mono(1)))
  {
    inPin.pinCaps =
    static_cast<CInputAudio::MIXCAPS>(
    inPin.pinCaps | inPin.MONO);

    //Восстановить состояние
    pMix->put_Mono(bres);
  }

  double level=0.0;
  pMix->get_MixLevel(&level);
  hr = pMix->put_MixLevel(level);
  if(S_OK == hr)
  {
    inPin.pinCaps =
    static_cast<CInputAudio::MIXCAPS>(
    inPin.pinCaps | inPin.MIXLEVEL);
  }

  pMix->get_Pan(&level);
  hr = pMix->put_Pan(level);
  if(S_OK == hr)
  {
    inPin.pinCaps =
```

```
          static_cast<CInputAudio::MIXCAPS>(
          inPin.pinCaps | inPin.PAN);
        }

        hr = pMix->get_Loudness(&bres);
        if((S_OK == pMix->put_Loudness(0))
          && (S_OK == pMix->put_Loudness(1)))
        {
          inPin.pinCaps =
          static_cast<CInputAudio::MIXCAPS>(
          inPin.pinCaps | inPin.LOUDNESS);

          //Восстановить состояние
          pMix->put_Loudness(bres);
        }

        pMix->get_Treble(&level);
        hr = pMix->put_Treble(level);
        if(S_OK == hr)
        {
          inPin.pinCaps =
          static_cast<CInputAudio::MIXCAPS>(
          inPin.pinCaps | inPin.TREBLE);
        }

        pMix->get_Bass(&level);
        hr = pMix->put_Bass(level);
        if(S_OK == hr)
        {
          inPin.pinCaps =
          static_cast<CInputAudio::MIXCAPS>(
          inPin.pinCaps | inPin.BASS);
        }

        pMix->Release();
      }

        vID.push_back(inPin);
     }

     pPin ->Release();
  }//while

  if(pEnum)  pEnum->Release() ;
}
```

После заполнения вектора производится поиск контакта линейного входа по его текстовому идентификатору. Другого способа поиска не существует, поскольку контакты звукового фильтра не имеют стандартизованных идентификаторов. Поиск по текстовому имени может быть не эффективным из-за различных способов сокращения длинных имен в национальных языках.

Поэтому в объекте функции FindInputLine поиск имени контакта производится в нижнем регистре для двух языков английского и русского. Из имени линейного входа выбирается наиболее вероятная часть, без которой не может быть осмысленного имени линейного входа.

```
struct FindInputLine :public std::unary_function<CInputAudio
&,bool>
{
  bool operator () (const CInputAudio& in) const
  {
    //Преобразуем широкую строку в простую строку
    CW2A pname(in.wpinID);

    //Объект локального контекста для русского языка
    locale loc("rus");

    //Перевести в нижний регистр
    //последний индекс строки должен быть равен ее
    //размеру, т.к. tolower использует интервал
    //begin,end, и если указать sizeLine - 1,
    // то один символ останется не обработанным.

    use_facet<ctype<char> >(loc).tolower(&pname[0]
            ,&pname[strlen(pname)]);

    //Создаем строку STL
    string name(pname);

    //Поиск подстроки для имени линейного входа
    //для английского
    if(name.find("line",0) == string::npos)
    {
      //для русского
      if(name.find("лин",0) == string::npos)
        return 0;
    }

    return 1;
  }
};
```

После определения свойств микшера и выявления управляемых контактов можно реализовывать интерфейс пользователя. При проверке на конкретном экземпляре звуковой карты было выяснено, что на линии записи для переключения входов установлен мультиплексор, который и необходимо реализовать для управления входами. Вся интерфейсная реализация возлагается на программиста, вопрос в том, следует ли этим заниматься, если нельзя осуществить управление линией воспроизведения, для которой следует реализовать отдельное решение.

Установка параметров органов управления

Первоначальная установка значений для регулятора громкости и регулятора стереобаланса производится в методах CSlider::InitSliderLevel и CSlider::InitSliderPan. Текущие значения устанавливаются в обработчиках ScrollSliderPan и ScrollSliderLevel при перемещении движка регуляторов. Текст метода ScrollSliderLevel приводится ниже. Структура остальных методов мало отличается от метода ScrollSliderLevel и с ними можно ознакомиться в проекте.

```
void CSlider::ScrollSliderLevel()
{
  CCapture* pCap =
  static_cast<CTVshowApp*>
                   (AfxGetApp())->myGetPtrCap();

  //Возвращаем свойства линейного входа
  CInputAudio inL;
  if(!pCap->GetCapsInputLine(inL))
    return;

  //Возвращаем указатель на фильтр звука
  IBaseFilter* pFA = pCap->GetPtrAudioCapture();
  if(!pFA)
      return;

  //Возвращаем указатель на контакт линейного входа
  IPin* pPin=0;
  if(!pCap->GetPtrPinFromID(pFA,inL.wpinID,pPin))
    return;

  //Запрос интерфейса на контакте линейного входа
  IAMAudioInputMixer *pMix = 0;
  if(FAILED(pPin->QueryInterface(&pMix)))
    return;

  //Устанавливаем уровень сигнала
  pMix->put_MixLevel( get_Value() /
                   static_cast<DOUBLE>(get_Max()));

  if(pPin)
  {
      pPin->Release(); pPin=0;
  };

    if(pMix)
  {
      Mix->Release(); pMix=0;
  };
}
```

Результаты экспериментальной проверки

Запустив программу на выполнение, можно выявить некоторые неожиданные эффекты. Следует помнить, что в данном устройстве звуковой выход подключается внешним физическим соединением с линейным входом звуковой карты. Обнаруженные проблемы могут быть присущи только звуковой карте данного конкретного производителя.

Для выявления особенностей поведения интерфейса при работе с определенным типом звуковой карты следует запустить системную программу SndVol32, управляющую микшером звуковой карты. Сделать это можно с помощью панели управления, иконка "Звуки и аудиоустройства". В открывшемся диалоге нажать кнопку "дополнительно". Далее в SndVol32 следует на линии воспроизведения включить "Лин. вход", отключить линию "Звук" (…SRC_WAVEOUT) и включить линию общей громкости. При просмотре телеканала в режиме предварительного просмотра должен, быть слышен нормальный звук. Это работает только линия воспроизведения звука. В этом режиме регулировки звука возможны только с помощью SndVol32, поскольку ни интерфейс IAMAudioInputMixer, ни тем более фильтр DirectSound Renderer не управляют линией воспроизведения.

Для проверки работы с линией записи отключим "Лин. вход" линии воспроизведения и подключим такой же вход на линии записи, то есть установим режим записи через мультиплексор выходной линии …DST_WAVEIN. Запустим программу на выполнение и проверим состояние входов линии записи SndVol32. Произошло переключение входов, подключен вход "Вых. микшера звукозаписи", то есть, подключен выход смесителя линии воспроизведения. Можно предположить, что целенаправленно без нашего участия ищется линия-источник …SRC_WAVEOUT на первой (нулевой) выходной линии. Линия …SRC_WAVEOUT для данного типа звуковой карты есть как на выходной линии воспроизведения, так и на выходной линии записи, что приводит к неправильной инициализации, подключается выход смесителя линии воспроизведения. Эту возможность звуковых карт следует анализировать отдельно, тем более, что документация MSDN прямо предупреждает о такой возможности. В таком режиме фильтр DirectSound Renderer не работает, и прослушать действительный сигнал записи нельзя. Пробуем переключить вход в правильное состояние и отключить ненужные входы на линии воспроизведения, чтобы не получить эффект эха. После этого проверяем работу регулировку на входе "Лин. вход". Регулировка производится и при этом ее изменение отображается в системном микшере, однако обратного отображения не происходит, при регулировке на входе "Лин. вход" в системном микшере в приложении это изменение естественно не отображается.

Таким образом, реализовать полноценное управление микшером звуковой карты с помощью интерфейса IAMAudioInputMixer не удается, он не учитывает некоторых особенностей звуковых карт. Все равно приходится привлекать системную программу SndVol32. Полностью решить проблему можно,

если реализовать управление системным микшером из своей программы, что будет наилучшим выходом из создавшегося положения. Чтобы не заниматься корректировкой кода этого проекта, просто оставляем его в этом виде и продолжим работу с проектом в папке ProjectState 6, где приступим к реализации управления системным микшером.

Реализация управления системным микшером

Для реализации управления системным микшером необходимо произвести некоторую подготовку классов микшера. В папке ProjectState6 в проекте создадим отдельную папку "Mixer" и скопируем туда файлы классов микшера из проекта "AudioMixer". Дополнительно в папке классов микшера создаем файл заголовка "headermixer.h", в который будем включать необходимые файлы основного проекта. В файлах .cpp микшера заменим строки #include "AudioMixer.h" строками #include "headermixer.h". Теперь при переносе классов микшера в любой проект достаточно внести изменения только в один файл. Если не использовать классы микшера CFader и CSwitch вне классов микшера, то файл headermixer.h становиться ненужным. В файле stdafx.h включаем мультимедийную библиотеку

```
#include <Mmsystem.h>
#pragma comment (lib,"winmm.lib")
```

Включение библиотек в файле stdafx.h зачастую удобней, чем в установках проекта, поскольку при переносе кода в другой проект можно забыть о переносе установок проекта или ошибиться.

Управление микшером может располагаться на нескольких вкладках, поэтому следует указатель на микшер сделать общедоступным из любой вкладки. Проще всего объявить его статическим в классе приложения

```
static  auto_ptr<CLineMixer> m_pMixer;
```

В файле .cpp класса приложения запишем определение указателя

```
auto_ptr<CLineMixer> CTVshowApp::m_pMixer;
```

Инициализация указателя может быть проведена в классе первой вкладки, где будет использоваться любой орган управления микшером. Следует учесть, что конструктор этой вкладки должен вызываться раньше конструкторов других вкладок, где установлены другие органы управления микшером. Если такой уверенности нет, то лучше инициализацию провести в классе главного окна.

Сначала создадим органы управления звуком для режима предварительного просмотра. Для этого следует установить органы управления выходной линией воспроизведения …DST_SPEAKERS или …DST_HEADPHONES, которые эквивалентны по назначению. Первая линия типовой компонент, но может быть вместо него и второй компонент. Поскольку с выхода телевизионной карты звук подается на линейный вход звуковой карты, то для полного

управления каналом звука следует установить и органы управления на линейном входе (..SRC_LINE).

Регуляторы управления линейным входом и общей громкостью разместим на вкладе класса CFView1 в горизонтальном положении.

В редакторе ресурсов на вкладке располагаем два статических окна для регуляторов. Код для установки регуляторов напишем в обработчике CFView1::OnInitialUpdate(). Обработка ошибок при невозможности установить орган управления не рассматривается. При необходимости можно просто скрыть статическое окно, сделав его невидимым.

```
.....................................................................
//Инициализация указателя на микшер
  CTVshowApp::m_pMixer.reset(
            CLineMixer::MakeMixer(0,this->m_hWnd));

  //Микшер открыт?
  if(!CTVshowApp::m_pMixer->m_hMix)
    return ;

  DWORD dwDestination = -1;
  DWORD bret =0;

  //Установка органов управления для выходной линии
  //воспроизведения
  bret = CTVshowApp::m_pMixer->MakeLineCtrlDST(
              GetDlgItem(IDC_VOLUME),1
              ,DST_SPEAKERS, dwDestination);
  if(!bret)
    CTVshowApp::m_pMixer->MakeLineCtrlDST(
              GetDlgItem(IDC_VOLUME),1
              ,DST_HEADPHONES, dwDestination);

  //и входных линий, связанной с данной выходной
  if(bret == CTRL_FADER && dwDestination != -1)
  {
    bret = CTVshowApp::m_pMixer->MakeLineCtrlSRC(
                GetDlgItem(IDC_LINE),1
                ,SRC_LINE,dwDestination);
  }
.....................................................................
```

В классах органов управления CFader и CSwitch в обработчиках следует вызвать методы для установки состояния органов управления.

Например, далее частично показан код обработчика регулятора CFader для горизонтального регулятора.

```
void CFader::HScroll(UINT /*nSBCode*/, UINT /*nPos*/)
{
  ASSERT(m_pOwner);
```

```
  //Для каждого регулятора производится проверка,
  //к какому классу он  принадлежит
  //Проверка указателя на принадлежность классу
  CLineMixer* pmix =  static_cast<CLineMixer*>(m_pOwner);

  if(pmix->IsKindOf(RUNTIME_CLASS(CLineMixer)))
  {

    //Установить значение параметра микшера
    pmix->SetValueFader(this,myGetFaderValueProc());

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

  //Проверка указателя на принадлежность классу
  CMux* pmux = static_cast<CMux*>(m_pOwner);

  if(pmux && pmux->IsKindOf(RUNTIME_CLASS(CMux)))
  {
    CLineMixer* pmix =
    static_cast<CLineMixer*>(pmux->m_pOwner);

    //Установить значение параметра микшера
    pmix->SetValueFader(this,myGetFaderValueProc());

    //Отобразить значение параметра микшера
    mySetWndBuddyText(pmix->GetValueFader(this));
  }
  //Далее такие же проверки делаются для регуляторов //других
  классов
  ..................................................................................
}
```

На рисунке ниже показан вид вкладки после установки регуляторов.

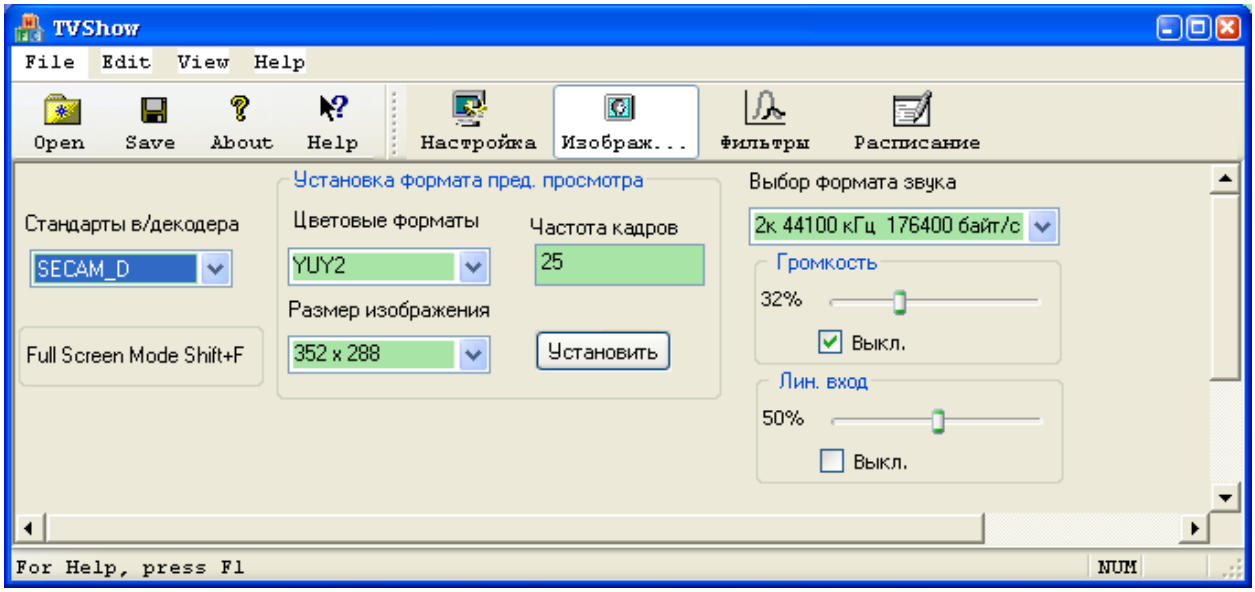

Рис.5. 3. Регуляторы системного микшера.

В классе окна CMsgWnd, обрабатывающего сообщения микшера, производится вызов виртуальной функции OnCtrlChange, которая перегружена в классе CLineMixer. Вызов этой функции заблокирован, когда окно получает сообщение при воздействии на микшер собственного приложения. Сообщение от микшера не несет информации об отправителе, поэтому производится анализ текущего обрабатываемого сообщения. Идентификацию своего приложения можно произвести разными способами. Можно делать проверку времени выполнения для собственных классов, как это делалось ранее, а можно определить имя окна своего приложения.

```
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);

  CWnd* pTopParent = pwnd->GetTopLevelParent();

  //Найти окно своего приложения
  if(pTopParent)
  {
    CString str;
    pFind->GetWindowText(str);

    if(str == "TVShow")
      return 0;
  }

  //if(pwnd && !(pwnd->IsKindOf(RUNTIME_CLASS(CSwitch))
  //      || pwnd->IsKindOf(RUNTIME_CLASS(CFader))))
  {
    m_pMix->OnSetControl(hmix,ctrlID);
  }

  return 0;
}
```

В функции OnSetControl производится только отображение состояния органа управления.

```
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 установлены органы
  //управления одной и той же линией с одинаковым ID,
  // то они будут отображать изменения,
  //при условии, что вызов метода не блокируется в
  //обработчике сообщения

  while(pos!= m_vCtrl.end() && (*pos).pCtrl)
  {
    CFader* pFd = static_cast<CFader*>((*pos).pCtrl);
    if(pFd && pFd->IsKindOf(RUNTIME_CLASS(CFader)))
    {
      //Вернуть 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)))
    {
      //Отображение состояния мультиплексора
      pMux->myDisplayStateMux();
    }

    //Поиск следующего органа управления со следующей      //по-
зиции
    pos = find_if(++pos,m_vCtrl.end()
                        ,bind2nd(FindCtrlID(),ctrlID));
  }
}
```

Управление системным микшером из приложения, а также использование фильтра DirectSound позволит реализовать полное управление звуковым ка-

налом и предотвратить возникновение побочных эффектов от неправильной начальной установки органов управления микшером. Предложенное решение нуждается в дополнении некоторыми возможностями, например, следует произвести при инициализации поиск всех нестандартных линий и установить их органы управления в положение, не препятствующее работе звукового канала. Для некоторых звуковых карт невозможно без экспериментальной проверки определить правильное положение нестандартного органа управления. В этом случае лучшим решением будет предоставление пользователю списка подобных органов управления. Вид графа с установленным фильтром DirectSound обеспечивающий запись файла с одновременным прослушиванием записываемого звукового сигнала показан на рисунке 5.4.

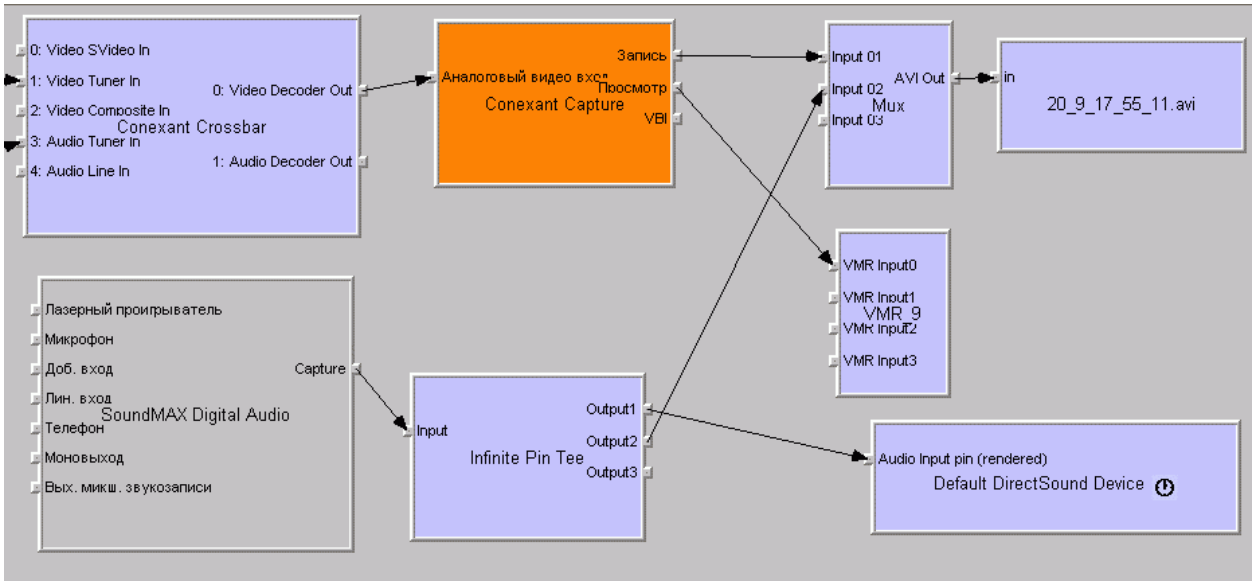

Рисунок 5.4. Запись файла с прослушиванием записываемого звука.