Глава 2
Построение фильтрового графа DirectShow

Глава 2
Построение фильтрового графа DirectShow 1
Подготовка проекта, выбор шаблона проекта.....1
Доработка шаблона проекта.....4
Смена представлений для одного документа.....12
Совместный старт двух окон.....15
Переключение нескольких представлений.....18
Установка DirectX 9.0 SDK, внесение изменений в проект.....28
Построение фильтрового графа DirectShow. 30
Класс фильтрового графа.....30
Реализация менеджера фильтрового графа.....32
Интерфейсы менеджера фильтрового графа.....32
Методы интерфейсов менеджера графа и построителя фильтра сбора 36
Методы интерфейса IFilterGraph 37
Методы интерфейса IGraphBuilder 38
Методы интерфейса IFilterGraph2 38
Методы интерфейса ICaptureGraphBuilder2 39
Создание менеджера графа и построителя графа сбора.....40
Системный перечислитель и моникеры 46
Перечисление устройств и фильтров 48
Реализация перечислителя устройств.....49
Реализация перечислителя фильтров (кодеков).....57
Реализация поиска в карте фильтров.....60
Установка в граф фильтра сбора.....62
Разрушение графа.....64
Просмотр результата в редакторе графа.....66

Подготовка проекта, выбор шаблона проекта

Реализация приложения DirectShow, как и любого другого на начальном этапе требует тщательного выбора шаблона проекта. Неправильно выбранный шаблон проекта повлечет за собой много переделок вплоть до его полной замены и потребует значительных трудозатрат.

Обычно при выборе шаблона проекта, прежде всего, руководствуются требованиями заказчика, особенностями аппаратной и программной реализации устройства и возможностями предоставляемой средой разработки. В данном случае, проект создается для изучения DirectShow, а также для проверки и отладки решений в процессе изучения. Поэтому, он должен легко перестраиваться и позволять создавать практически автономные блоки программы, каждый из которых предназначен для решения определенных задач, связанных единым замыслом. Устройство, для которого создается программа, является аналоговым телевизионным приемником, что сразу же предполагает большое количество разнообразных настроек и регулировок телевизионного приемника, которые необходимо будет отображать, как элементы интерфейса пользователя. Очевидно, что регулировки приемника по своему назначению будут делиться на группы. Для вывода изображения необходимо окно, отдельное от окон управления функциями приемника.

Эти главные требования наводят на мысль, что потребуется окно с переключающимися вкладками диалогов, где будут размещены органы управления, сгруппированные по назначению и отдельное, свободно плавающее, окно для вывода изображения. Окно диалога с вкладками должно допускать легкое расширение числа вкладок, а также изменение их размеров окна и прокрутку, поскольку экспериментальный характер программы не позволяет заранее определить границы диалога.

Посмотрим, что предлагает среда разработки для минимизации трудозатрат. Автор использует две версии Microsoft Visual Studio, VC 6.0 и .NET 2003 (VC 7.0), которые не конфликтуют на одном компьютере. Однозначно, шаблоны MDI, SDI и шаблон, основанный на диалоге, не подходят для структуры нашего приложения, поскольку потребуют дополнительных усилий по созданию отдельного окна для изображения, а большего VC 6.0 не предлагает. VC 7.0 предлагает нам более широкие возможности, такие как например, multi top-level documents (Рис.2. 1). Картинка в правом верхнем углу мастера подсказывает нам, что это, возможно, то, что нужно. Во всяком случае, есть основное окно и несколько окон верхнего уровня. Если изменить начальный шаблон таким образом, чтобы отображалось одно окно верхнего уровня, а в основном окне располагались переключаемые диалоги, то такой шаблон соответствовал бы сформулированным требованиям. Существование нескольких окон отображения верхнего уровня также может оказаться полезной воз-

можностью. Попытаемся произвести задуманные изменения типового шаблона проекта.

Дадим имя проекту TVshow, включим поддержку архитектуры Document/View и в качестве основного класса выберем CFormView для размещения элементов управления приемником. Архитектура Document/View поможет упростить сохранение и автоматическое восстановление настроек приемника и программы, что для пользователя немаловажно при эксплуатации устройства с большим количеством настроек. Также включим поддержку HTML Help, которая значительно упрощает создание справки. Проект находится в папке ProjectState1.

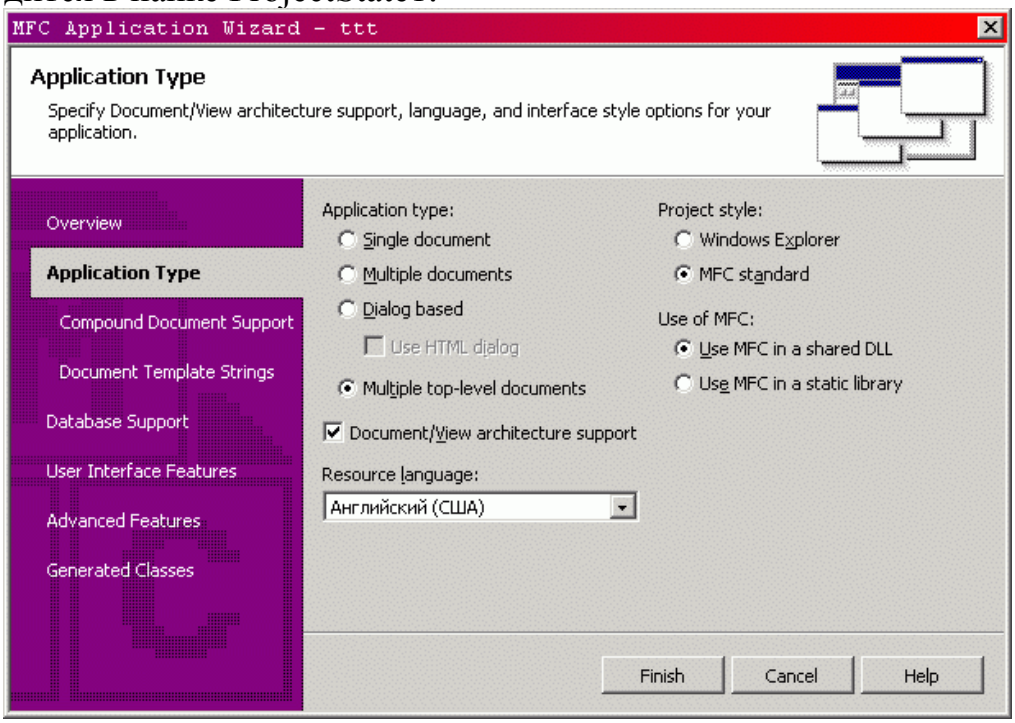


Рис.2. 1 Выбор типа приложения

Нажимаем OK, затем компилируем проект, запускаем его на выполнение и смотрим, что создала среда разработки. А создала она небольшое окно, заполненное формой класса CFormView.

Если в меню File выбрать пункт New Frame, то будет создано еще одно точно такое же, свободно плавающее окно.

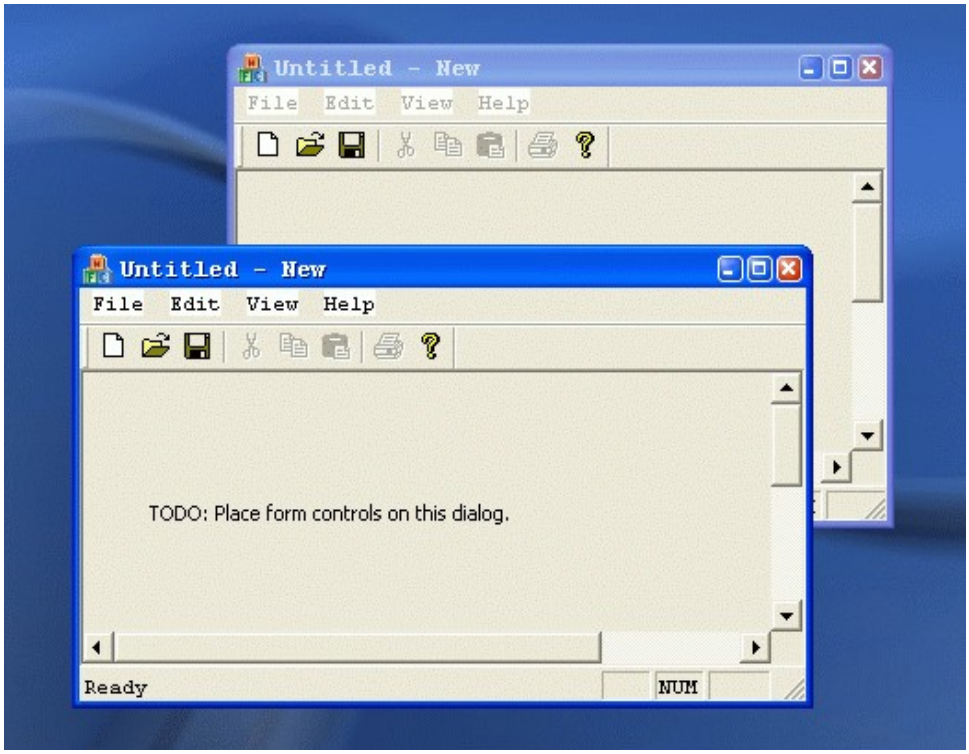


Рис.2. 2 Первичный вид окна приложения

Для наших целей этот шаблон уже неплох, окно для размещения элементов управления имеет полосы прокрутки, допускает изменение размеров. Результат показан на рисунке 2.2.

Однако это все же не совсем то, что нужно. Шаблон проекта нуждается в существенной доработке. Его следует изменить для полного соответствия следующим требованиям:

- При загрузке программы на выполнение должно формироваться только два окна. Одно для управления и второе для отображения.
- Окно управления должно содержать несколько сменяемых представлений класса CFormView с органами управления. Их количество должно легко изменяться.
- Окно отображения должно содержать представление класса CView для телевизионной картинки.

Таким образом, необходимо несколько окон класса CFormView разместить в одном основном окне в виде переключаемых вкладок диалогов, а окно верхнего уровня должно быть только одно и класса CView, для отображения фильма.

Доработка шаблона проекта

Первое действие это корректировка файла stdafx.h. Заменяем значения некоторых строк, определяющих использование дополнительных возможностей предоставляемых Windows 2000/XP. Если этого не сделать, то компилятор

при использовании некоторых идентификаторов, констант и т.д. будет выдавать ошибку. Новые значения выделены в тексте жирным шрифтом.

```
#ifndef WINVER    //allow use of features specific to
    //Windows 95 and Windows NT 4 or later.

#define WINVER 0x0501 //Change this to the appropriate
                        //  value to target Windows 98
                        //and Windows 2000 or later.
#endif

#ifndef _WIN32_WINNT    //Allow use of features specific //to
                        Windows NT 4 or   later.

#define _WIN32_WINNT 0x0501   //Change this to the
                      //appropriate value to target
                      //Windows 98 and Windows 2000 or //later.
#endif

#ifndef _WIN32_WINDOWS  //Allow use of features //specific to
               Windows 98 or later.

#define _WIN32_WINDOWS 0x0500 //Change this to the
                      //appropriate value to
                      //target  Windows Me or later.
#endif

#ifndef _WIN32_IE   //Allow use of features specific
                      //to IE 4.0 or later.

#define _WIN32_IE 0x0600    //Change this to the
                              //appropriate value to
                              //target IE 5.0 or later.
#endif
```

Второе действие корректировка файлов класса CTVshowApp. Здесь попытаемся ввести новый шаблон документа для окна вывода изображения класса CView, поскольку его нет вообще, а также введем еще одно представление на основе CFormView и осуществим переключение между основным представлением CFormView и дополнительным представлением CFormView. Отработав эти основные решения в дальнейшем, можно будет просто добавлять новые представления класса CFormView.

Итак, добавляем новый шаблон документа для окна вывода изображения класса CView. Для этого посмотрим на строки в функции InitInstance, где создается шаблон документа для главного окна. Здесь мы видим, что все окна создаются по одному шаблону.

```
  CMultiDocTemplate* pDocTemplate;
```

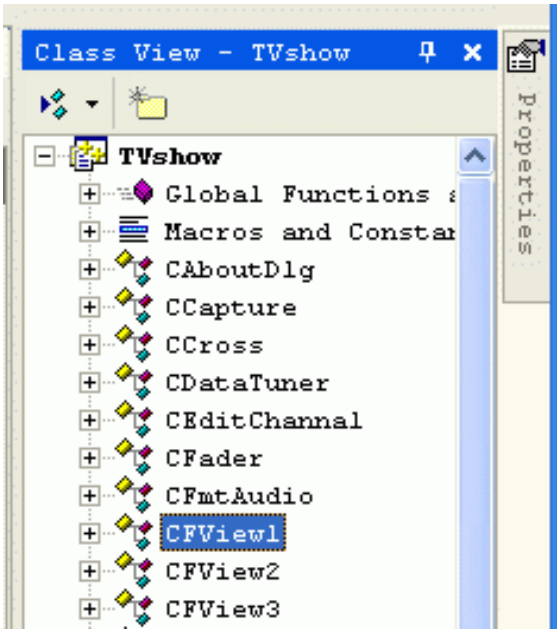
```
pDocTemplate = new CMultiDocTemplate(
  IDR_MAINFRAME,
  RUNTIME_CLASS(CTVshowDoc),
  RUNTIME_CLASS(CMainFrame),   //Main SDI frame
                                   //window
  RUNTIME_CLASS(CTVshowView));
if (!pDocTemplate)
  return FALSE;
m_pDocTemplate = pDocTemplate;
AddDocTemplate(pDocTemplate);
```

Добавляем новый шаблон. Класс документа в новом шаблоне оставим прежний CTVshowDoc, а классы рамки и представления должны быть другие. Поэтому создадим два новых класса один наследуемый от CFrameWnd и второй, наследуемый от CView. В таблицу строк добавим новую строку с произвольным идентификатором, например, IDR_WNDSHOW и текстом \nShow\nShow. Классы назовем CShowFrame и CShowView.

Процесс добавления нового класса в VC 7.0 несколько отличается от такого же действия в VC 6.0, здесь отсутствует ClassWizard и WizardBar. Все подробности можно найти в MSDN в разделе Where Are ClassWizard and WizardBar in Visual C++ .NET?. Коротко это действие выглядит так. В окне ClassView или другом, где есть имя проекта, сделайте правый щелчок на имени проекта и в открывшемся меню выберите пункт ADD/Add Class. В открывшемся окне выбираете класс MFC и нажимаете OK, после этого откроется окно мастера ClassWizard. Далее действуйте обычным образом.

После добавления этих классов в проекте появляются два новых файла класс CShowFrame и класс CShowView. Файлы класса CShowView пока не изменяются, а в классе CShowFrame следует перегрузить функцию Create. Это можно сделать и вручную, но с использованием оболочки IDE будет быстрее. Возможно, вы не еще использовали Visual C++ .NET и для ускорения процесса обучения поясню это, не совсем очевидное, действие.

В главном меню откройте окно свойств View/Properties Window и сделайте его скрывающимся автоматически. Для этого на заголовке окна щелкните по значку, или правым щелчком на заголовке окна откройте меню и установите режим Auto Hide. После этого окно минимизируется, и вы должны увидеть картинку, показанную на рис 3.



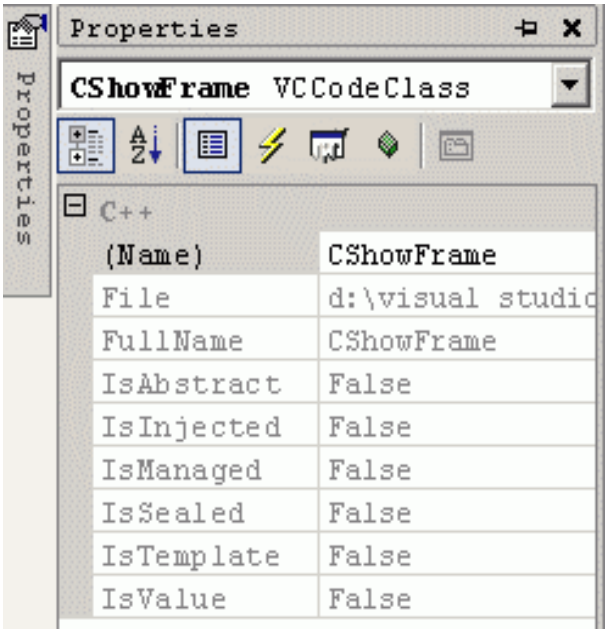
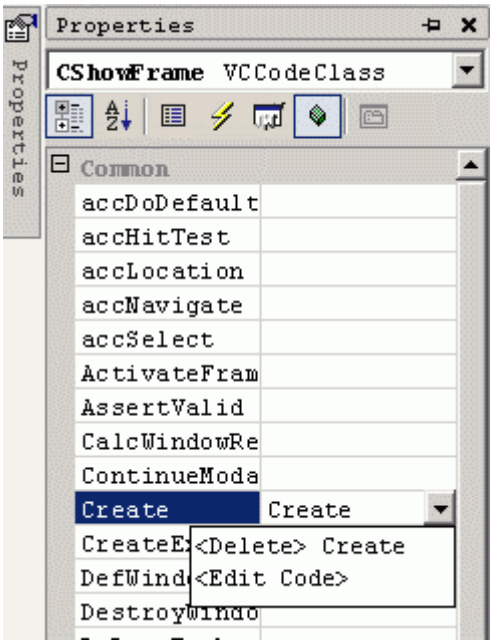


Рис.2. 3 Скрытое окно свойств

Рис.2. 4 Окно свойств класса

Затем выделяем класс CShowFrame и просто переводим курсор на панель Properties. Появляется окно свойств этого класса (рис 4). В данном окне три кнопки с символами, в соответствии с их порядком расположения, открывают окна, отображающие для данного класса события, сообщения и перегружаемые функции. После нажатия на зеленый кубик откроется окно, в котором выбираем функцию Create (рис. 2.5).



Предлагаемая последовательность действий самый короткий и быстрый путь для установки обработчиков событий, сообщений и перегружаемых функций. Эти действия также можно выполнить, используя горячие клавиши.

В созданной функции Create модернизируем вызов функции базового класса. Устанавливаем свое имя окна, удаляем меню и расширенный стиль окна, поскольку для окна отображения телевизионного изображения эти элементы не нужны.

Рис.2. 5 Выбор перегружаемой функции

```
BOOL CShowFrame::Create(LPCTSTR lpszClassName
      , LPCTSTR lpszWindowName
      , DWORD dwStyle, const RECT& rect
      , CWnd* pParentWnd, LPCTSTR lpszMenuName
      , DWORD dwExStyle , CCreateContext* pContext)
{
  return CFrameWnd::Create(lpszClassName, _T("TVshow")
      , dwStyle,rect, pParentWnd, NULL, 0, pContext);
}
```

Вернемся к покинутому файлу TVshow.cpp и продолжим создание нового шаблона для окна изображения. Прежде всего, добавляем в начале файла две строки, включающие заголовочные файлы новых классов

```
#include ".\showView.h"
#include ".\showFrame.h".
```

Затем добавляем указатель на шаблон окна отображения CMultiDocTemplate* m_pDocTemplateShow, который понадобится для идентификации шаблона. Если это сделать с помощью соответствующего мастера, то указатель будет инициализирован нулем в конструкторе автоматически, что позволит избежать возможных проблем, если при добавлении вручную этого не было сделано. Далее добавляем новый шаблон документа.

```
  m_pDocTemplateShow = new CMultiDocTemplate(
    IDR_WNDSHOW,
    RUNTIME_CLASS(CTVshowDoc),
    RUNTIME_CLASS(CShowFrame),
    RUNTIME_CLASS(CShowView));

  if (!m_pDocTemplateShow)
    return FALSE;
  AddDocTemplate(m_pDocTemplateShow);
```

Если откомпилировать проект в этом состоянии и выполнить, то мы увидим окно, маловразумительное для пользователя, не являющего программистом (рис. 2.6.).

Естественно, что от такого старта программы следует немедленно отказаться. Продолжим модернизацию проекта в этом направлении. Для этого внимательно изучим исходные тексты для обнаружения логики старта программы.

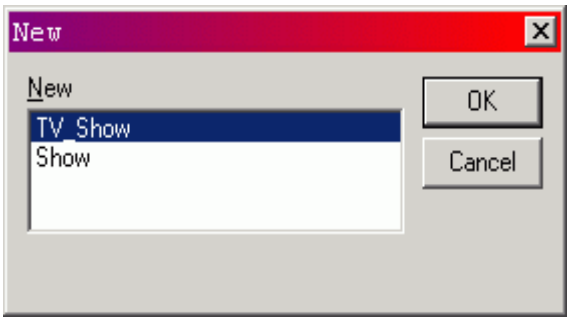


Рис.2. 6 Стартовое окно программы

Посмотрим внимательней на два основных файла, участвующих в старте программы, TVshow.cpp и MainFrm.cpp. Первый содержит две функции OnFileNewFrame и OnFileNew, второй LoadFrame, которые явно должны принимать участие в старте программы, загрузке нового окна и создании нового документа. Для определения последовательности их вызовов установим точки останова внутри функций и запустим программу в режиме отладки. После непродолжительных экспериментов обнаруживаем следующую логику работы.

- Первой вызывается функция OnFileNew() и, запустив на выполнение программу, видим окно, такое как на рис 6.
- Нажимаем ОК в этом окне и попадаем в LoadFrame. Запускаем опять программу и видим основное окно программы.
- Открываем меню File/New Frame и попадаем в OnFileNewFrame. Запустив программу, далее видим, второе такое же окно.

Логика работы ясна, далее изменим ее таким образом, чтобы при вызове функции OnFileNew загружался шаблон главного окна, в котором будут располагаться элементы управления телеприемником, а при выборе меню File/New Frame происходила загрузка окна изображения на основе второго шаблона.

В функции OnFileNew есть фрагмент кода, в котором вызывается функция базового класса CWinApp, формирующая первое окно (рис.6).

```
if (pFrame == NULL || pDoc == NULL)
{
  //If it's the first document, create as normal
  CWinApp::OnFileNew();
}
```

Вместо этого вызова напишем свою функцию, которая в зависимости от типа шаблона будет формировать соответствующее окно. Изобретать, что-либо новое не придется, для этой цели почти без изменений подойдет содержимое функции OnFileNewFrame. Назовем нашу функцию myNewFrame. Очень удобно в именах функций собственной разработки устанавливать префикс my, или любой другой. В этом случае в окне IntelliSense с длинным списком всех функций легко отыскать собственные функции, набрав только префикс (рис 7).

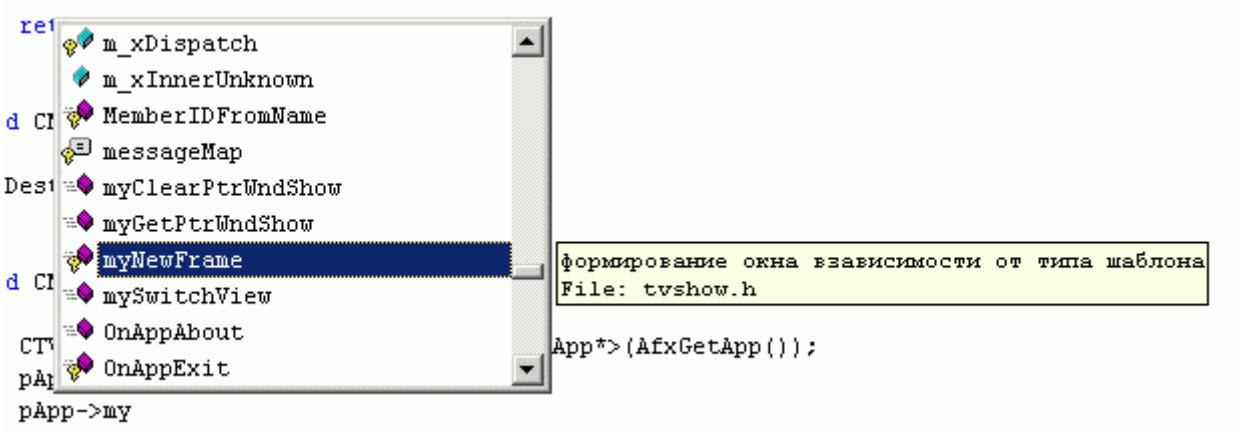


Рис.2. 7 Отображение собственных методов в окне IntelliSense

Ниже приводится текст измененного метода CTVshowApp::myNewFrame. Единственное существенное дополнение в этом методе это запоминание в классе CTVshowApp указателя на окно изображения для управления окном. Указатель объявлен защищенным и для его вызова объявлена функция myGetPtrWndShow.

```
CFrameWnd* myGetPtrWndShow(void)
  { return m_pFrameShow; };

void CTVshowApp::myNewFrame(
              CMultiDocTemplate* pDocTemplate)
{
  ASSERT(pDocTemplate != NULL);
  CDocument* pDoc = NULL;
  CFrameWnd* pFrame = NULL;

  //Создаем новый документ для шаблона
  pDoc = pDocTemplate->CreateNewDocument();
  if (pDoc != NULL)
  {
    //if creation worked, use create a new frame for
   //that   document.
    pFrame = pDocTemplate->CreateNewFrame(pDoc, NULL);
    if (pFrame != NULL)
    {
      //Set the title, and initialize the document.
      //If document initialization fails, clean-up the
  //frame window and document.
      pDocTemplate->SetDefaultTitle(pDoc);
      if (!pDoc->OnNewDocument())
      {
        pFrame->DestroyWindow();
        pFrame = NULL;
      }
      else
      {//Otherwise, update the frame
```

```
        //если это шаблон окна отображения сохраняем
  //его указатель для контроля и управления
        if(pDocTemplate == m_pDocTemplateShow)
          m_pFrameShow = pFrame;

        pDocTemplate->InitialUpdateFrame(pFrame,
      pDoc, TRUE);
      }
    }
  }
  //If we failed, clean up the document and show a
  //message to the user.
  if (pFrame == NULL || pDoc == NULL)
  {
    delete pDoc;
    AfxMessageBox(AFX_IDP_FAILED_TO_CREATE_DOC);
  }
}
```

Текст OnFileNew после модернизации будет выглядеть следующим образом.

```
void CTVshowApp::OnFileNew()
{
  CDocument* pDoc = NULL;
  CFrameWnd* pFrame;
  pFrame =
  DYNAMIC_DOWNCAST(CFrameWnd,CWnd::GetActiveWindow());

  if (pFrame != NULL)
    pDoc = pFrame->GetActiveDocument();

  if (pFrame == NULL || pDoc == NULL)
  {
    //Создаем главное окно с первым шаблоном при
    //первом запуске программы
    myNewFrame(m_pDocTemplate);

      //Не вызываем CWinApp::OnFileNew();  и тем самым
      //не отображаем окно выбора шаблона документа
  }

     //Далее удаляем все, чтобы не создавать новый
     //документ, который нам не нужен
}
```

Теперь при запуске программы будет формироваться главное окно со своим документом. Формирование нового документа при выборе меню File/New не нужно, документ будет единственный, поэтому удаляем этот пункт меню и соответствующую иконку из Toolbar. Осталось откорректировать функцию OnFileNewFrame, таким образом, чтобы при вызове меню File/

New Frame формировалось только окно изображения. Для этого просто удаляем все тело функции и делаем вызов myNewFrame с указателем на второй шаблон.

```
if(!m_pFrameShow)
  myNewFrame(m_pDocTemplateShow);
```

Не забываем в конструкторе, инициализировать m_pFrameShow нулем, если указатель добавлялся вручную. Одной проверки указателя на нуль недостаточно. Пользователь может закрыть окно, и затем опять его открыть. Следует предусмотреть обнуление указателя при закрытии окна изображения, иначе окно не будет формироваться. Для этого, добавим в классе CShowFrame обработчик сообщения WM_CLOSE, и в нем очистим указатель.

```
(static_cast<CTVshowApp*>(AfxGetApp()))-> myClearPtrWndShow();
```

Возможно, пользователю не следует предоставлять возможность закрытия и открытия окна изображения, но на данном этапе разработки это еще неясно. Если в дальнейшем появятся предпосылки для такого решения, то можно будет заблокировать открытие и закрытие окна пользователем и открывать и закрывать его вместе с главным окном.

После компиляции программы ее поведение уже соответствует нашим основным требованиям. Теперь необходимо реализовать в основном окне смену представлений, которые будут содержать органы управления телеприемником.

Смена представлений для одного документа

Задача состоит в том, чтобы для одного окна с одним документом создать несколько переключающихся представлений. То есть необходимо реализовать SDI с несколькими представлениями. В MSDN в разделе Adding Multiple Views to a Single Document демонстрируется техника смены представлений для одиночного документа. Однако, как это часто бывает, приведенный способ работает частично. Новое представление класса CFormView отображается неправильно, пример требует корректировки. В таких случаях поиск правильного решения может затянуться на достаточно длительное время. Возможно, в новой редакции ошибки будут исправлены, но как показывает практика, это маловероятно. Проще найти пример на данную тему и реализовать задачу. Всегда, в проблемных случаях следует начинать с поиска примера в MSDN, а не статьи на данную тему. Такой подход экономит много времени. В данном случае, используя модернизированный способ смены представлений, реализованный в примере Collect и сведения из статьи, удалось быстро и просто решить поставленную задачу.

Предварительно создаем еще один класс для второго представления под именем CFView1, производный от CFormView. Включаем его файл заголов-

ков #include ".\fview1.h" в файле TVshow.cpp. Для добавления нового представления в функции InitInstance после строк

```
if (!ProcessShellCommand(cmdInfo))
  return FALSE;
```

добавляется код, который создает новое представление с помощью вызова метода CFrameWnd::CreateView.

```
//Сохраняем указатель на основное (первое)
//представление
m_pMainView =
(static_cast<CFrameWnd*>(m_pMainWnd)->GetActiveView());

//Возвращаем указатель класса времени выполнения(run //time) для
нового представления класса CFView1
//из структуры CRuntimeClass
CRuntimeClass* pNewView = RUNTIME_CLASS(CFView1);

//Инициализируем новый контекст, указывающий на
//активный (существующий) документ. В этом контексте
//новое представление будет добавляться к документу
//после его создания

CCreateContext newContext;
newContext.m_pNewViewClass = pNewView;

//Для всех новых представления указываем один и
//тот же текущий документ
newContext.m_pCurrentDoc = (static_cast<CFrameWnd*> (m_pMain-
Wnd))->GetActiveDocument();

//Остальные члены структуры очищаем
newContext.m_pNewDocTemplate = NULL;
newContext.m_pLastView = NULL;
newContext.m_pCurrentFrame = NULL;

//создается новое представление с новым контекстом и //выполня-
ется его инициализация
m_pFView1 =
STATIC_DOWNCAST(CView ,(static_cast<CFrameWnd*>
          (m_pMainWnd))->CreateView(&newContext));

m_pFView1->OnInitialUpdate();
```

В этом коде используется, достаточно редко применяемая структура времени выполнения CRuntimeClass. Каждый класс, произведенный из CObject, содержит информацию времени выполнения о себе и базовом классе. Данная структура используется для дополнительной проверки типа объекта во время выполнения программы, или когда нужно написать специализированный код,

привязанный к классу объекта. В языке C++ отсутствует прямая поддержка информации времени выполнения. Дополнительная информация на эту тему содержится в разделе MSDN Accessing Run-Time Class Information. Также достаточно редко встречается и прямое использование структуры CCreateContext. Данная структура используется при создании рамки окна, представления и их связи с документом. Обычно данные этой структуры доступны в перегруженных функциях Create.

Для проверки функционирования кода требуется написать функцию, которая будет переключать два представления в главном окне. Она послужит в дальнейшем основой для реализации переключения нескольких представлений. За основу возьмем код, приведенный в разделе Adding Multiple Views to a Single Document. Принцип переключения представлений заключается в вызове виртуальной функции CFrameWnd::RecalcLayout. Функция RecalcLayout в свою очередь, вызывает функцию CFrameWnd::RepositionBars. Последняя функция, выполняет перерасчет клиентской области окна. Для правильной работы функции RecalcLayout требуется, чтобы идентификатор представления был равен константе AFX_IDW_PANE_FIRST, то есть всегда должно пересчитываться первое (видимое) представление. В результате, каждый раз при смене представления необходимо производить смену идентификаторов представлений. Для двух представлений реализация переключения достаточно простая.

```
CView* CTVshowApp::mySwitchView(CView* pView)
{
  CView* pOldView =(static_cast<CFrameWnd*>
 (m_pMainWnd))-> GetActiveView();

  //Активному представлению устанавливаем идентификатор
  //нового представления, а новому представлению
  //устанавливаем значение первого
  представления //AFX_IDW_PANE_FIRST, то есть, делаем обмен.
  //Это необходимо для правильной  работы функции
  //RecalcLayout при пересчете размеров представления

  //Видимое представление становится старым, его
  //идентификатор должен быть AFX_IDW_PANE_FIRST
  UINT idOld =
  ::GetWindowLong(pOldView->m_hWnd, GWL_ID);

  //Идентификатор представления, которое станет видимым
  UINT idNew = ::GetWindowLong(pView->m_hWnd, GWL_ID);

  //Производим обмен идентификаторов
  ::SetWindowLong(pOldView->m_hWnd, GWL_ID, idNew);
  ::SetWindowLong(pView->m_hWnd,GWL_ID,
                          AFX_IDW_PANE_FIRST);
  pOldView->ShowWindow(SW_HIDE);
  pView->ShowWindow(SW_SHOW);
```

```
  //Активизируем новое представление
 (static_cast<CFrameWnd*>
 (m_pMainWnd))->SetActiveView(pView);

  //Производим перерасчет представления,
  //ставшего видимым
  (static_cast<CFrameWnd*>
  (m_pMainWnd))->RecalcLayout();

  return pOldView;
}
```

Для проверки работы метода в меню View добавим временный пункт Switch, сделаем для него обработчик в классе CMainFrame и вызовем функцию mySwitchView. Предварительно в редакторе ресурсов пометим одну из форм своим текстом, чтобы их можно было различить.

```
void CMainFrame::OnViewSwitch()
{
  CTVshowApp* pApp = static_cast<CTVshowApp*>(AfxGetApp());

  pApp->mySwitchView(pApp->m_pFView1);
}
```

Проверка проходит нормально и можно считать, что основные решения отработаны. Недостатком является то, что расположение окон произвольно на рабочем столе, а также то, что после старта программы требуется дополнительно вызывать окно изображения. Желательно, показать основное окно в нижней части экрана, над ним окно изображения с пропорциями соответствующими пропорциям телевизионного экран. В таком виде окна программы будут более функциональны. Доработаем программу в этом направлении.

Совместный старт двух окон

Реализуем совместное появление обеих окон. Простейшим вариантом является вызов обработчика OnFileNewFrame()из класса CTVshowApp, который можно поместить в функции Create класса CMainFrame. Например, как в приведенных строках ниже.

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  ...
  ...
  DockControlBar(&m_wndToolBar);

  //Установить окно изображения
  (static_cast<CTVshowApp*>
  (AfxGetApp()))->OnFileNewFrame();
```

```
  return 0;
}
```

Код будет работать, но нам нужно будет установить новые размеры окна изображения, что обычно делается в перегруженной функции PreCreateWindow соответствующего класса. Произведем перегрузку PreCreateWindow в классе CShowFrame и напишем код, осуществляющий связанное размещение окон.

```
BOOL CShowFrame::PreCreateWindow(CREATESTRUCT& cs)
{
  CMainFrame* pMain =
  static_cast<CMainFrame*>(AfxGetMainWnd());
  ASSERT(pMain);
  CRect rect;

  //Текущее расположение основного окна
  pMain->GetWindowRect(&rect);

  //Стандартные размеры видео окна для SECAM
  cs.cx = 352;
  cs.cy = 288;

  //Видео окно установим над окном mainframe
  cs.x = rect.left;
  cs.y = rect.top - rect.Height();

  return CFrameWnd::PreCreateWindow(cs);
}
```

Запустив, программу в режиме Debug получаем исключение. Диагностический макрос ASSERT(pMain) сообщает, что указатель на главное окно равен нулю. Все правильно, вызов OnFileNewFrame происходит до окончательного формирования объекта CMainFrame.

Неплохо было бы обработать сообщение Windows о полном формировании объекта окна, но такого, похоже, не существует. Можно было бы попробовать использовать сообщение WM_SHOWWINDOW, которое посылается, когда окно скрывается или появляется, но его обработка может потребовать учет числа появлений окна и выделение первого появления, если в дальнейшем окно будет скрываться и появляться.

Исходя из этих соображений, было решено послать пользовательское сообщение в очередь сообщений для главного окна и вызвать OnFileNewFrame из обработчика этого сообщения. Вызываться обработчик будет только один раз и в дальнейшем о нем можно забыть. Формируем сообщение пользователя. В заголовочном файле класса CMainFrame пишем объявление сообщения

```
afx_msg LRESULT OnMyMessage(WPARAM,LPARAM);.
```

В файле реализации в карту сообщений добавляем новый вход `ON_MESSAGE(WM_USER+1,OnMyMessage)` и тело обработчика с вызовом OnFileNewFrame.

```
LRESULT CMainFrame::OnMyMessage(WPARAM wpar,
                                LPARAM lpar)
{
  //Открыть окно изображения
  (static_cast<CTVshowApp*>(AfxGetApp()))->OnFileNewFrame();

  return 0;
}
```

Запускаем программу и видим то, к чему мы стремились, (рис 2.8). Сохраним проект в данном состоянии в папке ProjectState1, а для дальнейшей работы сделаем его копию в папке ProjectState2. Таким образом, всегда можно вернуться к исходному состоянию перед существенным изменением проекта.

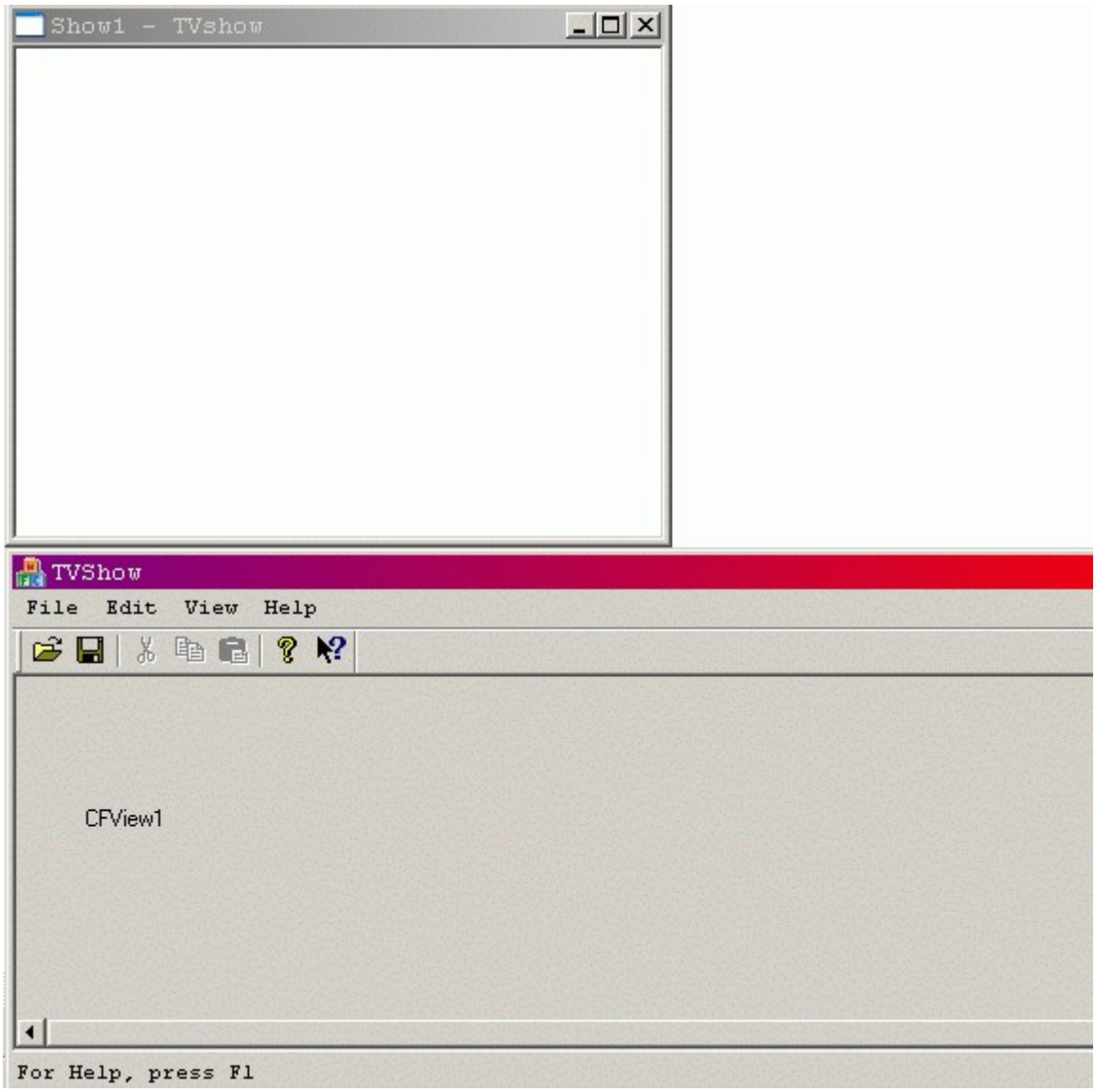


Рис.2. 8 Окна откорректированного проекта

Переключение нескольких представлений

В действительности число представлений может быть больше двух и тогда потребуется реализовывать переключатель представлений. Нам неизвестно, сколько потребуется представлений для отображения всех органов настройки и регулировки программы. Поэтому, нужно так реализовать смену представлений, чтобы в процессе работы можно было легко и просто наращивать переключатель и добавлять новые представления.

Сразу же напрашивается стандартное решение, использовать систему меню, но поскольку при работе с телеприемником, записи видео файлов и других действиях явно потребуется часто переключать представления то, система меню оказывается не очень неудобной.

Наилучший вариант это табулированное окно с несколькими закладками. К сожалению, MFC не имеет средств реализации таких окон. Существуют библиотеки, значительно расширяющие возможности MFC, но наиболее развитая библиотека является коммерческой и мало доступной по этой причине. Например, продукт фирмы Stingray имеет такие средства. Можно попытаться реализовать интерфейс программы, максимально приблизив его к табулированному окну. Для этой цели более всего подходит класс MFC CReBar. Он позволяет размещать в дочернем окне различные элементы управления, такие как панели инструментов, поля редактирования, списки. Таким образом, появляется возможность совместить традиционную панель инструментов с дополнительной панелью, на которой, разместим несколько кнопок для переключения представлений.

Класс CReBar обладает широкими возможностями и для их реализации лучше всего создать свой класс производный от CReBar, например, под именем CMyReBar. Производный класс также всегда упрощает расширение функций и внесение изменений. Новая панель класса CMyReBar может совмещаться с панелью инструментов, которую можно будет уменьшить или вообще заменить кнопками панели класса CMyReBar.

Для размещения рисунков на кнопках панели CMyReBar требуется создать по два рисунка для каждой панели в формате BMP. Для ускорения работы выберем произвольные готовые рисунки, которые потом можно будет легко изменить на подходящие по смыслу настроек для каждой панели. Первый рисунок предназначен для первоначального отображения, а второй для подсветки при наведении курсора. Проще всего скопировать подходящий рисунок и откорректировать его в соответствии со своими нуждами. Ниже показаны два произвольных рисунка для панели кнопок переключения представлений. Для замены традиционной панели инструментов также создаются два рисунка. Можно ограничиться и одним рисунком, но тогда будет отсутствовать подсветка активного состояния кнопки.

Рис.2. 9 Рисунки для панели CMyReBar

В классе CMyReBar перегружаем функцию Create, в которой реализуем стандартную панель инструментов и панель инструментов с четырьмя кнопками переключения представлений. В случае необходимости можно легко добавить дополнительную кнопку в соответствующую ей панель. Добавляем две переменные класса CToolBar m_wndToolBar, и m_wndToolBar_1 в файле заголовков, которые будут определять две панели кнопок. Комментарии в исходном тексте функции поясняют подробности реализации.

```
//Число кнопок
#define NBUT_TBAR_1 4
#define NBUT_TBAR   4

BOOL CMyReBar::Create(CWnd* pParentWnd,
                      DWORD dwCtrlStyle   ,DWORD dwStyle
                      , UINT nID)
{
  //Список рисунков
  CImageList img;

  CString str;
  CRect rectToolBar;
  REBARBANDINFO rbbi;

  //Создается объект CReBar
  BOOL bret = CReBar::Create(pParentWnd, dwCtrlStyle,
                             dwStyle, nID);

  //Создаем замену традиционной панели инструментов
  if (!m_wndToolBar_1.CreateEx(pParentWnd,
      TBSTYLE_TRANSPARENT| TBSTYLE_FLAT ,AFX_IDW_TOOLBAR +1))
  {
    TRACE0("Failed to create toolbar\n");
    return -1;      //fail to create
  }

  //Создаем  панель переключения представлений
  if (!m_wndToolBar.CreateEx(pParentWnd))
  {
    TRACE0("Failed to create toolbar\n");
    return -1;      //fail to create
  }

  //Установка свойств традиционной панели инструментов.
  //Определяется минимальный и максимальный
  //размер кнопок
```

```
m_wndToolBar_1.GetToolBarCtrl().SetButtonWidth(50, 50);

//Установка исходного списка рисунков и подсветки
img.Create(IDB_MAINFRAME_1HOT, 22,0,RGB(255, 0, 255));
m_wndToolBar_1.GetToolBarCtrl().SetHotImageList(
                                &img);  img.Detach();

img.Create(IDB_MAINFRAME_1, 22, 0, RGB(255, 0, 255));
m_wndToolBar_1.GetToolBarCtrl().SetImageList(&img);

img.Detach();

//Установка числа кнопок
m_wndToolBar_1.SetButtons(NULL, NBUT_TBAR_1);

//Инициализация кнопок
m_wndToolBar_1.SetButtonInfo(0, ID_FILE_OPEN,
    TBSTYLE_BUTTON, 0); str.LoadString(IDS_FILE_OPEN);
m_wndToolBar_1.SetButtonText(0, str);
m_wndToolBar_1.SetButtonInfo(1, ID_FILE_SAVE,
                            TBSTYLE_BUTTON, 1);

str.LoadString(IDS_FILE_SAVE);
m_wndToolBar_1.SetButtonText(1, str);
m_wndToolBar_1.SetButtonInfo(2, ID_APP_ABOUT,
  TBSTYLE_BUTTON, 2);

str.LoadString(IDS_APP_ABOUT);
m_wndToolBar_1.SetButtonText(2, str);
m_wndToolBar_1.SetButtonInfo(3, ID_HELP,
  TBSTYLE_BUTTON, 3);

str.LoadString(IDS_HELP);
m_wndToolBar_1.SetButtonText(3, str);

//Установка  свойств панели переключения
//представлений. Определяется минимальный и
//максимальный размер кнопок
m_wndToolBar.GetToolBarCtrl().SetButtonWidth(50, 80);

//Установка списка рисунков
img.Create(IDB_HOTTOOLBAR, 22, 0, RGB(255, 0, 255));
m_wndToolBar.GetToolBarCtrl().SetHotImageList(&img);
img.Detach();

img.Create(IDB_COLDTOOLBAR, 22, 0, RGB(255, 0, 255));
m_wndToolBar.GetToolBarCtrl().SetImageList(&img);
img.Detach();

//Установка числа кнопок
m_wndToolBar.SetButtons(NULL, NBUT_TBAR);
```

```
//Инициализация кнопок
m_wndToolBar.SetButtonInfo(0, ID_BUT_FORM_MAIN,
    TBSTYLE_BUTTON, 0);

str.LoadString(IDS_FORM_MAIN);
m_wndToolBar.SetButtonText(0, str);
m_wndToolBar.SetButtonInfo(1,ID_BUT_FORM1
            ,TBSTYLE_BUTTON, 1);

str.LoadString(IDS_FORM1);
m_wndToolBar.SetButtonText(1, str);
m_wndToolBar.SetButtonInfo(2, ID_BUT_FORM2
            , TBSTYLE_BUTTON, 2);

str.LoadString(IDS_FORM2);
m_wndToolBar.SetButtonText(2, str);
m_wndToolBar.SetButtonInfo(3, ID_BUT_FORM3,
  TBSTYLE_BUTTON,3);
str.LoadString(IDS_FORM3);
m_wndToolBar.SetButtonText(3, str);

//Устанавливаем последовательно все панели.
//Добавление первого ReBar
AddBar(&m_wndToolBar_1,NULL, NULL,
            RBBS_NOGRIPPER|RBBS_FIXEDBMP );

//Добавление второго ReBar.
//Установить стиль  RBBS_GRIPPERALWAYS
AddBar(&m_wndToolBar,NULL, NULL
      , RBBS_GRIPPERALWAYS | RBBS_FIXEDBMP);

//Параметры нужно устанавливать  после добавления
//всех панелей. Установка размеров кнопок для
//первой панели
m_wndToolBar_1.GetItemRect(0, &rectToolBar);
m_wndToolBar_1.SetSizes(rectToolBar.Size(),
                           CSize(20,20));

//Инициализация параметров для первой панели
rbbi.cbSize = sizeof(rbbi);
rbbi.fMask = RBBIM_SIZE | RBBIM_IDEALSIZE |
                         RBBIM_CHILDSIZE;
rbbi.cxMinChild = rectToolBar.Width();
rbbi.cyMinChild = rbbi.cyMaxChild = rectTool  Bar.Height();
rbbi.cx = rbbi.cxIdeal =
    static_cast<UINT>(rectToolBar.Width()* NBUT_TBAR_1);

//Установка параметров панели
GetReBarCtrl().SetBandInfo(0, &rbbi);

//Установка стиля первой панели
m_wndToolBar_1.SetBarStyle(m_wndToolBar_1.GetBarStyle()
```

```
          | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_FIXED);

  //Установка размеров кнопок для второй панели
  m_wndToolBar.GetItemRect(0, &rectToolBar); m_wndToolBar.Set-
Sizes(rectToolBar.Size(),
          CSize(30,20));

  //Установка параметров для второго bar
  rbbi.cbSize = sizeof(rbbi);
  rbbi.fMask = RBBIM_CHILDSIZE | RBBIM_IDEALSIZE | RBBIM_SIZE;

  rbbi.cxMinChild = rectToolBar.Width();
  rbbi.cyMinChild = rectToolBar.Height();
  rbbi.cx =
  rbbi.cxIdeal = rectToolBar.Width() * NBUT_TBAR;

  //Установка параметров панели
  GetReBarCtrl().SetBandInfo(1, &rbbi);

  //Установка стиля второй панели
  m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle()
                |CBRS_TOOLTIPS | CBRS_FLYBY
           |CBRS_SIZE_FIXED);

  //Установка для кнопок второй панели группового
 //стиля для   зависимого переключения кнопок
  m_wndToolBar.SetButtonStyle(0,TBBS_CHECKGROUP);m_wndToolBar.-
SetButtonStyle(1,TBBS_CHECKGROUP);
  m_wndToolBar.SetButtonStyle(2,TBBS_CHECKGROUP);
  m_wndToolBar.SetButtonStyle(3,TBBS_CHECKGROUP);

  //Установка исходного состояние переключателя
 //представлений,первая кнопка нажата
  m_wndToolBar.GetToolBarCtrl().CheckButton(
                           ID_BUT_FORM_MAIN);

  return bret;
}
```

Для сохранения преемственности поведения используем идентификаторы традиционной панели инструментов для первой панели инструментов, а для второй панели назначаем собственные идентификаторы ID_BUT_FORM… для каждой кнопки и записываем их в таблицу строк и таблицу акселераторов. Запись идентификаторов в таблицу акселераторов следует сделать обязательно, поскольку только после такой записи идентификаторы появляются в таблице событий (см. рис 9), что позволяет назначить обработчики событий и обработчики обновления интерфейса автоматически.

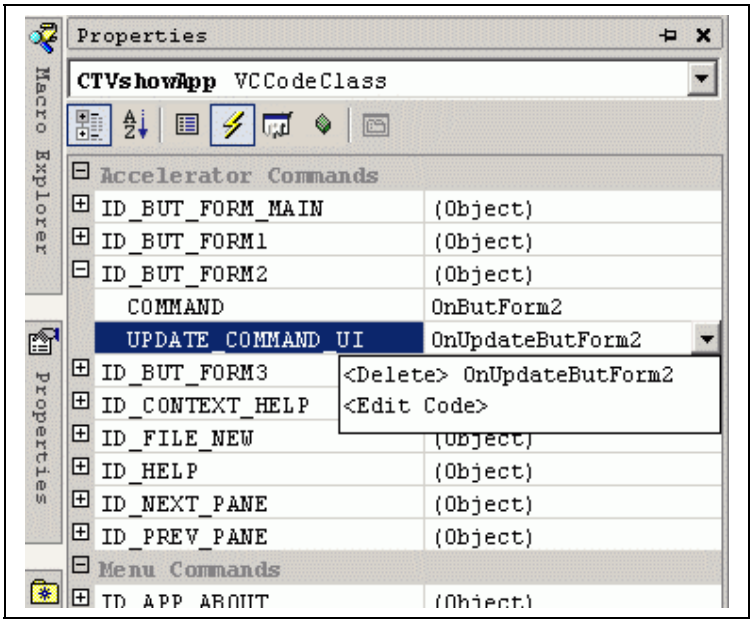


Рис.2. 10 Назначение обработчиков событий

Обработчики обновления интерфейса OnUpdateButForm… нужны обязательно, поскольку процесс настройки приемника и программы явно будет иметь зависимый характер и возможно придется блокировать некоторые кнопки переключателя представлений. Разместить обработчики можно в нескольких классах, но удобнее всего в классе приложения CTVshowApp, поскольку там размещается функция переключения представлений и указатели на представления.

Далее в функции CMainFrame::OnCreate удаляем традиционную панель инструментов и создаем объект класса CMyReBar. В заголовочном файле класса CMainFrame заменяем переменную CToolBar m_wndToolBar на CMyReBar m_reBar и не забываем сделать запись в начале файла #include "MyReBar.h".

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
    return -1;
  if (!m_wndStatusBar.Create(this) ||                    !m_wnd-
StatusBar.SetIndicators(indicators
        ,sizeof(indicators) / sizeof(UINT)))
  {
    TRACE0("Failed to create status bar\n");
    return -1;      //fail to create
  }
  m_reBar.Create(this);

  //Открыть окно изображения
  this->PostMessage(WM_USER+1);
```

```
  return 0;
}
```

После трансляции программы и ее запуска увидим измененный интерфейс программы.

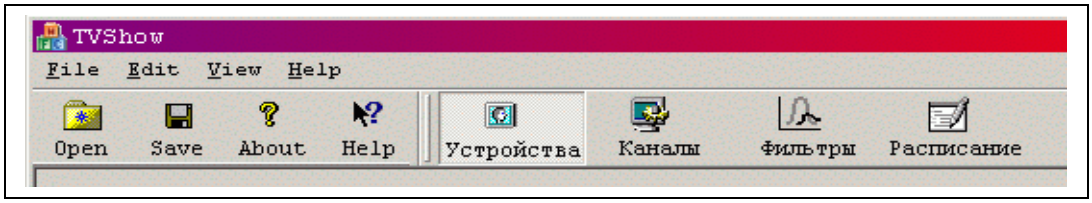


Рис.2. 11 Измененный вид интерфейса программы

В завершение реализуем переключение представлений. Для этого, каждому представлению следует назначить исходный собственный идентификатор, который при переключении представления, будет меняться на предопределенный идентификатор первого представления AFX_IDW_PANE_FIRST, как пояснялось ранее.
В файле реализации класса CTVshowApp определим константы для идентификаторов представлений.

```
//идентификаторы представлений
#define   IDVIEW_MAIN   2000
#define   IDVIEW_1    (IDVIEW_MAIN +1)
#define   IDVIEW_2    (IDVIEW_MAIN +2)
#define IDVIEW_3    (IDVIEW_MAIN +3)
```

В функции InitInstance установим идентификаторы для каждого представления, которое сменяет первое (IDVIEW_MAIN). Первое представление получает значение AFX_IDW_PANE_FIRST при инициализации без нашего вмешательства. Ниже жирным шрифтом выделены строки, которые необходимо добавить для назначения идентификаторов представлениям.

```
…….
  m_pFView1->OnInitialUpdate();
  ::SetWindowLong(m_pFView1->m_hWnd,GWL_ID,IDVIEW_1);
…….
  m_pFView2->OnInitialUpdate();
  ::SetWindowLong(m_pFView2->m_hWnd,GWL_ID,IDVIEW_2);
…….
  m_pFView3->OnInitialUpdate();
  ::SetWindowLong(m_pFView3->m_hWnd,GWL_ID,IDVIEW_3);
```

При смене представлений, значение идентификатора важно только для активного представления (AFX_IDW_PANE_FIRST), для остальных следует только обеспечить правильный порядок замены. Достаточно назначить идентификатор AFX_IDW_PANE_FIRST представлению, чья кнопка нажата, а исходный идентификатор этой кнопки присвоить представлению, которое было

активным. Другими словами, произвести обмен идентификаторами. При нажатии на кнопку переключателя соответствующему представлению назначается его исходный идентификатор, который затем переназначается активному представлению, а тому, чья кнопка нажата, устанавливается значение AFX_IDW_PANE_FIRST (рис.2.12). При назначении идентификатора определяется указатель представления, которое станет первым (активным).

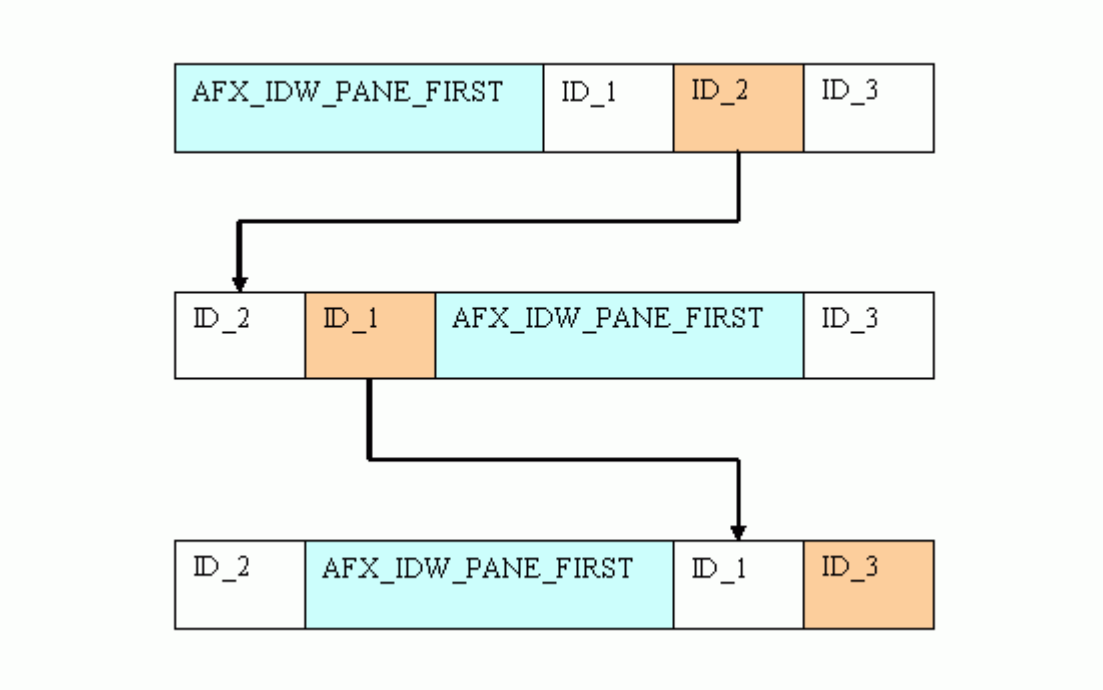


Рис.2. 12 Принцип переключения представлений.

Для восстановления значений идентификаторов пишем простой метод.

```
CView* CTVshowApp::mySetViewID(long id)
{
  CView* pview;
  //Определяем указатель на представление,
  //соответствующий его идентификатору
  switch(id)
  {
  case 0: pview = m_pMainView;
    break;
  case 1: pview = m_pFView1;
    break;
  case 2: pview = m_pFView2;
    break;
  case 3: pview = m_pFView3;
    break;
  default: pview = m_pMainView;
  }

  //Восстанавливаем ID представления
```

```
  ::SetWindowLong(pview->m_hWnd, GWL_ID,IDVIEW_MAIN + id);

  return pview;
}
```

Метод вызывается в соответствующем обработчике кнопки, перед переключением представления.

```
void CTVshowApp::OnButForm3()
{
  //Восстановление ID представления
  CView * pV = mySetViewID(3);

  //Переключение представления
  mySwitchView(pV);
}
```

Метод mySwitchView даже несколько упрощается по сравнению с тестовым вариантом.

```
CView* CTVshowApp::mySwitchView(CView* pView)
{
     CView* pOldView =(static_cast<CFrameWnd*>
                    (m_pMainWnd))->GetActiveView();

     //Активному представлению устанавливаем ID нового //пред-
     ставления, а новому устанавливаем
     //ID=AFX_IDW_PANE_FIRST, для обеспечения
     //работы RecalcLayout,
     //возвращаем  ID представления нажатой клавиши
    UINT idNew = ::GetWindowLong(pView->m_hWnd, GWL_ID);

    //Назначаем  AFX_IDW_PANE_FIRST представлению
    //нажатой клавиши
    ::SetWindowLong(pView->m_hWnd, GWL_ID, AFX_IDW_PANE_FIRST);

    //Назначаем  старому представлению ID нового
    ::SetWindowLong(pOldView->m_hWnd, GWL_ID, idNew);

    //Скрываем старое представление и показываем
    //новое представление
      pOldView->ShowWindow(SW_HIDE);
      pView->ShowWindow(SW_SHOW);

     //Устанавливаем видимое представление активным
      (static_cast<CFrameWnd*>(m_pMainWnd))
                              ->SetActiveView(pView);

     //Перерасчет клиентской области и перерисовка
      (static_cast<CFrameWnd*>(m_pMainWnd))->RecalcLayout();
```

```
    pView->Invalidate();

   return pOldView;
  }
```

Проводим проверку функционирования переключения представлений и подготовительная работа практически завершается. Возможные улучшения интерфейса программы можно производить по мере возникновения необходимости. Эти два метода без труда можно объединить в один при последующей модернизации проекта, что упростит код.
Полностью версия проекта в данном состоянии находится в папке ProjectState2.

Установка DirectX 9.0 SDK, внесение изменений в проект

Среда разработки DirectX 9.0 SDK доступна для загрузки на сайте Microsoft. Общий размер загружаемого файла составляет приблизительно 234 MB. Установка DirectX 9.0 SDK не имеет, каких либо особенностей, за исключением того, что пользователю предлагается выбрать один из двух вариантов, DirectX SDK отладочный вариант или окончательный (retail). Отладочный вариант SDK работает медленней, но какой бы вариант DirectX ни был загружен, предоставляется возможность оперативной перезагрузки.

Все приложения, написанные с применением DirectShow, используют заголовочный файл Dshow.h, который следует включить в файл stdafx.h. Некоторые интерфейсы DirectShow требуют дополнительных заголовочных файлов, в этом случае они указываются в документации DirectX. Дополнительно используются две библиотеки Strmiids.lib и Quartz.lib. Первая библиотека экспортирует идентификаторы классов CLSID и идентификаторы интерфейсов IID, а вторая экспортирует функцию AMGetErrorText. Функция AMGetErrorText, позволяет выводить текстовые сообщения об ошибках, используя текущие языковые установки. Если ее использование не предполагается, то библиотеку Quartz.lib можно не включать. Однако ее применение существенно облегчает обработку ошибок и избавляет от составления текстов собственных сообщений. После установки DirectX следует проверить и внести изменения в установки проекта. Вначале проверяем пути к SDK DirectX, для чего открываем меню Tool / Options и в открывшемся окне проверяем пути к файлам заголовков и библиотекам, как показано на рисунке 2.13.

Если пути к DirectX не указаны, их следует дописать вручную в первую строку соответствующего списка. Желательно удалить всякую оптимизацию кода проекта на этапе его разработки, и вообще лучше ее не использовать. На рисунке 2.14 показана соответствующая вкладка свойств проекта.

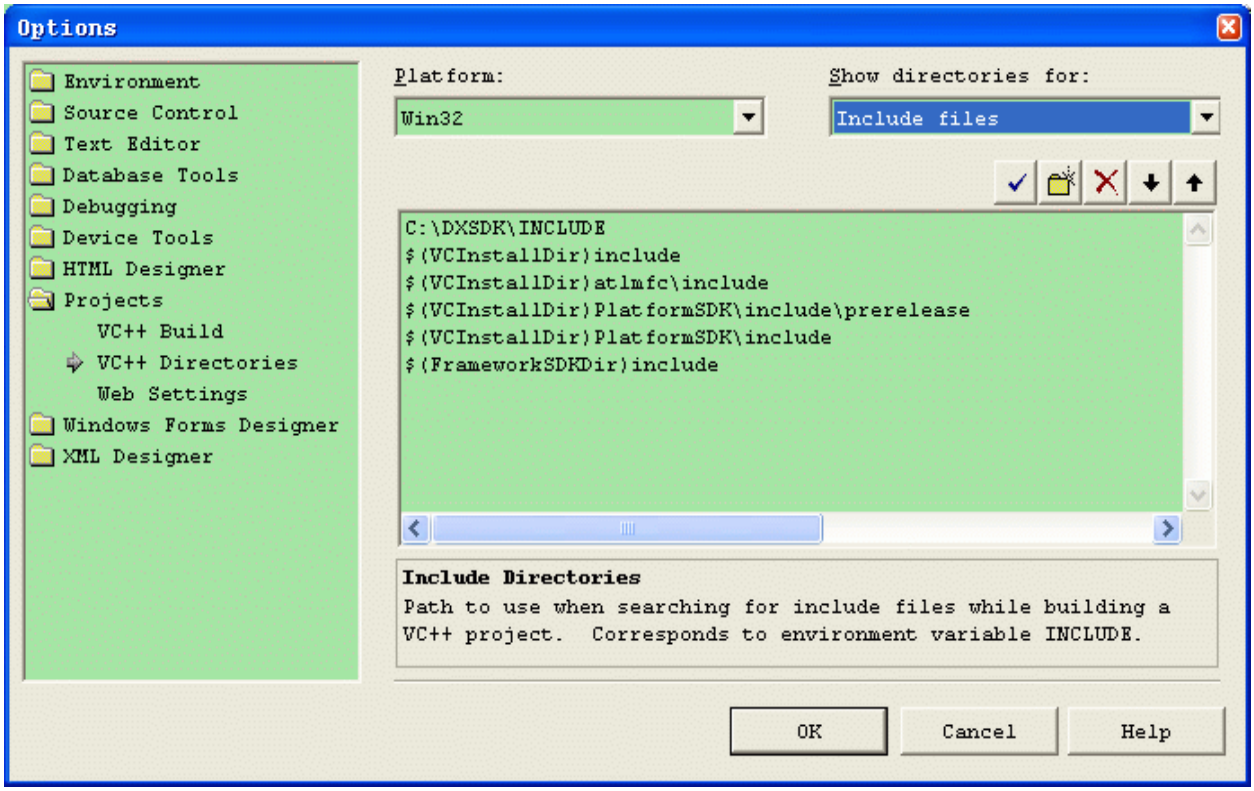


Рис.2. 13 Пути к библиотекам и файлам DirectX

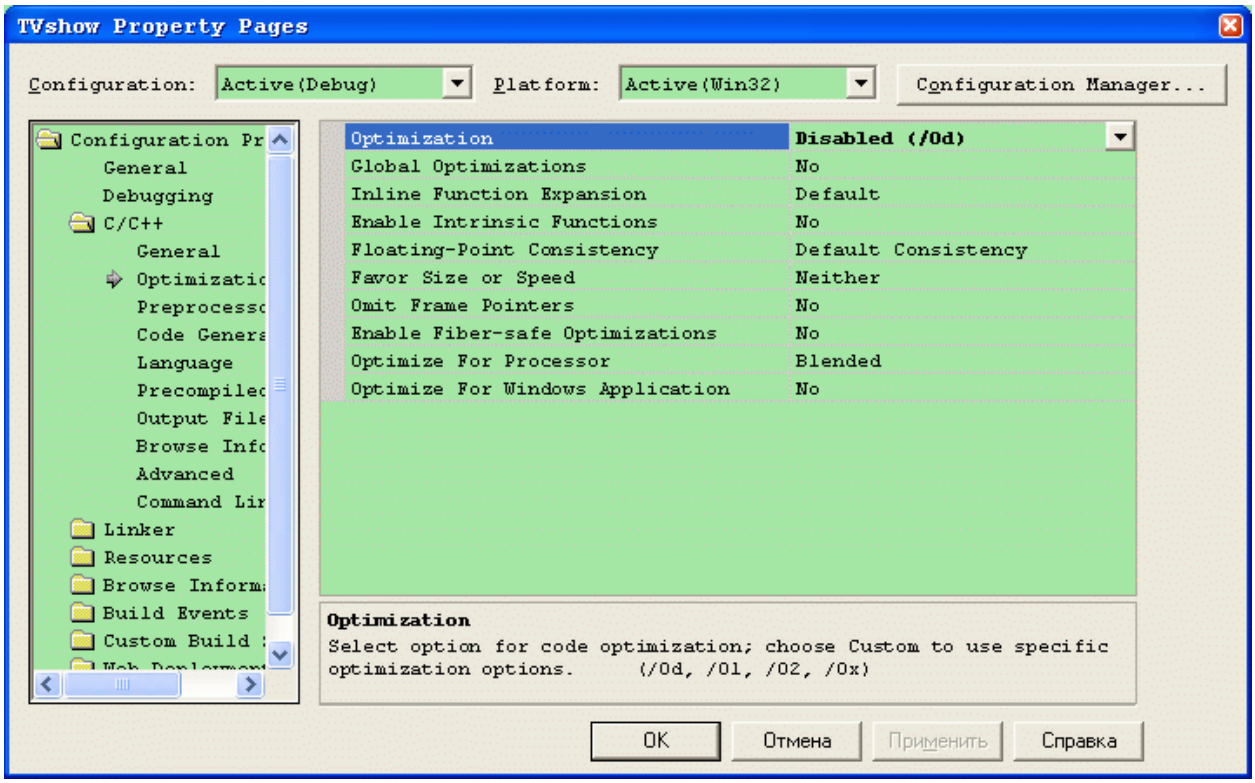


Рис.2. 14 Удаление оптимизации кода

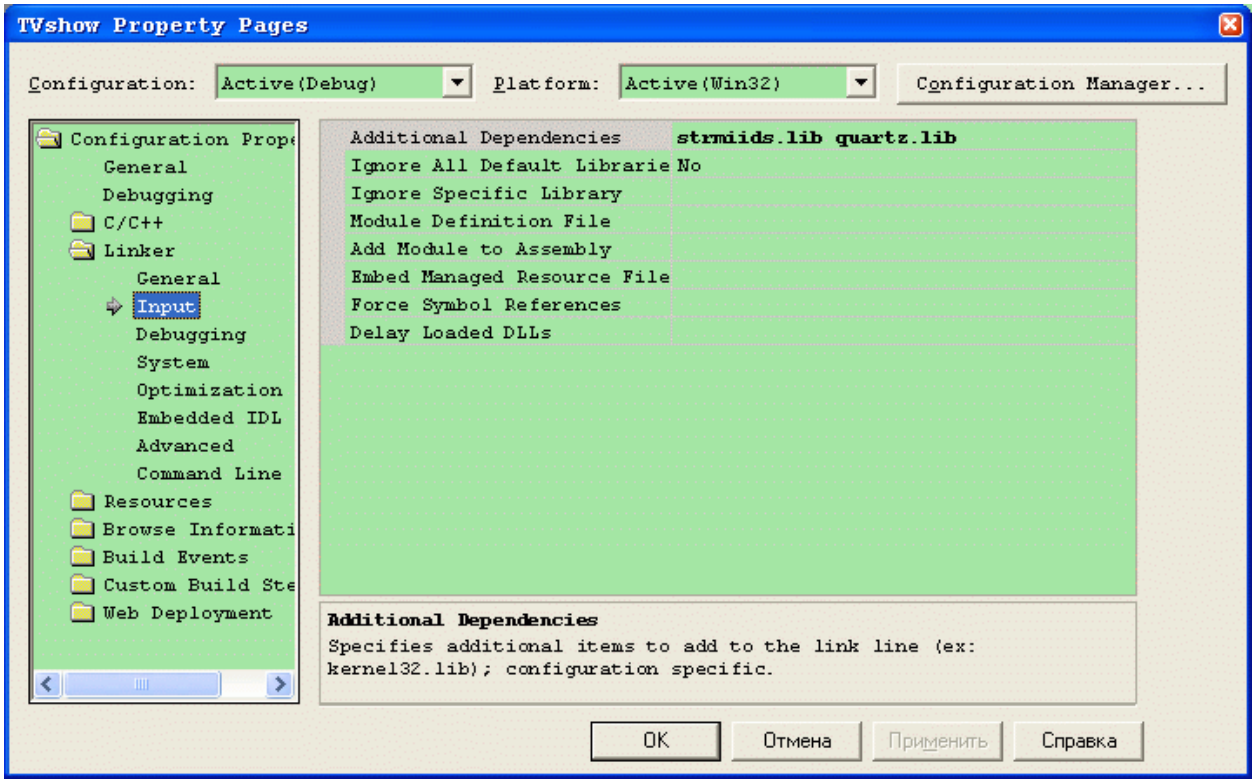


Рис.2. 15 Включение библиотек DirectX

На рисунке 2.15 показана вкладка свойств проекта, для включения библиотек DirectX, пути к ним уже указаны, как пояснялось ранее. Установки следует делать для режима Debug и Release, что обычно упускается из виду. После внесения этих изменений проект готов к работе с применением DirectShow. Дополнительные файлы, требуемые при использовании некоторых интерфейсов, могут быть включены в файле stdafx.h.

Построение фильтрового графа DirectShow.

Класс фильтрового графа

На первом этапе построения фильтрового графа следует решить следующие задачи:

- Реализовать класс, который будет определять все функции устройства в фильтровом графе. Поскольку программа пишется для телевизионного устройства, центральным компонентом которого является фильтр сбора данных (Capture), то и класс можно создать под именем CCapture.
- Создать менеджер фильтрового графа.
- Найти в системе фильтр сбора и установить его в граф.

После первичной реализации класса CCapture с помощью мастера или вручную следует решить важнейшую задачу, в каком классе проекта создавать объект CCapture. Класс CCapture должен играть роль сервера, то есть обслуживать запросы пользователя по управлению фильтровым графом. Интерфейс пользователя будет располагаться на нескольких вкладках. От того, каким образом классы вкладок будут передавать запросы классу CCapture и связываться между собой, зависит эффективность и сложность реализации проекта в целом. В классе приложения CTVshowApp сохраняются все указатели на классы представлений (вкладок), поэтому если объект класса CCapture создать в классе приложения, то и реализация взаимной связи вкладок и класса CCapture будет максимально простой. Объект любой вкладки сможет получить указатель на класс CCapture через класс приложения. Другим важным условием упрощения взаимосвязи классов будет условие, при котором класс CCapture не должен по своей инициативе посылать запросы любому другому классу. В результате все запросы от классов вкладок будут односторонние. В исключительных случаях, когда классу CCapture потребуется активизировать объект вкладки, его окну можно послать пользовательское сообщение. Класс CCapture должен будет сохранять свое текущее состояние, тогда не потребуется взаимных пересекающихся запросов между классами вкладок для определения состояния элементов управления. Любой объект вкладки, при необходимости, может запросить текущее состояние параметра телеприемника у класса CCapture. На рисунке 2.16 показана структура связей класса CCapture. Пунктиром показаны сообщения классам вкладок

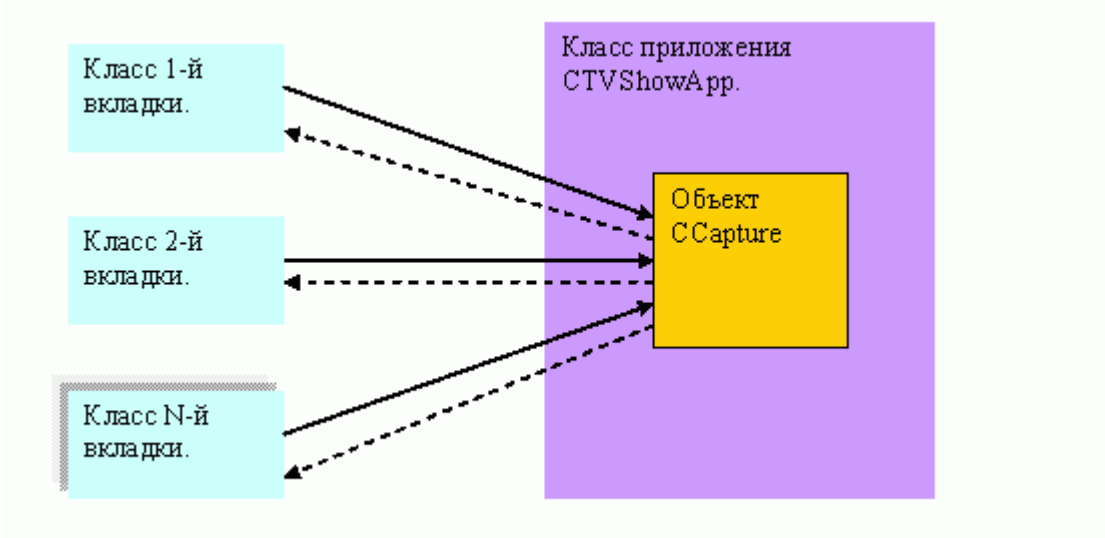


Рис.2. 16 Связи класса CCapture

Создаем объект CCapture в методе CTVshowApp::InitInstance() после инициализации библиотеки COM на куче и сразу позаботимся об его удалении при завершении программы.

```
…………………………………………………………………………………………………………………………
  CoInitialize(NULL);
  //Создание объекта DirectShow
  m_pCap = new CCapture;
…………………………………………………………………………………………………………………………
```

Для удаления объекта напишем метод myDestroyCapture.

```
void CTVshowApp::myDestroyCapture(void)
{
  if(m_pCap)
  { delete m_pCap; m_pCap = 0; }
}
```

Удаление объекта производим в обработчике CMainFrame::OnClose().

```
void CMainFrame::OnClose()
{
  CTVshowApp* pApp = static_cast<CTVshowApp*>(AfxGetApp());

  //Удаление объекта сбора
  pApp->myDestroyCapture();

  //Удаление библиотеки COM
  CoUninitialize();

  CFrameWnd::OnClose();
}
```

Теперь проект подготовлен для разработки программы управления телевизионным приемником.

Реализация менеджера фильтрового графа

Построение фильтрового графа и управление им осуществляет менеджер графа (Filter Graph Manager). Объект менеджера графа является центральным компонентом в DirectShow. Приложение использует интерфейсы менеджера графа для построения и управления фильтровым графом. Менеджер графа также выполняет синхронизацию потоков данных и обработку сообщений. Построение фильтрового графа всегда начинается с создания объекта менеджера графа. После построения объекта менеджера графа можно строить сам граф с использованием различных фильтров DirectShow.

Интерфейсы менеджера фильтрового графа

Менеджер графа предоставляет несколько интерфейсов для управления фильтровым графом и для его построения. Часть интерфейсов предоставляется соответствующими фильтрами, но не прямо, а через менеджер графа.

Ниже приводится краткое описание интерфейсов. После создания объекта менеджера графа метод CoCreateInstance возвращает указатель на интерфейс IGraphBuilder. Интерфейсы, выделенные жирным шрифтом, являются интерфейсами, обеспечивающими построение и перестройку фильтрового графа.

Интерфейс	Описание
IFilterGraph	Обеспечивает методы для построения фильтрового графа. Является базовым для интерфейса IGraphBuilder.
IGraphBuilder	Обеспечивает методы для построения фильтрового графа. Является производным от интерфейса IFilterGraph. Приложение должно использовать этот интерфейс раньше, чем IFilterGraph.
IFilterGraph2	Расширяет интерфейсы IFilterGraph и IGraphBuilder. Добавляет три метода ля построения фильтрового графа.
IAMGraphStreams	Обеспечивает управление графом при обработке данных от источников реального времени ("живые" источники). Управляет синхронизацией и временем обработки выборки для "живого" источника.
IAMStats	Позволяет возвратить статистику производительности из менеджера графа.
IBasicAudio	Осуществляет управление громкостью и балансом на звуковом фильтре (канале).
IBasicVideo	Осуществляет управление параметрами изображения на отображающем фильтре.
IBasicVideo2	Интерфейс производный от IBasicVideo. Добавляет один метод.
IFilterChain	Предоставляет интерфейсы для управления цепочкой фильтров (участком фильтрового графа).
IFilterMapper2	Управляет регистрацией и поиском фильтров в регистре.
IGraphConfig	Поддерживает динамическую реконфигурацию фильтров в графе без остановки графа.
IGraphVersion	Возвращает номер текущей версии графа.
IMediaControl	Управляет состоянием графа, пуск, стоп и пауза.
IMediaEvent	Управляет событиями, происходящими в графе.
IMediaEventEx	Интерфейс производный от IMediaEvent, дополняет и расширяет его.
IMediaEventSink	Уведомляет менеджер графа о событиях, происходящих в графе. Не применяется приложением.

IMediaFilter	Управляет состоянием фильтра (потока). Не применяется приложением, кроме двух методов, управляющих ссылочным временем.
IMediaPosition	Управляет передвижением по потоку. Интерфейс предназначен для применения с Visual Basic.
IMediaSeeking	Управляет передвижением по потоку. Интерфейс предназначен для применения с C/C++.
IQueueCommand	Интерфейс предназначен для создания очереди команд (методов), выполняющихся менеджером графа в указанное время.
IRegisterServiceProvider	Интерфейс регистрирует объект как сервис. Сервис это интерфейс, который клиент получает через COM. Можно зарегистрировать менеджер графа и затем получить его из другого объекта.
IResourceManager	Обеспечивает управление общими системными ресурсами, например звуковой картой. Приложения обычно не используют этот интерфейс.
IServiceProvider	Обеспечивает запрос сервиса COM. Относится к общим интерфейсам COM.
IVideoFrameStep	Обеспечивает покадровый просмотр видео потока. Просмотр в обратном направлении не поддерживается. Интерфейс работает с аппаратным декодером, который должен поддерживать покадровый просмотр.
IVideoWindow	Устанавливает свойства для окна просмотра видео, позицию размеры и т.п.

Таблица 1 Интерфейсы менеджера графа

Кроме перечисленных интерфейсов для построения графа сбора данных важную роль играет интерфейс ICaptureGraphBuilder2, который реализуется на вспомогательном объекте построителя графа сбора. Создается объект с помощью вызова метода CoCreateInstance с идентификатором CLSID_CaptureGraphBuilder2.

При построении фильтрового графа DirectShow, в дополнение к перечисленным интерфейсам, используются и некоторые другие интерфейсы.

Интерфейс	Описание
ICreateDevEnum	Интерфейс перечислителя устройств и фильтров. Возвращается при создании объекта системного перечислителя.

IDvdGraphBuilder	Построитель фильтрового графа для DVD.
IEnumFilters	Перечисление фильтров в графе. Интерфейс предоставляется при вызове метода менеджера графа IFilterGraph::EnumFilters.
IAMGraphBuilderCallback	Реализует механизм повторного вызова (callback) при построении графа.
IAMFilterGraphCallback	Принимает сообщения (callback) при построении графа, если контакт не может быть подключен.

Таблица 2 Дополнительные интерфейсы для построения графа.

Три интерфейса менеджера графа IFilterGraph, IGraphBuilder, IFilterGraph2 и интерфейс построителя графа сбора ICaptureGraphBuilder2 предоставляют методы, позволяющие построить граф тремя способами:

- Приложение может указать графическому менеджеру, строить граф полностью автоматически. Используется при решении типовых задач, например, воспроизведение файла.
- Использование смешанного режима построения графа. Часть графа строится автоматически, а другая часть строиться приложением в ручном режиме.
- Приложение будет полностью строить граф, выбирая и соединяя каждый фильтр индивидуально.

Первый способ построения графа чаще всего используется при решении стандартных задач, таких как, например, воспроизведение файла AVI, MPEG, WAV, MP3 и некоторых других типов файлов. В этом случае вызывается метод IGraphBuilder::RenderFile, который требует в качестве параметра только имя файла. Файл может быть удаленным или локальным. При автоматическом построении графа менеджер осуществляет поиск фильтров в регистре, используя карту фильтров. Соединение фильтров производится в соответствии с медиа типом потока на контактах фильтра. Поиск фильтра начинается с определения медиа типа данных в файле по его расширению. Менеджер осуществляет поиск соответствия в регистре для зарегистрированных расширений файлов. Для поиска менеджер создает экземпляр карты фильтров и вызывает метод IFilterMapper2::EnumMatchingFilters. Применение этого метода показано в примере, приведенном для системного перечислителя. Если такое соответствие не установлено, тогда менеджер читает файл для поиска проверочных байтов (check bytes), по которым определяется тип медиа данных. После установки соответствия ищется фильтр источник способный читать данный медиа тип. Процесс поиска носит итеративный характер и использует следующие критерии для отбора фильтров.

- Фильтр выбирается по категории фильтра, которая определяет его область использования.

- Определяются медиа типы данных для входных и выходных контактов фильтра.

- Окончательно фильтр выбирается по параметру предпочтения (merit) фильтра. В граф устанавливается фильтр, имеющий наивысший параметр предпочтения. Фильтры, имеющие более низкие параметры предпочтения при совпадении всех остальных данных, устанавливаются в граф только приложением, поскольку они проектируются для специальных целей.

Если фильтр источник найден, тогда менеджер создает экземпляр фильтра и устанавливает его в граф , вызывая метод IGraphBuilder::AddSourceFilter. После добавления фильтра менеджер проверяет контакты на соответствие медиа типу. После отбора первого фильтра, ищется второй фильтр для подключения к первому. Менеджер проверяет совпадение медиа типов выходного контакта первого фильтра и входного контакта второго фильтра и вызывает метод IGraphBuilder::Connect. Если попытка соединения фильтров оказалась неудачной, поиск повторяется до тех пор, пока поток не будет обработан. После построения графа, приложение нуждается только в интерфейсе IMediaControl для управления процессом воспроизведения файла.

Второй способ предполагает участие в построении графа приложения. Чаще всего сначала граф строится в автоматическом режиме, а затем производятся определенные коррективы, некоторые фильтры удаляются, другие устанавливаются и подключаются. Например, перекодировку файлов из одного формата в другой можно сделать из графа воспроизведения файла созданной в автоматическом режиме. Для этого удаляются фильтры воспроизведения, и устанавливается фильтр кодера и фильтр записи файлов.

Третий способ создания графа требует точного знания спецификации каждого фильтра, устанавливаемого в граф. Каждый фильтр устанавливается и соединяется вручную, при этом отпадает необходимость в использовании карты фильтров. Этот способ позволяет перестраивать граф с максимальной скоростью.

Методы интерфейсов менеджера графа и построителя фильтра сбора

Методы интерфейсов менеджера графа и интерфейса построителя графа сбора играют основную роль в построении графа. Их можно считать интерфейсами верхнего уровня. Многие методы, указанных интерфейсов, реализуют сложные алгоритмы поиска и соединения фильтров в одном методе, скрывая детали и подробности процесса. Методы этих интерфейсов используют методы других интерфейсов, которые можно условно считать интерфейсами более низкого уровня, например интерфейс контакта IPin. Часть методов

интерфейсов нижнего уровня не разрешается (не рекомендуется) для прямого применения приложением, что указывается в описании метода. Такие методы используются в методах интерфейсов верхнего уровня и при реализации фильтров. Другую часть методов интерфейсов нижнего уровня разрешается использовать в приложении. Некоторые методы интерфейсов верхнего уровня также не разрешается (не рекомендуется) использовать в приложении. Однако это не догма, и в некоторых ситуациях просто необходимо использовать такие методы. Методы, не рекомендуемые для использования приложением, могут быть спроектированы для работы в определенном окружении, и к их применению следует подходить внимательно.

Далее кратко рассмотрим свойства интерфейсов менеджера графа и интерфейса построителя графа и методов, которые они используют. Более подробно особенности методов этих интерфейсов будут рассматриваться при их применении.

Методы интерфейса IFilterGraph

Интерфейс IFilterGraph предоставляет методы для построения фильтрового графа. Он является базовым интерфейсом для интерфейса IGraphBuilder. Интерфейс предоставляется объектом менеджера графа. Приложение должно использовать производные интерфейсы раньше, чем базовые.

Методы	Описание
AddFilter	Устанавливает фильтр в граф и присваивает ему, указанное имя. Имя можно не указывать, тогда оно будет генерироваться автоматически.
RemoveFilter	Удаляет фильтр из графа.
EnumFilters	Создает перечислитель для фильтров в графе.
FindFilterByName	Ищет фильтр по имени, которое было назначено ему при установке в граф.
ConnectDirect	Производит прямое соединение двух контактов без установки промежуточных фильтров.
Reconnect	Разрывает существующее соединение и производит соединение с тем же самым контактом. Не разрешается использовать в приложении. Метод вызывается фильтрами при построении графа.
Disconnect	Отключает указанный контакт, но соединение не разрывает. Для разрыва соединения необходимо отключать и второй контакт.
SetDefaultSyncSource	Указывает менеджеру графа установить источник синхронизации по умолчанию. Метод не нужно вызывать при первоначальном построении графа, поскольку менеджер графа делает это автоматически.

Таблица 3 Интерфейс IFilterGraph

Методы интерфейса IGraphBuilder

Методы данного интерфейса позволяют строить фильтровый граф. Интерфейс является производным от интерфейса IFilterGraph и предоставляется объектом менеджера графа. Интерфейс добавляет методы строящие граф по неполной информации. При использовании этих методов приложение предоставляет частичную информацию, по которой менеджер графа самостоятельно отбирает нужные фильтры, зарегистрированные в системе, устанавливает их в граф и соединяет, используя алгоритм "интеллектуального" соединения (Intelligent Connect).

Метод	Описание
Connect	Соединяет два контакта. При необходимости устанавливает промежуточные фильтры.
Render	Строит ветвь графа, необходимую для воспроизведения (отображения) данных, начиная с указанного выходного контакта.
RenderFile	Строит фильтровый граф для воспроизведения указанного файла. Метод не удаляет никаких фильтров из графа. При повторном использовании предыдущие фильтры остаются в графе и при запуске графа работают вместе, воспроизводя оба файла.
AddSourceFilter	Устанавливает в граф фильтр источник для указанного файла.
SetLogFile	Назначает файл для документирования действий при построении графа. Используется при отладке.
Abort	Просит менеджер графа закончить текущую операцию кратчайшим способом.
ShouldOperationContinue	Запрашивает менеджер графа о возможности продолжения текущей операции. Приложение не должно вызывать этот метод.

Таблица 4 Интерфейс IGraphBuilder

Методы интерфейса IFilterGraph2

Интерфейс IFilterGraph2 расширяет возможности интерфейсов IFilterGraph и IGraphBuilder.

Методы	Описание
AddSourceFilterForMoniker	Устанавливает фильтры, указанные моникером.
ReconnectEx	Производит повторное соединение контактов и указанным медиа типом.
RenderEx	Выполняет построение ветви графа для воспроизведения данных, используя фильтры уже установленные в граф.

Таблица 5 Интерфейс IFilterGraph2

Примечание.
Моникеры и их применение в DirectShow рассматриваются далее.

Методы интерфейса ICaptureGraphBuilder2

Интерфейс ICaptureGraphBuilder2 предоставляет методы для построения графа сбора и задач построения специализированного графа. Интерфейс предоставляется на отдельном (вспомогательном) объекте построителя графа сбора (Capture Graph Builder).

Методы	Описание
AllocCapFile	Распределяет память для записываемого файла.
ControlStream	Устанавливает время старта и остановки на одном или более потоках записи данных.
CopyCaptureFile	Копирует данные из одного записанного файла в другой. Обычно используется при перезаписи из предварительного распределенного файла в окончательный файл, при этом размер конечного файла уменьшается. Поддерживаются только файлы AVI.
FindInterface	Ищет в графе указанный интерфейс, начиная с указанного фильтра. Метод производит установку и соединение необходимых WDM фильтров при запросе интерфейса с фильтра сбора. Фильтр сбора должен быть WDM фильтром.
FindPin	Возвращает контакт на фильтре соответствующий, указанным критериям.
GetFiltergraph	Возвращает указатель на менеджер графа (IGraphBuilder), который используется построителем графа сбора.
RenderStream	Соединяет выходной контакт фильтра источника с воспроизводящим (отображающим) фильтром с установкой промежуточных фильтров. Может использовать указан-

	ный фильтр компрессора и устанавливать воспроизводящий фильтр по умолчанию.
SetFiltergraph	Предоставляет построителю графа сбора указатель на менеджер графа, который он должен использовать.
SetOutputFileName	Создает записывающую ветвь графа для файлов AVI.

Таблица 6 Интерфейс ICaptureGraphBuilder2

Создание менеджера графа и построителя графа сбора

Создается менеджер графа обычным способом с помощью метода CoCreateInstance.

```
HRESULT hr;

IGraphBuilder* pGraph = NULL;
hr = CoCreateInstance(CLSID_FilterGraph, NULL,
      CLSCTX_INPROC_SERVER, IID_IGraphBuilder,
      reinterpret_cast<void **>(&pGraph));
```

В вызове CoCreateInstance может использоваться один из двух глобальных идентификаторов, показанных в таблице.

CLSID	Описание
CLSID_FilterGraph	Создает менеджер графа в общем рабочем потоке.
CLSID_FilterGraphNoThread	Создает менеджер графа в потоке приложения.

Таблица 7 Идентификаторы CLSID менеджера графа

Обычно используется идентификатор CLSID_FilterGraph. Оба идентификатора создают одинаковые объекты, но они используют различные потоковые модели.

- Идентификатор CLSID_FilterGraph создает менеджер графа в рабочем потоке, который является общим для всех экземпляров менеджера внутри процесса. Этот поток распределяет сообщения посылаемые фильтрами и управляет временем жизни любого окна, которые создают фильтры.
- Идентификатор CLSID_FilterGraphNoThread создает менеджер графа в потоке приложения. При создании менеджера с этим идентификатором, поток, создавший менеджер, должен иметь полный цикл обработки сообщений, в противном случае граф будет неработоспособен. Также, перед завершением приложения, необходимо будет освободить менеджер графа и все объекты графа, фильтры, контакты и так далее.

В приложении MFC можно использовать оба идентификатора без изменения кода приложения или его идеологии. При использовании второго идентификатора запускается на один поток меньше. Этот поток создает окно "ActiveMovie".

После создания менеджера графа создается объект построителя для графа сбора. Несмотря на наличие в имени построителя слова "сбор", он выполняет построение не только графа сбора, но и многих других видов графа.

```
hr = CoCreateInstance(CLSID_CaptureGraphBuilder2
    , NULL, CLSCTX_INPROC, IID_ICaptureGraphBuilder2
    , reinterpret_cast<void **>(&m_pBuilder));
```

Затем вызывается метод, который передает в объект построителя графа сбора указатель на объект менеджера графа. Этим построителю графа сбора указывается, с каким менеджером граф он будет взаимодействовать..

```
hr= m_pBuilder->SetFiltergraph(pGraph);
```

В соответствии с описанием реализуем свой метод InitBuilder в классе CCapture, создающий объект менеджера графа и построителя графа. В этом методе удобно сразу осуществить установку фильтра сбора в граф, поэтому в качестве параметра в метод InitBuilder будем передавать указатель на интерфейс моникера фильтра сбора в качестве параметра. Поиск фильтра сбора и получение его моникера производится с помощью системного перечислителя и будет рассмотрен позже. Также, несколько позже, в разделе " Системный перечислитель и моникеры " будет рассмотрено применение моникеров в DirectShow и их основные свойства.

Примечание

Использование интерфейсов требует постоянного учета ссылок и освобождения интерфейса, когда в нем потребности. Эта процедура является источником постоянных ошибок. В библиотеке ATL 7.0 имеются шаблонные классы интерфейсных указателей, которые избавляют от этой проблемы и позволяют работать с интерфейсными указателями практически также, как с обычными.

К этим классам относятся:

- CComPtr, выполняющий автоматический подсчет ссылок.
- CComQIPtr, выполняет запрос интерфейса и подсчет ссылок.
- CInterfaceArray обеспечивает массив для хранения интефейсных указателей.
- CInterfaceList обеспечивает список для хранения интефейсных указателей.

- CComGITPtr класс для работы с глобальной таблицей нтерфейсов (GIT).

Далее при работе с интерфейсными указателями будут использоваться три первых класса.

При создании менеджера графа и построителя графа сбора возвращаются указатели на интерфейсы IGraphBuilder и ICaptureGraphBuilder2. Из последнего интерфейса можно получить указатель на IGraphBuilder, вызвав метод ICaptureGraphBuilder2::GetFiltergraph, а из интерфейса IGraphBuilder можно запросить указатель на IFilterGraph. Таким образом, можно сделать переменной класса только указатель на интерфейс ICaptureGraphBuilder2 и при необходимости получать остальные. Однако в этом случае при реализации абсолютного большинства методов, каждый раз придется делать несколько однотипных вызовов для получения указателей на основные интерфейсы менеджера графа. Можно поступить иначе, дополнительно к указателю на ICaptureGraphBuilder2, сразу запросить указатель на интерфейс IFilterGraph2, который будет являться переменной класса. Этот указатель, являясь последним в цепи наследования, позволит получить доступ к методам интерфейсов IGraphBuilder, IFilterGraph, IFilterGraph2. Достаточно сделать членами класса два указателя на интерфейсы IFilterGraph2 и ICaptureGraphBuilder2, чтобы получить доступ ко всем интерфейсам построителя и менеджера графа. Такой подход существенно сокращает затраты на дополнительные вызовы при реализации методов класса CCapture.

Объявляем две переменных класса CComPtr<IFilterGraph2> m_pGraph2 и CComPtr<ICaptureGraphBuilder2> m_pBuilder, являющиеся указателями на менеджер графа и на построитель графа сбора. Третья переменная m_bGraph будет указывать на то, что менеджер графа и сам граф создан.

Указатель на моникер рассмотрим позже.

```
HRESULT CCapture::InitBuilder(CComPtr<IMoniker> &
                                    pMoniker)
{
  HRESULT hr = S_OK;

  //Граф уже создан
  ASSERT(m_bGraph == 0);

  //Только один граф
  if(m_bGraph)
    return hr=E_FAIL;

  try
  {
    //Создается объект  менеджера
    CComPtr<IGraphBuilder> pGraph;
    if(S_OK != (hr=CoCreateInstance( CLSID_FilterGraph
```

```
        ,NULL, CLSCTX_INPROC ,IID_IGraphBuilder
        ,reinterpret_cast<void **>(&pGraph))))
    throw hr;


  //Создается  вспомогательный объект  построителя //графа
  сбора
  if(S_OK != (hr=CoCreateInstance(
        CLSID_CaptureGraphBuilder2
        ,NULL, CLSCTX_INPROC ,IID_ICaptureGraphBuilder2
        ,reinterpret_cast<void **>(&m_pBuilder))))
    throw hr;

//Граф создан
m_bGraph = TRUE;

//Передача указателя на объект менеджера в объект
//построителя графа сбора
if(S_OK != (hr=m_pBuilder->SetFiltergraph(pGraph)))
  throw hr;

//Запрашиваем указатель на интерфейс IFilterGraph2
if(S_OK != (hr=pGraph->QueryInterface(&m_pGraph2)))
  throw hr;

//Устанавливаем в граф фильтр сбора видео
if(S_OK != (hr=m_pGraph2->AddSourceFilterForMoniker(
      pMoniker, NULL,
      GetNameFilterFromMoniker(pMoniker).c_str(), &m_pCapV)))
  throw hr;

//Установку фильтра сбора можно выполнить и таким способом, //
связав моникер с объектом

//if(S_OK != (hr=pMoniker->BindToObject(
        0,0,IID_IBaseFilter,
//      reinterpret_cast<void**>(&m_pCapV))))
//throw hr;

//Добавляем  фильтр сбора в граф
//if(S_OK != (hr=m_pGraph2->AddFilter(m_pCapV,NULL)))
//  throw hr;

}
catch(HRESULT)
{
  if(hr != S_OK)
  {
    DestroyGraph();
  }
}
```

```
  return hr;
}
```

Установку фильтра сбора можно произвести двумя способами:

- Первый способ показан в закомментированной части метода, для его использования следует активизировать фильтр сбора с помощью моникера (BindToObject) и затем установить в граф (AddFilter) .
- Второй способ использует указатель на расширенный интерфейс менеджера графа и построителя IFilterGraph2.Если получен указатель на интерфейс IFilterGraph2, то тогда установку фильтра в граф можно произвести, вызвав метод этого интерфейса AddSourceFilterForMoniker, вместо двух вызовов, как показано в закомментированной части метода.

Имя фильтра может быть установлено по собственному усмотрению, либо можно получить "дружественное" имя из моникера. Можно запросить из моникера еще и "отображаемое" имя, которое в действительности лучше никому кроме программистов не показывать. Это имя используется при создании моникера и для предъявления пользователю непригодно. В этом легко убедится, запросив "дружественное" имя для фильтра сбора.

```
LPOLESTR name = 0;
hr = pMoniker->GetDisplayName(pBindCtx,0,&name);
```

Метод возвращает строку, которая явно не пригодна в качестве имени устройства.

```
"@device:pnp:\\?\pci#ven_109e&dev_036e&subsys_00000000&rev_11#4&
2e98101c&0&50f0#{65e8773d-8f56-11d0-a3b9-00a0c9223196}\global"
```

В методе GetNameFilterFromMoniker показано, как получить "дружественное" имя фильтра.

```
wstring CCapture::GetNameFilterFromMoniker(
                              CComPtr<IMoniker> & pMoniker)
{
  //Проверка имени
  CComPtr<IPropertyBag> pBag;

  //Связаться с хранилищем объекта (фильтра)
  HRESULT hr = pMoniker->BindToStorage(0, 0,
          IID_IPropertyBag,reinterpret_cast<void **>(&pBag));

  if (FAILED(hr))
    return wstring();

  VARIANT var;
  VariantInit(&var);
```

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

  if (FAILED(hr))
    return wstring();

  return wstring(var.bstrVal);
}
```

Если "дружественное" имя отсутствует можно попытаться прочитать описание устройства.

```
  hr = pBag->Read(L" Description", &var, NULL);
```

Кроме "дружественного" имени также можно прочитать строку CLSID, что может быть полезно в некоторых случаях.

```
  hr = pBag->Read(L"CLSID", &var, NULL);
```

Других имен свойств моникера обнаружить не удалось, в документации они присутствуют только в небольших примерах кода на определенную тему.

Примечание

Метод возвращает строку типа wstring библиотеки STL. Использование строк и других средств библиотеки STL значительно облегчает программирование классов, где нет прямой необходимости в использовании MFC. Подключение необходимых средств библиотеки STL произведено в файле Afx-Std.h.

Далее необходимо создать метод для поиска, установленного в системе, фильтра сбора данных. Это можно сделать с помощью системного перечислителя, который перечисляет моникеры устройств в определенной категории. Когда пользователю будет предоставлена возможность выбора устройства, можно будет вызвать метод InitBuilder. В конце метода InitBuilder, следует добавить вызов метода AddToRot, как показано ниже.

```
#ifdef _DEBUG
AddToRot(m_pGraph2, &m_dwReg);
#endif //_DEBUG
```

Это позволит просмотреть реализованный граф с помощью утилиты редактора графа.

Системный перечислитель и моникеры

В составе DirectShow имеется множество фильтров, которые поддерживают большое количество различных форматов данных и обеспечивают высокий уровень аппаратной независимости. Фильтры, предоставляемые DirectX, всегда присутствуют в системе. Все фильтры DirectX зарегистрированы в операционной системе (регистре). Фильтры делятся на категории, по типу применения. Таблицы категорий фильтров приведены в справочном разделе. Каждый фильтр имеет свой идентификатор CLSID и относится к определенной категории, которая тоже имеет свой идентификатор CLSID. Фильтры сторонних разработчиков тоже регистрируются в системе, но в процессе удаления приложения они могут быть удалены и сняты с регистрации. Поэтому список фильтров DirectShow при просмотре утилитой редактора графа может изменяться при установке и удалении программ. Фильтры сторонних производителей могут регистрироваться, а затем использоваться в графе при условии, что они выполнены с соблюдением соглашений DirectShow. Фильтры выполнение по другим технологиям, даже производства Microsoft, DirectShow не поддерживает.

Кроме этих фильтров существует вторая группа фильтров DirectShow, которая представляет устройства WDM и фильтр KSProxy. На системе пользователя может быть установлено несколько устройств и все они могут поддерживаться одним фильтром. Это возможно для устройств поддерживающих WDM драйвер и KSProxy фильтр, которые позволяют применять общие фильтры для разных устройств. Один фильтр из этой группы может представлять несколько различных устройств и имеет один CLSID для всей группы устройств. Фильтры, зарегистрированные в системе, требуют определенного механизма для их распознавания, в особенности это касается фильтров, которые представляют несколько устройств и имеют один общий CLSID. Например, фильтр сбора видеоинформации телеприемника, который расположен на карте PCI и фильтр звукового интегрированного устройства "территориально" размещаются в одном файле фильтра ksproxy.ax и имеют один CLSID.

Механизмом распознавания устройств является системный перечислитель, который является служебным компонентом DirectShow и создается приложением. При перечислении, если фильтр найден, возвращается моникер фильтра, позволяющий в дальнейшем использовать этот фильтр. Моникер хранит информацию об объекте и предоставляет её для использования с целью управления объектом. Он идентифицирует свой объект, которым в данном случае, является устройство. Эти фильтры не могут (не должны) создаваться прямым вызовом функции CoCreateInstance, они активизируются только с помощью моникера. Все фильтры, представляющие устройства должны создаваться с помощью системного перечислителя. Только тогда можно точно определить устройство, которое производитель предлагает для использова-

ния. Это связано с тем, что фильтр ksproxy.ax может содержать фильтр устройства, которое отсутствует на карте. Например, производитель выпускает в продажу несколько вариантов одного многофункционального устройства и сопровождает устройство одним фильтром ksproxy, в котором моникер указывает на COM объект представляющий необходимое устройство. Иллюстрацией может служить фильтр настройки телеприемника, который имеет отдельный идентификатор CLSID и располагаться в отдельном файле. Создание объекта с помощью вызова функции CoCreateInstance в таком случае вполне возможно и выполняется успешно, но результат может быть совсем не тот, что при использовании системного перечислителя. Экземпляр фильтра оказался не идентичен экземпляру, созданному с использованием системного перечислителя. Во всяком случае, такой результат был получен при экспериментальном исследовании, когда неожиданно удалось создать фильтр с цифровым спутниковым приемником, в то время как устройство аналогового телевидения не имеет такого приемника. Учитывая, что аналоговый приемник и спутниковый приемник реализованы на чипсете одного типа Conexant 878, можно предположить, что фильтр ksproxy обеспечивает обе разновидности чипсета, а моникер указывает на "правильное" устройство.

Фильтры, которые имеют свой отдельный CLSID и не являются фильтрами, которые представляют устройства, создаются как обычные COM объекты с помощью вызова функции CoCreateInstance. Они также могут создаваться с помощью системного перечислителя и моникера. Этот путь более сложный, чем прямое создание объекта, но он бывает необходим, когда нужно предоставить выбор фильтра пользователю.

Моникер достаточно часто используемый объект в технологии COM, но плохо освещен в доступной литературе. В модели COM нет способа идентификации конкретного объекта. Если существует несколько одинаковых объектов, которые обрабатывают разные данные, то возникает задача идентификации объектов. Моникер призван решать эту задачу, он идентифицирует определенный объект и сам является объектом COM. Для идентификации объекта моникер сохраняет определенную информацию о нем. В состав этой информации входит идентификатор объекта CLSID и текстовая строка с уникальным содержанием. Информация об объекте записывается в моникер однократно при его создании, и таким образом моникер может указывать только на определенный объект. Текстовая строка может содержать имя устройства или путь к файлу, тогда это будет файловый моникер. Моникер предоставляет интерфейс IMoniker.

Моникер может создать (активизировать) объект и возвратить его интерфейс или активизировать место хранения объекта. В первом случае возвращается указатель на объект, а во втором указатель на место хранения объекта. Для связи с хранилищем используется метод IMoniker::BindToStorage, а для создания самого объекта метод

IMoniker::BindToObject. Все устройства и фильтры, установленные в системе идентифицируются и активизируются моникерами с помощью системного перечислителя.

Моникер устройства может быть создан на основе строки, предоставляемой контексту связывания. Контекст связывания это вспомогательный COM объект, создающий и сохраняющий временную информацию в процессе создания моникера, он предоставляет интерфейс IBindCtx. Объект контекста предоставляет доступ к этой информации, но потребность в ее изменении возникает достаточно редко. Например, моникер по умолчанию, для категории фильтров сбора видео, может быть реализован следующим образом.

```
//strMon строка, отображаемого имени
WCHAR strMon[] =
L"@device:*:{860BB310-5D01-11d0-BD3B-00A0C911CE86}";

//Создание контекста
IBindCtx *pbc;
hr = CreateBindCtx(0, &pbc);

ULONG chEaten = 0;
IMoniker *pMoniker = 0;

//Создание моникера
hr = MkParseDisplayName(pbc, strMon, &chEaten, &pMoniker);
pbc->Release();

if (SUCCEEDED(hr))
{
     //Далее можно вернуть имя или активизировать фильтр.
     pMoniker->Release();
}
```

Существуют и другие разновидности моникеров, но создавать их не придется, а только использовать. Подробней ознакомиться с моникерами, их созданием и использованием можно в статье опубликованной в журнале RSDN (http://www.rsdn.ru/article/com/monikers.xml).

Перечисление устройств и фильтров

Перечисление устройств и фильтров в приложении DirectShow является одной из первых задач после запуска программы. Прежде всего, приложение должно предоставить пользователю список доступных устройств, с которыми оно может работать или предупредить о том, что в системе отсутствуют необходимые устройства и работа приложения невозможна. Перечисление устройств удобно производить в том классе, где пользователю будет предъявляться информация об устройствах. Кроме устройств, пользователю необходимо предоставить данные об установленных в системе фильтрах кодеков, которые он может выбрать и подключить для записи сжатых файлов. Имея та-

кую информацию, он сможет определить достаточно ли системных компонентов или необходима установка дополнительных фильтров или кодеков.

Таким образом, возникает задача определения наличия системных устройств на компьютере пользователя. Для ее решения DirectShow предоставляет два компонента:

- Системный перечислитель, который производит поиск фильтров в определенных категориях.

- Карта фильтров, поиск в которой осуществляется по критериям заданным приложением. Поиск фильтров производится в регистре. Карта фильтров работает менее эффективно по сравнению с перечислителем. Карта используется чаще всего, когда требуется найти фильтр с определенным набором медиа типов, но не принадлежащий к какой либо чистой категории. Например, фильтр декодера или фильтр отображения видео или звука. Поиск в карте фильтров достаточно непроизводительная процедура.

Все фильтры DirectShow делятся на категории в соответствии с назначением. Каждая категория кодируется уникальным числовым идентификатором (CLSID). Для перечисления фильтров в определенной категории необходимо передать его CLSID в качестве параметра методу ICreateDevEnum::CreateClassEnumerator. В приложении, в разделе "Категории фильтров" (таблицы 1-4) приведены CLSID фильтров DirectShow. Они определены в файле Uuids.h и некоторых других. Этот файл включается в проект, когда в него включается файл Dshow.h. Для облегчения работы с оригинальной документацией наименования категорий приведены в английской транскрипции, а свободный перевод дан во второй графе.

Реализация перечислителя устройств

Системный перечислитель предоставляет единый подход для перечисления фильтров, зарегистрированных в системе пользователя. Более того, он различает разные устройства, поддерживаемые одним и тем же фильтром.

Для использования системного перечислителя следует:

- Создать системный перечислитель, вызвав функцию CoCreateInstance с идентификатором класса CLSID_SystemDeviceEnum.

- Создать перечислитель для определенной категории, вызвав метод ICreateDevEnum::CreateClassEnumerator с указанием категории фильтра. Этот метод возвратит интерфейсный указатель IEnumMoniker, если категория не пустая. Если категория пустая метод возвращает код S_FALSE.

- Для перечисления всех моникеров используется метод IEnumMoniker::Next, который возвратит интерфейсный указатель IMoniker. При достижении конца перечисления метод возвратит S_FALSE.

- Для возврата имени устройства вызывать метод IMoniker::BindToStorage.

- Для инициализации фильтра и включения его в граф последовательно вызываются методы IMoniker::BindToObject и IFilterGraph::AddFilter. Также можно использовать метод IFilterGraph2::AddSourceFilterForMoniker, который объединяет, указанные два метода и сразу производит установку фильтра в граф по его моникеру.

Основной метод системного перечислителя, позволяющий извлечь информацию об устройстве это IMoniker::BindToStorage. Рассмотрим подробнее его применение. Метод возвращает интерфейсный указатель на хранилище, в котором хранится информация об объекте. Хранилище также является объектом COM.

```
HRESULT BindToStorage(
IBindCtx * pbc,
IMoniker * pmkToLeft,
REFIID riid,
void ** ppvObj
);

Параметры

pbc
 (in).
Указатель на интерфейс контекста IBindCtx, используемый в про-
цессе операции связывания. Если параметры контекста не нужны для
операции, указатель должен быть равен NULL.

pmkToLeft
 (in).
Указатель на левый моникер, если используемый моникер  является
частью композитного моникера или NULL.

riid
 (in).
Ссылка на идентификатор интерфейса хранилища.
Обычно хранилище предоставляет один из интерфейсов IStorage,
IStream, ILockBytes. Системный перечислитель использует другой
интерфейс хранилища IPropertyBag, который реализован как контей-
нер для хранения текста.  Он рекомендуется для использования в
```

элементах ActiveX, поэтому текст в нем храниться как переменная типа VARIANT. Интерфейс имеет два метода Read и Write.
Метод Read может читать три свойства. Имя свойства является текстовой строкой.

- FriendlyName. Имя устройства.
- CLSID. CLSID фильтра.
- Description. Описание устройства. Используется для видеокамер.
- DevicePath. Уникальная строка, однозначно идентифицирующая устройство. Может использоваться для распознавания двух одинаковых устройств.

ppvObj
(out).
Адрес указателя на интерфейс IPropertyBag. Если произошла ошибка при вызове метода адрес указателя возвращается равным нулю. После окончания работы с объектом его необходимо освободить, вызвав метод IUnknown::Release.

Примечание

При вызове методов системного перечислителя необходимо проверять результат вызова. Все COM объекты возвращают код типа HRESULT, для ускоренной проверки, которого предоставляются два макроса SUCCEEDED и FAILED. Первый макрос возвращает TRUE для всех кодов, которые больше или равны нулю. Второй макрос возвращает TRUE для всех кодов, которые меньше нуля, то есть для кодов ошибок. Эти макросы достаточно часто используются в DirectShow, но здесь скрыт источник неприятностей. Некоторые методы возвращают код S_FALSE, значение которого равно единице. Код S_FALSE говорит о том, что при удачном завершении результат операции не достигнут. Для каждого метода, который может вернуть это значение, результат операции может интерпретироваться по-разному, все зависит от ситуации, не всегда возврат S_FALSE может быть ошибкой или нормальным завершением. Например, при попытке разъединения уже разъединенных фильтров метод IFilterGraph::Disconnect возвращает код S_FALSE, что не является ошибкой, если проектируемый метод разъединяет фильтры. В таком случае проектируемый метод должен интерпретировать код S_FALSE как S_OK, результат достигнут, и не важно были фильтры разъединены ранее или это сделал проектируемый метод. Возможны ситуации, когда использо вание одного из этих макросов с кодом S_FALSE может неправильно интерпретироваться и в последующем вызвать исключение. Если метод возвращает код S_FALSE, то макросы SUCCEEDED и FAILED лучше не использовать или применять после тщательного анализа ситуации.

Приступим к непосредственной реализации системного перечислителя. Исходя из общей решаемой задачи, прежде всего, следует определить наличие устройств сбора видео и звуковой информации, а также видео и звуковых кодеков, которые пользователь может применять при записи файлов. Создадим метод для перечислителя видео и звуковых устройств в классе первой вкладки CTVshowView. Напишем метод myEnumDevice, который будет перечислять устройства сбора видеоданных и звуковых данных. Хранить моникеры будем в двух массивах класса CInterfaceArray библиотеки ATL, что избавляет от заботы по их освобождению. Назначим переменные массивов для видео устройств m_arIMonDev_V и для звуковых устройств m_arIMonDev_A. Вызов метода произведем в конце метода CTVshowView::OnInitialUpdate().

```
void CTVshowView::myEnumDevice(void)
{
  HRESULT hr;
  string serr;
  //Указатель на системный перечислитель
  CComPtr<ICreateDevEnum> pDevEnum;

  //Указатель на перечислитель моникеров
  CComPtr<IEnumMoniker> pEnumCat;

  //Указатель на интерфейс хранилища свойства объекта
  CComPtr<IPropertyBag> pPropBag;

  //Указатель на моникер
  CComPtr<IMoniker> pMoniker;

  //Имя устройства
  VARIANT varName;

  //Инициализация переменной
  varName.vt = VT_BSTR;

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

    if(hr != S_OK)
      throw hr;

    ULONG cFetched;

    //Создается класс перечислителя для видеоустройств.
    //Категорию фильтра выбираем из таблицы категорий.
```

```
hr = pDevEnum->CreateClassEnumerator(
            CLSID_VideoInputDeviceCategory
                  , &pEnumCat,0);

if(hr == S_FALSE)//Категория пустая
  AfxMessageBox(_T("Не найдено ни одно
                      видеоустройство "));

if(FAILED(hr))
  throw hr;

if(hr== S_OK)
{
  //Перечисление видеоустройств в цикле.
  while(
    pEnumCat->Next(1, &pMoniker, &cFetched) == S_OK)
  {
    //Получить указатель на хранилище
    if(S_OK == pMoniker->BindToStorage(0,
         0,IID_IPropertyBag,
          reinterpret_cast<void **>(&pPropBag)))
    {
      //Читаем дружественное имя устройства из
      //хранилища
      if(S_OK == pPropBag->Read(L"FriendlyName"
                                  , &varName, 0))
      {
        //Преобразуем строку из BSTR (макрос ATL7)
        CString sName = COLE2T(varName.bstrVal);

        //Записываем моникер в массив
        size_t idx = m_arIMonDev_V.Add(pMoniker);

        //Сохраняем имя, сохраняем индекс массива,
//хранящего моникеры
        myInsertNameVideo(
                      sName,static_cast<UINT>(idx));
      }
      pPropBag = NULL;
    }
    pMoniker = NULL;
  }
}

pEnumCat = NULL;
//Создается класс перечислителя для Audio
//возвращается моникер устройства
hr = pDevEnum->CreateClassEnumerator(
                  CLSID_AudioInputDeviceCategory
                  , &pEnumCat, 0);

if(hr == S_FALSE)//Категория пустая
```

```
    {
      serr =
      _T("Не найдено ни одно звуковое устройство ");
      throw hr;
    }

    if(FAILED(hr))
          throw hr;

    if(hr== S_OK)
    {
      //Перечисление аудиоустройств в цикле.
      while(
        pEnumCat->Next(1, &pMoniker, &cFetched) == S_OK)
      {
        //Получить указатель на хранилище
        if( S_OK == pMoniker->BindToStorage(0,0
                  ,IID_IPropertyBag,
                  reinterpret_cast<void **>(&pPropBag)))
        {
          //Читаем дружественное имя устройства из
          //хранилища
          if(S_OK == pPropBag->Read(L"FriendlyName"
                                      , &varName, 0))
          {
            //Преобразуем строку из BSTR (макрос ATL7)
            CString sName = COLE2T(varName.bstrVal);

            //Записываем моникер в массив
            size_t idx = m_arIMonDev_A.Add(pMoniker);

            //Сохраняем имя, сохраняем индекс массива,
    //хранящего моникеры
            myInsertNameAudio(sName
                                ,static_cast<UINT>(idx));
          }
          pPropBag = NULL;
        }
        pMoniker = NULL;
      }
      //Установить нулевую строку списка видимой
      m_listAudio.SetCurSel(0);
    }
  }
  catch(HRESULT h)
  {
    serr.insert(0,"Метод myEnumDevice\n\r");
    CTVshowApp::myMessageError(h, serr);
  }
}
```

Примечание

В данном методе и предыдущем обработка ошибок осуществляется с использованием блоков try и catch. Во многих работах по программированию можно встретить мнение, что применение блоков try catch "загрязняет и утяжеляет" код. С этим утверждением нельзя согласиться, во многих случаях блок try и catch может сделать код более прозрачным для понимания и обеспечить вызов единого метода для вызова сообщения об ошибке. Если проектируемый метод содержит последовательные множественные вызовы функций и после каждого очередного вызова функции требует выхода из тела метода при возникновении исключения, то один единственный блок try catch помогает просто и прозрачно реализовать такой метод, без использования повторяющихся конструкций if else. Как и любое другое средство языка C++, блоки try catch желательно использовать без излишеств.

В классе CTVshowApp myMessageError написан статический метод, который производит расшифровку результата выполнения метода (код HRESULT) и добавляет свою строку, в которой можно указать точку возникновения ошибки и дополнительное сообщение. Расшифровка результата выполнения производится с помощью метода DirectShow AMGetErrorText. Данный метод локализован для языка используемого в системе, в данном случае русского. Это неплохо, но перевод часто не соответствует английскому тексту сообщения или, что еще хуже, смыслу кода ошибки, пример будет приведен далее. Поэтому следует дополнить метод выводом исходного кода ошибки. В программе многие проверки и блокировки будут опускаться, поскольку она предназначена только для обучения, проверки решений и проведения экспериментов. В коммерческой программе блоки try catch могут оказаться очень полезными.

```
void CTVshowApp::myMessageError(HRESULT hr
                                , string strErr)
{
  string s;
  TCHAR buf[MAX_ERROR_TEXT_LEN];
  if(AMGetErrorText(hr,buf,MAX_ERROR_TEXT_LEN))
  {
    s = buf;
    s+= "\n\r" + strErr;
  }
  else
    s = "Сообщение об ошибке не расшифровано";

  ::MessageBox(NULL,s.c_str(),NULL,MB_ICONERROR |
                   MB_SYSTEMMODAL | MB_OK);
}
```

Метод IEnumXXXX::Next возвращает массив элементов из перечисляемой последовательности. В данном случае возвращается один элемент. Подроб-

ней с описанием метода можно ознакомиться в MSDN. Для поиска описания используйте строку IEnumXXXX.

Найденные имена устройств необходимо сохранить и отобразить в списке для выбора пользователем. В качестве списка можно выбрать для звуковых устройств выпадающий список (Combo Box). Для видео устройств такой список не очень удобен, поскольку желательно отобразить пользователю все устройства, тем более, что обычно их не много. Возможно, придется выбирать два устройства одновременно. Также неплохо для устройства отобразить дополнительную информацию, например его состояние. Поэтому для видео устройств выберем управляемый список (List Control) класса CListCtrl, который имеет широкие возможности по отображению списка в различных вариантах. Этот класс списка также позволяет отобразить состояние устройства с помощью иконки, что может быть полезно в некоторых случаях. Выберем в свойствах вид списка report и удалим сортировку (No Sort Header). Для списка звуковых устройств также следует установить значение свойства сортировки FALSE. В редакторе ресурсов установим шаблоны списков в форме для класса CTVshowView и назначим две защищенные переменные m_listVideo и m_listAudio в этом же классе.

При перечислении устройств в категории может быть обнаружено не одно устройство, либо может быть обнаружен фильтр, для которого нет аппаратного устройства. В программе все эти случаи не обрабатываются. Однако сузить поиск, если точно не известно имя устройства, или другой признак идентификации устройства, который предоставляет хранилище, можно таким способом.

При перечислении устройств запросить у объекта специфический интерфейс перед запросом имени. Например, для устройства аналогового телевидения это может быть интерфейс видео декодера IAMAnalogVideoDecoder.

```
CComPtr<IAMAnalogVideoDecoder> pVD;
hr = pMoniker->BindToObject(0,0,IID_IAMAnalogVideoDecoder
                    ,reinterpret_cast<void**>(&pVD)));
```

Если интерфейс присутствует, то можно предъявлять устройство пользователю.

Второй вариант предусматривает предоставление пользователю всех устройств в данной категории и обработку ошибок при выборе устройства. Устройство, не прошедшее проверку, в списке обозначается иконкой как не работоспособное, а моникер можно удалить из массива или присвоить ему нулевое значение, предварительно освободив (указателю CComPtr только присваивается нуль). Если все устройства работоспособны, программа должна построить несколько графов, если это необходимо.

Методы myInsertNameVideo и myInsertNameAudio реализованы в классе CTVshowView и устанавливают строки с именами устройств в указанные списки, а индексы массивов записываются как параметры соответствующих строк. Метод CTVshowView::myInitListVideo выполняет первичную инициализацию списка и вызывается в OnInitialUpdate. Их реализация не имеет каких-либо особенностей и с ней можно ознакомиться в исходных кодах проекта.

Моникеры, как и любой объект COM, необходимо освободить после окончания работы с ним. Моникеры устройств и фильтров могут использоваться в течение всей работы приложения, и освобождать их следует перед закрытием программы. Лучшее место для этого обработчик события WM_CLOSE, реализованный в классе CMaineFrame. В деструкторе класса CTVshowView освобождать моникеры нельзя, поскольку к моменту вызова деструктора списки уже разрушаются, и обращение к ним вызывает исключение. Поскольку для хранения моникеров используются массивы CInterfaceArray, то эта проблема решается автоматически без участия программиста.

Для поверки производим компиляцию проекта и запускаем программу на исполнение. В списках должны отобразиться устройства установленные в системе пользователя (рис. 1).

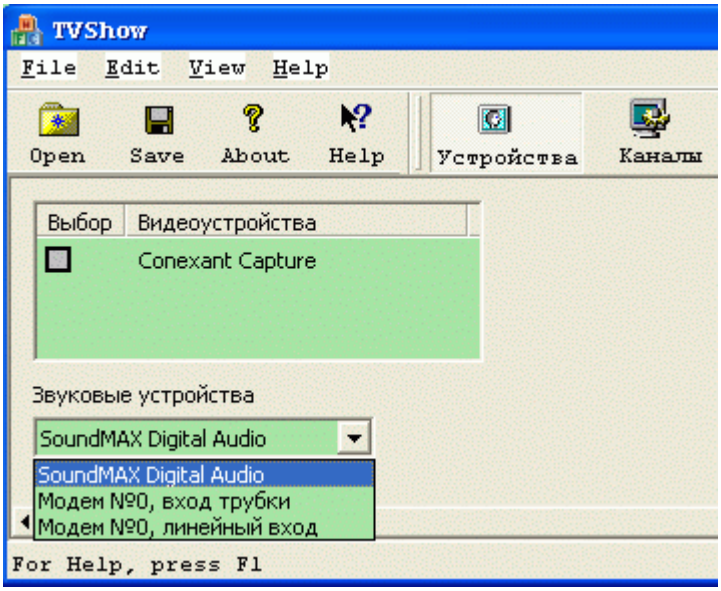


Рис.2. 17 Окно программы после установки фильтра сбора

Реализация перечислителя фильтров (кодеков)

Перечислитель фильтров реализуется аналогично перечислителю устройств, необходимо только заменить категории для перечисления на CLSID_VideoCompressorCategory и CLSID_AudioCompressorCategory и дать имя методу myEnumFilter. Запись имен фильтров и указателей на моникеры

реализуется также как и в случае перечислителя устройств. В отличие от перечислителя устройств, необходимо решить какие фильтры предоставить пользователю для выбора и установки в граф . Система всегда имеет в своем составе несколько кодеков. Другая часть кодеков поставляется в составе DirectX и Windows Media Player. Установка новых версий DirectX и Windows Media Player может повлечь за собой установку дополнительных кодеков. Установка приложений использующих кодеки также может увеличить их число, причем некоторые кодеки могут быть предназначены только для работы с этим приложением, другие же могут работать с DirectShow. Не все приложения при удалении корректно удаляют свои компоненты, в результате в системе могут существовать десятки и даже сотни разных кодеков.

Все кодеки могут быть разделены на три группы:

- Тип кодеков видео для Windows или просто видео кодеки, которые загружаются менеджером видеокомпрессора (VCM) или звукового компрессора (ACM). Эти кодеки устанавливаются в граф с помощью фильтра обертки AVI декомпрессора. Применение кодеков, реализованных по этой технологии, не рекомендуется.

- Кодеки, выполненные как фильтры DirectShow. Многие производители выполняют кодеки как фильтры DirectShow, например, фильтр Frauenhofer MP3. Эти фильтры используются в графе обычным способом, при условии, что производитель фильтра предоставляет его для свободного использования. Есть фильтры, которые имеют защиту и могут использоваться только в составе своего приложения.
  Дополнительное ограничение налагается на фильтры производства Microsoft, такие как Windows Media™ Audio, Windows Media Video и фильтр Microsoft MPEG-4. Эти фильтры не могут быть установлены в граф вручную, а только при использовании фильтров ASF Reader и ASF Writer. Таким образом, данные фильтры можно использовать только при записи (воспроизведении) файлов в формате ASF.

- Кодеки, выполненные как объекты DMO (DirectX Media Objects). Эти кодеки могут использоваться как в графе DirectShow, так и в приложениях не использующих DirectShow.
  Однако их применение, как и в предыдущем случае, ограничено.

Для решения тривиальной задачи записи видеофайлов пользователю следует предоставить фильтры звуковых кодеков из категории CLSID_AudioCompressorCategory и видео кодеков из категории CLSID_VideoCompressorCategory. Кодеки, предъявляемые пользователю, следует подвергнуть отбору, поскольку:

- Не все кодеки, поставляемые с Windows, могут работать в приложении DirectShow.
- В системе пользователя может быть множество кодеков сторонних производителей, часть из которых может быть предназначена только для работы в конкретном приложении или при соблюдении определенных условий.
- Пользователь может не понимать, по каким признакам выбирать кодек. Появление специфического сообщения об ошибке при установке не надлежащего кодека пользователь может воспринять как общую ошибку приложения.

Можно ввести режим отображения для опытного пользователя, в котором показать все кодеки, а в обычном режиме показывать только те кодеки, которые полностью функционируют в составе приложения. Звуковые кодеки обычно работают без проблем, и нужды в их отборе нет.

Примечание.

Установку кодеков сторонних производителей следует производить, соблюдая некоторые предосторожности. Даже непродолжительные опыты по применению некоторых кодеков показали их небезопасность для системы и приложения**. Для безопасной установки кодеков на платформе Windows XP следует обязательно сделать контрольную точку для восстановления системы, поскольку удаление нежелательного кодека обычным способом не всегда возможно**. Приходится искать его компоненты, вручную очищать регистр и тратить на это много времени. Кроме того, при установке кодека могут быть заменены некоторые системные файлы, что крайне нежелательно. При установке некоторых кодеков отмечались нарушения в работе среды разработки VC++.Net 2003 в режиме Debug при отладке приложения DirectShow. После отката системы в состояние до установки кодеков, нарушения исчезали.

Для отображения фильтров используем два выпадающих списка (CComboBox), один для видео кодеков и один для звуковых кодеков. Разместим их в классе для второго представления CFview2. Для перечисления кодеков напишем метод CFView2::myEnumFilter аналогичный методу для перечисления устройств. Вызов метода производится в методе CFView2::OnInitialUpdate. После трансляции и запуска программы на исполнение на вкладке фильтры можно увидеть списки кодеков установленных в системе. В системе автора отображаются кодеки, показанные на рисунке 2.6.

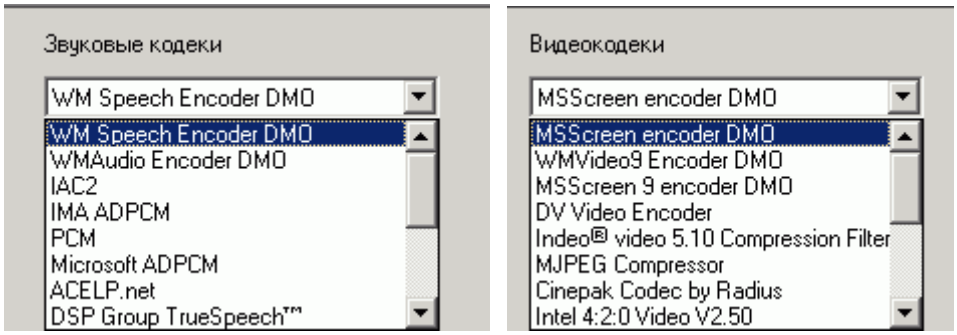


Рис.2. 18 Списки системных кодеков.

Решение по дополнительной селекции отображаемых кодеков следует принимать в каждом конкретном случае отдельно.

Реализация поиска в карте фильтров

При необходимости поиска фильтра по определенным критериям используется карта фильтров (Filter Mapper). Поиск в карте работает медленней, чем системный перечислитель. Использовать карту следует для поиска фильтров, которые не имеют чистой категории. В этом случае можно отыскать фильтр по комбинации медиа типов, категориям контактов и другим критериям. Например, для поиска в карте фильтров, можно использовать метод показанный ниже. В приложении он не используется. Для подробного изучения поиска в карте фильтров можно воспользоваться примером приложения в документации по адресу (SDK root)\Samples\C++\DirectShow\Misc\Mapper.

```
void CTVshowApp::myFindMapper(void)
{
  IFilterMapper2 *pMapper = NULL;
  IEnumMoniker *pEnumCut = NULL;
  //Имя устройства
  VARIANT varName;
  //Инициализация переменной
  varName.vt = VT_BSTR;

  HRESULT hr = CoCreateInstance(CLSID_FilterMapper2,
              NULL, CLSCTX_INPROC, IID_IFilterMapper2,
              reinterpret_cast<void **>(&pMapper));

  if (FAILED(hr))
  {
    //Обработка ошибок
  }

  GUID arrayInTypes[2];
  arrayInTypes[0] = MEDIATYPE_Video;
  arrayInTypes[1] = MEDIASUBTYPE_dvsd;
```

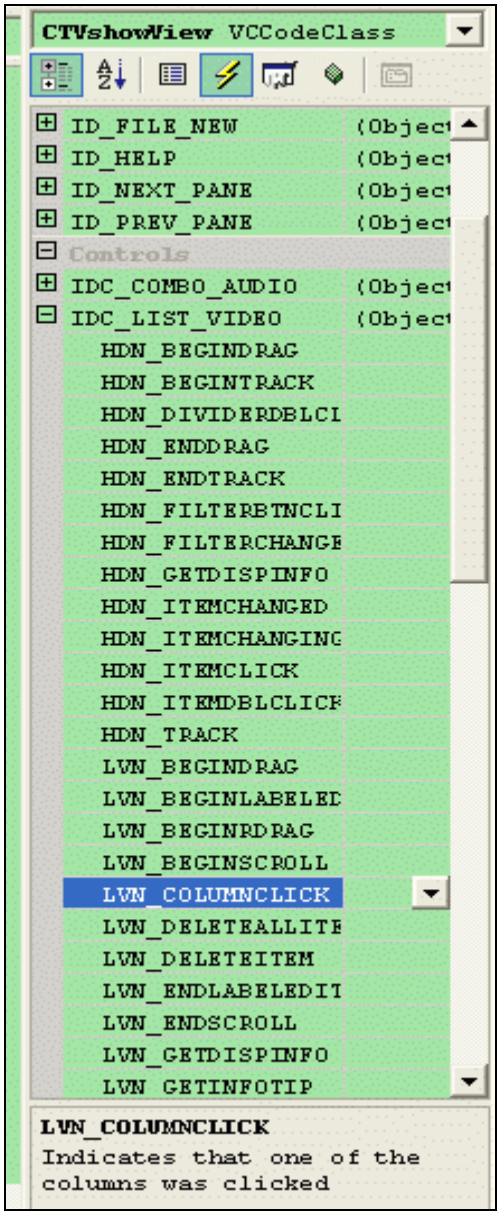
```
hr = pMapper->EnumMatchingFilters(
      &pEnumCut,
      0,     //Резерв.
      TRUE,  //Искать в точном соответствии с медиа
      //типом, категорией и средой контактов
      MERIT_DO_NOT_USE+1, //Минимальное значение
                          //предпочтения.
      TRUE,     //Должен ли иметь фильтр не менее
          //одного входного контакта
      1,        //Число пар медиа типов и подтипов
                //для входного контакта фильтра
      arrayInTypes, //Массив медиа типа и подтипа для
              //входного контакта фильтра,
                      //размер массива равен числу пар
                      //медиа типов, умноженному на два
      NULL,        //Носитель для входного контакта.
      NULL,        //Категория входного контакта.
      FALSE,       //Отображающий фильтр (renderer)?
      TRUE,        //Должен ли иметь фильтр не менее
              //одного выходного контакта?
      0,           //Число пар медиа типов и подтипов
                      //для выхода.
      NULL,        //Массив медиа типа и подтипа для
              //выходного контакта фильтра.
      NULL,        //Носитель для выходного контакта.
      NULL);       //Категория выходного контакта.

//перечисление моникеров.
IMoniker *pMoniker;
ULONG cFetched;
while (pEnumCut->Next(1, &pMoniker, &cFetched) ==
                                            S_OK)
{
  IPropertyBag *pPropBag = NULL;
  hr = pMoniker->BindToStorage(0, 0, IID_IPropertyBag,
                                  (void **)&pPropBag);

  if (SUCCEEDED(hr))
  {
    //Читаем дружественное имя  фильтра компрессора из //храни-
    лища
    if(S_OK == pPropBag->Read(L"FriendlyName", &varName, 0))
  {
    //Преобразуем строку из BSTR (макрос ATL7)
    CString sName = COLE2T(varName.bstrVal);
  }

    //Создание (активизация) экземпляра фильтра
    IBaseFilter *pFilter;
    hr = pMoniker->BindToObject(NULL, NULL,
                      IID_IBaseFilter,(void**)&pFilter);
```

```
      //Здесь следует добавить фильтр в граф для
      //дальнейшего использования
      ……..
      //Освобождение
        pPropBag->Release();
      }
      pMoniker->Release();
    }
    //освобождение
    pMapper->Release();
    pEnumCut->Release();
  }
```

Критерии поиска, используемые в методе, установлены для фильтра декодера цифрового видео (DV Video Decoder).

Установка в граф фильтра сбора

Установка в граф фильтра сбора производится после того, как пользователь выбрал из списка соответствующее устройство. Это событие (NM_CLICK) обрабатывается в обработчике OnNMClickListVideo в классе CTVshowView. Данный обработчик будет являться основным методом для инициализации фильтрового графа. Он будет определять основную последовательность построения графа.



Создание обработчика для списка класса CListCtrl в среде VC 7.0 не совсем очевидно. Для этого нужно для класса CTVshowView открыть вкладку свойств и отыскать идентификатор списка IDC_LIST_VIDEO. Раскрыв список событий для этого идентификатора, нужно найти требуемое событие и назначить обработчик. На рисунке 2.19 показана соответствующая вкладка и выбрано произвольное событие из-за большого размера списка событий.Далее в обработчике отыскивается индекс массива. По индексу массива извлекается указатель на моникер. Затем вызывается метод myInitBuilder, который выполняет построение менеджера графа и устанавливает в граф фильтр сбора.

Рис.2. 19 Создание обработчика для списка устройств.

Текст метода OnNMClickListVideo приводится ниже.

В программе не осуществлена поддержка

функционирования нескольких устройств одновременно. При необходимости такой поддержки следует провести анализ массива моникеров для отбора реально работающих устройств и создание нескольких графов. В методе производится только проверка на работоспособность устройства, удаление моникера из массива и удаление строки с именем устройства из списка.

```
void CTVshowView::OnNMClickList Video(NMHDR *pNMHDR,
                            LRESULT *pResult)
{
  //Преобразуем в указатель на структуру NMITEMACTIVATE
  LPNMITEMACTIVATE lpnmitem =
        reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR) ;

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

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

  if(uFlags & LVHT_ONITEMSTATEICON &&
            !m_listVideo.GetCheck(lpnmitem->iItem))
  {
    //В иконке
    //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
      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 захвата
                                                  звука"));
      return;
    }
  }
  *pResult = 0;
}
```

Если метод InitBuilder выполнился с ошибкой, следует разрушить граф и произвести подготовку к повторной попытке пользователя создать граф.

Разрушение графа

Перед окончанием работы приложения или при полной перестройке графа, старый граф должен быть разрушен, все фильтры удалены и освобождены все указатели на интерфейсы. Разрушение графа производится после его остановки. Простейший путь разрушения графа и удаления всех фильтров это освобождение старого менеджера и создание нового. После этого остается только освободить все интерфейсы, которые на момент разрушения графа были не освобождены. Если интерфейсы используют указатели классов CComPtr, то при закрытии приложения их не нужно освобождать, но, если приложение не закрывается, а только перестраивается граф, то для повторного использования указателей в объекте сбора следует все указатели на интерфейсы освободить, приравняв их нулю.

Данный способ разрушения графа работает при условии, что он полностью построен. Хотя бы один выходной контакт фильтра сбора должен быть обработан для выполнения конечной задачи, записи файла или предварительного

просмотра. Если граф построен частично, то для удаления фильтров следует использовать альтернативный способ разрушения, поскольку иначе он корректно не удаляется. В документации этот нюанс разрушения графа не описан, но практическая проверка с помощью графического редактора показывает именно такой результат. Альтернативный способ разрушения графа это перечисление всех фильтров и удаление их из графа. Это самый надежный способ.

Для разрушения графа в случае завершения программы и в случае выхода из метода с ошибкой, когда часть графа построена, реализуем универсальный метод DestroyGraph. В начале вызываем метод удаляющий граф из таблицы запущенных объектов (ROT), затем удаляем все фильтры из графа и освобождаем интерфейсы, невзирая на то, что они созданы как члены класса CComPtr. Освободить интерфейсы необходимо для случая, когда граф разрушается, а объект CCapture нет, и указатели будут использоваться повторно.

```
void CCapture::DestroyGraph(void)
{
  #ifdef _DEBUG
    RemoveFromRot(m_dwReg);
  #endif //_DEBUG

  if(!m_pGraph2) return;
  if(!m_bGraph)
    return; //граф уже разрушен

  //Удаление фильтров в графе.
  CComPtr<IEnumFilters> pEnum;
  HRESULT hr = m_pGraph2->EnumFilters(&pEnum);
  if (S_OK == hr)
  {
    CComPtr<IBaseFilter> pFilter;
    while (S_OK == pEnum->Next(1, &pFilter, NULL))
    {
      //Удалить фильтр.
      m_pGraph2->RemoveFilter(pFilter);

      //Обязательно сбросить перечислитель, после
      //удаления фильтра.
      pEnum->Reset();
      pFilter = NULL;
    }
    pEnum = NULL;
  }

  //Граф разрушен
  m_bGraph = FALSE;
  m_pGraph2 = 0;

  //Указатель на  построитель графа сбора
```

```
  m_pBuilder = 0;

  //Указатель на фильтр сбора видео
  m_pCapV = 0;
}
```

В данном варианте удаляются все фильтры независимо от степени завершенности графа и гарантировано освобождаются все интерфейсы. Метод RemoveFilter автоматически разъединяет фильтры, а затем удаляет их из графа. В дальнейшем при объявлении новых указателей членов класса CCapture их необходимо будет освобождать в методе DestroyGraph.

После завершения приложения можно проверить правильность завершения всех потоков DirectShow в режиме отладки. В окне Debug при правильном завершении в последних строках отображаются сообщения о завершении потоков и приложения в целом с нулевым кодом завершения.

```
The thread 'Win32 Thread' (0x9a4) has exited with code 0 (0x0).
The thread 'Win32 Thread' (0x838) has exited with code 0 (0x0).
The thread 'Win32 Thread' (0xa18) has exited with code 0 (0x0).
The program '[812] TVshow.exe: Native' has exited with code 0
(0x0).
```

При завершении с ошибкой в окне Debug увидим соответствующее сообщение. Поток 0xb20 завершился с ошибкой.

```
The thread 'Win32 Thread' (0xb20) has exited with code
-2147024637 (0x80070103).
```

Данная ошибка относится к потоку, который обслуживает таблицу запущенных объектов, и она не является критической, во всяком случае, никаких мер принимать не нужно. Метод DestroyGraph вызывается при возникновении исключения, а также в деструкторе класса.

Просмотр результата в редакторе графа

На этом этапе можно уже подключить редактор графа и увидеть первые результаты работы приложения.

Далее запускаем приложение в режиме отладки, а затем запускаем утилиту GraphEdit. В окне утилиты открываем меню File / Connect to Remote Graph, и наблюдаем окно, в котором отображается строка с идентификатором графа (Рис. 20).

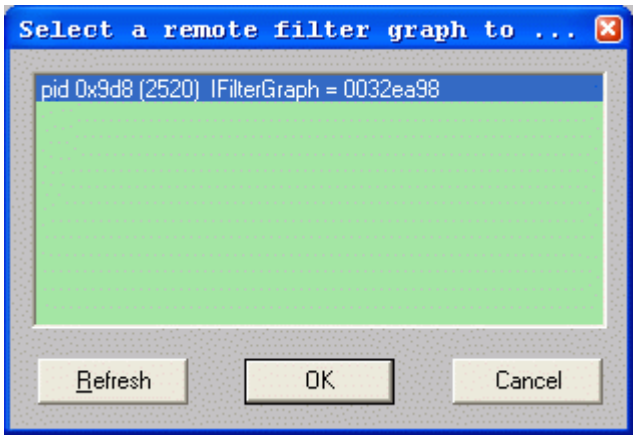


Рис.2. 20 Отображение идентификатора графа

Нажимаем OK и видим установленный в граф фильтр сбора (Рис. 21).

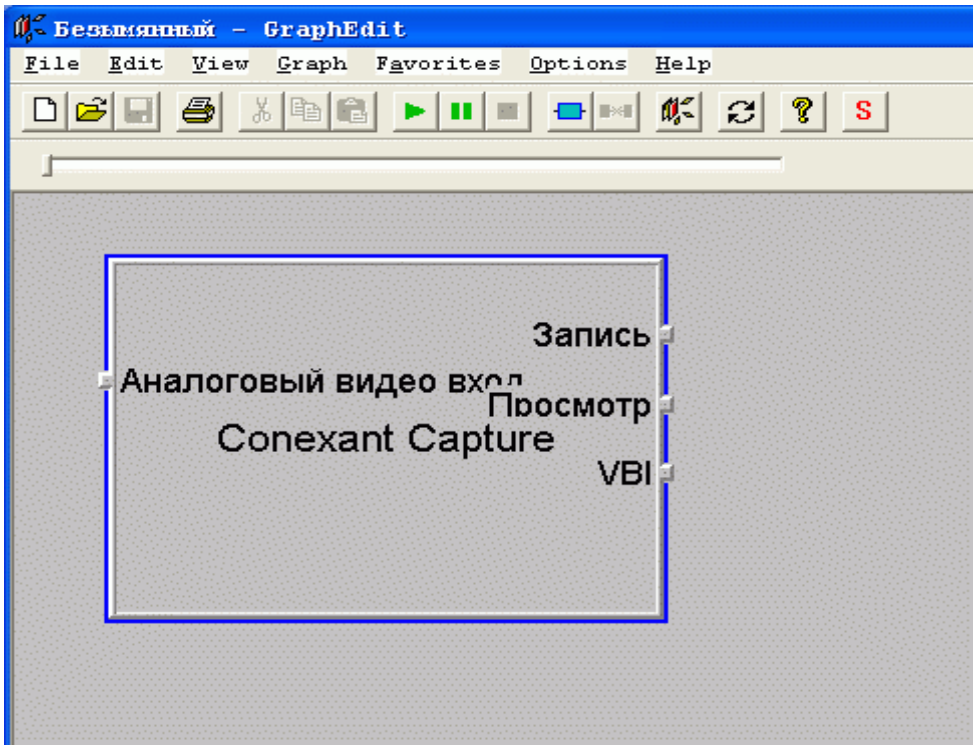


Рис.2. 21 Отображение графа в окне утилиты GraphEdit

Исходные коды проекта находится в папке ProjectState3.