Глава 3

Построение графа до фильтра сбора

Глава 3
Построение графа до фильтра сбора....1
Принципы построения графа сбора....1
Определение конфигурации устройства сбора....3
Поиск фильтров с помощью метода FindInterface....4
Поиск фильтров с помощью системного перечислителя....20
Методы Render....24
Фильтр переключателя видео входов (Crossbar)....29
Класс фильтра переключателя 32
Интерфейс переключения видео входов 41
Фильтр настройки телеприемника TV Tuner....43
Принципы настройки на телевизионные каналы....46
Класс фильтра настройки приемника и интерфейс пользователя....49
Установка режима приемника....52
Переключение входов....56
Отображение доступных стандартов ТВ....58
Организация данных для отображения телевизионных каналов....60
Реализация настройки на каналы....65
Интерфейс управления фильтром настройки....77
Завершение основного этапа построения графа....83
Инициализация фильтра сбора....89
Интерфейс IAMAnalogVideoDecoder....90
Методы для работы с ТВ стандартами....91
Интерфейс пользователя....93

Принципы построения графа сбора

Построение графа сбора производится вокруг предварительно установленного в граф фильтра сбора, который является центральным фильтром графа. Сначала построение графа ведется от фильтра сбора вверх по потоку, при этом выявляются и устанавливаются все фильтры, которые входят в состав устройства. Затем производится определение возможностей фильтра сбора записывать и отображать данные. Для этого производится обнаружение выходных контактов фильтра сбора и определение их функциональных возможностей. Фильтры сбора могут иметь различный набор выходных контактов. В минимальном варианте фильтр сбора может иметь только один выходной контакт записи. После обнаружения выходных контактов фильтра сбора начинается построение графа вниз по потоку. Каждый выходной контакт фильтра сбора является началом ветви графа определенного назначения. На завершающей стадии построения графа сбора, может быть построено несколько ветвей графа в зависимости от решаемой задачи и наличия выходных контактов. Например, строится одна ветвь графа для предварительного просмотра, вторая ветвь для записи сжатых или не сжатых файлов и может быть третья ветвь графа для просмотра телетекста и дополнительной информации.

Для решения задачи видеозаписи и просмотра существует несколько интерфейсов, для устройств, поддерживающих модель драйвера WDM. В таблице 2 приводится список этих интерфейсов и их краткая характеристика. При построении графа будет рассмотрено применение практически всех указанных интерфейсов, за исключением интерфейса IAMBufferNegotiation, который используется в особых случаях.

Интерфейсы	Описание
IAMAnalogVideoDecoder	Управление аналого-цифровым преобразованием в фильтре сбора WDM.
IAMBufferNegotiation	Управление распределением буфера на контакте.
IAMCopyCaptureFileProgress	Интерфейс обратного вызова (callback) для приема сообщений при операции копирования файлов.
IAMCrossbar	Интерфейс, управляющий переключением источников видео сигнала.
IAMDroppedFrames	Определение производительности фильтра сбора в процессе работы (run-time).
IAMStreamControl	Управление временем пуска и останова на индивидуальном потоке данных.

IAMStreamConfig	Управление выходным форматом видео изображения и звука.
IAMTVTuner	Управление настройкой приемника.
IAMVfwCaptureDialogs	Отображение диалога для фильтра сбора VFW. Может предоставляться и другими фильтрами для управления их свойствами.
IAMVideoControl	Управление конечным изображением, переключает изображение зеркально, вертикально, горизонтально. Управляет внешним переключателем потока данных. Устройство должно поддерживать, указанные возможности.
IAMVideoProcAmp	Регулировки видеосигнала, такие как яркость, контрастность и другие.
ICaptureGraphBuilder2	Построение и настройка фильтрового графа.
IFileSinkFilter2	Установка имени и атрибутов записываемого файла.

Таблица 1 Интерфейсы сбора видео данных

Определение конфигурации устройства сбора

На начальном этапе построения графа сбора следует определить конфигурацию устройства, то есть определить какие фильтры входят в его состав и отыскать их интерфейсы. Как уже упоминалось ранее, типичное устройство аналогового телевидения содержит фильтр сбора, фильтр настройки ТВ приемника, фильтр переключателя видео входов. Многофункциональный телеприемник может дополнительно включать фильтр приемника звуковых программ .

Отправной точкой поиска является фильтр сбора, который должен быть к этому моменту уже установленным в граф. Поиск может производиться двумя способами:

- Первый способ состоит в том, чтобы, используя метод FindInterface построителя графа сбора (ICaptureGraphBuilder2), найти интерфейсы необходимые для реализации задач стоящих перед приложением. Например, для построения графа аналогового телевидения следует искать интерфейсы IAMTVTuner, IAMCrossbar, IAMTVAudio**.** Если найдены первые два интерфейса, то это свидетельствует о том, что существуют фильтры устройств, которые позволяют реализовать граф аналогового телевидения. Последний интерфейс позволит реализовать приемник звуковых программ . Поиск производится вверх по потоку. В процессе поиска метод стартует с указанного фильтра, находит, устанавливает фильтры в граф и соединяет их. Если требуемый интерфейс

найден, то легко находится и указатель на сам фильтр. Для этого достаточно запросить указатель на интерфейс IBaseFilter из найденного интерфейса. Любой фильтр DirectShow предоставляет этот интерфейс, который считается указателем на фильтр.

Метод работает автоматически и выполняет практически всю работу по построению графа. Недостатком такого подхода к построению графа можно считать достаточно большое время построения, заметное для пользователя. Положительной стороной использования данного метода является то, что он позволяет реализовать приложение без конкретизации устройства. Во многих случаях можно осуществить поддержку нескольких устройств различных производителей, которые имеют требуемые интерфейсы. Этот способ скорее предназначается для приложений, которые не предполагают поддержки всех свойств устройства конкретного типа (производителя). Например, для программы, которая производит запись файлов важно получить данные от любого источника, обеспечив типовую настройку устройства. В то же время остается сомнение, что с помощью данного способа, не имея дополнительных данных для устройства, можно правильно построить граф для некоторых устройств. Например, существуют карты телевизионных приемников, которые имеют два переключателя видеосигналов (IAMCrossbar). Переключатели соединяются в определенном порядке, и конфигурация их соединения может быть различной. Как отреагирует метод FindInterface на такую ситуацию, и каким образом произведет соединение, неизвестно, подробный алгоритм работы метода в документации не описан. В примерах и документации DirectShow предлагается только этот способ построения графа для аналогового телевидения.

- Второй способ состоит в использовании системного перечислителя. В этом случае перечисляются устройства в своих категориях. Выбор и установка фильтра устройства в граф может производится по его имени. После установки всех фильтров в граф производится их соединение. Достоинством этого способа является скорость работы. Недостатком можно считать то, что необходимо заранее знать имена фильтров, чтобы осуществить их поиск. Данный способ хорош для приложений ориентированных на поддержку одного или небольшого числа определенных устройств, когда заведомо известны имена установленных устройств, для которых пишется приложение. Этот способ построения графа следует предпочесть другим, когда программа пишется для поддержки конкретного устройства и предполагается использовать все возможности предоставляемые устройством.

Поиск фильтров с помощью метода FindInterface

Данный метод осуществляет поиск интерфейса на фильтре. Поиск можно производить для каждого фильтра в графе, начиная с указанного фильтра. Поиск можно ограничить по направлению вверх или вниз по потоку, категорией контакта или категорией медиа типа. Используя FindInterface можно значительно упростить поиск в графе, поскольку отпадает необходимость в перечислении фильтров и поиске на самом фильтре.

```
HRESULT FindInterface(
const GUID  *pCategory,
const GUID  *pType,
IBaseFilter     *pf,
REFIID    riid,
void      **ppint
);
```

Параметры.

pCategory

(in) Указатель на GUID критерия поиска. Может иметь четыре значения.

1. &LOOK_UPSTREAM_ONLY - поиск вверх по потоку.
2. &LOOK_DOWNSTREAM_ONLY - поиск вниз по потоку.
3. Член структуры AMPROPERTY_PIN_CATEGORY -указывает категорию (GUID) контакта для поиска. Например, категория контакта &PIN_CATEGORY_CAPTURE. Категории контактов указаны в разделе справки.
4. NULL. В этом случае поиск производится в таком порядке:
   - На фильтре.
   - На контактах.
   - Вниз по потоку.
   - Вверх по потоку.

pType

(in) Указатель на GUID главного медиа типа выходного контакта или NULL. Список главных типов приводится в разделе справки.

pf

(in) Указатель на фильтр, с которого начинается поиск.

riid

(in) Идентификатор интерфейса (IID), который требуется найти.

ppint

(out) Адрес переменной для приема указателя на найденный интерфейс. После использования интерфейса его следует освободить.

Коды возврата
Возвращается величина типа HRESULT. Возможные значения показаны ниже.

Возвращаемая величина	Описание
S_OK	Завершение без ошибок.
E_FAIL	Ошибка.
E_NOINTERFACE	Интерфейс не поддерживается.
E_POINTER	Указатель на интерфейс NULL.

Примечание.
Данный метод нельзя использовать для поиска интерфейса IVideoWindow, управляющего окном отображения. Этот интерфейс возвращается менеджером графа по запросу. В противном случае менеджер графа будет некорректно реагировать на события связанные с отображением.

В процессе поиска происходит опрос объектов в графе в следующем порядке:

- Фильтр.
- Контакты фильтра.

Поиск можно ограничить установкой параметров pCategory и pType.

Если параметр pCategory имеет значение LOOK_DOWNSTREAM_ONLY или LOOK_UPSTREAM_ONLY, то поиск происходит вверх или вниз по потоку. Параметр pType игнорируется.

Если параметр pCategory указывает категорию контакта, то тогда учитывается параметр pType и он также должен быть указан. В этом случае метод ищет интерфейс на контактах фильтров, что находятся выше по потоку, начиная с указанного фильтра. Поиск ниже по потоку ограничен выходными контактами фильтра. Категория контакта указывается при поиске интерфейса на контактах фильтра сбора. Следует учесть, что фильтр сбора может иметь раздельные контакты для сбора и предварительного просмотра. Некоторые фильтры имеют контакт видео порта (PIN_CATEGORY_VIDEOPORT) вместо контакта предварительного просмотра. В этом случае, если производить поиск только для категории контакта предварительного просмотра (PIN_CATEGORY_PREVIEW) и медиа типа MEDIATYPE_Video, метод опре-

делит видео порт как контакт предварительного просмотра и приложение не сможет различить видео порт.

Если устройство сбора использует WDM драйвер, то при поиске интерфейсов для фильтра настройки (TV Tuner) и (или) фильтра переключателя видео входов (Analog Video Crossbar) они автоматически устанавливаются в граф. Метод также производит соединение установленных фильтров.

Для дальнейшего построения графа вокруг уже установленного фильтра сбора создадим метод CCapture::InitCaptureVideo(HWND hwndView), где параметр hwndView является дескриптором окна отображения приложения, в котором будет отображаться видео. Он понадобится нам в дальнейшем для инициализации фильтра отображения. В методе напишем код для поиска интерфейсов IAMTVTuner для фильтра настройки телеприемника и IAMCrossbar для фильтра переключателя видео входов. На первом этапе закомментируем часть кода, которая создает объекты классов переключателя и настройки приемника. Сейчас важно определить, как метод будет производить поиск и установку фильтров в граф. Если метод найдет интерфейс фильтра настройки IAMTVTuner , то он должен будет найти и установить в граф фильтр переключателя видео входов, поскольку переключатель входов всегда находится ниже по потоку. Возможен вариант существования специального устройства сбора без фильтра настройки или без переключателя. Для нестандартного варианта устройства, метод все равно должен найти все фильтры выше по потоку и установить их. Так предполагается из описания метода, но как в действительности он работает в этом случае, можно определить только экспериментально, документация не посвящает нас в такие тонкости. Поэтому для страховки, следует поискать интерфейс переключателя IAMCrossbar, после неудачного поиска интерфейса фильтра настройки. Можно поступить и другим способом. После окончания поиска фильтра настройки перечислить фильтры в графе и найти фильтр переключателя. Это несколько ускорит поиск нужных фильтров.

```
HRESULT CCapture::InitCaptureVideo(HWND hwndView)
{
  //Нет фильтра сбора в графе
  if(!m_pCapV)
    return E_FAIL;

  HRESULT hr=S_OK;
  string sErr;

  //Указатель на первый переключатель видео входов
  CComPtr<IBaseFilter> pCross1;
  //Указатель на фильтр настройки приемника видео
  CComPtr<IBaseFilter> pTunerV;
  //Указатель на фильтр звуковых программ приемника
```

```
  CComPtr<IBaseFilter> pTunerA;

  try
  {
    //Построения графа сбора  через поиск интерфейса
    //IAMTVTuner

    CComPtr<IAMTVTuner> pTuner;
    hr = m_pBuilder->FindInterface(&LOOK_UPSTREAM_ONLY,
                &MEDIATYPE_Video,m_pCapV,IID_IAMTVTuner
                ,reinterpret_cast<void **>(&pTuner));

    //Если интерфейс найден, реализуем класс для фильтра //на-
    стройки приемника
    if(hr == S_OK && pTuner)
    {
      //Создаем объект класса CTuner
      //m_pAMTunerV = CTuner::MakeTuner(pAMTVTuner);

      //Перечисление фильтров в графе для поиска
      //фильтра переключателя

        CComPtr<IEnumFilters> pEnum;
        hr = m_pGraph2->EnumFilters(&pEnum);

        if (S_OK == hr)
        {
          //запрашиваем интерфейс на каждом фильтре
          CComPtr<IBaseFilter> pFilter;
          while (S_OK == pEnum->Next(1, &pFilter, NULL))
          {
            CComQIPtr<IAMCrossbar> pCross(pFilter);
            if(pCross)
            {
              //Фильтр переключателя был установлен
              //создать объект переключателя
              //  m_pCrossBar1 =
              //CCross::MakeCrossbar(pCross);
              break;
            }
            pFilter = NULL;
          }
          pEnum = NULL;
        }

        if(!m_pCrossBar1)
          return FALSE;
}
else
{
  //поиск фильтра переключателя, если не найден фильтр //на-
стройки
```

```
  CComPtr<IAMTVTuner> pCross;
  //hr = m_pBuilder->FindInterface(
            //&LOOK_UPSTREAM_ONLY
            //,&MEDIATYPE_Video,m_pCapV,IID_IAMCrossbar
            //,reinterpret_cast<void **>(&pCross));

    //если интерфейс найден, реализуем класс для фильтра   //на-
стройка
    if(hr == S_OK && pCross)
    {
      //m_pCrossBar1 = CCross::MakeCrossbar(pCross);
    }
  }
}
  catch(HRESULT)
  {
    if(hr != S_OK)
    {
      DestroyGraph();
    }
  }

  return hr;
}
```

Проверим работу метода в режиме поиска только фильтра настройки приемника. При удачном завершении поиска построитель графа должен автоматически установить фильтр настройки приемника и соединить его контакты с другими фильтрами. Для вызова метода в обработчике CTVshowView::OnNMClickListVideo добавим следующие строки.

```
if(S_OK != pCap->InitCaptureVideo(hwnd))
{
  string sErr = "Ошибка инициализации фильтра сбора";
  CTVshowApp::myMessageError(E_FAIL,sErr);
  return;
}
```

Полностью код обработчика можно посмотреть в папке проекта, версия ProjectState3.

Запускаем программу в режиме отладки и с помощью утилиты GraphEdit проверим результат. В окне утилиты видим картинку, показанную на рисунке 3.1.

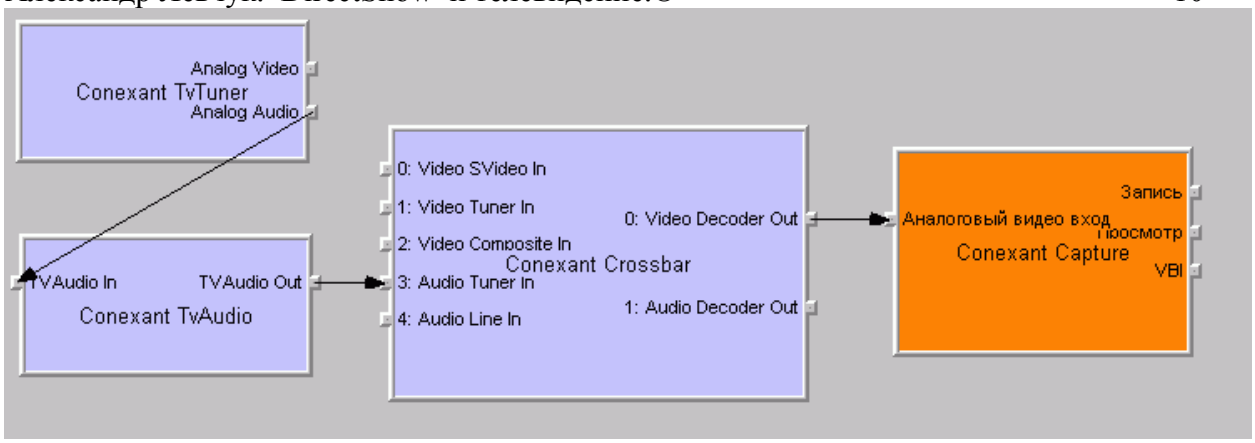

Рис.3. 1 Результат поиска интерфейса фильтра настройки приемника

Как видно на рисунке метод установил все фильтры и соединил их, в том числе и фильтр приемника звуковых программ , но не полностью. Нет подключения контакта Analog Video фильтра TvTuner к входному контакту переключателя Video Tuner In. Это ошибка, суть которой попытаемся выяснить с помощью утилиты GraphEdit . Как упоминалось ранее, для фильтров WDM потоки данных передаются в режиме ядра, а соединение фильтров отображает только управление фильтром со стороны DirectShow. Проверить работу графа, когда он подкючен к приложению невозможно, но его можно сохранить на диске, закрыть GraphEdit, затем опять открыть в GraphEdit и произвести необходимые проверки. Проделав необходимые манипуляции, достроим граф автоматически с контакта фильтра сбора Preview. Для этого следует сделать правый щелчок на контакте Preview и в открывшемся меню выбрать пункт "Render Pin". В граф будет установлен отображающий фильтр и появляется возможность проверить его функционирование в режиме предварительного просмотра. Для управления фильтрами в графе следует на фильтре сделать правый щелчок открывшемся меню выбрать пункт "Filter Properties". Откроется диалог со свойствами фильтра, позволяющий произвести его настройку. Диалог свойств фильтра настройки и фильтра сбора показаны на рисунке ниже. Первый позволяет произвести настройку на каналы, а второй выбрать стандарт телевидения. После настройки фильтров можно запустить граф и увидеть телевизионную картинку. Для ответа, почему не происходит соединения, и в тоже время граф работает, немного поэкспериментируем с графом.

Учтите, что описанные результаты могут отличаться от результатов ваших экспериментов. На результат эксперимента влияет много факторов, от модели устройства до версии его драйвера и того насколько разработчики устройства следовали условиям DirectShow.

Проверим, действительно ли контакт остался неподключенным или GraphEdit нам не показал его подключение. Маловероятно, что утилита GraphEdit виновата в этом, но проверка позволит исключить ее влияние. Сначала сохраним граф в файле и закроем GraphEdit. Напишем вспомогательный метод, определяющий неподключенные контакты на фильтре и возвра-

щающий их имена. Он поможет в процессе экспериментов вывести сообщение с именами неподключенных контактов на фильтре.

```
HRESULT CCapture::GetUnconnectedPin(IBaseFilter
          *pFilter  ,PIN_DIRECTION pinDir )
{
  CComPtr<IEnumPins> pEnum ;
  CComPtr<IPin> pPinl,pTmp ;
  PIN_INFO info ={0};

  vector<string> namePin;

  //Создать перечислитель контактов
  HRESULT hr = pFilter->EnumPins(&pEnum);
  if (FAILED(hr))
    return hr;

  while (pEnum->Next(1, &pPinl, NULL) == S_OK)
  {
    PIN_DIRECTION pinDirl;
    //Проверяем направление контакта
    pPinl->QueryDirection(&pinDirl);
    if (pinDirl == pinDir)
    {
      if(pPinl->ConnectedTo(&pTmp) ==
                    VFW_E_NOT_CONNECTED)
      {
        //Контакт не подключен,получить информацию о
        //контакте
        pPinl->QueryPinInfo(&info);

        //Преобразование в простую строку
        string stmp = COLE2T(info.achName);

        //Запись строк в вектор
        namePin.push_back(stmp);
      }

      pTmp = NULL;
    }
    pPinl = NULL;
  }

  //Вывод
  string str;
  size_t count = namePin.size();
  for(size_t i=0; i< count; ++i)
  {
    str += namePin[i] + "\n\r";
  }
```

```
  str.insert(0,"Неподключенные контакты\n\r");
  AfxMessageBox(str.c_str());

  return hr;
}
```

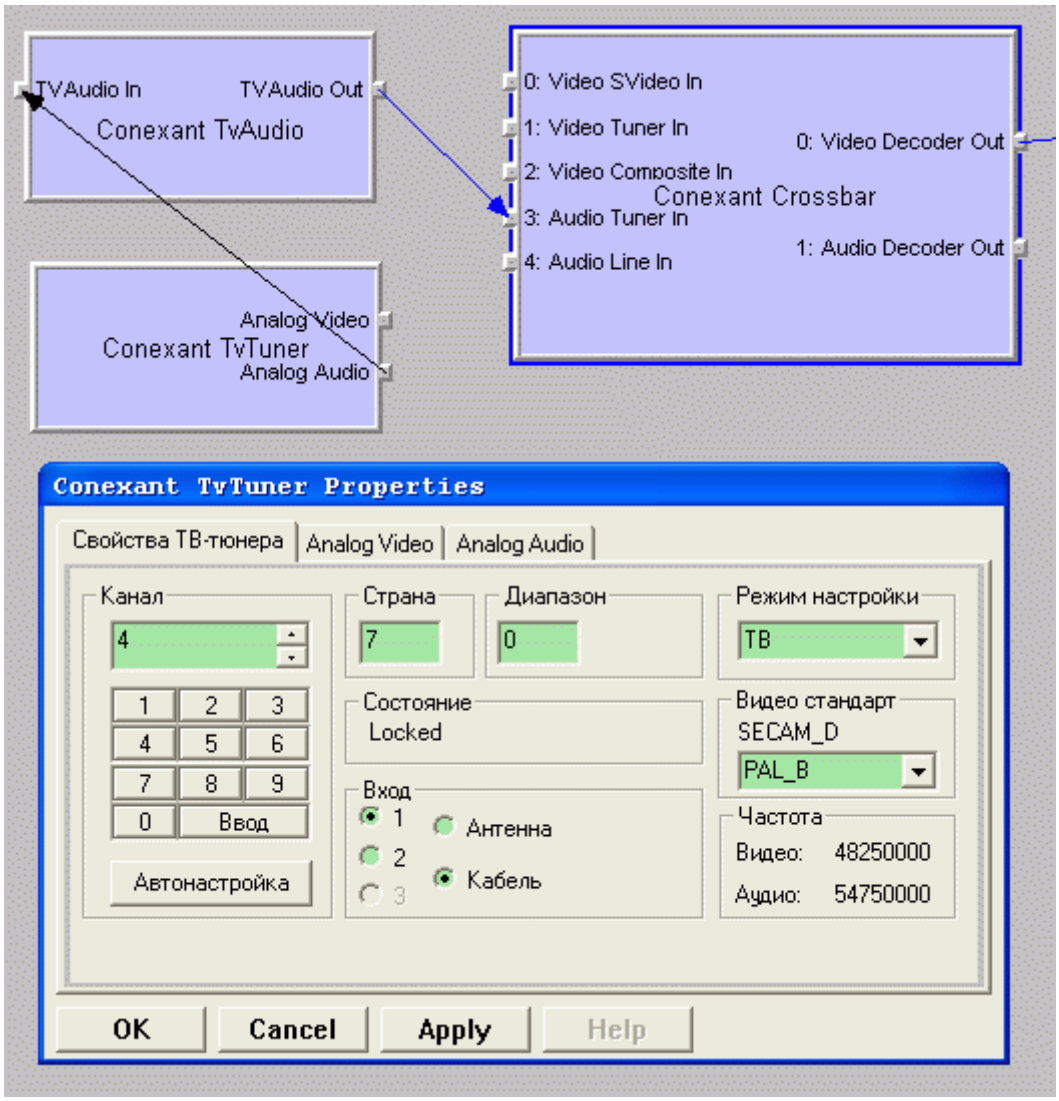

Рис.3. 2. Диалог фильтра настройки приемника.

Вызываем метод в блоке if в методе InitCaptureVideo.

```
if(hr == S_OK && pTuner)
{
  CComQIPtr<IBaseFilter> pF(pTuner);
  GetUnconnectedPin(pF,PINDIR_OUTPUT);

  //создаем объект класса CTuner
  //m_pAMTunerV = CTuner::MakeTuner(pAMTVTuner);
}
```

Запускаем приложение в режиме отладки. В выведенном сообщении указывается не подключенный контакт Video Tuner In. Таким образом, можно исключить влияние утилиты GraphEdit на результат.

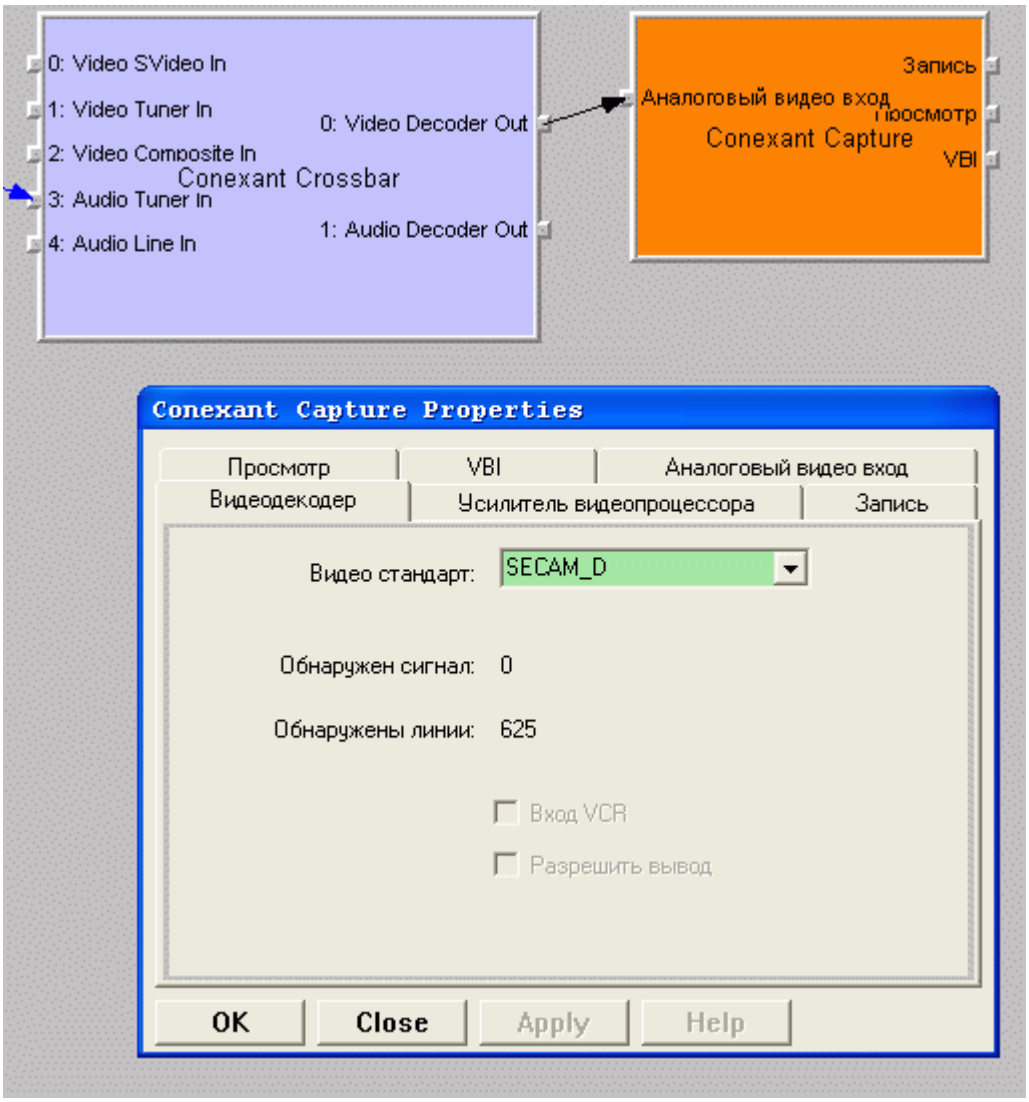

Рис.3. 3. Диалог настройки фильтра сбора.

Далее проведем некоторые эксперименты с графом, сохраненным в файле. Запустим GraphEdit и загрузим сохраненный файл. Попробуем подключить выходной контакт Analog Video фильтра TvTuner к входному контакту Video Tuner In фильтра Crossbar. Сделать это можно двумя способами:

- Первый способ это сделать щелчок правой кнопкой мыши на контакте и выбрать пункт меню Render Pin. Делаем. Попытка оказывается неудачной, в ответ получаем такое сообщение (Рис.4). В этом сообщении английский текст не соответствует коду ошибки. Он должен быть следующим "No combination of filters could be found to render the stream", тогда и перевод будет соответствовать содержанию сообщения.

- Второй способ состоит в ручном соединении контактов, перетаскиванием выходного контакта на входной контакт. Опять пробуем. В ответ получаем несколько иное сообщение об ошибке (Рис.5).

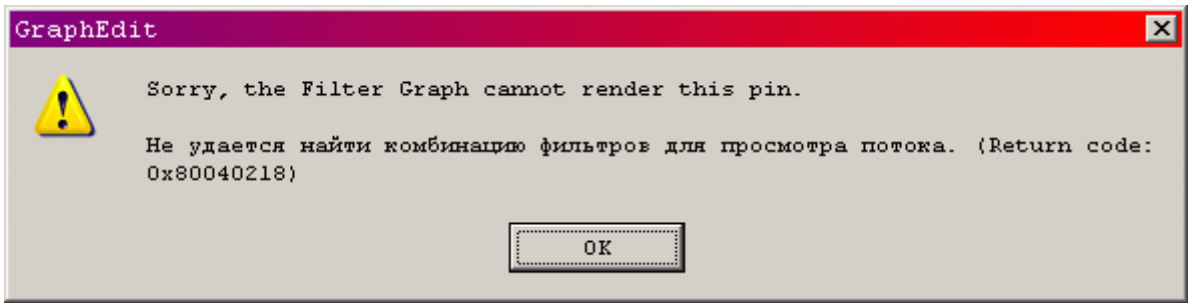

Рис.3. 4 Сообщение GraphEdit.

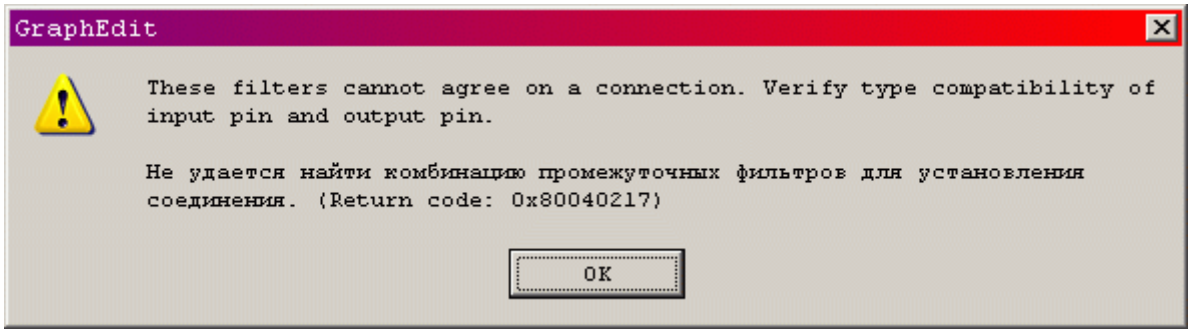

Рис.3. 5 Сообщение GraphEdit.

В этом сообщении перевод на русский язык сделан точно, в соответствии с кодом ошибки, но английский текст опять не соответствует описанию ошибки, текст должен быть такой "No combination of intermediate filters could be found to make the connection". Вывод неутешительный, не следует доверять текстам сообщений DirectShow, следует ориентироваться на код ошибки и расшифровывать его по таблице описания кодов ошибок. Можно поэкспериментировать еще. Отключим опцию "интеллектуального" соединения (меню Graph/Connect Intelligent) и повторим соединение вторым способом. Получаем третье сообщение.

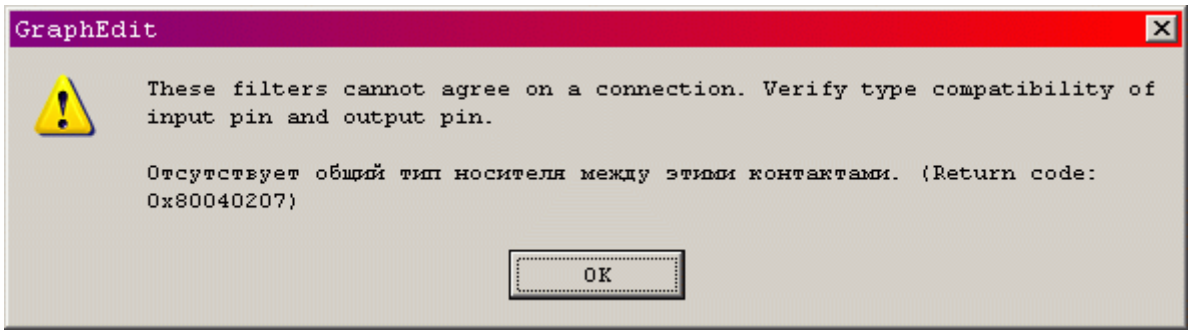

Рис.3. 6 Сообщение GraphEdit.

Текстовое сообщение на английском языке опять не имеет ничего общего с приведенным ниже кодом ошибки, который описывается такой строкой (цитата) "There is no common media type between these pins". Перевод звучит так, "Нет общего медиа типа между двумя этими контактами", что соответствует русскому переводу сообщения.

Следует учесть, что фильтры WDM, представляющие устройства, не используют при соединении контактов согласование медиа типов. Они используют для установки соединения согласование сред (mediums), то есть физических шин. Ничто не мешает проверить на контактах и медиа типы.
Опять делаем щелчок правой кнопкой мыши на выходном контакте фильтра TvTuner и проверяем его свойства. В свойствах привилегированный медиа тип отсутствует. А в фильтре Crossbar медиа тип указан AnalogVideo. Проверим и звуковой контакт фильтра TvTuner, предварительно разорвав соединение. Для этого контакта указан медиа тип AnalogAudio.

Возможно, что разработчики драйвера допустили ошибку, не указав медиа тип контакта, но все же это не развеивает сомнений, ведь при соединении фильтры WDM не используют медиа типов.

Контакты фильтров WDM поддерживают среду передачи данных, которая физически представляется шиной (bus) передачи сигналов. Среда определяет метод передачи данных, соответствующий определенной шине. Контакт может поддерживать одну или несколько сред, или не поддерживать ни одной.

Контакты могут соединяться, если они поддерживают одинаковую среду. Среду в DirectShow описывает структура REGPINMEDIUM, которая эквивалентна структуре KSPIN_MEDIUM, используемой для потоковых драйверов ядра. Член clsMedium структуры REGPINMEDIUM указывает идентификатор класса (CLSID)среды.

Контакты, имеющие идентификатор CLSID равный GUID_NULL или KSMEDIUMSETID_Standard не должны соединяться, поскольку эти величины означают, что контакты не поддерживают среду. Для возврата среды контакта вызывается метод IKsPin::KsQueryMediums, который возвращает указатель на структуру KSMULTIPLE_ITEM, за которой следуют структуры REGPINMEDIUM. Каждая структура идентифицирует определенную среду, поддерживаемую контактом.

Кроме того, контакты соединяются, при условии, что фильтр требует только одного подключения этого контакта. То есть, число подключений контакта должно быть равным единице. Для определения числа подключений контакта следует возвратить свойство
KSPROPERTY_PIN_NECESSARYINSTANCES. Для проверки этих двух условий напишем вспомогательный метод CCapture::GetMediumPin. Основой метода послужил пример приведенный в статье WDM Class Driver Filters документации. Его функционирование требует дополнительного включения файлов ks.h и ksproxy.h. Дополнительные условия функционирования метода можно прочитать в комментариях в файле StdAfx.h. Методу передается указатель на контакт и указатель на фильтр.

```
HRESULT CCapture::GetMediumPin(IPin* pPin
                              , IBaseFilter* pF)
{
  //Структура, описывающая число и размер свойств
  //контакта.
    KSMULTIPLE_ITEM *pmi = NULL;
    BOOL bfMatch = false;

  //запрашивается интерфейс IKsPin для данного контакта
  CComQIPtr<IKsPin> pKsPin(pPin);
  if(!pKsPin)
    return E_FAIL;

  //Возвращаем GUID среды, которая поддерживается
  //контактом
    HRESULT hr = pKsPin->KsQueryMediums(&pmi);

    if (FAILED(hr))
          return hr;  //Pin does not support mediums.

    if (pmi->Count)
    {
        //Получить укзатель на структуру описывающую
        //среду.
        REGPINMEDIUM *pTemp = (REGPINMEDIUM*)(pmi + 1);
        for (ULONG i = 0; i < pmi->Count; i++, pTemp++)
        {
            bfMatch = true;
        }
    }

    CoTaskMemFree(pmi);

  //Определение числа подключений, которые требует
  //фильтр.
  //Описание данного интерфейса в MSDN на узле Microsoft
  //http://msdn2.microsoft.com/en-us/library/ms810540.aspx

  IKsPinFactory *pPinFactory;
  hr = pPin->QueryInterface(IID_IKsPinFactory,
                   (void **)&pPinFactory);
  if (FAILED(hr))
      return hr;  //Pin does not support mediums.

  //Возвращается ID фабрики контактов
  ULONG ulFactoryId;
  hr = pPinFactory->KsPinFactory(&ulFactoryId);
    if (FAILED(hr))
        return hr;  //Pin does not support mediums.

  //возвращается экземпляр свойства из фильтра.
  IKsControl *pKsControl;
```

```
  hr = pF->QueryInterface(IID_IKsControl,
                          (void **)&pKsControl);
    if (FAILED(hr))
        return hr;  //Pin does not support mediums.


  KSP_PIN ksPin;
  ksPin.Property.Set = KSPROPSETID_Pin; //GUID
  ksPin.Property.Id = KSPROPERTY_PIN_NECESSARYINSTANCES;


  ksPin.Property.Flags = KSPROPERTY_TYPE_GET;
  ksPin.PinId = ulFactoryId;
  ksPin.Reserved = 0;

//  KSPROPERTY ksProp;
  ULONG ulInstances = 0, bytes = 0;
  hr = pKsControl->KsProperty((PKSPROPERTY)&ksPin
              , sizeof(ksPin),  &ulInstances
              , sizeof(ULONG), &bytes);

  if (hr == S_OK && bytes == sizeof(ULONG))
  {
    if (ulInstances == 1)
    {
      //Фильтр запрашивает одно соединение.
      //This pin is OK.
    }
  }

  return E_NOTIMPL;
}
```

Для поиска контактов на фильтре напишем еще один метод, который пригодится в дальнейшем. Он будет возвращать указатель на контакт на фильтре. В качестве параметра поиска служит направление контакта и его номер на фильтре. В методе GetPinFilter используется метод ICaptureGraphBuilder2::FindPin, который будет подробно рассмотрен далее.

```
CComPtr<IPin> CCapture::GetPinFilter(IBaseFilter* pFilter
                ,PIN_DIRECTION pinDir,long numPin/*=0*/)
{
  CComPtr<IPin> pP;

  //ищем контакт на фильтре
  if(S_OK != m_pBuilder->FindPin(pFilter, pinDir,NULL,
                              NULL,FALSE,numPin,&pP))
    return CComPtr<IPin>(NULL);

  return CComPtr<IPin>(pP);
}
```

Вызовы методов можно произвести в конце CCapture::InitCaptureVideo и произвести исследование возможности ручного соединения фильтров в соответствии с упомянутой методикой.

```
hr = pTuner->QueryInterface(IID_IBaseFilter
              ,reinterpret_cast<void **>(&pTunerV));

CComPtr<IPin> pP1,pP2;

pP1 = GetPinFilter(pTunerV,PINDIR_OUTPUT);
pP2 = GetPinFilter(pTunerV,PINDIR_OUTPUT,1);

hr = GetMediumPin(pP1,pTunerV);
hr = GetMediumPin(pP2,pTunerV);
```

К сожалению, после проверки, не удалось сделать однозначного вывода о причинах разного поведения контактов фильтра при соединении. Для выходных контактов фильтра TvTuner интерфейс IKsPinFactory оказался не реализованным. Можно сделать вывод, что фильтр настройки реализован с некоторыми отступлениями от рекомендаций DirectShow.

В подобных случаях всегда фирма изготовитель устройства рекомендует решать проблемы путем установки новой версии драйвера. Можно было не заниматься поиском решения в данной ситуации, если бы была уверенность в том, что новая версия драйвера не содержит старой проблемы. На сайте производителя устройства была новая версия драйвера. К сожалению, результат остался прежний и с новой версией драйвера.

На данном этапе нужно найти решение проблемы или смирится с такой ущербной реализацией графа. Ничего катастрофического в этом нет, граф функционирует, но в дальнейшем всегда будет оставаться сомнение, что причиной какой-то ошибочной ситуации является отсутствие данного соединения. Можно еще попытаться произвести построение графа ручным способом.

Разорвем все соединения графа (кнопка на tool bar) и попытаемся опять соединить контакт Analog Video фильтра TvTuner и контакт Video Tuner In фильтра Crossbar, поочередно первым и вторым способом. И первый, и второй способ приводят к положительному результату. Более того, если разорвать только соединение фильтра Crossbar и фильтра сбора, то проверяемое соединение также выполняется. Вероятней всего алгоритм FindInterface имеет более жесткий критерий соединения фильтров, чем алгоритм метода Render.

Это уже решение проблемы, причем в нескольких вариантах:

1. После вызова методов FindInterface для всех фильтров разорвать все соединения и вызвать метод IGraphBuilder::Render для каждого контакта фильтра TvTuner.
2. Разорвать соединение фильтра Crossbar и фильтра сбора и вызвать метод IGraphBuilder::Render для неподключенного контакта.

3. Используя системный перечислитель, установить фильтры в граф и вызвать метод IGraphBuilder::Render для каждого контакта фильтра TvTuner.

Первые два способа потребуют использования перечислителя контактов на фильтре для поиска неподключенного контакта.

Первый способ потребует дополнительного перечисления всех фильтров в графе для разрыва соединений.

Второй способ потребует одного разрыва соединения и поиска неподключенного контакта на фильтре TvTuner. Более детальный анализ первых двух способов решения проблемы, скорее всего, выявит дополнительные издержки реализации.

Третий способ самый простой в реализации и наиболее логичный. Кроме того, следует вспомнить, что системный перечислитель работает быстрее поиска в карте фильтров и в целом третий способ реализации графа может оказаться эффективней. Применение этого способа требует точного знания имен фильтров, то есть теряется возможность поддержки широкого круга устройств.

Следует сказать, что влияния описанной проблемы на функционирование устройства обнаружено не было. Это не говорит о том, что не может быть другой проблемы, которая будет влиять на функционирование устройства, существенно снижая его потребительские свойства. Далее будет описана проблема в фильтре настройки и способ ее устранения. Подобные проблемы являются дополнительным аргументом в пользу построения графа для поддержки конкретных устройств. Одновременно это даст больше уверенности в том, что будет обеспечена поддержка и для других устройств, не имеющих проблем с драйверами.

Поиск фильтров с помощью системного перечислителя

Для поиска остальных фильтров устройства и их установки в граф можно использовать системный перечислитель. Каждый требуемый фильтр ищется в своей категории WDM фильтров. Категории фильтров описаны в справочном разделе. В любой категории может быть несколько фильтров, например, на компьютере пользователя может быть установлена карта приемника спутникового телевидения и карта приемника аналогового телевидения. В этом случае потребуется дополнительная идентификация фильтров либо по имени фильтра, либо по коду CLSID. Таким образом, поиск становиться аппаратнозависимым. Может быть, это и не является серьезным недостатком, поскольку неизвестно как поведет себя метод FindInterface, когда два разных устройства будут предлагать одинаковые интерфейсы, например настройки приемника. Такие ситуации требуют тщательной проверки.

Для реализации поиска WDM фильтров с помощью системного перечислителя необходимо написать два метода. Первый метод AddKernelFilter будет искать фильтр в его категории по имени, и устанавливать в граф, а второй

строить граф. Если имя фильтра точно неизвестно, можно ввести поиск фильтра по фрагменту имени и дополнительно проверить на предоставление специфического для данного фильтра интерфейса.

В методе AddKernelFilter перечисляются фильтры в указанной категории. Найденный фильтр устанавливается в граф. Можно вместо найденных имен фильтрам присваивать свои имена, более короткие или сохранять найденные. Тогда можно будет производить поиск фильтра с помощью метода IFilterGraph::FindFilterByName, без использования перечислителя фильтров в графе.

Проект на этой стадии сохраним в папке ProjectState3, сделаем его копию в папку ProjectState4 и продолжим работу над новой версией проекта.

```
HRESULT CCapture::AddKernelFilter(const GUID & category
                        , IBaseFilter** pFilter/*out*/
                        , wstring nameFilter)
{
  HRESULT hr = E_FAIL;

  //указатель на системный перечислитель
  CComPtr<ICreateDevEnum> pDevEnum;
  //указатель на перечислитель моникеров
  CComPtr<IEnumMoniker> pEnumCat;
  //указатель на моникер
  CComPtr<IMoniker> pMoniker;

  //создается системный перечислитель
  hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL
              ,CLSCTX_INPROC,IID_ICreateDevEnum
              ,reinterpret_cast<void **>(&pDevEnum));

  if(hr != S_OK)
    return hr;

  ULONG cFetched;
  //создается класс перечислителя  для категории
  hr = pDevEnum->CreateClassEnumerator(category,
                                &pEnumCat, 0);
  if(hr != S_OK)
    return hr;

  //перечисление  фильтров  в цикле.
  while(
        pEnumCat->Next(1, &pMoniker, &cFetched) == S_OK)
  {
    //проверка имени
    CComPtr<IPropertyBag> pBag;
    hr = pMoniker->BindToStorage(0, 0, IID_IPropertyBag,
      reinterpret_cast<void **>(&pBag));

    if (FAILED(hr))
    {
```

```
      pMoniker = NULL;
      continue; //Следующий моникер.
    }

    VARIANT var;
    VariantInit(&var);

    //читать дружественное имя фильтра
    hr = pBag->Read(L"FriendlyName", &var, NULL);
    pBag = NULL;

    //отбор фильтра по имени
    if(nameFilter == var.bstrVal)
    {
      //установка фильтра
      if(S_OK !=
            (hr= m_pGraph2->AddSourceFilterForMoniker(
           pMoniker,NULL,var.bstrVal,pFilter)))
        return hr;

      //фильтр найден
      hr = S_OK;
      break;
    }
    else hr = E_NOTIMPL;

    VariantClear(&var);
    pMoniker = NULL;
  }//while

  return hr;
}
```

Второй метод RenderAllPins предназначается для построения ветви графа для каждого неподключенного выходного контакта фильтра. В методе перечисляются контакты на фильтре, определяются неподключенные выходные контакты и строится ветвь графа от каждого неподключенного контакта. Поскольку в этом методе используется интерфейс IPin, необходимо дать краткую характеристику его методам.

Интерфейс IPin предоставляется всеми входными и выходными контактами. Менеджер графа использует многие методы интерфейса для выполнения соединений и приложение должно использовать методы менеджера графа для соединения фильтров. Методы, разрешенные для вызова в приложении, выделены жирным шрифтом.

Method	Description
Connect	Соединяет два контакта. Не использовать в приложении.
ReceiveConnection	Принимает соединение с другим контактом. Метод вызывается фильтрами в процессе соединения. Не использовать в приложении.
Disconnect	Разрывает соединение на контакте. Не использовать в приложении и фильтрах. Вызывать посредством менеджера графа.
ConnectedTo	Возвращает указатель на контакт, с которым соединен текущий контакт.
ConnectionMediaType	Возвращает медиа тип для текущего подключенного контакта.
QueryPinInfo	Возвращает информацию о контакте. Имя контакта, направление контакта, указатель на фильтр, которому принадлежит контакт.
QueryId	Возвращает строковый идентификатор контакта. Может не совпадать с именем контакта.
QueryAccept	Определяет, будет ли контакт принимать выборку с указанным медиа типом.
EnumMediaTypes	Перечисляет на контакте предопределенные медиа типы.
QueryInternalConnections	Возвращает контакты, имеющие внутреннее соединение (внутри фильтра) с этим контактом.
EndOfStream	Метод сообщает контакту, что данных больше не будет. Не использовать в приложении.
BeginFlush	Начинается операция сброса. Не использовать в приложении.
EndFlush	Начинается операция сброса. Не использовать в приложении.
NewSegment	Метод сообщает контакту, что медиа выборки принятые после этого вызова группируются в сегмент данных. Не использовать в приложении.
QueryDirection	Запрашивается направление контакта (вход, выход).

Таблица 2 Методы интерфейса IPin.

В методе RenderAllPins в процессе перечисления контактов производится отбор контактов по направлению. Направление контакта определяется перечислением PIN_DIRECTION.

```
typedef enum  _ PinDirection{
    PINDIR_INPUT,
    PINDIR_OUTPUT
  } PIN_DIRECTION;
```

Подключение контактов и построение ветвей графа производится с помощью метода IGraphBuilder::Render. Этот метод использует алгоритм "интеллектуального" соединения. Описание метода IGraphBuilder::Render и его расширенной версии IFilterGraph2::RenderEx приводится ниже.

Методы Render

Оба метода IGraphBuilder::Render и IFilterGraph2::RenderEx выполняют сходную задачу. Они строят граф, начиная с указанного выходного контакта с использованием алгоритма "интеллектуального" соединения. Отличие заключается в том, что метод RenderEx может ограничиться подключением только тех отображающих фильтров, которые были ранее установлены в граф. Отображающий фильтр должен иметь свободный входной контакт. Промежуточные трансформирующие фильтры метод будет устанавливать в граф самостоятельно. При соединении фильтров метод не соединяет фильтры, чьи имена начинаются с тильды (~).

```
HRESULT Render(
  IPin *ppinOut
);
```

Параметры

ppinOut
[in]

Указатель на выходной контакт.

Возвращаемые величины

Величина	Описание
S_OK	Успешно.
VFW_S_AUDIO_NOT_RENDERED	Частично выполнено. Звук не подключен.
VFW_S_DUPLICATE_NAME	Успешно. Менеджер графа изменил имя фильтра во избежание дублирования.
VFW_S_PARTIAL_RENDER	Частично выполнено. Некоторые потоки не поддерживают формат.
VFW_S_VIDEO_NOT_RENDERED	Частично выполнено. Видео поток не воспроизводится.

E_ABORT	Операция прервана.
E_OUTOЗВУКОВЫХ ПРОГРАМ-MEMORY	Недостаточно памяти.
E_POINTER	Нулевой указатель.
VFW_E_CANNOT_CONNECT	Не найдена комбинация промежуточных фильтров для соединения.
VFW_E_CANNOT_RENDER	Не найдена комбинация промежуточных фильтров для отображения потока.
VFW_E_NO_ACCEPTABLE_TYPES	Разные медиа форматы у контактов.
VFW_E_NOT_IN_GRAPH	В графе нет фильтра с таким контактом.

IFilterGraph2::RenderEx

```
HRESULT RenderEx(
  IPin *pPinOut,
  DWORD dwFlags,
  DWORD *pvContext
);
```

Параметры

pPinOut
[in]

Указатель на выходной контакт.

dwFlags
[in]

Флаг, определяющий как использовать отображающий фильтр. Если он имеет значение AM_RENDEREX_RENDERTOEXISTINGRENDERERS, то метод использует уже установленный ранее в граф отображающий фильтр и не добавляет новых. Флаг не влияет на установку дополнительных трансформирующих фильтров. Для успешного выполнения метода в графе должен быть соответствующий отображающий фильтр с неподключенным входным контактом. При нулевом значении флага метод ведет себя как метод Render.

pvContext
[in, out] Зарезервирован, должен быть NULL

Возвращаемые величины.
Возвращает общие значения HRESULT, указанные в справочном разделе.

Рассмотрим практические результаты использования метода IGraphBuilder::Render для построения графа. В процессе соединения метод

Render строит ветви графа, начиная с фильтра настройки приемника, который уже установлен в граф. Остальные фильтры также установлены в граф. Ветку графа для видео тракта метод строит до фильтра сбора включительно без проблем. Проверка этого метода при построении ветки графа от звукового контакта фильтра настройки приемника показала, что он работает неустойчиво, часто в одних и тех же условиях соединения не происходило, или же наоборот соединение происходило правильно. Причина неустойчивой работы методов Render и RenderEx заключается в самих методах. После многократных экспериментов и проверок было обнаружено, что случаи неустойчивой работы метода Render предположительно происходят по следующей причине. Метод Render при обработке звукового тракта доходит до подключения выходного контакта Audio Decoder переключателя и пытается установить (найти) WDM фильтр сбора звука, которого нет в графе, и который он не может установить самостоятельно. В абсолютном большинстве случаев метод прекращает работу, оставляя ветвь графа недостроенной, и заканчивается успешно. Иногда метод переходит на другой участок своего алгоритма и не строит ветвь графа, от звукового контакта фильтра настройки приемника. В этом случае он заканчивается с ошибкой. Если установить в граф звуковой фильтр сбора заранее, то проблемы связанные с аварийным завершением метода исчезают. В этом случае метод Render работал без аварийного завершения, но с различным результатом, иногда он устанавливал воспроизводящий звуковой фильтр по умолчанию "Default DirectSound Device", иногда нет, чаще не устанавливал. В целом сложно определить причину таких сбоев, которые не имеют четко выраженной закономерности. Следует отметить, что код метода RenderAllPins и в целом класса CCapture при этом не изменялся. Напрашивается вывод, что алгоритм метода одну и ту же ситуацию оценивает различно. Установка фильтра "Default DirectSound Device" приводит к другой проблеме, которую рассмотрим позже. Фильтр "Default DirectSound Device" в графе для данного устройства не нужен и его следует удалить.

Можно использовать расширенный вариант этого метода IFilterGraph2::RenderEx. Проблема метода RenderEx та же самая, если в графе нет звукового фильтра сбора и дополнительно еще и воспроизводящего фильтра "Default DirectSound Device", он не может завершить построение ветви графа, и возвращается почти всегда с ошибкой, за редким исключением. После установки этих двух фильтров метод RenderEx начинает работать более устойчиво, но соединения звукового фильтра и фильтра "Default DirectSound Device" не производит.

Можно сделать вывод, что для уверенного соединения фильтров в нестандартных конфигурациях лучше использовать прямое (ручное) соединение фильтров, не полагаясь на алгоритм "интеллектуального" соединения. Для звуковой ветви графа методы Render желательно не использовать вообще. Построение звуковой ветви графа оказывается более сложной задачей, чем видео ветви. Это проблема DirectShow, способы решения которой рассматриваются в других разделах.

Существует еще один метод ICaptureGraphBuilder2::RenderStream, использующий алгоритм "интеллектуального" соединения. В описанной ситуации он вообще отказывается производить соединения ветви графа для приемника звуковых программ.

Ниже приведен текст метода RenderAllPins, который для описываемой ситуации позволяет правильно построить участок графа обрабатывающий звук. Отладочные сообщения помогут выявить характер происходящих сбоев.

```
HRESULT CCapture::RenderAllPins(IBaseFilter* pFilter)
{
  HRESULT hr =S_OK;

  if(!pFilter)
    return E_INVALIDARG;

  CComPtr<IEnumPins> pEnum;
  CComPtr<IPin> pPin;

  //указатель на подключенный контакт
  CComPtr<IPin> pConnect;

  //перечислитель контактов на фильтре
    hr = pFilter->EnumPins(&pEnum);
    if (hr != S_OK) return hr;

  //перечисление контактов на фильтре
  while (pEnum->Next(1, &pPin, NULL) == S_OK)
    {
        //проверка направления
        PIN_DIRECTION pinDir;
        pPin->QueryDirection(&pinDir);
        if (pinDir == PINDIR_OUTPUT)
        {
      //проверка на подключение
      hr = pPin->ConnectedTo(&pConnect);

      //подключаем
      if(!pConnect && hr == VFW_E_NOT_CONNECTED)
            {
        hr = m_pGraph2->Render(pPin);

//отладочное сообщение при не соединении
//контакта фильтра
#ifdef _DEBUG

  if(hr != S_OK)
  {
    string sErr = "RenderAllPins, ошибка метода Render";
    CTVshowApp::myMessageError(hr,sErr);
    //GetUnconnectedPin(pFilter,PINDIR_OUTPUT );
  }
```

```
#endif //_DEBUG
            }
      pConnect = NULL;
    }
    pPin = NULL;
    }//while

//удалить фильтр Direct Sound
  CComPtr<IBaseFilter> pF;
  if(S_OK == m_pGraph2->FindFilterByName(
                      L"Default DirectSound Device",&pF))
  {
    m_pGraph2->RemoveFilter(pF);

    //Контрольное сообщение для определения случаев
    //установки и удаления фильтра Default DirectSound Device
    #ifdef _DEBUG

      MessageBox(NULL,_T("Удален фильтр Default
                 DirectSound Device"), _T("DEBUG"),MB_OK);
    #endif //_DEBUG
  }
    return hr;
}
```

При успешном завершении метода все соединения будут выполнены, при условии предварительной установки в граф звукового фильтра сбора. Установка фильтра сбора звука будет рассмотрена далее при описании построения ветви графа для сбора звука.

Вызов RenderAllPins производится в методе CCapture::InitCaptureVideo, который будет рассмотрен позже. Сейчас следует подробно ознакомиться с фильтрами, участвующими в построении графа и реализацией классов, которые будут управлять их функционированием.

Фильтр переключателя видео входов (Crossbar)

Фильтр переключателя видео входов предназначен для управления коммутацией видео и звуковых сигналов карты телеприемника. Карта телеприемника обычно содержит один переключатель видео входов, но существуют устройства, содержащие два переключателя видео входов. Вариант построения фильтрового графа с двумя переключателями здесь не рассматривается.

Прежде чем приступить к непосредственной реализации класса переключателя необходимо ознакомится с интерфейсами, которые поддерживает фильтр переключателя, методами этих интерфейсов и принципом работы переключа-

теля. Фильтр переключателя предоставляет интерфейсы IAMCrossbar, ISpecifyPropertyPages, IPersistPropertyBag, IPersistStream. Основным интерфейсом, который определяет функционирование переключателя, является IAMCrossbar.

Переключатель видео входов представляет собой матрицу, имеющую n входов и m выходов. Таким образом, матрица является переключателем на n положений и m направлений. Входные и выходные контакты матрицы нумеруются раздельно. Начинается нумерация с нуля. Контакты матрицы, кроме номера, характеризуются еще физическим типом и индексом связи. Физический тип определяется перечислением PhysicalConnectorType.

Индекс связи устанавливает соответствие между каналом переключения видео сигнала и каналом переключения звукового сигнала. Индекс связи текущего видео контакта указывает номер звукового контакта, который должен подключаться к своему выходу одновременно с видео контактом. Другими словами, индекс связи определяет контакты, которые должны переключаться одновременно. Например, если подключается один из видео входов с номером M, имеющий индекс связи N, то звуковой вход номер N также должен быть подключен к свому выходу. При переключении контактов определяющим является видео канал, поскольку несколько разных видео входов могут быть иметь одинаковый индекс связи, указывающий на один и тот же звуковой вход. В то же время звуковой вход не может иметь больше одного индекса связи, который указывает только на один из видео входов. Это видно на примере линейного входа звука (Рис.5). На рисунке 3.5 показаны внутренние соединения переключателя при подключении входа "SVideo" и сопряженного с ним звукового входа "Линия".

Функционирование переключателя определяется интерфейсом IAMCrossbar. Рассмотрим методы этого интерфейса. Их немного, и все они предназначены для управления матрицей переключателя.

Метод	Описание
CanRoute	Определяет, соединены два контакта или нет.
get_CrossbarPinInfo	Возвращает информацию о контакте.
get_IsRoutedTo	Возвращает входной контакт, подключенный к указанному выходному контакту.
get_PinCounts	Возвращает число входных и выходных контактов.
Route	Подключает входной контакт к выходному.

Таблица 3 Методы интерфейса IAMCrossbar.

Все методы используют для адресации контактов только их индексы. Методы интерфейса IAMCrossbar не позволяют произвести идентификацию контактов по именам и возврат указателя на контакт. Так метод интерфейса get_CrossbarPinInfo, предоставляющий информацию о контакте возвращает направление контакта, его индекс, индекс связи и физический тип. Этих дан-

ных недостаточно для предоставления пользователю удобной информации. Необходимо предоставить имя контакта. Для получения полной информации недостаточно одного интерфейса IAMCrossbar, следует использовать еще интерфейсы IPin и IBaseFilter, которые предоставят дополнительные данные о контактах. Тогда можно будет связать индекс контакта с его именем и указателем на интерфейс IPin.

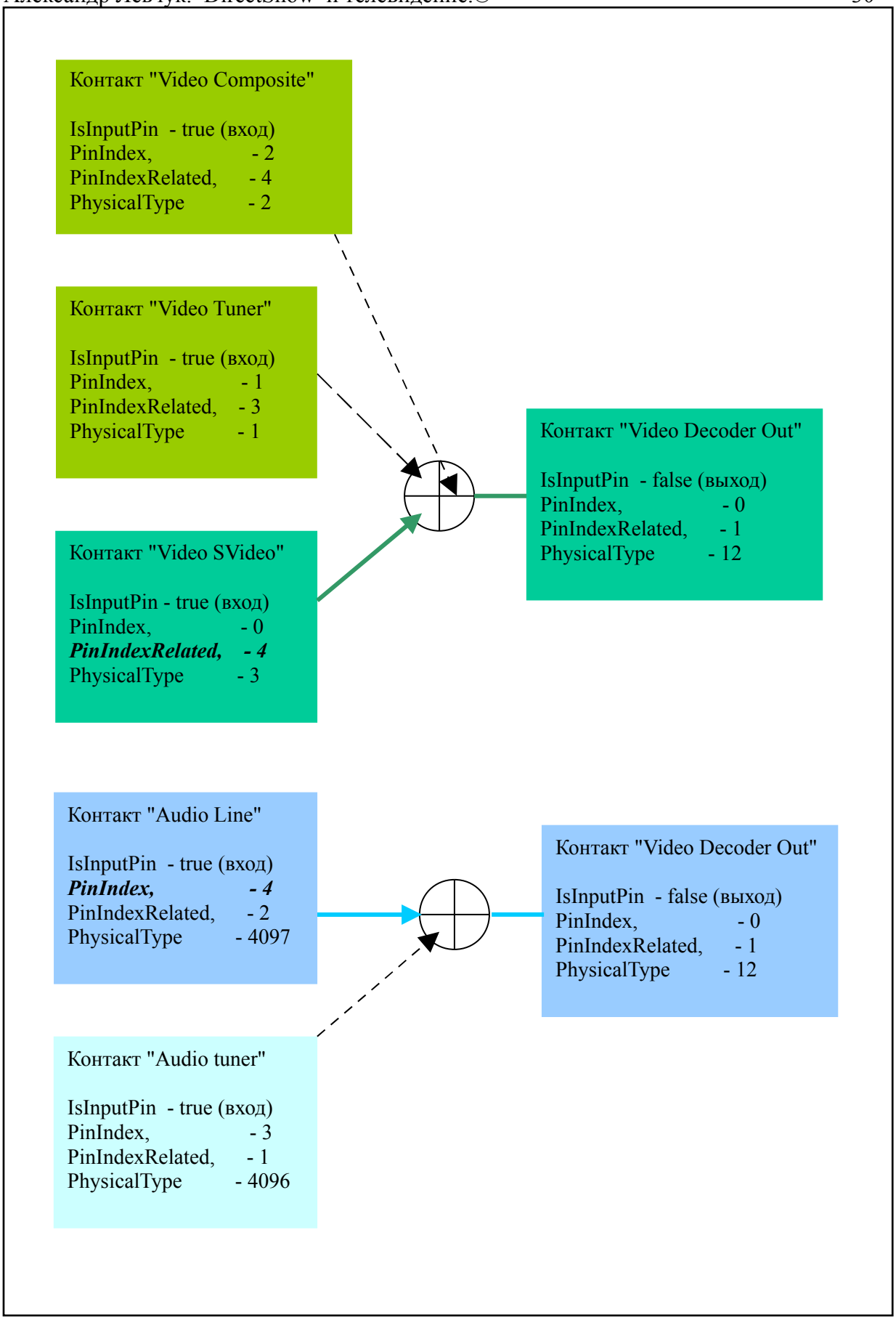

Рис.3. 7 Внутренние соединения переключателя

Устройство может иметь два переключателя. Производителю в рамках WDM драйвера предоставляется большая свобода по реализации устройства. Наличие второго переключателя порождает дополнительное множество вариантов его применения, учесть которые можно только для конкретного устройства. Производитель может реализовать второй переключатель видео входов, отличающийся от первого переключателя. Совместное использование двух различных переключателей также может быть обусловлено рядом ограничений производителя и дополнительных условий. Таким образом, написать приложение с использованием двух переключателей, без учета особенностей конкретного устройства вряд ли возможно. В рамках данного проекта применение второго переключателя не рассматривается.

Переключатель видео входов может сохранять свое состояние после закрытия приложения. Если это свойство поддерживается, то при запуске приложения можно отобразить подключенный вход. Для этого фильтр должен поддерживать интерфейс IPersistPropertyBag и автоматически сохранять и восстанавливать свое состояние. Проверка показала существование поддержки данного свойства для испытуемого образца устройства.

Если построение переключателя и логика его работы отличается от описываемого образца, то некоторые методы класса необходимо будет откорректировать в соответствии с конкретным типом устройства.

Класс фильтра переключателя

С учетом вышесказанного, реализуем класс переключателя для решения следующих основных задач:

- Выполнять соединение входов и выходов фильтра переключателя.
- Предоставить пользователю все видеовходы для управления переключателем в виде имен.
- Имена входов должны использоваться для идентификации контакта переключателя.
- Каждый вход переключателя должен дополнительно идентифицироваться указателем на переключатель видео входов для различения первого и второго переключателя.

Реализация этих возможностей должна позволить управлять как одним, так и двумя переключателями. Класс переключателя реализуем на куче. В классе фильтра сбора объявим указатель на объект переключателя CCross* m_pCrossBar1. Для хранения данных в классе переключателя объявляем структуру PINDATA, которая будет содержать все данные необходимые для идентификации контакта переключателя. Методы класса и их назначения перечислены в объявлении класса.

```
class CCross
{
public:
  //Псевдоконструктор
```

```
  static CCross* MakeCrossbar(
                CComPtr<IAMCrossbar> pAMCross);
  ~CCross(void);

private:
  struct PINDATA //Данные на контакт
  {
    //Направление контакта
    bool bInput;
    //Номер контакта
    long pinIndex;
    //Индекс связи
    long indexRelated;
    //Физический тип
    long physicalType;
    //ID контакта
    WCHAR wsIDPin[MAX_PIN_NAME];
  };

private:
  //Закрытый конструктор
  CCross(CComPtr<IAMCrossbar>);

  //Подсчет числа объектов для ограничения числа
  //переключателей
  static size_t m_numObj;

  //копирование запрещено, конструктор копировщик не //определя-
  ется
  CCross(const CCross& rCross);

  //Вектор для всех контактов
  vector<PINDATA> m_vPin;

  //Указатель на интерфейс IAMCrossbar фильтра
  CComPtr<IAMCrossbar> m_pAMCross;

  //Создает массив (вектор) контактов
  void CreateVectorPin(void);

  //Заполняет данные на контакт по указанному ID
  BOOL GetPinData(PINDATA& pinData);

public:
  //Возвращает вектор строк с идентификаторами
  //видеовходов
  void GetInputIDVideo(vector<string>& vInputV);

  //Подключение выбранного входа видео
  BOOL ToggleInputV(string sPinID);

  //Возвращает строковий идентификатор, подключенного
  //входного видео контакта и физический тип контакта
```

```
  string IsConnectToV(LONG& physicalConnectorType);
};
```

Класс имеет некоторые особенности. Для запрета создания более двух объектов конструктор в классе объявляется закрытым, также для запрета копирования объявляется конструктор копировщик без его реализации. Для создания объектов класса на куче объявляется статический псевдоконструктор MakeCrossbar и статическая переменная m_numObj для подсчета числа создаваемых объектов. Реализация указанных членов класса позволяет контролировать число создаваемых объектов.

```
//Определение статического члена для подсчета объектов
size_t CCross::m_numObj = 0;

//Псевдоконструктор
CCross* CCross::MakeCrossbar(CComPtr<IAMCrossbar> pAMCross)
{
  if(!pAMCross) return NULL;

  if(m_numObj> 1)
  {
    AfxMessageBox("Не может быть больше двух объектов
                                 переключателя");
    return NULL;
  }

  ++m_numObj; //Считать объекты

  return new CCross(pAMCross);
}

//Конструктор
CCross::CCross(CComPtr<IAMCrossbar> pCross) : m_pAMCross(NULL)
{
  if(pCross)
  {
    m_pAMCross = pCross;

  //Заполнить массив данных для контактов
    CreateVectorPin();
  }
}

//Деструктор
CCross::~CCross(void)
{
  --m_numObj;
}
```

Псевдоконструктор перед вызовом закрытого конструктора проверяет число созданных объектов и при превышении их количества более двух блокирует создание третьего объекта. В деструкторе счетчик уменьшается при удалении объекта. При вызове псевдоконструктора создается список контактов переключателя содержащий все необходимые данные на контакт. В состав этих данных, кроме информации, предоставляемой методом get_CrossbarPinInfo, включается имя контакта. Есть две возможности получить имя контакта. Первая это получить имя контакта из метода QueryPinInfo интерфейса IPin, а вторая использовать в качестве имени строковый идентификатор, возвращенный из метода QueryId того же интерфейса. Возвращаемые методами имена оказались одинаковыми, но в описании метода QueryId есть предупреждение, что строка идентификатора может не соответствовать строке имени, которое возвращает QueryPinInfo. Кроме того, в пользу идентификатора контакта говорит то, что он позволяет получить указатель на контакт, используя метод IBaseFilter::FindPin. Поэтому использование идентификатора позволяет решить две проблемы одновременно. Даже если идентификатор будет отличаться от имени контакта, то уж точно не настолько, чтобы исказился смысл названия контакта. Для полной идентификации контакта достаточно данных, объявленных в структуре PINDATA.

При вызове псевдоконструктора вызывается метод CreateVectorPin, который заполняет вектор данными для всех контактов переключателя. Этот вектор используется в дальнейшем для получения необходимой информации о контакте. Возможные физические типы входов видео канала и звукового канала определены в перечислении PhysicalConnectorType.

```
typedef enum
{
    PhysConn_Video_Tuner = 1,
    PhysConn_Video_Composite,
    PhysConn_Video_SVideo,
    PhysConn_Video_RGB,
    PhysConn_Video_YRYBY,
    PhysConn_Video_SerialDigital,
    PhysConn_Video_ParallelDigital,
    PhysConn_Video_SCSI,
    PhysConn_Video_AUX,
    PhysConn_Video_1394,
    PhysConn_Video_USB,
    PhysConn_Video_VideoDecoder,
    PhysConn_Video_VideoEncoder,
    PhysConn_Video_SCART,

    PhysConn_Audio_Tuner = 4096,
    PhysConn_Audio_Line,
    PhysConn_Audio_Mic,
    PhysConn_Audio_AESDigital,
    PhysConn_Audio_SPDIFDigital,
    PhysConn_Audio_SCSI,
```

```
    PhysConn_Audio_AUX,
    PhysConn_Audio_1394,
    PhysConn_Audio_USB,
    PhysConn_Audio_AudioDecoder,
} PhysicalConnectorType;



void CCross::CreateVectorPin(void)
{
  PINDATA pinData;

  PIN_INFO info ={0};
  LPWSTR wsTmp;

  CComPtr<IEnumPins> pEnum;
  CComPtr<IPin> pP;

  //По указателю на интерфейс получаем указатель
  //на фильтр и перечисляем все контакты на
  //переключателе
  CComQIPtr<IBaseFilter> pFilter(m_pAMCross);
  if(!pFilter) return;

  LONG j =0,i = 0; //Счетчики контактов
  if(S_OK != pFilter->EnumPins(&pEnum))
    return;

  //Перечисляем контакты на фильтре
  while(pEnum->Next(1, &pP, NULL) == S_OK)
  {
    pP->QueryPinInfo(&info);

    //заполняем вектор для входных pin
    if(info.dir == PINDIR_INPUT)
    {
      LONG indexRelated =0,physicalType =0;
      pinData.bInput = TRUE;
      if(FAILED(
          m_pAMCross->get_CrossbarPinInfo(pinData.bInput
                        ,i,&indexRelated,&physicalType)))
        return;

      //запись данных
      pinData.physicalType = physicalType;
      pinData.indexRelated = indexRelated;
      pinData.pinIndex = i;

      //ID контакта для идентификации контакта
      if(SUCCEEDED(pP->QueryId(&wsTmp)))
      {
        wcscpy(pinData.wsIDPin,wsTmp);
```

```
        //Освободить строку
        CoTaskMemFree(wsTmp);
      }
      else
        *pinData.wsIDPin = 0;//Очистить

      //Запись в вектор
      m_vPin.push_back(pinData);

      ++i;
    }
    else if(info.dir == PINDIR_OUTPUT)
    {
      //заполняем список для выходных pin
      pinData.bInput = FALSE;
      if(FAILED(
      m_pAMCross->get_CrossbarPinInfo(pinData.bInput,j
                          , &pinData.indexRelated
                          ,&pinData.physicalType)))
        return;

      pinData.pinIndex = j;

      //ID контакта для идентификации контакта
      if(SUCCEEDED(pP->QueryId(&wsTmp)))
      {
        wcscpy(pinData.wsIDPin,wsTmp);

        //Освободить строку
        CoTaskMemFree(wsTmp);
      }
      else
        *pinData.wsIDPin = 0;//Очистить

      //Запись в вектор
      m_vPin.push_back(pinData);

      ++j;
    }
    //освободить указазатель на контакт
    pP=0;
  }//while
}
```

Второй метод GetInputIDVideo(vector<string>& vInputV) класса CCross позволяет получить вектор строковых идентификаторов всех видеовходов. Пользователю необходимо предоставить набор входов переключателя для подключения различных источников видеосигнала. Это должны быть все видео входы, которые имеются на карте, звуковые входы показывать не нужно, поскольку они переключаются вместе с видеовходами, и забота об их переключении лежит на классе CCross.

```
void CCross::GetInputIDVideo(vector<string>& vInputV)
{
  PINDATA pinData;

  //Размер вектора
  size_t count = m_vPin.size();

  string s;

  //Поиск видеовходов в векторе контактов
  for(size_t i =0; i< count; ++i)
  {
    pinData = m_vPin[i];

    //Только видеовходы
    if(pinData.physicalType < 4096 && pinData.bInput)
    {
      s = CW2A(pinData.wsIDPin);
      vInputV.push_back(s);
    }
  }
}
```

Третий метод CCross::IsConnectToV возвращает строковый идентификатор текущего соединенного контакта переключателя. Не следует забывать, что речь идет о соединении внутренних контактов переключателя, а не контактов фильтра.

```
string CCross::IsConnectToV(LONG& physicalConnectorType)
{
  PINDATA pinData;

  size_t count = m_vPin.size();
  if(count == 0)
    return string();

  LONG input = 0;
  string s;

  //Поиск видеовыходов
  for(size_t i =0; i< count; ++i)
  {
    pinData = m_vPin[i];

    //Если это контакт видеодекодера
    if(pinData.physicalType ==
          PhysConn_Video_VideoDecoder &&                    !
        pinData.bInput)
    {
      //Найти входной контакт, соединенный с контактом //видеоде-
      кодера
```

```
      if(SUCCEEDED(m_pAMCross->get_IsRoutedTo(
                      pinData.pinIndex,&input)))
      {
        //Возвращаем физический тип входного контакта
        physicalConnectorType =
                          m_vPin[input].physicalType;

        //Возвращаем имя  подключенного видеовхода
        s = CW2A(m_vPin[input].wsIDPin);
        break;
      }
    }
  }

  return s;
}
```

Четвертый метод CCross::ToggleInputV осуществляет соединение входного контакта видео и сопряженного с ним контакта звука переключателя видео входов. В качестве параметра методу передается имя входного видеоконтакта, который должен быть соединен с выходным контактом переключателя.

```
BOOL CCross::ToggleInputV(string sPinID)
{
  BOOL bret = TRUE;
  PINDATA pinInput = {0};

  size_t count = m_vPin.size();
  if(count == 0)
    return FALSE;

  //Копируем в константу поиска идентификатор
  wcscpy(pinInput.wsIDPin,CA2W(sPinID.c_str()));

  //Возвращаем данные входного контакта
  if(!GetPinData(pinInput)) return FALSE;

//Найти выходной контакт видео, с которым может быть соединен //
входной. Соединить его и соединить
//сопрягаемые  контакты звука

  for(size_t i =0; i< count; ++i)
  {
    if(m_vPin[i].physicalType ==
                  PhysConn_Video_VideoDecoder &&
                  !m_vPin[i].bInput)
    {
      //Проверка на возможность соединения
      if(S_OK == m_pAMCross->CanRoute(m_vPin[i].pinIndex
                                ,pinInput.pinIndex))
      {
        //Соединить видеоконтакты
```

```
        if(S_OK == m_pAMCross->Route(m_vPin[i].pinIndex
                                    ,pinInput.pinIndex))
        {
          //Соединение звуковых контактов
          if(S_OK !=
                  m_pAMCross->Route(m_vPin[i].indexRelated
                                    ,pinInput.indexRelated))
          {
            bret = 0; //Соединение для звука не выполнено
          }
          break;
        }
      }
    }
  }

  return bret;
}
```

Кроме упомянутых методов класс содержит вспомогательный метод BOOL GetPinData(PINDATA & pinData), который возвращает данные на контакт по строке ID в структуре PINDATA. С текстом метода можно ознакомиться в исходных кодах проекта.

Интерфейс переключения видео входов

Для выбора видео входа пользователю предоставляется выпадающий список (CComboBox), отображающий имена входов. Каждая строка имени входа сопровождается указателем на переключатель видео входов, который необходим для идентификации переключателя, если он не один. Заполнение списка входов происходит после выбора устройства пользователем. С исходными текстами методов интерфейса пользователя можно ознакомиться в архиве проектов. Они не являются специфическими методами, использующими DirectShow, и служат для управления графом.
Вызов метода производится при выборе устройства в методе CTVshowView::OnNMClickListVideo. При выборе входа пользователем в обработчике события вызывается метод ToggleInputV, производящий переключение входов.

Кроме рассмотренного интерфейса IAMCrossbar, фильтр может поддерживать интерфейсы ISpecifyPropertyPages, IPersistPropertyBag, IPersistStream. Интерфейсы IPersistPropertyBag и IPersistStream предназначены для сохранения и загрузки индивидуальных свойств объекта. Ранее уже упоминался интерфейс IPersistPropertyBag, который использовался объектом моникера для хранения и загрузки своих свойств. Рассматриваемый фильтр переключателя

поддерживает эти интерфейсы, и они могут быть использованы для сохранения дополнительных свойств объекта. В данном случае нет нужды сохранять дополнительную информацию, достаточно того, что фильтр сохраняет состояние переключателя. Интерфейс ISpecifyPropertyPages может быть более полезен для приложения, не использующего MFC, поскольку предоставляет страницы свойств со всей необходимой информацией для управления переключателем. Этот интерфейс поддерживается многими фильтрами, в том числе некоторыми кодеками и поэтому в классе CCapture создадим метод CCapture::DisplayPropertyPages, который будет отображать страницы свойств для любого фильтра поддерживающий этот интерфейс. При вызове этого метода отображается страница свойств фильтра. Для примера на рисунке 3.6 показана страница свойств переключателя. С помощью этого диалога можно производить переключение входов, но для пользователя он не удобен и мало понятен. Обычно страницы свойств используются в приложениях, которые не привязаны к конкретному устройству, а используют только свойства общие для широкого круга устройств.

```
BOOL CCapture::DisplayPropertyPages(
                                    IBaseFilter* pFilter)
{
  ASSERT(pFilter);
  if(!pFilter)
    return FALSE;

  HRESULT hr;
    CAUUID cauuid ={0};

  //Получить интерфейс из фильтра
  CComQIPtr<ISpecifyPropertyPages> pSpec(pFilter);
  if(!pSpec)
    return FALSE;

  //Заполнить структуру CAUUID
  if(FAILED(hr = pSpec->GetPages(&cauuid)))
    return FALSE;

  //Запрашиваем имя фильтра
  FILTER_INFO filterInfo ={0};
  if(S_OK != pFilter->QueryFilterInfo(&filterInfo))
    return FALSE;

  //Отобразить страницы свойств
  if (hr == S_OK && cauuid.cElems > 0)
  {
    OleCreatePropertyFrame(NULL,0,0
              ,filterInfo.achName,1
              ,reinterpret_cast<IUnknown**>(&pFilter)
              ,cauuid.cElems,cauuid.pElems,0,0,NULL);

    //Освободить
```

```
    if(filterInfo.pGraph)
      filterInfo.pGraph->Release();
  }

  //Освободить структуру CAUUID
  CoTaskMemFree(cauuid.pElems);

  return TRUE;
}
```

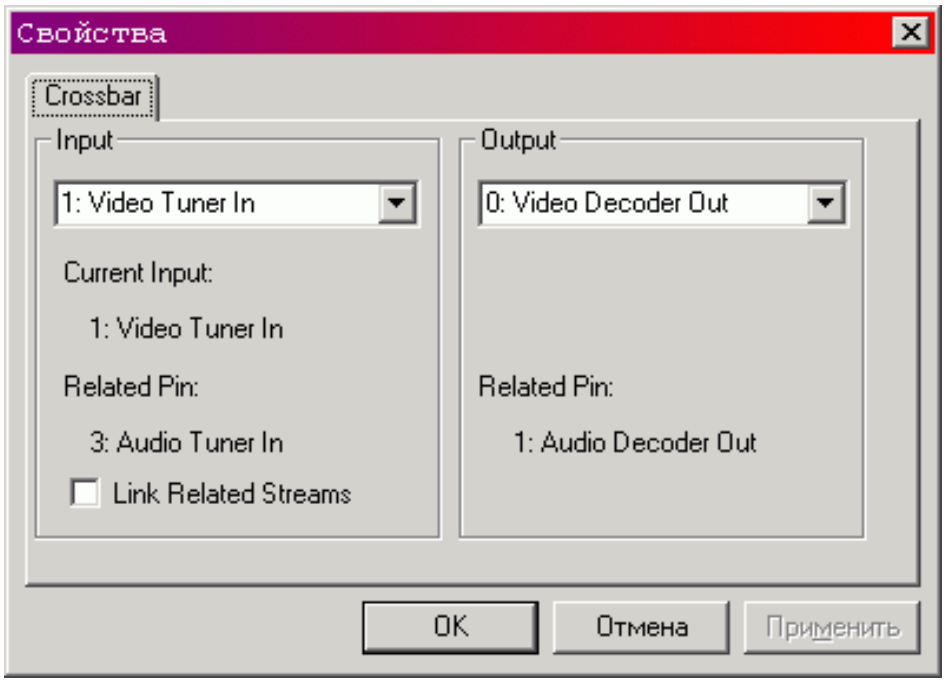

Рис.3. 8 Страница свойств переключателя видео входов

Для проверки работы метода и для вызова страниц свойств фильтров, удобно создать кнопку "Тест" и в обработчике этой кнопки производить вызов метода. Например, таким образом, вызывается страница свойств фильтра сбора.

```
void CTVshowView::OnBnClickedTest()
{
  CCapture* pCap = myGetPtrCap();
  pCap->Test();
}

void CCapture::Test(void)
{
  DisplayPropertyPages(m_pCapV);
}
```

Как видно из рисунка 3.8, для того чтобы пользователь правильно произвел переключение, он должен понимать, что такое индекс связи контактов (Related Pin). Это маловероятно, поэтому предоставление пользователю подобного интерфейса нецелесообразно.

Фильтр настройки телеприемника TV Tuner

Фильтр настройки предназначен для настройки телевизионного приемника на частотные каналы широковещательного и кабельного телевидения, а также для управления переключением антенных входов телеприемника. Основные интерфейсы, используемые фильтром для настройки IAMTuner и IAMTVTuner. Если приемник многофункциональный, то эти интерфейсы также поддерживают настройку других устройств, входящих в состав карты. Например, в составе карты может быть приемник FM, настройка которого также поддерживается, указанными интерфейсами. Интерфейс IAMTVTuner является производным от интерфейса IAMTuner и поддерживает настройку аналогового телевизионного приемника и приемника звуковых программ. Список методов и их краткое описание для этих интерфейсов приведены в таблицах 4 и 5.

Интерфейс IAMTuner.

Метод	Описание
ChannelMinMax	Возвращает минимальное и максимальное число доступных каналов настройки или возвращает границы частотного диапазона в режиме настройки приемника звуковых программ (FM).
GetAvailableModes	Возвращает поддерживаемые режимы многофункционального приемника.
get_Channel	Возвращает ТВ канал, на который настроен приемник.
get_CountryCode	Возвращает установленный код страны (региона), который определяет частоты настройки и стандарт ТВ.
get_Mode	Возвращает текущий режим многофункционального приемника.
get_TuningSpace	Возвращает сохраняемый индекс таблицы частот для региона.
Logon	Регистрирует пользователя в системе.
Logout	Удаляет пользователя из системы.
put_Channel	Настраивает ТВ приемник на канал.
put_CountryCode	Устанавливает код страны (региона) для использования соответствующих частот и стандарта ТВ.
put_Mode	Устанавливает режим многофункционального приемника.

put_TuningSpace	Устанавливает индекс таблицы частот для региона.
RegisterNotificationCallBack	Позволяет объекту, который реализует интерфейс IAMTunerNotification принимать сообщения об изменении состояния приемника. (Метод не реализован).
SignalPresent	Возвращает уровень сигнала на канале.
UnRegisterNotificationCallBack	Удаляет регистрацию объекта для приема сообщений. (Метод не реализован.)

Таблица 4 Интерфейс IAMTuner.

Интерфейс IAMTVTuner.

Методы	Описание
get_AvailableTVFormats	Возвращает все доступные аналоговые стандарты ТВ.
get_TVFormat	Возвращает текущий используемый стандарт ТВ.
AutoTune	Производит настройку на канал с точной подстройкой частоты канала.
StoreAutoTune	Сохраняет данные настройки на все каналы в текущем настроечном пространстве (в регистре).
get_NumInputConnections	Возвращает число входов (разъемов) фильтра настройки приемника.
put_InputType	Устанавливает тип входа (кабель или антенна), который определяет использование таблицы частот TSn-0 или TSn-1.
get_InputType	Возвращает установленный тип входа (кабель или антенна), который определяет использование таблицы частот TSn-0 или TSn-1.
put_ConnectInput	Подключает вход фильтра настройки приемника.
get_ConnectInput	Возвращает подключенный вход фильтра настройки приемника.
get_VideoFrequency	Возвращает текущую видеочастоту.
get_AudioFrequency	Возвращает текущую звуковую частоту.

Таблица 5 Интерфейс IAMTVTuner.

Все перечисленные методы просты в использовании, однако необходимо отметить, что поскольку реализация методов данных интерфейсов является прерогативой производителя драйверов, то некоторые нюансы поведения ме-

тодов остаются вне поля зрения документации DirectShow и требуется тщательная проверка их функционирования. Это замечание в частности касается возвращаемых кодов. В документации оговорено, что при успешном выполнении методы возвращают код S_OK, все остальные коды возвратов остаются для программиста неизвестными. В тоже время, например реализация метода put_Channel допускает, что при настройке на канал метод может возвращать код S_FALSE. Это происходит в случае, когда приемник при слабом сигнале имеет строчную синхронизацию и отображает устойчивую картинку, но декодер не может уверенно выделить сигнал строчной частоты и выдает сигнал об ее отсутствии. Для проверяемого устройства при слабом сигнале или его отсутствии метод всегда возвращал S_OK, но другие реализации могут возвращать S_FALSE. Таким образом, следует обратить особое внимание на контроль кодов возврата данных интерфейсов.

Принципы настройки на телевизионные каналы

Фильтр (устройство) TV Tuner содержит внутренний список частотных таблиц. Каждая таблица содержит список частот для широковещательных передач и кабельного телевидения для нескольких стран (региона). Выбор внутренних частотных таблиц производится по коду страны. Если приемник многофункциональный, выбор таблиц частот также зависит от режима работы многофункционального приемника, например TV или звуковых программ и определяется при вызове метода IAMTuner::put_Mode. Код страны может быть установлен при вызове метода IAMTuner::put_CountryCode или он устанавливается по умолчанию, используя данные операционной системы. Далее из таблиц для выбранного региона (страны) производится выбор частотной таблицы для широковещательных передач или кабельных сетей. Выбор производится установкой типа входа "антенна/кабель", с помощью метода IAMTVTuner::put_InputType. Если регион не имеет стандартизованной таблицы для кабельного диапазона, то производится генерирование универсальной таблицы кабельных частот. Кабельные операторы свободны в выборе частот и часто смешивают частоты разных стандартов. Для настройки на кабельные каналы в тех странах, где отсутствует их стандартизация, существует универсальная частотная таблица (Unicable) перекрывающая весь диапазон частот. Часто устройства имеют только универсальную кабельную таблицу частот.

Следует отметить, что переключение типа входа антенна/кабель не имеет никакого отношения к физическим входам устройства, которые переключают входные кабельные разъемы. Название этого переключателя не совсем удачное и может ввести в заблуждение. Физические входы устройства переключаются при установке номера входа, они отличаются типом входного разъема, но часто объединены по высокочастотному сигналу и их переключение не оказывает влияния на режим работы.

Для настройки на канал приложение вызывает метод IAMTuner::put_Channel, указывая индекс канала. Метод осуществляет выборку частоты из таблицы и настройку приемника. Существует возможность на-

стройки на канал с использованием метода IAMTVTuner::AutoTune, который производит точную подстройку частоты в пределах нескольких килогерц. Метод не работает с настроечными пространствами, сохраняемыми в регистре. Настроечные пространства служат для корректировки частот, и будут рассматриваться далее. Результат точной настройки на все каналы сохраняется в регистре в текущем настроечном пространстве при вызове метода IAMTVTuner::StoreAutoTune. Во всяком случае, так гласит документация, на самом деле в данной реализации метод StoreAutoTune работает частично, по сравнению с заявленными функциями. В тоже время метод AutoTune работает с настроечными пространствами. Поэтому для каждого устройства следует тщательно проверять поведение методов.

Приложение может проверить качество настройки и обнаружения сигнала, вызвав метод IAMTuner::SignalPresent.

В некоторых странах собственная индексация каналов (номер программы), предоставляемая пользователю, не совпадает с индексацией частотной таблицы фильтра. Для таких стран приложение должно поддерживать свою таблицу перекодировки для обеспечения правильной настройки на канал.

Частотные таблицы фильтра могут быть предназначены только для определенной страны (региона) и не соответствовать таблицам страны, где используется приемник. Кроме того, внутренняя таблица фильтра, которая недоступна для записи, может содержать ошибки. Для исправления ситуации, существует механизм, позволяющий заменить отдельные записи внутренней частотной таблицы, вплоть до полной замены таблицы. Это позволяет установить правильную частоту канала в любом случае, в том числе и при ошибке во внутренней таблице фильтра. Новые таблицы частот сохраняются в регистре и называются настроечным пространством (Tuning Space). При существовании настроечного пространства метод IAMTuner::put_Channel может использовать для настройки эти данные вместо данных внутренней таблицы фильтра.

Для создания настроечного пространства в регистре создается раздел HKEY_LOCAL_MACHINE\Software\Microsoft\TV System Services\TVAutoTune\ , в котором имеется две частотные таблицы, одна для широковещательных частот, а вторая для кабельных частот. Частотная таблица для широковещательных каналов содержится в подразделе регистра с именем TSn-0, где n это номер настроечного пространства, а нуль указывает на таблицу широковещательных каналов. Удобно устанавливать номер настроечного пространства равный коду страны. Для кабельных частот имя таблицы TSn-1. Настроечное пространство имеет более высокий приоритет, чем внутренняя частотная таблица фильтра. Если имеется частотная таблица с нулевым номером (TS0-0, TS0-1), то частота канала будет выбираться из этой таблицы по умолчанию, при условии, что там есть записи. Если записи нет, выбирается частота из таблицы устройства. Для использования других настроечных пространств существует метод IAMTuner::put_TuningSpace, устанавливающий текущее настроечное пространство.

Для каждой частоты в настроечном пространстве указывается параметр типа DWORD, где имя параметра это номер канала, а значение частота. Ниже, в качестве примера, приводится несколько строк из файла .reg для записи в регистр данных настроечного пространства кабельного телевидения. Полностью файлы .reg находятся в папке проекта ProjectState4.

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\TV System Services
\TVAutoTune\TS7-1]
“1”=dword:02b275d0
“2”=dword:02ba16f0
“3”=dword:02c1b810
“4”=dword:02e03c90
“5”=dword:02f71ff0
“6”=dword:033428f0
"7"=dword:034b0c50
```

Количество каналов в частотной таблице можно увеличить выше установленного предела во внутренней таблице частот фильтра. Например, можно указать частоту для 400 канала. В этом случае метод IAMTuner::ChannelMinMax будет возвращать максимальное значение числа каналов равное 400. В данной реализации настройка на канал с номером большим, чем номер во внутренней таблице не выполняется. В результате настроится на канал 400 невозможно. Создание записей в регистре для настроечных пространств выполняется приложением.

Выбор настроечного пространства осуществляется при вызове метода IAMTuner::put_TuningSpace. Метод должен вызываться после установки типа входа (IAMTVTuner::put_InputType), который определяет, какую из частотных таблиц настроечного пространства следует использовать. Далее при описании класса будут рассмотрены некоторые варианты создания настроечных пространств и особенности реализации методов для работы с настроечными пространствами.

При проведении экспериментов с устройством было обнаружено, что список частотных таблиц определяется при установке драйверов и делится он на две группы. Одна группа включает все страны со стандартом PAL и SECAM, а вторая страны со стандартом NTSC. Например, если выбрана страна из первой группы, то в дальнейшем установка кода страны из второй группы невозможна без переустановки драйверов. При попытке установить код страны из другой группы метод IAMTuner::put_CountryCode возвращается с кодом ошибки.

Кроме рассмотренных настроечных пространств Microsoft предлагает унифицированную модель настройки для любых сетей телевизионного вещания, в том числе и для аналоговых сетей. Для этой модели настройки в регистре (Windows XP) создается системный раздел HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tuning Spaces, в котором сохраняется информация для настройки в цифровых и аналоговых телеви-

зионных сетях вещания. Данную модель настройки можно использовать и для настройки чисто аналогового приемника, но это не дает никаких преимуществ, поскольку после некоторой дополнительной работы все равно возвращается интерфейс IAnalogTVTuningSpace, который является усеченным вариантом интерфейса IAMTVTuner. Также потребуется использование еще нескольких объектов и интерфейсов, которые не дают новых возможностей по сравнению с интерфейсами аналогового телевидения, но требуют дополнительных усилий программиста.

Класс фильтра настройки приемника и интерфейс пользователя

Для решения задач связанных с выполнением настроек телеприемника создадим класс фильтра настройки. Класс напишем по образу и подобию класса переключателя с ограничением количества создаваемых объектов.

```
class CTuner
{
public:
  //Псевдоконструктор
  static CTuner*
          MakeTuner(CComPtr<IAMTVTuner> pAMTVTuner);

  ~CTuner(void);

private:
  //Указатель на интерфейс фильтра
  CComPtr<IAMTVTuner> m_pTuner;

  //Закрытый конструктор
  CTuner(CComPtr<IAMTVTuner>);

  //Подсчет числа объектов для ограничения числа
  //объектов
  static size_t m_numObj;

  //Копирование запрещено, конструктор копировщик
  //не определяется
  CTuner(const CTuner& rTuner);
};
```

Фильтр настройки приемника имеет много настроечных параметров. Часть этих параметров необходимо предоставить для управления пользователю, другая часть может использоваться по умолчанию. Ограничимся наиболее употребительными из них:

- Выбор режима работы приемника (приемник звуковых программ, TV).

- Переключение частотных таблиц приемника (антенна/кабель).
- Настройка на выбранный канал.
- Поиск каналов.
- Установка кода страны.
- Сведения о доступных телевизионных стандартах.

Обилие настроек пользователя предполагает интенсивное взаимодействие класса настройки с классом интерфейса пользователя. Логика работы класса состоит в настройке приемника в ответ на действия пользователя. Из списка настроек исключим установку кода страны, поскольку этот параметр устанавливается при установке драйвера устройства. При необходимости установку кода страны несложно добавить в состав органов управления. Если существует необходимость смены частотной таблицы при смене местоположения компьютера, то удобней предоставить пользователю возможность выбора региона, вместо выбора страны. Число регионов значительно меньше числа стран. Список регионов приводится ниже.

1. Australia
2. China
3. Czech Republic
4. Eastern Europe
5. France
6. French Overseas Territories
7. Ireland
8. Italy
9. Japan
10. New Zealand
11. NTSC (United States)
12. United Kingdom
13. Western Europe

При выборе региона можно устанавливать код одной из стран в данном регионе, поскольку частотная таблица одна для всех стран региона. Коды стран и стандарты ТВ приведены в справочном разделе.

Пользователю для управления настройкой приемника предоставим следующие органы управления.

1. Список для выбора канала. Список должен отображать основные данные канала и его состояние, допускать редактирование имен каналов и сохранять данные, выполнять сортировку каналов для обеспечения быстрого доступа по имени канала, его состоянию и т.д.
2. Включение режима поиска каналов.

3. Переключатель режима работы приемника.
4. Переключатель входов приемника.
5. Переключатель частотных таблиц (тип входа).
6. Отображение доступных ТВ стандартов приемника.

Элементы интерфейса пользователя расположим на первой вкладке, класс CTVshowView. На рисунке 3.7 показаны органы управления настройкой приемника.

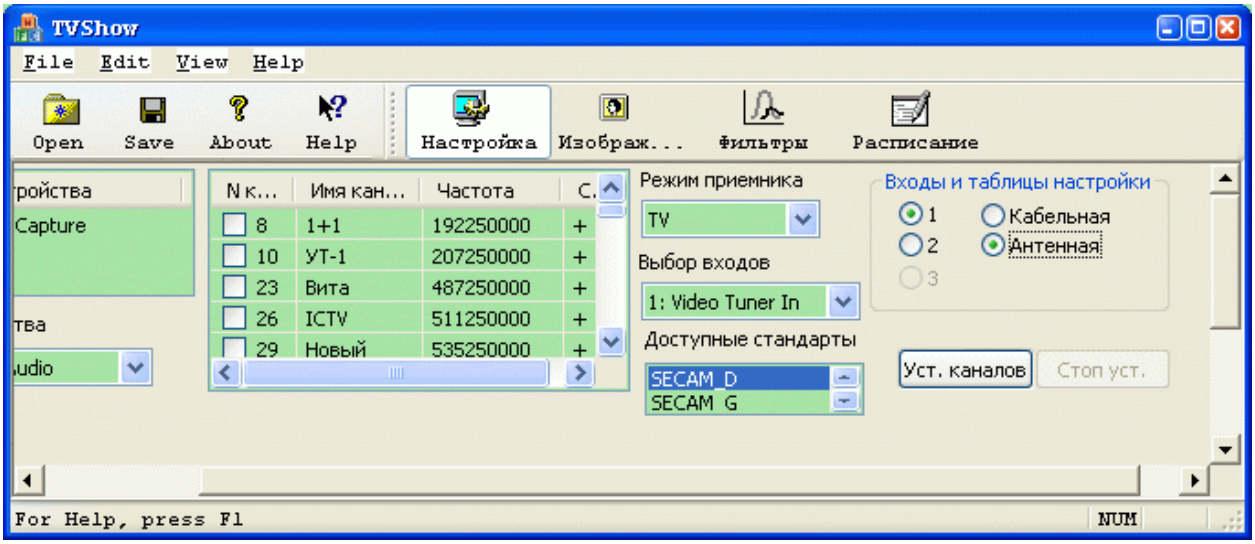

Рис.3. 9 Управление фильтром настройки приемника

Для этих органов управления назначим переменные в классе CTVshowView.

```
//Список каналов
CListCtrl m_listChannal;

//Установка режима приемника TV/Приемник
//звуковых программ
CComboBox m_listMode;

//Стадарт ТВ
CComboBox m_standartTV;

//Выбор  входов
BOOL m_bInput;       //Входы 1,2,3
CButton m_wInput1;     //Первый вход
CButton m_wInput2;     //Второй вход
CButton m_wInput3;     //Третий вход

//Кнопка кабель (выбор частотной таблицы)
CButton m_wCable;

//Кнопка антенна (выбор частотной таблицы)
CButton m_wAntenna;
```

```
  //Признак типа входа антенна/кабель
  //(переключение частотных таблиц)
  BOOL m_bInputType;

  //Кнопка поиска каналов
  CButton m_butTuneChannal;

  //Кнопка остановки поиска каналов
  CButton m_wStopTune;
```

Для всех переменных создаем обработчики. Для списка каналов будет создано несколько обработчиков, которые будут рассматриваться позже, а сейчас займемся созданием методов класса CTuner для простых органов управления.

Установка режима приемника

Многофункциональный приемник может иметь несколько режимов работы, которые определены в перечислении AMTunerModeType.

```
typedef enum tagAMTunerModeType
    {
        AMTUNER_MODE_DEFAULT      = 0x0000,
        AMTUNER_MODE_TV           = 0x0001,
        AMTUNER_MODE_FM_RADIO     = 0x0002,
        AMTUNER_MODE_AM_RADIO     = 0x0004,
        AMTUNER_MODE_DSS          = 0x0008
    } AMTunerModeType;

AMTUNER_MODE_TV         - режим телевизионного приемника.
AMTUNER_MODE_FM_RADIO   - режим приемника FM
AMTUNER_MODE_AM_RADIO   - режим приемника AM.
AMTUNER_MODE_DSS        - режим цифрового спутникового
                              приемника.
```

Для отображения режима приемника напишем метод CTuner::GetNameMode, который будет определять символьные эквиваленты режимов.

```
string CTuner::GetNameMode(long mode)
{
  string s;
  switch(mode)
  {
  case AMTUNER_MODE_DEFAULT:
    s= "По умолчанию";
    break;
  case AMTUNER_MODE_TV:
```

```
    s= "TV";
    break;
  case AMTUNER_MODE_ЗВУКОВЫХ ПРОГРАММ_RADIO:
    s= "FM Radio";
    break;
  case AMTUNER_MODE_AM_RADIO:
    s= "AM Radio";
    break;
  case AMTUNER_MODE_DSS:
    s= "Цифровой спутниковый приемник(DSS)";
    break;
  }

  return s;
}
```

Режим приемника устанавливается при вызове метода CTuner::SetMode, код, которого показан ниже. Установка режима приемника определяет частотный диапазон настройки. В режиме ТВ фильтр настройки обеспечивается настройку телевизионной части приемника, в режиме приемника звуковых программ обеспечивается настройка приемника звуковых программ. Независимо от режима, для настройки используются одни и те же интерфейсы. Изменяются только возвращаемые значения некоторых методов. Например, в режиме звуковых программ метод IAMTuner::ChannelMinMax возвращает не номера каналов, а предельные значения частот настройки. Работа в режиме приемника не рассматривается, читатель может самостоятельно написать код для управления приемником звуковых программ, используя в качестве примера настройку телевизионного приемника.

```
BOOL CTuner::SetMode(long mode)
{
  //установить новый режим
  if(S_OK != m_pTuner->put_Mode(
                      static_cast<AMTunerModeType>(mode)))
    return FALSE;

  return TRUE;
}
```

Возврат режима приемника производит метод GetMode.

```
long CTuner::GetMode(void)
{
  //возвратить режим
  AMTunerModeType mode;
  if(S_OK == m_pTuner->get_Mode(&mode))
    return mode;

  return 0;
}
```

Метод IAMTuner::GetAvailableModes возвращает все режимы, поддерживаемые приемником. Для отображения списка доступных режимов метод заполняет вектор символьных имен режимов и вектор констант, определяющих режимы приемника.

```
void CTuner::GetVectorAvailableModes(
                vector<string>& vStr, vector<long>& vMode)
{
  HRESULT hr = S_OK;
  LONG mode = 0;
  AMTunerModeType modeCurrent;

  //Определить доступные режимы
  hr = m_pTuner->GetAvailableModes(&mode);

  if(hr != S_OK || mode == 0)
  {
    string serr = "Ошибка GetVectorAvailableModes";
    CTVshowApp::myMessageError(hr, serr);
    return;
  }

  //Определить текущий режим
  m_pTuner->get_Mode(&modeCurrent);

  //Заполнить вектор имен  и вектор режимов
  if(mode & AMTUNER_MODE_TV)
  {
    vStr.push_back(GetNameMode(AMTUNER_MODE_TV));
    vMode.push_back(AMTUNER_MODE_TV);
  }

  if(mode & AMTUNER_MODE_FM_RADIO)
  {
    vStr.push_back(GetNameMode(AMTUNER_MODE_FM_RADIO));
    vMode.push_back(AMTUNER_MODE_FM_RADIO);
  }

  if(mode & AMTUNER_MODE_AM_RADIO)
  {

  vStr.push_back(
                GetNameMode(AMTUNER_MODE_AM_RADIO));
    vMode.push_back(AMTUNER_MODE_AM_RADIO);
  }

  if(mode & AMTUNER_MODE_DSS)
  {
    vStr.push_back(GetNameMode(AMTUNER_MODE_DSS));
    vMode.push_back(AMTUNER_MODE_DSS);
```

```
  }

  if(mode & AMTUNER_MODE_DEFAULT)
  {
    vStr.push_back(GetNameMode(AMTUNER_MODE_DEFAULT));
    vMode.push_back(AMTUNER_MODE_DEFAULT);
  }

//Определить текущий режим и передвинуть его на нулевую //позицию в векторе
  string sCurrent = GetNameMode(modeCurrent);
  if(vStr.size() > 1 )
  {
    rotate(vStr.begin(),find(vStr.begin(),vStr.end()
                              ,sCurrent),vStr.end());

    rotate(vMode.begin(),find(vMode.begin(),vMode.end()
                              ,modeCurrent),vMode.end());
  }
}
```

Для поиска в векторе текущего режима и перестановки в нулевую позицию используется два алгоритма STL, алгоритм find, который находит режим и алгоритм rotate, производящий циклический сдвиг вектора в указанную позицию.

Переключение входов

Теперь напишем свои методы для переключения входов фильтра приемника. Для переключения входов фильтра интерфейс IAMTVTuner предлагает три метода. Непосредственно переключение входов производит метод put_ConnectInput, который подключает вход с указанным индексом. Фильтр может иметь несколько входов, например вход для подключения телевизионного кабеля (антенны) и вход для подключения антенны приемника FM, AM или другого типа. Каждый вход может иметь свой тип разъема (соединителя). Переключение входов приемника в большинстве случаев не имеет эффекта, поскольку физически антенные входы объединены в фильтре сложения сигналов. Возможно, существуют устройства, имеющие раздельные усилители сигналов для приемника ТВ и приемника FM, тогда переключение входов может иметь эффект.

Методы get_InputType и put_InputType, хотя и имеют названия, которые позволяют отнести их к методам, управляющих входами, в действительности управляют переключением частотных таблиц и на переключение входов не влияют. Выбор частотной таблицы определяется индексом, который в свою очередь определен в перечислении TunerInputType.

```
  typedef enum
  {
    TunerInputCable,
```

```
    TunerInputAntenna
  }TunerInputType;
```

В зависимости от установленного типа входа фильтра выбирается частотная таблица либо внутренняя, либо из текущего настроечного пространства. При установленном типе входа TunerInputCable используется частотная таблица кабельного телевидения, а для типа входа TunerInputAntenna таблица широковещательного телевидения.

Следует заметить, что в документации DirectShow вообще не удалось найти внятного упоминания о высокочастотных входах фильтра настройки приемника. Возможно, что это связано с тем, что переключение входов не оказывает никакого влияния на функционирование устройства и можно вообще в интерфейсе их не показывать, при условии, что устройство действительно допускает такую возможность. При завершении приложения фильтр сохраняет индекс подключенного входа, но не сохраняет его тип, который всегда равен TunerInputCable. Таким образом, сохранение типа входа (номера частотной таблицы) и его восстановление является задачей приложения.

Метод для подключения входа приемника.

```
BOOL CTuner::SetConnectInput(long indInput)
{
  HRESULT hr = m_pTuner->put_ConnectInput(indInput);

  if(S_OK == hr)
    return TRUE;

  return FALSE;
}
```

Метод, возвращающий число входов приемника.

```
LONG CTuner::GetNumInput(void)
{
  LONG num = 0;

  if(S_OK != m_pTuner->get_NumInputConnections(&num))
    return 0;

  return num;
}
```

Метод, возвращающий индекс подключенного входа и его тип (индекс частотной таблицы).

```
BOOL CTuner::GetInputTableFreq(TunerInputType& input,LONG& in-
dex)
{
```

```
  LONG ind=0;
  TunerInputType inputType;

  //опр. индекс подключенного входа
  if(S_OK == m_pTuner->get_ConnectInput(&ind))
  {
    //опр. тип входа
    if(S_OK == m_pTuner->get_InputType(ind,&inputType))
    {
      input = inputType;
      index = ind;
      return TRUE;
    }
  }

  return FALSE;
}
```

Метод, переключающий тип входа (частотную таблицу).

```
BOOL CTuner::SetInputTableFreq(long idxFreq)
{
  //индекс подключенного входа
  long ind;

  if(S_OK != m_pTuner->get_ConnectInput(&ind))
    return FALSE;

  //устанавливаем таблицу
  if(S_OK != m_pTuner->put_InputType(ind,
                          static_cast<TunerInputType>(idxFreq)))
    return FALSE;

  return TRUE;
}
```

Для управления входами фильтра используются обработчики в классе CTVshowView OnBnClickedRInput1, OnBnClickedRInput2, OnBnClickedRInput3, которые обрабатывают нажатия на кнопки переключения входов.

Для обработки переключения типа входа (выбора частотных таблиц) используются обработчики CTVshowView::OnBnClickedRantenna() и CTVshowView::OnBnClickedRcable. В этом же обработчике заполняется вектор данных каналов с помощью метода myFillListChannalFromVector.

Предварительная инициализация производится в методе OnInitialUpdate, в котором блокируются все элементы интерфейса до момента установки исходного состояния фильтра, который определяется режимом работы. При этом запрещаются все элементы управления.

Полностью инициализация фильтра настройки рассматривается в разделе "Интерфейс управления фильтром настройки".

Отображение доступных стандартов ТВ

Метод IAMTVTuner::get_AvailableTVFormats позволяет отобразить совокупность входных стандартов ТВ, которые может обработать фильтр настройки приемника. Фильтр настройки автоматически обрабатывает все стандарты ТВ, которые возвращает метод. Метода для настройки фильтра на определенный стандарт нет, фильтр не настраивается на обработку какого либо определенного стандарта. Он просто предоставляет для отображения те стандарты, с которыми может работать. Данное устройство на стадии установки драйверов делит все стандарты на две группы, NTSC это одна группа, SEKAM и PAL вторая группа. Метод get_AvailableTVFormats отображает доступные стандарты только из группы выбранной на стадии установки драйверов. Для просмотра доступных стандартов используется элемент управления CListBox.

Метод CTuner::GetVectorAvailableTVStandards заполняет вектор символьных имен стандартов ТВ и вектор констант, которые определяют стандарты.

```
void CTuner::GetVectorAvailableTVStandards(
            vector<string>& vStr, vector<long>& vSt)
{
  HRESULT hr = S_OK;
  string serr = "Ошибка GetVectorAvailableTVStandards";

  //Стандарты ТВ доступные и текущий
  LONG stAvailable = 0,stCurrent=0;

  //Определить доступные стандарты
  hr = m_pTuner->get_AvailableTVFormats(&stAvailable);
  ASSERT(stAvailable);

  if(hr != S_OK )
  {
    CTVshowApp::myMessageError(hr, serr);
    return;
  }
  //Определить текущий стандарт
  hr = m_pTuner->get_TVFormat(&stCurrent);

  if(hr != S_OK )
  {
    CTVshowApp::myMessageError(hr, serr);
    return;
  }
```

```
  //Заполнить вектор имен  и вектор режимов
  LONG mask=1;//Маска
  LONG tmp;

  for(int i=0; i<21; ++i)
  {
    tmp = mask & stAvailable;
    if(tmp)
    {
      vStr.push_back(GetNameStandartTV(tmp));
      vSt.push_back(mask);
    }
    mask <<= 1;
  }

//Определить текущий стандарт и передвинуть его на
//нулевую позицию в векторе
  string s = GetNameStandartTV(stCurrent);
  rotate(vStr.begin(),find(vStr.begin()
                   ,vStr.end(),s),vStr.end());
  rotate(vSt.begin(),find(vSt.begin()
                   ,vSt.end(),stCurrent),vSt.end());
}
```

В методе GetVectorAvailableTVStandards вызывается статический метод GetNameStandartTV, возвращающий символьное имя стандарта ТВ. Он определен в файле Constants.h. В этом же файле определена структура, определяющая символьные имена стандартов. Ознакомиться с кодом файла можно в проекте. Метод GetTVStandard возвращает текущий стандарт ТВ, который установлен и используется.

```
LONG CTuner::GetTVStandard(void)
{
  LONG fm ;

  //определить текущий стандарт
  if(m_pTuner->get_TVFormat(&fm ) == S_OK)
    return fm ;

  return -1;
}
```

К сожалению, фильтр настройки приемника не определяет стандарт принимаемой телепередачи. В результате невозможно осуществить автоматическое переключение видеодекодера фильтра сбора на принимаемый сигнал текущего стандарта.

Организация данных для отображения телевизионных каналов

Пользователю для удобной работы чаще всего необходимо предоставить номер канала, символьное имя канала, и состояние настройки на канал. Также пользователь должен иметь возможность изменять имя канала и иметь возможность произвести сортировку каналов по нескольким параметрам.

Для обработки данных каналов лучше всего применить библиотеку STL, что позволит резко снизить затраты труда и повысить эффективность обработки. Чтобы использовать возможности STL в полной мере, необходимо организовать данные в соответствии с требованиями библиотеки. Это можно сделать несколькими способами, но предпочтительней использовать универсальную организацию данных в виде отдельного класса, который позволяет передавать их в любой алгоритм STL. Этот подход очень экономит время, так как не требует переделок структуры данных в процессе работы. При окончательной отладке программы можно просто преобразовать данные в структуру, если класс окажется избыточным. Организацию сложных данных следует рассмотреть подробнее, поскольку в литературе по STL этот вопрос освещается плохо, в основном все примеры даются для простых типов данных. Данные каналов представим в виде класса, который расположим в заголовочном файле класса CTuner.

```
class CDataTuner
{
  public:
    CDataTuner() {};      //конструктор
    ~CDataTuner() {};     //деструктор

  //конструктор копировщик
  CDataTuner(const CDataTuner & dataTime);

  //оператор присваивания
  const CDataTuner & operator= (const CDataTuner&) ;

  //данные
    string m_nameChannal;   //имя канала
    long m_numChannal;        //номер канала
    long m_freq;              //частота канала
    char m_state;             //состояние настройки

  //оператор меньше (сортировка по умолчанию)
  BOOL operator < (const CDataTuner & ) const;

  //оператор равно
  BOOL operator == (const CDataTuner& ) const;
};
```

Класс, кроме данных, содержит несколько дополнительных членов, которые необходимы для работы практически всех алгоритмов. Это конструктор копировщик, создающий копию объекта и три перегруженных оператора, присваивания, меньше и равно. Оператор равно используется реже чем оператор меньше и если он будет не нужен, то его можно легко удалить при окончательной отладке программы.

Реализация класса.

```
//Конструктор копировщик
CDataTuner::CDataTuner(const CDataTuner& dataTuner)
{
  *this = dataTuner;
}

//Оператор присваивания
const CDataTuner & CDataTuner::operator= (const CDataTuner& dt)
{
  if(this != &dt)
  {
    m_nameChannal   = dt.m_nameChannal;
    m_numChannal    = dt.m_numChannal;
    m_freq          = dt.m_freq;
    m_state         = dt.m_state;
  }

  return *this;
}

//Оператор меньше (сортировка по номеру канала, по
//умолчанию)
BOOL CDataTuner::operator < (const CDataTuner& dt) const
{
  return  m_numChannal < dt.m_numChannal;
}

//Оператор равно,если номера каналов равны
BOOL CDataTuner::operator == (const CDataTuner& dt) const
{
  return m_numChannal == dt.m_numChannal;
}
```

Алгоритм сортировки sort и его разновидности используют оператор "меньше" по умолчанию. Перегруженный оператор "меньше" сравнивает номера каналов, и поэтому сортировка будет происходить по номерам каналов по возрастанию. Если необходимо отсортировать данные по другому критерию, то в алгоритм передается функция предикат, выполняющая сравнение элементов по другому критерию.

Предикатом называется функция, возвращающая логическое значение, которое зависит только от передаваемых в нее параметров. Это важное условие

не должно нарушаться. Библиотека STL требует, чтобы при неизменности входных данных предикат всегда возвращал постоянный результат. Например, в предикате нельзя установить внутренний счетчик, который будет изменять возвращаемый результат в зависимости от своего значения. Только параметры могут определять результат. Предикаты бывают унарные и бинарные. Унарный предикат проверяет один аргумент, бинарный сравнивает два аргумента.

Предикат должен быть объявлен как глобальная функция. Можно сделать предикат и членом класса данных, но это только будет увеличивать размер объекта данных и не принесет особого выигрыша. Наибольший выигрыш и наиболее универсальный подход состоит в использовании предикатов в виде объектов функций (функторов), которые передаются алгоритмам. Объекты функций часто используются в STL и являются очень полезным инструментом.

Объект функции это объект класса, который ведет себя как функция, то есть объект вызывается как функция с круглыми скобками и передачей параметров. Для этого в классе объекта функции переопределяется оператор круглые скобки. Можно написать собственный объект функции, но в таком случае его возможности будут ограничены. Для того чтобы объект функции мог использовать все возможности библиотеки STL его следует объявить производным от одной из двух специальных структур, как показано ниже.

Объекты функций обладают целым рядом преимуществ по сравнению с обычными функциями предикатами. Они могут включать другие функции и иметь атрибуты, работают они обычно быстрее функций.

Для сортировки по другим критериям напишем несколько объектов функций, которые будут использоваться в разных алгоритмах.

Сортировка по частоте.

```
struct SortDataTunerFreq :public
        std::binary_function<CDataTuner,CDataTuner,bool>
{
  bool operator () (const CDataTuner& e1
                            ,CDataTuner& e2) const
  {
    return e1.m_freq < e2.m_freq;
  }
};
```

Сортировка по имени канала.

```
struct SortDataTunerNameChannal :public
        std::binary_function<CDataTuner,CDataTuner,bool>
{
  bool operator () (const CDataTuner& e1
                           ,CDataTuner& e2) const
  {
```

```
    return e1.m_nameChannal < e2.m_nameChannal;
  }
};
```

Сортировка по состоянию настройки. Для примера, настроенный канал будем отмечать знаком "+". Сортировка по состоянию канала будет передвигать на первые позиции вектора настроенные каналы.

```
struct SortDataTunerState :public std::unary_function<CData-
Tuner,bool>
{
  bool operator () (const CDataTuner& e1) const
  {
    return e1.m_state == '+';
  }
};
```

Метод CTuner::SortDataTuner выполняет один из видов сортировки в том векторе, который определяется по типу входа. Вид сортировки указывается параметром. Для всех сортировок, кроме сортировки по состоянию канала используется один алгоритм sort с разными предикатами. Для сортировки по состоянию канала применяется алгоритм stable_partition, который перемещает в начало отсортированные элементы с сохранением относительного порядка следования. Таким образом, после сортировки по состоянию канала все работающие каналы будут перемещены в начало списка и отсортированы по возрастанию.

```
void CTuner::SortDataTuner(long typeSort)
{
  TunerInputType inputType; LONG ind;

  if(!GetInputType(inputType,ind))
    return;

  switch(inputType)
  {
  case TunerInputCable:
    {
      switch(typeSort)
      {
      case 0: //Сортировка по номеру канала
        sort(m_vDataChannalCable.begin()
                      ,m_vDataChannalCable.end());
        break;
      case 1: //Сортировка по имени канала
        sort(m_vDataChannalCable.begin()
                    ,m_vDataChannalCable.end()
                    ,SortDataTunerNameChannal());
        break;
      case 2: //Сортировка по частоте
```

```
        sort(m_vDataChannalCable.begin()
                    ,m_vDataChannalCable.end()
                    ,SortDataTunerFreq());
        break;
      case 3: //Сортировка по состоянию каналов
        stable_partition(m_vDataChannalCable.begin()
                             ,m_vDataChannalCable.end()
                             ,SortDataTunerState());
        break;
      }
    }
    break;

  case TunerInputAntenna:
    {
      switch(typeSort)
      {
      case 0: //Сортировка по номеру канала
        sort(m_vDataChannalAnt.begin()
                       ,m_vDataChannalAnt.end());
        break;

      case 1: //Сортировка по имени канала
        sort(m_vDataChannalAnt.begin()
                         ,m_vDataChannalAnt.end()
                         ,SortDataTunerNameChannal());
        break;

      case 2: //Сортировка по частоте
        sort(m_vDataChannalAnt.begin()
                    ,m_vDataChannalAnt.end()
                    ,SortDataTunerFreq());
        break;

      case 3: //Сортировка по состоянию каналов
        stable_partition(m_vDataChannalAnt.begin()
                           ,m_vDataChannalAnt.end()
                           ,SortDataTunerState());
        break;
      }
    }
  }
}
```

Реализация настройки на каналы

Интерфейсы IAMTuner и IAMTVTuner предоставляют несколько методов для организации настройки на каналы. Часть этих методов упоминалась ранее. Ниже приводится краткое описание методов по данным документации DirectShow. Все особенности настройки, определенные экспериментальным путем, описываются далее. Возможно, что в других условиях, полученные ре-

зультаты могут не подтвердится или проявятся дополнительные эффекты. Следует учитывать, что реализация интерфейса определяется производителем устройства (драйвера). В результате может сложиться ситуация, когда при соответствии каждого отдельного метода описанию DirectShow их совместное применение может иметь некоторые отличия для различных устройств.

- Метод put_CountryCode устанавливает код страны для указания внутренней частотной таблицы устройства, метод get_CountryCode возвращает код страны.
- Метод put_TuningSpace производит установку текущего настроечного пространства. Метод get_TuningSpace возвращает текущее настроечное пространство.
- Метод ChannelMinMax возвращает минимальное и максимальное значение числа каналов в текущей частотной таблице.
- Метод put_Channel производит выборку частоты из таблицы по номеру канала и настройку приемника на эту частоту. Метод get_Channel возвращает номер канала.
- Метод AutoTune производит настройку на канал с подстройкой частоты в некоторых пределах.
- Метод StoreAutoTune сохраняет результаты настройки в текущем настроечном пространстве в регистре. Метод в текущей версии работает частично, он создает настроечное пространство в регистре, но записей в него не производит.
- Метод SignalPresent определяет наличие или отсутствие полезного сигнала.
- Методы get_VideoFrequency и get_AudioFrequency возвращают значения видео и звуковой частоты, на которую настроен канал.
- Метод put_InputType устанавливает тип частотной таблицы, get_InputType возвращает установленный тип частотной таблицы.

Перечисленные методы позволяют производить настройку каналов двумя способами.

- Первый способ предполагает использование метода AutoTune, который производит точную настройку на частоту канала в определенных пределах. В документации указано, что этот метод не работает с настроечными пространствами и рекомендуется для упрощенных реализаций. В действительности в данной реализации метод работает с настроечным пространством.
- Второй способ использует метод put_Channel для настройки на канал и метод SignalPresent, который возвращает результат настройки. Метод put_Channel использует предварительно установленное настроечное пространство для преобразования номера канала в частоту. Метод

SignalPresent возвращает результат настройки с одним из трех возможных значений. Результат настройки сохраняется в регистре с помощью метода IAMTVTuner::StoreAutoTune. В действительности в данной реализации метод StoreAutoTune не сохраняет данных в регистре.

Процесс настройки на каналы занимает много времени. Для эфирного телевидения, где число каналов менее ста еще можно допустить поиск и настройку на все каналы при каждом запуске приложения. Для кабельного телевидения, где общее число каналов немногим менее четырехсот, время ожидания настойки на все каналы недопустимо велико. Учитывая тенденцию увеличения числа каналов, необходимо предусмотреть возможность ускорения процесса настройки. Кроме того, пользователю кроме списка каналов и частот настройки желательно предоставить некоторые сервисные возможности. Например, предоставить состояние канала, есть вещание на данном канале или нет, сортировку каналов по нескольким признакам, изменение названия канала, частоты канала и некоторые другие возможности.

Настроечное пространство имеет данные только о канале, частота которого отличается от значения частоты внутренней таблицы устройства. Остальные данные о каналах необходимо сохранять в регистре или в отдельном файле состояния программы. Сохранение дополнительных данных в отдельном файле, который хранится в установочной директории программы, позволяет сократить время их загрузки и отображения. Хранение достаточно большого количества данных в регистре не рекомендуется, время поиска в регистре может занимать много времени.

Теперь, когда стала понятной организация данных для каналов, можно приступить к реализации методов настройки на каналы. Методы будем создавать с учетом использования настроечных пространств. Для хранения текущих данных каналов создадим два вектора, один для кабельных каналов и второй для каналов эфирного телевидения.

```
vector<CDataTuner> m_vDataChannalCable;
vector<CDataTuner> m_vDataChannalAnt;
```

Эти два вектора будут использоваться следующим образом. При старте программы они будут инициализироваться данными из файла. При первом старте программы после ее установки, когда файл данных еще не существует, векторы должны заполняться при поиске каналов по указанию пользователя. После окончания работы, данные из векторов, будут сохраняться в файле.

Таким образом будет создана достаточно комфортная обстановка для работы пользователя. Ему не нужно будет ожидать каждый раз окончания длительной процедуры настройки на каналы при запуске программы. В тоже время он сможет при необходимости произвести настройку на каналы и установить все данные в исходное состояние в аварийной ситуации или при появлении в сетке вещания новых каналов. Каждый раз при старте программы необходимо проводить проверку наличия настроечного пространства и при необ-

ходимости его создавать. Для этой цели напишем метод CTuner::MakeTuneSpace. Как говорится в документации, создание настроечного пространства и запись в него корректирующих данных производит метод IAMTVTuner::StoreAutoTune. Метод не имеет параметров. В действительности метод StoreAutoTune только создает настроечное пространство, но не производит в него записей. В документации DirectShow указывается, что для использования метода необходимо и достаточно установить код страны. Это не так, в данном случае необходимо указывать номер настроечного пространства, только тогда метод будет создавать нужное настроечное пространство. Если не указать номер настроечного пространства, то не зависимо от кода страны будет создаваться настроечное пространство с кодом нуль.

Код страны определяет внутреннюю частотную таблицу устройства, из которой выбирается значение частоты, если его нет в таблице настроечного пространства. Настроечное пространство указывается при вызове метода put_TuningSpace, а частотная таблица внутри пространства указывается при вызове метода put_InputType.

```
BOOL CTuner::MakeTuneSpace(long nTuneSpace)
{
  HRESULT hr;
  TunerInputType type;

  //Возвращаем тип исходной частотной таблицы
  hr = m_pTuner->get_InputType(0,&type);
  if(hr!=S_OK)
    return FALSE;

  //Устанавливаем настроечное пространство
  hr = m_pTuner->put_TuningSpace(nTuneSpace);
  if(hr!=S_OK)
    return FALSE;

  //Устанавливаем тип частотной таблицы TSn-1
  hr = m_pTuner->put_InputType(0,TunerInputCable);
  if(hr!=S_OK)
    return FALSE;

  //Создаем настроечное пространство, если его нет
  hr = m_pTuner->StoreAutoTune();
  if(hr!=S_OK)
    return FALSE;

  //Устанавливаем тип частотной таблицы TSn-0
  hr = m_pTuner->put_InputType(0,TunerInputAntenna);
  if(hr!=S_OK)
    return FALSE;

  //Создаем настроечное пространство, если его нет
  hr = m_pTuner->StoreAutoTune();
  if(hr!=S_OK)
```

```
    return FALSE;

  //Востанавливаем тип частотной таблицы
  hr = m_pTuner->put_InputType(0,type);
  if(hr!=S_OK)
    return FALSE;

  return TRUE;
```

Вызов метода производим в методе CTuner::MakeTuner, после строки CTuner* p = new CTuner(pAMTVTuner); Таким образом, производится установка настроечного пространства по текущему коду страны, которое будет использоваться в процессе работы приложения.

```
  if(p)
  {
    //Резервировать емкость векторов
    p->m_vDataChannalCable.reserve(400);
    p->m_vDataChannalCable.reserve(100);

    //Создать настроечные пространства по номеру страны
    long countyCode;
    p->m_pTuner->get_CountryCode(&countyCode);

    p->MakeTuneSpace(countyCode);
  }
```

Первоначально, при старте приложения после его установки, пользователь должен произвести поиск всех доступных каналов. В процессе эксплуатации приложения также может возникнуть необходимость в проведении повторного поиска каналов. Для поиска каналов напишем метод CTuner::FindChannals. Метод будет производить поиск всех каналов в соответствующей частотной таблице и с использованием настроечного пространства. После настройки на очередной канал определяется состояние канала и производиться запись данных в текущий вектор и в файл, сохраняющий данные для последующего восстановления при очередном запуске приложения.

```
BOOL CTuner::FindChannals()
{
  long lMin = 0, lMax = 0;    //Номера каналов
  LONG countryCode = 0;       //Код страны
  LONG freqV = 0;               //Частота видеосигнала

  long SignalStrength = AMTUNER_NOSIGNAL;
  HRESULT hr;

  //Тип входа (частотная таблица)
```

```
  TunerInputType inputType;
  LONG ind;

  //Очистить вектор
  if(GetInputTableFreq(inputType,ind))
  {
    switch(inputType)
    {
      case TunerInputCable:
        m_vDataChannalCable.clear();
        break;
      case TunerInputAntenna:
        m_vDataChannalAnt.clear();
    }
  }

  //Определить число каналов
  if (S_OK != m_pTuner->ChannelMinMax(&lMin, &lMax))
      return FALSE;

  //Произвести настройку каналов
  CDataTuner dataTuner;

  for (long i = lMin; i <= lMax; ++i)
  {
    //Прекратить настройку
    if(m_bStopTune)
      break;

    //Настройка на канал по данным настроечного
    //пространства или по таблице устройства
    hr = m_pTuner->put_Channel(i
                      , AMTUNER_SUBCHAN_DEFAULT
                      , AMTUNER_SUBCHAN_DEFAULT);

    if (S_OK == hr)
    {
      //Определить наличие сигнала
      m_pTuner->SignalPresent(&SignalStrength);

      //Определить частоту сигнала
      m_pTuner->get_VideoFrequency(&freqV);

      //Формировать символьный номер канала (n;)
      char buf[16];
      ultoa(i,buf,10);
      string numChannal(buf);

      //Заполнить данные канала
      //Условное имя канала
      m_dataTuner.m_nameChannal = 'K' + numChannal;

      //Номер канала
```

```
      m_dataTuner.m_numChannal = i;

      //Частота настройки
      m_dataTuner.m_freq = freqV;

      //Признак состояния канала
      switch(SignalStrength)
      {
        //Сигнал обнаружен
      case AMTUNER_SIGNALPRESENT:
                m_dataTuner.m_state = '+';
        break;

      //Сигнал не обнаружен
      case AMTUNER_NOSIGNAL:
                m_dataTuner.m_state = '-';
        break;

      //Сигнал пока не обнаружен
      case AMTUNER_HASNOSIGNALSTRENGTH:
                m_dataTuner.m_state = '?';
        break;
      default:;
      }

      //Передать данные канала для отображения
      //(из m_dataTuner) в список каналов
      SendMessageDataTuner();

      //Записать данные в вектор
      switch(inputType)
      {
        case TunerInputCable:
            m_vDataChannalCable.push_back(m_dataTuner);
          break;
        case TunerInputAntenna:
            m_vDataChannalAnt.push_back(m_dataTuner);
      }
    }
  }

  //Сохранить данные настройки в регистре
  //метод не работает в данной реализации устройства
  //hr = m_pTuner->StoreAutoTune();

  //Запись в файл
  if(!WriteVectorToFile())
    return FALSE;

  m_bStopTune = 0; //Разрешить настройку

  return 1;
}
```

В методе FindChannals существует возможность прерывания процесса настройки, который может быть весьма продолжительным. Для этой цели введена переменная m_bStopTune, которая обрабатывается при нажатии на кнопку остановки настройки в обработчике CTVshowView::OnMyMessageDataTuner.

После настройки на канал, когда собраны все данные о нем, окну класса, который содержит список, отправляется сообщение, что он может забрать данные канала для отображения. Это сделано с целью отображения процесса настойки, который протекает долго и пользователь должен видеть, что происходит настройка. Метод SendMessageDataTuner передает в класс окна указатель на себя, и затем объект окна в обработчике сообщения забирает данные канала. Этот метод посылает пользовательское сообщение WM_DATATUNER (WM_USER+1) после очередной настройки на канал, которое обрабатывается в классе CTVshowView. Прерывание цикла настройки нельзя произвести другим способом, например, при нажатии на кнопку в классе CTVshowView, поскольку сообщение от кнопки поступает на обработку только после окончания всего цикла настройки. Можно прервать цикл настройки и другими способами, но данный способ наиболее простой. Основан он на том, что функция SendMessage не возвращается до тех пор, пока не будет обработана, что позволяет определить состояние кнопки прерывания настройки в классе CTVshowView. Отображение каждого настроенного канала позволяет пользователю наблюдать за процессом настройки, что значительно информативней, чем индикатор длинного процесса.

```
void CTuner::SendMessageDataTuner(void)
{
  //получить указатель на представление
  CTVshowApp* pApp =
          static_cast<CTVshowApp*>(AfxGetApp());
  CTVshowView* pView =
          static_cast<CTVshowView*>( pApp->myGetPtrMainView());

  //отправить сообщение
  pView->SendMessage(WM_DATATUNER, 0
                  ,reinterpret_cast<LPARAM>(this));
}
```

После завершения настройки, когда сформирован текущий вектор данных для каналов, производится его запись в файл, сохраняющий все данные и настройки пользователя. После поиска каналов, с целью упрощения, файл полностью обновляется, что приводит к потере пользовательских данных. Желательно было бы корректировать в файле только изменившиеся данные. Файл храниться в установочной директории программы и обновляется перед завер-

шением программы. Обновление производится в обработчике CMainFrame::OnClose().

Поскольку при построении этого класса принято решение не использовать MFC, а процесс обработки файла с помощью API достаточно трудоемкий, можно облегчить задачу с помощью класса ATL CAtlFile. Поначалу была попытка воспользоваться STL и ее библиотекой классов ввода вывода (IOStream) для реализации неформатированного ввода-вывода. Этот путь оказался сложнее, использование класса CAtlFile оказалось намного проще. Структура файла простая. Первым записывается размер вектора данных, затем следуют записи для каждого канала. Перед записью имени канала записывается размер строки имени. Запись файла выполняет метод WriteVectorToFile, чтение файла выполняет метод ReadVectorFromFile. Ознакомиться с кодом этих методов можно в проекте.

При отладке методов MakeTuneSpace и FindChannals окончательно выявились особенности работы с настроечными пространствами. Обнаруженные особенности могут быть присущи только для устройства данного производителя. Они состоят в следующем:

- Для того чтобы при настройке на каналы использовалось настроечное пространство недостаточно установки кода страны и номера настроечного пространства при инициализации в методе MakeTuneSpace. Дополнительно необходимо производить установку кода страны перед вызовом метода put_Channel или AutoTune. Только в таком случае производится выборка частоты из частотной таблицы настроечного пространства. Следует отметить, что настроечное пространство, установленное при инициализации, так и остается установленным, не изменяется, но не используется до повторной установки кода страны. Для использования настроечного пространства в начале метода FindChannals следует добавить следующие строки.

```
long countryCode;
if(S_OK !=
         m_pTuner->get_CountryCode(&countryCode))
     return FALSE;

if(S_OK !=
           m_pTuner->put_CountryCode(countryCode))
     return FALSE;
```

Возможно, такое поведение присуще только для устройства данного производителя и для других устройств не подтвердится. В описании интерфейсов и примерах, приводимых в статьях документации DirectShow нигде нет указания на такое или подобное условие. В исходных кодах примеров DirectShow настроечные пространства не используются. Может быть, это странное условие для использования настроечных пространств есть попытка производителя затруднить использование своего устройства с программами сторонних производителей? Вряд ли,

ведь для этого есть общепринятые способы лицензирования программного продукта. Во всяком случае, повторная установка кода страны не повредит в любом случае, являясь дополнительной страховкой.

- Метод put_CountryCode при вызове с любым кодом страны, всегда обращается к регистру и проверяет наличие настроечных пространств TS0-0, TS0-1.
- Метод put_TuningSpace при вызове обращается к регистру и читает данные из текущей частотной таблицы настроечного пространства.
- Методы put_Channel и AutoTune не обращаются к регистру.
- Метод StoreAutoTune при вызове обращается к регистру, создает настроечное пространство, но не делает никаких записей в нем. Он делает попытку удалить величину AutoTune из регистра.

Проверить факт выборки частоты из настроечного пространства можно очень просто. Достаточно сделать запись в частотную таблицу неверного кода частоты, который будет отображаться в списке каналов. Коды частот для региона Восточной и Западной Европы приведены в справочном разделе, пример записи кода в регистр, приводился выше. Для записи кода частоты в текущее настроечное пространство служит метод CTuner::WriteChannalTuneSpace, который вызывается при корректировке частоты настройки пользователем.

При выборе в списке каналов определенного канала для просмотра, вызывается метод CTuner::PutChannal. Метод настраивает приемник на канал в соответствии с данными настроечного пространства и корректирует данные для канала в соответствующем векторе данных, а также в списке каналов. Для проверки обращения за данными частоты в настроечное пространство, в режиме отладки, устанавливаем код одной из стран, где поддерживается стандарт SECAM, и делаем неправильную запись частоты для канала, на котором происходит вещание. Пример приведен для 8-го канала метрового диапазона, правильная частота 191250000, неправильная 193250000, код страны 375 установлен в методе CTuner::MakeTuner. Если приемник будет настроен на неправильную частоту, значит, обращение к настроечному пространству произошло и пользователь увидит сообщение о настройке на нестандартную частоту.

```
BOOL CTuner::PutChannal(long nChannal)
{
  HRESULT hr= S_OK;

  long SignalStrength = AMTUNER_NOSIGNAL;

  long countryCode;
  if(S_OK != m_pTuner->get_CountryCode(&countryCode))
    return FALSE;
```

```
  //Для использования настроечного пространства
  //следует установить код страны, независимо
  //от установки при инициализации
  if(S_OK != m_pTuner->put_CountryCode(countryCode))
    return FALSE;

  //Произвести настройку каналов
  if (S_OK != m_pTuner->put_Channel(
                 nChannal, AMTUNER_SUBCHAN_DEFAULT
                 , AMTUNER_SUBCHAN_DEFAULT))
    return 0;

  //Произвести настройку каналов
  hr = m_pTuner->put_Channel(
                 nChannal, AMTUNER_SUBCHAN_DEFAULT
                 , AMTUNER_SUBCHAN_DEFAULT);

  //Определить наличие сигнала
  if(hr == NOERROR && S_OK ==
             m_pTuner->SignalPresent(&SignalStrength))
  {
    char state;
    switch(SignalStrength)
    {
    //сигнал обнаружен
    case AMTUNER_SIGNALPRESENT: state = '+';
      break;

    //сигнал не обнаружен
    case AMTUNER_NOSIGNAL: state = '-';
      break;

    //сигнал пока не обнаружен
    case AMTUNER_HASNOSIGNALSTRENGTH: state = '?';
      break;
    }

    LONG freqV = 0; //частота видеосигнала
    if (S_OK != m_pTuner->get_VideoFrequency(&freqV))
      return FALSE;

#ifdef _DEBUG
//Для проверки использования частоты из настроечного //про-
странства  в регистре, в TS375-0
//делаем неправильную запись
    if(nChannal == 8 && freqV == 193250000)
      AfxMessageBox("Настройка на нестандартную частоту
                                 из TSn-0 193250000",MB_OK);
#endif //_DEBUG

    //Проверка настройки на канал
    if (S_OK == hr)
    {
```

```
      //Вернуть данные из вектора
      m_dataTuner.m_numChannal = nChannal;
      LONG ind;

      if(GetDataFromVector(m_dataTuner,ind))
      {
        //Обновить значения частоты в векторе
        if(freqV != m_dataTuner.m_freq || state !=
                                m_dataTuner.m_state)
        {
          m_dataTuner.m_freq = freqV;
          m_dataTuner.m_state = state;

          //Корректировка данных в векторе
          ReplaceDataVector(m_dataTuner);

          //Передать данные канала для отображения
          //(из m_dataTuner)
          SendMessageDataTuner();
        }

      }
    }
  }

  return TRUE;
}
```

Метод GetDataFromVector возвращает данные канала из вектора и индекс вектора. Для определения индекса используется функция STL distance, которая определяет расстояние между итераторами. Данная функция определяет расстояние между итераторами для всех типов итераторов. Для вектора можно также использовать операцию вычитания итераторов, например
indv = static_cast<LONG>(pos - m_vDataChannalAnt.begin()) или
indv = static_cast<LONG>(pos - &m_vDataChannalAnt[0]).

Интерфейс управления фильтром настройки

Некоторые простые элементы управления фильтром настройки упоминались и описывались ранее. К ним относятся переключатели входов и выбор режима приемника. Более сложные элементы управления, такие как элементы настройки и поиска каналов, которые требуют определенной обработки данных, будут рассмотрены в этом разделе. Первичная инициализация элементов интерфейса производится при запуске приложения в методе CTVshowView::OnInitialUpdate().

При выборе видео устройства, в обработчике CTVshowView::OnNMClickListVideo вызывается метод myInitTuner инициализирующий фильтр настройки. При инициализации определяются доступные режимы для устройства, и в зависимости от текущего режима разрешаются или запрещаются соответствующие элементы интерфейса пользователя. Также заполняются списки каналов и доступных стандартов. В режиме приемника звуковых программ запрещаются все элементы, не имеющие отношения к этому режиму. Работа в режиме приемника звуковых программ не рассматривается, поскольку принципы настройки аналогичны, используются те же интерфейсы, и программа легко может быть дополнена этим режимом. Заполнение списка каналов производится из файла данных. Если файл отсутствует, пользователю выдается сообщение о необходимости выполнить поиск каналов. Методы, которые выполняют задачи по заполнению списков данными, являются достаточно простыми и с ними можно ознакомиться в проекте.

В методе myInitTuner производится установка стандарта ТВ в фильтре сбора, в соответствии со стандартом принимаемого канала фильтра настройки. Фильтр настройки возвращает стандарт ТВ, который установлен для страны пребывания. Следует помнить, что фильтр не возвращает стандарт текущей телепередачи. Таким образом, если региональная телестанция ведет передачу в стандарте отличном от стандарта принятого в стране пребывания, фильтр настройки не определит стандарт телепередачи. Кроме того, фильтр сбора не устанавливает автоматически стандарт принимаемого канала, он всегда устанавливает стандарт по умолчанию NTSC. Необходимо согласовать стандарт видеодетектора фильтра сбора со стандартом страны пребывания. В противном случае происходит искажение изображения.

Также в обработчике OnNMClickListVideo вызывается метод myInitCrossBar, который осуществляет заполнение списка видеовходов и соединение внутренних контактов переключателя видеовходов.

Поиск каналов производится при нажатии кнопки "Уст. каналов". Остановка поиска производится при нажатии на кнопку "Стоп уст.". Обработка этих событий производится в обработчиках CTVshowView:: OnBnClickedButTunechannal и CTVshowView::OnMyMessageDataTuner. Первый обработчик вызывает метод для включения поиска каналов.

Второй обработчик обрабатывает пользовательское сообщение, которое посылается из цикла поиска каналов для проверки состояния кнопки остановки. В обработчике пользовательского сообщения производится запись данных настройки на очередной канал в список каналов. Также производится проверка очереди сообщений на наличие сообщения от кнопки остановки поиска. Если сообщение от кнопки есть в очереди, то оно удаляется, а в классе CTuner устанавливается флаг прерывания настройки.

В списке каналов при нажатии на кнопку заголовка вызывается обработчик CTVshowView::OnHdnItemclickListChannal. В обработчике вызывается метод для сортировки по соответствующему критерию. В качестве номера критерия сортировки используется номер нажатой кнопки. В данном случае не используется сортировка, реализованная для класса CListCtrl, поскольку ее возможности значительно проще. Применение STL позволяет с меньшими затратами получить большие возможности по сортировке.

Для настройки на выбранный канал в списке каналов производится одиночный щелчок в поле номера канала (check box). Обработчик OnNMClickListChannal обрабатывает это событие. Класс CListCtrl в режиме report использует независимое переключение, то есть каждый переключатель устанавливается и выключается независимо от состояния других переключателей. Для выбора канала переключатель должен быть зависимым и позволять выбор только одного канала. После обработки щелчка вызывается метод для установки выбранного канала и устанавливается режим предварительного просмотра канала.

Для удобной работы со списком каналов наиболее важны две сервисных функции. Первая это возможность присваивать каналам собственные имена. Вторая это возможность корректировки частоты канала, которая должна быть записана в частотную таблицу текущего настроечного пространства. Операция необходима в случае ошибки во внутренней частотной таблице, а также для установки частот кабельных каналов. Нет другой возможности программно, без участия пользователя, решить эту проблему. Для удобного управления кабельными каналами также следует предусмотреть режим удаления ненужных каналов из частотной таблицы настроечного пространства.

В классе списка CListCtrl отсутствует возможность редактирования полей таблицы. Для реализации такой возможности необходимо создание собственного режима редактирования для колонок имен канала и частоты. С этой целью создается класс CEditChannal производный от CEdit.

```
class CEditChannal : public CEdit
{
friend class CTVshowView;

public:
  CEditChannal(CTVshowView* pV);
  virtual ~CEditChannal();

protected:
  //Указатель на родительское окно
  CTVshowView* m_pParent;

  //Шрифт окна редактирования
  CFont m_font;
```

```
  //Текст из окна
  string m_str;

  //Текущие строка и подстрока редактирования
  int m_nEditItem;
  int m_nEditSubItem;

  DECLARE_MESSAGE_MAP()
  afx_msg void OnClose();

public:
  virtual BOOL PreTranslateMessage(MSG* pMsg);

  //Установка системного шрифта MS Shell Dlg
  //для диалога

  void mySetFont(void);

};
```

При вызове конструктора класса ему передается указатель на родительское окно. Указатель необходим для возврата введенного текста в список каналов. Объект шрифта используется для установки шрифта в окне редактирования. Установка шрифта несколько отличается от обычной практики установки объекта шрифта. Время жизни объекта шрифта определяется временем жизни окна редактирования, поэтому объект шрифта является членом класса. Остальные подробности показаны в тексте метода mySetFont. При создании окна, для него, по умолчанию, устанавливается системный шрифт, который отличается от шрифта диалога и выглядит в окне редактирования непривлекательно. Лучше заменить его системным шрифтом для диалогов. Для корректной установки системного шрифта для диалога следует воспользоваться механизмом замены шрифта. Для этого указывается имя шрифта "MS Shell Dlg" или "MS Shell Dlg 2" для операционных систем 2000/XP. Этот условный шрифт меняется на системный шрифт для диалогов с учетом интернационализации.

Установка системного шрифта MS Shell Dlg.

```
void CEditChannal::mySetFont(void)
{
  LOGFONT lf = {0};
  //Определяем размер шрифта
  lf.lfHeight =
  -MulDiv(8,GetDeviceCaps(GetDC() -> m_hDC
                                ,LOGPIXELSY),72);
   //Русский набор
  lf.lfCharSet = RUSSIAN_CHARSET;
```

```
  //Системный шрифт диалога
  strcpy(lf.lfFaceName,"MS Shell Dlg 2");

  m_font.CreateFontIndirect(&lf);  //Создаем шрифт

  //Устанавливаем шрифт
  //Объект шрифта не должен удаляться !!!
  SetFont(&m_font);
}
```

Для ввода текста из окна редактирования удобно использовать клавишу Enter. К сожалению, обработка нажатия этой клавиши и некоторых других производится нестандартно. Диспетчер диалоговых окон перехватывает нажатия таких клавиш и отправляет их элементам, которые используются в диалоговых окнах по умолчанию. Как известно, завершение ввода в поле редактирования нажатием клавиши Enter приводит к закрытию диалогового окна. Это не совсем удобно, а в данном случае окно еще и не будет удаляться. В литературе встречаются разные способы решения этой проблемы, но все они либо громоздки, либо не работают для клавиши Enter. Эта клавиша стоит особняком даже в ряду особых клавиш диалогового окна. В тоже время перехватить сообщение от любой клавиши очень просто, если помнить о существовании замечательной виртуальной функции PreTranslateMessage, которую и будем использовать для решения этой проблемы. Функция PreTranslateMessage позволяет перехватить сообщение до его распределения. Тексты методов, приведенных ниже, иллюстрируют сказанное.

```
BOOL CEditChannal::PreTranslateMessage(MSG* pMsg)
{
  if(pMsg->wParam == VK_RETURN)
  {
    pMsg->wParam = NULL;

    //Возвращаем текст из окна редактирования
    char  s[MAX_NANE];
    GetWindowText(s,MAX_NANE);
    m_str = s;

    //Удалить свое окно
    PostMessage(WM_CLOSE);
  }
  return CEdit::PreTranslateMessage(pMsg);
}
```

Создание окна редактирования производиться в обработчике OnLvnItemActivateListChannal, который для списка в режиме report обрабатывает двойной щелчок. Для этой цели также можно использовать сообщение NM_DBLCLK.

После ввода текста объект окна должен быть корректно удален во избежание утечки памяти. Для этого используется посылка сообщения WM_CLOSE своему же окну. Это сообщение обрабатывается в обработчике, в котором производится вызов метода для обновления данных канала и удаление объекта окна.

```
void CEditChannal::OnClose()
{
  CEdit::OnClose();
  //Записать изменения в регистр и удалить объект
  //окна редактирования

  m_pParent->myReWriteValueReg(m_str);
}
```

Обновление данных производится в методе myReWriteValueReg, либо только векторе, если корректируется имя канала, либо еще дополнительно в частотной таблице настроечного пространства, если корректируется частота. Для поиска и замены записи в векторе используется метод CTuner::ReplaceDataVector.

Удаление объекта окна редактирования производится в деструкторе класса CTVshowView и при прокрутке. Это вызвано тем, что пользователь может начать прокрутку с не удаленным окном редактирования. В результате окно будет отображаться неправильно. Поэтому необходимо либо писать дополнительный код для правильного отображения окна, либо просто его удалить. Второй вариант проще и не обременителен для пользователя.

```
void CTVshowView::myDestroyEditChannal(void)
{
  if(m_pEditChannal)
    {delete m_pEditChannal; m_pEditChannal=0;}
}
```

Удаляем окно редактирования и в деструкторе, если пользователь закрыл программу с существующим окном.

```
CTVshowView::~CTVshowView()
{
  if(m_pEditChannal) delete m_pEditChannal;
}
```

Обработчик сообщения прокрутки LVN_BEGINSCROLL в классе CListCtrl доступен только для Windows XP и при установленном Internet Explorer 5.5 или выше. В этом случае появляется возможность использования библиотеки Comclt32.dll 6.0, которая поддерживает это сообщение. Для этого проект иметь файл манифеста, который создается автоматически при создании

проекта, если установлена соответствующая опция. Файл манифеста может быть написан и вручную. После создания манифеста необходимо в файле ресурсов TVshow.rc2 дописать такую строку.

CREATEPROCESS_MANIFEST_RESOURCE_ID RT_MANIFEST ".\\res\\TVshow.manifest"

Далее создается обработчик, который будет успешно функционировать при правильном соблюдении, указанных условий.

```
void CTVshowView::OnLvnBeginScrollListChannal(
                NMHDR *pNMHDR, LRESULT *pResult)
{
  //This feature requires Internet Explorer 5.5 or //greater.The
  symbol _WIN32_IE must be >= 0x0560.

  LPNMLVSCROLL pStateChanged =
      reinterpret_cast<LPNMLVSCROLL>(pNMHDR);

  //удалить окно редактирования при прокрутке
  myDestroyEditChannal();

  *pResult = 0;
}
```

С текстами остальных методов можно ознакомиться в исходных текстах проектов.

Завершение основного этапа построения графа

На данном этапе построения графа сделана вся необходимая предварительная работа для завершения построения графа. Ранее описывалась проблема, связанная с построением звуковой ветви графа. Эта проблема частично решается путем опережающей установки в граф звукового фильтра сбора. Для этого в классе CCapture напишем метод InitCaptureAudio, который будет устанавливать звуковой фильтр сбора в граф до установки фильтра сбора видео.

```
BOOL CCapture::InitCaptureAudio(CComPtr<IMoniker> & pMoniker)
{
  //установка фильтра звукового фильтра сбора
  if(S_OK !=
      m_pGraph2->AddSourceFilterForMoniker(pMoniker
      ,NULL,GetNameFilterFromMoniker(pMoniker).c_str()
      ,&m_pCapA))
    return FALSE;
```

```
  return TRUE;
}
```

Вызов этого метода произведем в обработчике CTVshowView:: OnNM-ClickListVideo, который приобретает вид близкий к завершению. В нем остаются закомментированными строки, которые служат для продолжения инициализации в других классах. После основной инициализации фильтра сбора в обработчике CTVshowView::OnNMClickListVideo, необходимо продолжить его инициализацию для отображения настроек фильтра на других вкладках. Инициализация определенных элементов интерфейса носит зависимый характер и должна проводиться в определенном порядке. Управлять инициализацией элементов интерфейса в классе CFView1 из класса CTVshowView означало бы существенно запутать логику программы и усложнить ее переделку и сопровождение. Значительно удобней передать управление дальнейшей инициализацией второй вкладке, послав ей пользовательское сообщение. В таком случае инициализация продолжиться в соответствующем обработчике, и мы полностью освободимся от необходимости связывать классы вкладок между собой. Сообщения посылаются в классы в порядке, который определяется логикой инициализации. Если последовательность инициализации не имеет значения или нет необходимости ожидать выполнения обработчика, вместо функции SendMessage можно использовать функцию Post Message. Такое решение оказалось проще, чем создание специальных методов для продолжения инициализации и связи объектов вкладок.

```
void CTVshowView::OnNMClickListVideo(NMHDR *pNMHDR,
                            LRESULT *pResult)
{
  //Преобразуем в указатель на структуру NMITEMACTIVATE //для
  получения  дополнительной информации

  LPNMITEMACTIVATE lpnmitem =
            reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR) ;

  //Тестируем точку click на попадание в иконку выбора //устрой-
  ства состояние иконки изменяется после выхода //из обработчика
  UINT uFlags;
  m_listVideo.HitTest(lpnmitem->ptAction,&uFlags);

  //Получаем указатель на класс фильтра сбора
  CCapture* pCap = myGetPtrCap();

  //Точка click в иконке и она не помечена
  if(uFlags & LVHT_ONITEMSTATEICON &&
            !m_listVideo.GetCheck(lpnmitem->iItem))
  {
    //Поддержка нескольких устройств не реализована
    //если устройств несколько и все работоспособны,
    //то необходимо реализовывать несколько графов
    //.............................
```

```
if(m_arIMonDev_V.IsEmpty())
  return;

//вернуть моникер из массива
DWORD_PTR idx =
m_listVideo.GetItemData(lpnmitem->iItem);

ASSERT(idx < m_arIMonDev_V.GetCount());

//Вызываем метод для инициализации построителя графа
if(S_OK != pCap->InitBuilder(m_arIMonDev_V[idx]))
{
  //Обработка устройств не прошедших проверку
  //Можно проверить на отсутствие устройства
  //hr==VFW_E_NO_CAPTURE_HARDWARE
  //и сделать сообщение

  //Удалить устройство не прошедшее инициализацию
  m_arIMonDev_V.RemoveAt(idx);
  m_arIMonDev_V.FreeExtra();

  //Удалить строку из списка
  m_listVideo.DeleteItem(lpnmitem->iItem);
  return;
}

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

//Установить звуковой фильтр сбора по умолчанию
idx = m_listAudio.GetItemData(0);

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

//Проверить существование родительского окна
//для предварительного просмотра
CTVshowApp* pApp =
        static_cast<CTVshowApp*>(AfxGetApp());
HWND hwnd = pApp->myGetHandleWndShow();
```

```
    if(!hwnd)
    {
      AfxMessageBox(
        _T("Установите окно для предварительного
                                        просмотра"));
      return;
    }

    //Инициализируем фильтр сбора видео
    if(S_OK != pCap->InitCaptureVideo(hwnd))
    {
      pCap->DestroyGraph();
      string sErr =
          "Ошибка инициализации фильтра сбора";
      CTVshowApp::myMessageError(E_FAIL,sErr);
      return;
    }

    //Инциализация переключателя видео входов
    myInitCrossBar();

    //Инициализация фильтра TV Tuner
    myInitTuner();

    //посылаем сообщение объекту CFView1 сообщение об //оконча-
    нии предварительной инициализации фильтров
    //pApp->myGetPtrView1()->SendMessage(WM_ENDINIT);

    //посылаем сообщение объекту CFView2 сообщение об //оконча-
    нии предварительной инициализации фильтров.
    //Не используется.
    //pApp->myGetPtrView2()->SendMessage(WM_ENDINIT);

  }

  //если флажок снимается граф разрушаем
  if(uFlags & LVHT_ONITEMSTATEICON &&
            m_listVideo.GetCheck(lpnmitem->iItem))
  {
    pCap->DestroyGraph();
  }

  *pResult = 0;
}
```

Далее раскомментируем создание объектов классов CCross и CTuner в методе CCapture::InitCaptureVideo. Поскольку эти объекты создаются на куче необходимо добавить строки

```
  //удалить объект переключателя
  if(m_pCrossBar1)
```

```
      { delete m_pCrossBar1; m_pCrossBar1 =0; }

  //удалить объект настройки
  if(m_pAMTunerV)
      {delete m_pAMTunerV; m_pAMTunerV = 0;}
```

для их удаления в методе CCapture::DestroyGraph, до удаления фильтров. Метод InitCaptureVideo завершен для построения основной части графа, которая производит сбор видео и звуковой информации. Код метода приведен ниже.

```
HRESULT CCapture::InitCaptureVideo(HWND hwndView)
{
  //Нет фильтра сбора в графе
  if(!m_pCapV)
    return E_FAIL;

  HRESULT hr=S_OK;
  string sErr;

  //Указатель на первый переключатель
  CComPtr<IBaseFilter> pCross1;

  //Указатель на фильтр настройки приемника видео
  CComPtr<IBaseFilter> pTunerV;

//Указатель на фильтр приемника звуковых программ
  CComPtr<IBaseFilter> pTunerA;

  try
  {
    //Указатель на первый фильтр от которого будем
    //строить ветви графа
    CComPtr<IBaseFilter> pFConnect;

    if(S_OK == AddKernelFilter(AM_KSCATEGORY_CROSSBAR
          , &pCross1,L"Conexant Crossbar"))
    {
      //запрашиваем указатель на интерфейс переключателя
      CComQIPtr<IAMCrossbar> pAMCrossbar(pCross1);

      if(pAMCrossbar)
      {
        pFConnect = pCross1;
        m_pCrossBar1 =
              CCross::MakeCrossbar(pAMCrossbar);

        //далее следует поискать второй переключатель
        //и обработать ситуацию
        //CComPtr<IAMCrossbar> pCrosss2;
      }
    }
```

```
    //поиск и установка фильтра приемника звуковых
    //программ
    hr =AddKernelFilter(AM_KSCATEGORY_TVAUDIO
          , &pTunerA,L"Conexant TvAudio");

    //установка фильтра настройки ТВ приемника
    if(S_OK == AddKernelFilter(AM_KSCATEGORY_TVTUNER
          , &pTunerV,L"Conexant TvTuner"))
    {
      //запрашиваем указатель на интерфейс приемника
      CComQIPtr<IAMTVTuner> pAMTVTuner(pTunerV);

      if(pAMTVTuner)
      {
        pFConnect = pTunerV;
        //создаем объект класса CTuner (фильтр настройки
  //приемника)
        m_pAMTunerV = CTuner::MakeTuner(pAMTVTuner);
      }
    }

  //Соединение фильтров графа, начиная с последнего      //уста-
новленного фильтра и до фильтра сбора
  //включительно. Возможно, что не подключение
  //одного из контактов не нарушит работоспособности   графа. //
  Поэтому только сообщаем.
    if(pFConnect)
    {
      hr = RenderAllPins(pFConnect);

      if(hr!= S_OK)
      {
        sErr = "Невозможно произвести соединение
                                          фильтров";
        CTVshowApp::myMessageError(hr,sErr);
      }
    }

  }
  catch(...)
  {
    DestroyGraph();
    sErr = "Неизвестное исключение (InitCaptureVideo)";
    CTVshowApp::myMessageError(hr= E_FAIL,sErr);
  }
  return hr;
}
```

После запуска приложения, проводим проверку построения графа с помощью утилиты графического редактора. Результат преставлен на рисунке 3.8.

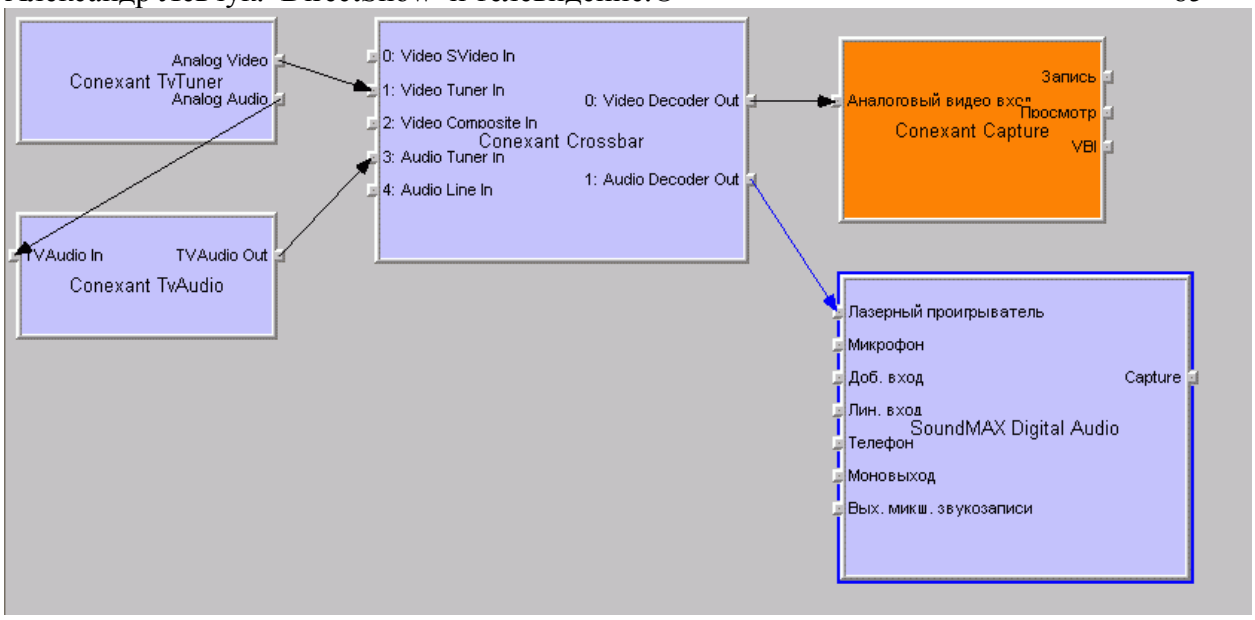

Рис.3. 10 Граф сбора

Инициализация фильтра сбора

Первая вкладка (класс CTVshowView) заполнялась оперативными элементами интерфейса, к которым пользователь может обращаться наиболее часто. На второй вкладке (класс CFView1) разместим элементы интерфейса, предназначенные для настроек и регулировок фильтра сбора. Вся инициализация фильтра сбора будет производиться в обработчике CFView1::OnMyMessageEndInit, который обрабатывает пользовательское сообщение, отправленное из класса первой вкладки. В инициализации некоторое участие также будет принимать метод OnInitialUpdate.

Фильтр сбора аналогового телевидения может поддерживать несколько интерфейсов, которые управляют различными регулировками:

1. IAMAnalogVideoDecoder. Управляет процессом аналого-цифрового преобразования. Для приложения представляет интерес переключение стандартов ТВ, и возможно отображение сбора строчной синхронизации. Последний параметр косвенно отображается при настройке на канал (состояние настройки).
2. IAMVideoProcAmp. Управляет регулировками изображения, такими как яркость, контрастность и другие.
3. IAMDroppedFrames. Возвращает информацию о производительности фильтра сбора в процессе работы. Возвращает число кадров изображения, удаленных из обработки и некоторые другие параметры. Позволяет сделать вывод о производительности компьютера в целом при сборе видео данных.

4. IAMVideoCompression. Интерфейс, который поддерживается некоторыми программными фильтрами сжатия и фильтрами сбора, имеющими аппаратную реализацию сжатия данных
5. IAMVideoControl Управление конечным изображением, переключает изображение зеркально, вертикально, горизонтально. Управляет внешним переключателем потока данных. Устройство должно поддерживать, указанные возможности.

Из перечисленных интерфейсов первые три необходимы в той или иной мере пользователю. Интерфейс IAMVideoCompression предоставляют незначительное число устройств, и он больше распространен в устройствах для профессионального применения. Применение данного интерфейса будет рассмотрено для программных фильтров сжатия. Последний интерфейс применим в основном для специальных эффектов и при управлении видео камерой, он не будет рассматриваться.

Таким образом, для первичной инициализации остается интерфейс IAMAnalogVideoDecoder, который требуется для согласования ТВ стандартов приемника и фильтра сбора. Ранее при рассмотрении инициализации приемника этот вопрос затрагивался. Остальные интерфейсы используются в процессе отображения и будут рассмотрены в соответствующих разделах.

Следует напомнить, что стандарт ТВ используется в двух фильтрах, фильтре настройки приемника и фильтре сбора. Фильтр настройки приемника позволяет только определить доступные стандарты (IAMTVTuner) с которыми может работать и возвратить стандарт установленный для страны пребывания, но не позволяет их устанавливать.

Фильтр сбора позволяет производить все операции со стандартами. Таким образом, возможно, что некоторые устройства могут производить перекодировку из одного стандарта в другой при просмотре и записи файла. Используемое устройство не имеет такой возможности и, следовательно, необходимо согласовывать стандарт фильтра приемника и фильтра сбора для обеспечения правильного отображения.

Интерфейс IAMAnalogVideoDecoder

Интерфейс позволяет управлять некоторыми свойствами аналого-цифрового преобразования фильтра сбора. Некоторые методы интерфейса могут не реализовываться устройством, для этих случаев предусмотрен код возврата равный константе E_PROP_ID_UNSUPPORTED. Такие методы для испытуемого устройства отмечены в таблице знаком звездочка (*).

Интерфейс IAMAnalogVideoDecoder

Методы	Описание
get_AvailableTVFormats	Возвращает стандарты ТВ поддерживаемые де-

	кодером.
*get_HorizontalLocked	Определяет, произошел ли захват горизонтальной синхронизации.
*get_NumberOfLines	Возвращает число строк в видеосигнале.
*get_OutputEnable	Определяет, разрешена ли шина видеопорта.
get_TVFormat	Возвращает текущий видеоформат сигнала.
*get_VCRHorizontalLocking	Возвращает источник сигнала, видеомагнитофон или вещательный источник. Определение производится по точности сохранения временных соотношений, которые у видеомагнитофона ниже.
*put_OutputEnable	Управляет шиной видеопорта.
put_TVFormat	Устанавливает видео формат.
*put_VCRHorizontalLocking	Устанавливает источник сигнала, лента или вещательный источник. Указание позволяет видеодекодеру понизить требования к стабильности временных соотношений в сигнале.

Таблица 6 Интерфейс IAMAnalogVideoDecoder

Методы для работы с ТВ стандартами

Для управления установкой ТВ стандартов напишем в классе фильтра сбора несколько методов. Первый метод будет определять доступные для использования стандарты. Он будет заполнять вектор строк с именами стандартов и вектор констант, определяющих стандарты.

```
BOOL CCapture::GetVectorAvailableTVStandards(
                 vector<string>& vStr, vector<long>& vSt)
{
  HRESULT hr = S_OK;

  //Стандарты ТВ доступные и текущий
  LONG stAvailable = 0,stCurrent=0;

  //Получить указатель на интерфейс видеодекодера
  //фильтра сбора
  CComQIPtr<IAMAnalogVideoDecoder> pVD(m_pCapV);
  if(!pVD) return FALSE;

  //Определить все доступные стандарты для фильтра
  //сбора
  hr = pVD->get_AvailableTVFormats(&stAvailable);
  if(S_OK != hr)
    return FALSE;
```

```
  //Определить текущий стандарт фильтра сбора
  hr = pVD->get_TVFormat(&stCurrent);
  if(S_OK != hr)
    return FALSE;

  //Заполнить вектор имен  и вектор констант ТВ
  //стандартов
  LONG mask=1; //Маска (стандарт AnalogVideo_NTSC_M)
  LONG tmp;

  //Определить имена стандартов и их константы
  for(int i=0; i<21; ++i)
  {
    tmp = mask & stAvailable;

    if(tmp)
    {
      vStr.push_back(GetNameStandartTV(tmp));
      vSt.push_back(mask);
    }
    mask <<= 1;
  }

  //Определить текущий стандарт и передвинуть его
  //на нулевую позицию в векторах
  string s = GetNameStandartTV(stCurrent);

  rotate(vStr.begin(),find(vStr.begin()
                       ,vStr.end(),s),vStr.end());

  rotate(vSt.begin(),find(vSt.begin()
                       ,vSt.end(),stCurrent),vSt.end());

  return TRUE;

}
```

Метод GetNameStandartTV и определение имен стандартов можно посмотреть в файле Constants.h. Второй метод CCapture:: GetTVStandard возвращает текущий ТВ стандарт фильтра сбора.

```
LONG CCapture::GetTVStandard(void)
{
  LONG standard =0;

  CComQIPtr<IAMAnalogVideoDecoder> pVD(m_pCapV);
  if(!pVD)
    return standard;

  if(S_OK == pVD->get_TVFormat(&standard))
    return standard;
```

```
  return standard;
}
```

Третий метод устанавливает ТВ стандарт для фильтра сбора.

```
BOOL CCapture::SetTVStandard(AnalogVideoStandard standard)
{
  CComQIPtr<IAMAnalogVideoDecoder> pVD(m_pCapV);

  if(!pVD)
    return FALSE;

  if(pVD->put_TVFormat(standard) == S_OK)
    return TRUE;

  return FALSE;
}
```

Этих методов достаточно для управления установкой и отображения текущего стандарта ТВ фильтра сбора.

Интерфейс пользователя

Для отображения и установки стандарта ТВ на фильтре сбора на второй вкладке установим элемент управления CComboBox и создадим обработчик CFView1::OnCbnSelchangeComboFormatVd. При установке элемента управления в его свойствах следует отменить сортировку. Первоначальное заполнение элемента управления и установка текущего ТВ стандарта производится в обработчике CFView1::OnMyMessageEndInit.

На этом этапе разработки проекта можно считать завершенной основную инициализацию фильтров. Проект в этой стадии находится в папке Project-State4. Далее приступим к построению графа для осуществления предварительного просмотра.