Глава 7 Запись файлов

Глава 7 Запись файлов....1
Запись файлов в DirectShow....1
Простой способ записи файлов в формате AVI и ASF....3
Реализация простого способа записи файлов AVI и ASF....6
Расширенный способ записи файлов в формате AVI и ASF....11
Реализация расширенного способа записи файлов AVI и ASF....18
Управление профилями фильтра ASF....31
Дополнительные способы управления записью файлов....38
Синхронизация потоков при записи файлов AVI....43
Управление звуком и видео при записи файлов AVI....44
Установка видеоформата на контакте Capture фильтра захвата....46
Использование кодеров при записи файлов AVI....49
Отображение свойств компрессора....58
Использование кодеров сторонних производителей....64
Отображение производительности фильтра захвата....76

Запись файлов в DirectShow

DirectShow позволяет производить запись видеофайлов двух форматов, определяемых подтипами:

MEDIASUBTYPE_Avi, для записи файлов типа AVI.

MEDIASUBTYPE_Asf, для записи файлов типа ASF.

Для этих двух форматов файлов DirectShow предоставляет фильтры записи файлов. Основные данные фильтров записи файлов проведены в справочном разделе. Запись файлов AVI производится с участием двух фильтров, фильтра мультиплексора (AVI Mux) и собственно самого фильтра записи файлов (File Writer). Фильтр мультиплексора многовходовый и производит формирование потока данных в формате AVI. Фильтр записи файлов принимает поток данных от мультиплексора и производит запись на диск, как простой поток байтов без дополнительного форматирования.

Фильтр записи файлов ASF содержит внутренний мультиплексор и принимает на входные контакты поток данных непосредственно от фильтров захвата (capture) видео и звука. Он не требует промежуточных фильтров. Фильтр записи файлов (WM ASF Writer) в формате ASF является фильтром оберткой для записывающего объекта платформы Windows Media Format SDK. Фильтр обрабатывает любое число входных потоков и создает файлы в формате Advanced Systems Format (ASF). Файл формата ASF является форматом файла

контейнера для хранения как сжатых так и не сжатых звуковых и видео данных. Спецификация файлов формата ASF здесь не рассматривается и при необходимости с ней можно ознакомиться в папке Doc на компакт-диске. Файл ASF может содержать практически любые типы данных. Фильтр записи ASF файлов может использоваться для различных видов обработки потоков данных, цифровое видео (DV), сжатие и восстановление звука, преобразование файлов AVI или MPEG для передачи по сетевым каналам связи. Данный фильтр является единственным способом создания файлов ASF в DirectShow.

Фильтр ASF использует Windows Media Format SDK для преобразования несжатых видео и звуковых данных в Windows Media контент, с применением кодеков Windows Media. Если контент содержит только звуковые данные, файл имеет расширение ".wma", если файл содержит видео данные или видео и звуковые данные он имеет расширение ".wmv". Также файл может иметь обобщенное расширение ".asf". Любой тип файла может включать и метаданные. Список интерфейсов фильтра и другие данные приведены в справочном разделе.

Для определения типа кодирования данных фильтр использует набор входных параметров, называемых профилем. Профиль предоставляет фильтру конфигурационную информацию потока данных, на основании которой выбирается кодек и устанавливаются его параметры. При создании фильтра он конфигурируется автоматически с использованием предопределенных системных профилей и учетом параметров входных потоков данных. При выборе профиля учитываются параметры сжатия данных, скорость передачи, частота кадров, число потоков, требуемое качество сжатия и другие. Из параметров профиля фильтр определяет тип записываемого файла, число входных контактов для установки и медиа форматы записываемых потоков. Для управления профилями фильтра необходимо установить Windows Media Format SDK.

Каждый из указанных фильтров предоставляет несколько интерфейсов позволяющих управлять режимами работы и потоками данных. Для простых решений нет нужды в использовании многих из этих интерфейсов, достаточно настроек фильтров по умолчанию.

Для записи файлов также можно использовать фильтры сторонних производителей, которые обеспечивают запись файлов других форматов, например MPEG2. Запись файлов в формате MPEG2 не поддерживается DirectShow, Microsoft не предоставляет фильтров для записи файлов MPEG2. Для воспроизведения DirectShow предоставляет фильтры для файлов формата MPEG2.

Простой способ записи файлов в формате AVI и ASF

Для записи файлов необходимо построить ветку графа, начиная с контактов записи (capture) фильтра захвата изображения и фильтра захвата звука. Запись файлов AVI и ASF в простом варианте не представляет особой сложности. Рассмотрим на примере вариант записи файлов двух форматов. Допустим, что требуется производить запись отдельного файла одного из двух форматов. Начало записи файла будет происходить при нажатии кнопки "Запись", остановка и освобождение файла при нажатии кнопки "Стоп". Анализ и выбор формата записываемого файла производится по расширению имени файла. Повторная процедура производит запись файла с новым именем.

Построение ветки графа для реализации записи файлов форматов AVI и ASF можно произвести полуавтоматически с использованием вспомогательного объекта построителя, который предоставляет интерфейс ICaptureGraphBuilder2. Некоторые методы этого интерфейса рассматривались ранее, сейчас рассмотрим метод SetOutputFileName, который формирует участок графа для записи файла. Метод устанавливает в граф фильтры для записи либо файла типа AVI, либо файла ASF. Для простейшего способа записи вполне достаточно только этого метода.

Второй интерфейс, который может быть понадобиться при записи файлов это IFileSinkFilter, и производный от него IFileSinkFilter2. Эти интерфейсы предоставляются на фильтрах записи файлов. Интерфейс IFileSinkFilter2 полностью заменяет IFileSinkFilter и добавляет два метода, управляющие перезаписью файлов.

Если производится запись файла AVI, метод SetOutputFileName устанавливает два соединенных фильтра, фильтр мультиплексора AVI Mux и фильтр записи файла. При записи файлов ASF устанавливается только фильтр записи файла. После вызова этого метода остается только подключить фильтр записи к фильтрам захвата изображения и звука. Это можно сделать различными способами, как ручным соединением фильтров, так и автоматически с возможностью установки промежуточного фильтра компрессора. Фильтр компрессора может понадобиться для записи сжатых файлов AVI. При записи файлов ASF использовать компрессор не нужно, поскольку файл сжимается в соответствии с установленным по умолчанию профилем. Другое дело, что профиль по умолчанию может не удовлетворить пользователя и для полноценного использования возможностей фильтра потребуется управление системными профилями, что невозможно при простейшем варианте записи.

Запись начинается сразу после пуска графа. Для остановки записи следует остановить граф и удалить фильтр (или фильтры) записи. Для записи нового файла вся процедура повторяется.

Все опыты по записи файлов производятся с размером изображения 352x288 пикселей для стандарта SECAM. При установке другого видео формата (размера изображения) запись файла становиться невозможной. Подробней об установке формата на контакте Capture фильтра захвата будет рассказано далее.

Рассмотрим метод ICaptureGraphBuilder2::SetOutputFileName и методы интерфейса IFileSinkFilter, используемые для реализации простого способа записи файлов.

```
HRESULT SetOutputFileName(
  const GUID    *pType,
  LPCOLESTR   lpwstrFile,
  IBaseFilter     **ppf,
  IFileSinkFilter   **pSink
);
```

Параметры.
pType
[in]
Указатель на GUID, определяющий выходной медиа подтип или идентификатор (CLSID) фильтра мультиплексора или фильтра записи файла. Медиа подтип может иметь одно из следующих значений.

Value	Описание
MEDIASUBTYPE_Avi	Файл формата AVI.
MEDIASUBTYPE_Asf	Файл формата ASF.

lpwstrFile
[in]
Указатель на строку типа WCHAR, содержащую имя файла.

ppf
[out]
Адрес указателя, принимающего интерфейс IBaseFilter мультиплексора.

pSink
[out]
Адрес указателя, принимающего интерфейс IFileSinkFilter фильтра записи файла. Может быть NULL.

Коды возврата.

Код	Описание
S_OK	Без ошибок.
E_FAIL	Ошибка.

E_POINTER	Нулевой указатель.

Если интерфейс IFileSinkFilter поддерживается на мультиплексоре, то метод вызывает IFileSinkFilter::SetFileName для установки имени файла. Если мультиплексор не поддерживает этот интерфейс, то дополнительно устанавливается фильтр записи файла, который автоматически соединяется с мультиплексором и имя файла устанавливается фильтром записи файла. Описание методов интерфейса IFileSinkFilter приводится ниже.

Для заказных фильтров мультиплексора, которые не поддерживают соединение на выходном контакте до подключения входного, метод будет возвращаться с ошибкой.

Интерфейс IFileSinkFilter имеет два метода SetFileName и GetCurFile. Метод SetFileName создает файл, если он не существует, в противном случае файл переписывается без удаления.

```
HRESULT SetFileName(
  LPCOLESTR pszFileName,
  const AM_MEDIA_TYPE *pmt
);
```

Параметры.

pszFileName
[in]
Указатель на строку с именем файла.

pmt
[in]
Указатель на структуру, описывающую медиа формат выборки записываемой в файл и медиа тип для входного контакта фильтра записи. В документации отсутствуют указания на какие-либо особенности его использования. Нулевое значение указателя не препятствует нормальной записи файла.

Возвращает величины HRESULT.

Метод GetCurFile возвращает имя файла и медиа тип текущего файла.

```
HRESULT GetCurFile(
  LPOLESTR *ppszFileName,
  AM_MEDIA_TYPE *pmt
);
```

Параметры.

ppszFileName
[out]
Адрес указателя на строку типа OLESTR, которая принимает имя файла. Строка должна быть освобождена, вызовом CoTaskMemFree после ее использования.

pmt
[out]
Указатель на структуру AM_MEDIA_TYPE, которая принимает медиа тип. Может быть равным NULL, если тип не нужен. Если медиа тип используется, то структура должна освобождаться после использования.

Коды возврата.

Код	Описание
S_OK	Без ошибок.
E_FAIL	Нет открытого файла. Если файл не открыт, метод может возвращаться без ошибок, но возвращает NULL в первом параметре. Он должен быть проверен перед использованием имени файла и медиа типа.
E_OUTOFMEMORY	Недостаточно памяти.
E_POINTER	Нулевой указатель в параметре ppszFileName.

Интерфейс IFileSinkFilter2 производный от IFileSinkFilter и полностью заменяет его. Этот интерфейс добавляет возможность удалять старый файл перед перезаписью, что позволяет избавиться от мусора, который остается от предыдущего файла. Интерфейс имеет два метода, которые должны вызываться после установки имени файла.

```
HRESULT SetMode(DWORD dwFlags);
HRESULT GetMode(DWORD *dwFlags);
```

Флаг имеет только одно значение AM_FILE_OVERWRITE, который указывает на то, что старый файл должен быть удален.

Реализация простого способа записи файлов AVI и ASF

Перед написанием метода, производящего запись файлов, и проверки его работы, реализуем минимальный интерфейс пользователя, который будет иметь поле редактирования для ввода пути к папке хранения записанного файла и поле редактирования для имени файла. Также необходимы две кнопки для старта записи и остановки записи.

Для реализации интерфейса записи файлов используем класс CFView2. Элементы интерфейса размещаем на вкладке этого класса. Чтобы не вводить постоянно путь к папке при запуске программы и новое имя файла каждый раз при записи, удобно сформировать по умолчанию путь к папке пользователя "Мои видеозаписи", а имя файла формировать по текущей дате с точностью до секунд, чтобы можно было сформировать новое имя файла при записи коротких файлов.

Тогда, для смены типа записываемого файла, достаточно отредактировать только расширение файла. Для формирования пути к папке записи следует определить местонахождение этой папки для текущего пользователя, поскольку на компьютере может быть несколько пользователей. Самый простой способ отыскать папку "Мои видеозаписи", это воспользоваться возможностями, которые предоставляет Shell API, при условии, что пользователь не удалил папку "Мои видеозаписи". Иначе необходимо будет ее создать. Метод CFView2::myGetPathMyVideo, возвращает полный путь к папке "Мои видеозаписи".

Для формирования имени файла напишем отдельный метод CFView2::myGetFileNameFromDate, который из текущей даты будет формировать имя файла в формате "день_месяц_часы_минуты_секунды.avi". Вызов методов производится в методах CFView2::OnInitialUpdate() при запуске приложения для первоначальной установки пути и имени файла, а также в обработчике кнопки "Запись" CFView2::OnBnClickedButWritefile. С текстами методов можно ознакомиться в архиве проектов.

В методе CCapture::InitWriteToFile производит разбор полного пути к файлу и на основании расширения файла формирует медиа подтип файла. При разборе пути к файлу в качестве примера, используется локальный контекст русского языка библиотеки STL для перевода расширения файла в нижний регистр. Этот фрагмент можно упростить, поскольку расширение файла практически всегда записывается в английской транскрипции, но в других случаях пример может быть полезен. Далее метод строит ветку графа для записи файла.

Метод реализован в простейшем варианте, каждый раз при записи нового файла старая записывающая секция удаляется и формируется новая. Для удаления фильтров используется метод интерфейса динамической реконфигурации графа IGraphConfig:: RemoveFilterEx, который описан ниже. Можно также использовать метод IFilterGraph::RemoveFilter. Удаление старой записывающей секции производится в обработчике кнопки "Стоп" CCapture::StopCapture(), так запись начнется быстрее, чем, если бы старые фильтры удалять в методе InitWriteToFile.

```
BOOL CCapture::InitWriteToFile(LPCWSTR nameFile)
{
HRESULT hr;
//Определить тип фильтра записи файла
wstring ws(nameFile);

//Извлечь расширение имени файла
wstring::size_type idx;
idx = ws.find_last_of('.');

//Не найден символ точка
```

```
if(idx == wstring::npos)
return wstring(L"");

//Число символов неправильное
wstring wext = ws.substr(idx);
if(wext.size() != 4)
return wstring(L"");

//Русский локальный контекст
std::locale loc("rus");

//Копируем расширение файла в символьный буфер
wchar_t wbuf[5] = {0}; //Размер на 1 больше!!!
wext.copy(wbuf,4);

//Перевод в нижний регистр
std::use_facet<std::
ctype<wchar_t> >(loc).tolower(&wbuf[0],&wbuf[4]);

//Выбор фильтра записи файла
wext.assign(wbuf);

GUID *guid = NULL;
if(wext == L".avi")
guid = const_cast<GUID*>(&MEDIASUBTYPE_Avi);

BOOL bfileASF = FALSE;
if(wext == L".asf")
{
guid = const_cast<GUID*>(&MEDIASUBTYPE_Asf);
bfileASF = TRUE;
}

if(guid == NULL)
return FALSE;

//формирование записывающей секции
CComPtr<IBaseFilter> pMux;
CComPtr<IFileSinkFilter> pSink;

hr = m_pBuilder->SetOutputFileName(guid
,nameFile,&pMux,&pSink);

//Этот интерфейс позволяет усекать файл до нуля перед
// перезаписью
//CComQIPtr<IFileSinkFilter2> pSink2(pSink);
//if(pSink2)
  //{
  //
  //}

if(hr != S_OK)
```

```
return FALSE;

//Подключаем канал видео
hr = m_pBuilder->RenderStream(&PIN_CATEGORY_CAPTURE,
  &MEDIATYPE_Video, m_pCapV, NULL, pMux);

if(hr != S_OK)
return FALSE;

//Подключаем канал звука, этот метод устанавливает
//фильтр Smart Tee, который не нужен, его не
//используем
//hr = m_pBuilder->RenderStream(
//&PIN_CATEGORY_CAPTURE
//,&MEDIATYPE_Audio,m_pCapA,NULL,pMux);

  //находим контакт звука и подключаем звук без
//Smart Tee
  CComPtr<IPin> pP;
  if(S_OK == m_pBuilder->FindPin(m_pCapA,PINDIR_OUTPUT
,&PIN_CATEGORY_CAPTURE
,&MEDIATYPE_Audio,TRUE,0,&pP))
{
if(S_OK !=  m_pGraph2->Render(pP);
return FALSE;
}

m_bCaptureAVI = 1;

//Запрашиваем указатель на записывающий фильтр.
//Если записывается файл ASF, то указатели pSink
//и pMux равны.
//Записываем указатели на фильтры
CComQIPtr<IBaseFilter> pFSink(pSink);
if(pFSink && bfileASF)
{
m_pFSinkASF = pSink;
return TRUE;
}
else if(pFSink)
m_pFSink = pSink;

if(pMux)
m_pFMux = pMux;

return TRUE;
}
```

При остановке записи, независимо от типа записываемого файла, следует удалить записывающую ветвь графа, не разрушая остальной части графа и запустить режим предварительного просмотра. Это можно сделать в обработчике кнопки остановки записи. Сразу после удаления фильтров производится

пуск графа и восстанавливается предварительный просмотр. Если этого не сделать на экране останется последний записанный кадр и для запуска графа нужно будет ввести отдельную кнопку. Иногда это может быть полезно для точной фиксации окончания записи файла. Для выполнения данной задачи в обработчике кнопки вызывается метод CCapture::StopCapture(), текст показан ниже. В методе StopCapture() удаление фильтров можно производить двумя способами.

Первый способ использует метод IGraphConfig::RemoveFilterEx. Метод, по умолчанию, удаляет только указанный фильтр, если же установлен флаг, то удаляются все фильтры, имеющие соединения, начиная с указанного фильтра. Метод не требует предварительного разъединения фильтров, но граф должен быть остановлен. Если указанный фильтр не отключен от фильтров выше по потоку, то они также удаляются.

```
HRESULT RemoveFilterEx(
  IBaseFilter *pFilter,
  DWORD Flags
);

Параметры
pFilter
[in]
Указатель на удаляемый фильтр.

Flags
[in]
Комбинация флагов из перечисления REM_FILTER_FLAGS. Пока суще-
ствует только один флаг REMFILTERF_LEAVECONNECTED, удаляющий все
фильтры имеющие соединения.

Метод возвращает S_OK при успешном выполнении или ошибку
HRESULT.
```

Второй способ для удаления фильтров предполагает использование метода IFilterGraph::RemoveFilter, который имеет только один параметр и вызывается с помощью менеджера графа. Метод StopCapture, в качестве примера, удаляет фильтры первым способом, но в данном случае, целесообразней использовать второй способ удаления фильтров. Поскольку данный способ записи очень прост, то отдельной версии проекта для него нет.

```
BOOL CCapture::StopCapture()
{
HRESULT hr;

//Остановка графа
```

```
if(!StopGraph())
return FALSE;

//Удаление из графа фильтров записи файла.
//Запрос интерфейса динамической реконфигурации и
//удаление
CComQIPtr<IGraphConfig> pCnf(m_pGraph);

if(pCnf)
{
//Флаг REMFILTERF_LEAVECONNECTED удаляет все фильтры
//имеющие соединения
if(m_pFMux && (S_OK ==
              pCnf->RemoveFilterEx(m_pFMux,NULL)))
m_pFMux = NULL;
else
return E_FAIL;

if(m_pFSink && (S_OK == (hr=
pCnf->RemoveFilterEx(m_pFSink,NULL))))
m_pFSink = NULL;

if(m_pFSinkASF && (S_OK ==(hr=
pCnf->RemoveFilterEx(m_pFSinkASF, NULL))))
m_pFSinkASF = NULL;
}
else
return FALSE;

//Пуск графа
if(!StartPreview())
return FALSE;

return TRUE;
}
```

Расширенный способ записи файлов в формате AVI и ASF

Недостатком предыдущего способа записи файлов является достаточно долгий поиск в системном регистре фильтров необходимых для записи файла, что приводит к дополнительной задержке начала записи. Если требуется максимально сократить задержку перед началом записи или ускорить переключение между типами записываемых файлов, то необходимо построить управление веткой записи файлов таким образом, чтобы не приходилось каждый раз удалять и создавать фильтры записи.

Первое и самое простое решение установить в граф две ветви графа для записи файлов двух типов и производить их оперативное переключение для записи файлов определенного формата. Со стороны DirectShow, каких либо ограничений и предупреждений на установку и использование в графе двух разнотипных фильтров записи файла найдено не было. Проверка этого решения обнаружила много препятствий, которые исключают возможность такого применения фильтров записи. Первое и, пожалуй, самое главное препятствие состоит в том, что фильтр записи файла ASF после установки в граф блокирует управление графом до момента его полного подключения к фильтрам захвата. Таким образом, не удается организовать режим предварительного просмотра с установленным и неподключенным фильтром ASF. Блокировка управления графом не полная, в некоторых случаях удается установить фильтр в граф, сохранив предварительный просмотр. Например, фильтр ASF устанавливается в запущенный граф, но дальнейшее управление графом происходит с нарушениями, которые вызывают сомнения в управляемости графом. При такой установке отмечается невозможность определения текущего состояния графа. Метод IMediaControl::GetState возвращается с ошибкой. Фрагмент кода показан ниже.

```
OAFilterState state; //Состояние фильтрового графа
CComQIPtr<IMediaControl> pCtrl(m_pGraph2);
ASSERT(pCtrl);

HRESULT hr;
if(FAILED(pCtrl->GetState(30,&state)))
{
  return E_FAIL;
}
```

Также, в этом случае, после остановки записи файла и отключения фильтра невозможно запустить граф для предварительного просмотра. Существует интерфейс IFilterChain, который позволяет производить запуск и остановку ветки графа. К сожалению, он имеет существенное ограничение, поскольку цепочкой фильтров считается последовательность из фильтров, имеющих только один вход и один выход. Таким образом, ветка графа записи файлов не попадает под определение цепочки фильтров, и интерфейс IFilterChain не управляет в полном объеме фильтром записи.

Все эти ограничения требуют удаления фильтра ASF после остановки записи файла для организации предварительного просмотра. К счастью, есть интерфейс IGraphConfig, который поддерживает динамическую реконфигурацию графа. Он позволяет удалять фильтры в кэш и извлекать их оттуда, что резко сокращает затраты на повторную установку фильтров в граф. Методы интерфейса приводятся в таблице.

Метод	Описание
Reconnect	Выполняет динамическую реконфигурацию двух контактов, выходного и входного. Входной контакт фильтра должен поддерживать интерфейс динамической реконфигурации IPinConnection.
Reconfigure	Закрывает граф и вызывает функцию обратного вызова (callback) в приложении или фильтре для выполнения динамической реконфигурации.
AddFilterToCache	Переводит фильтр из графа к кэш.
RemoveFilterFromCache	Удаляет фильтр из кеша.
EnumCacheFilter	Перечисляет фильтры в кеше.
GetStartTime	Возвращает значение времени (reference time) для последнего запуска графа.
PushThroughData	Проталкивает данные через граф на указанный контакт.
SetFilterFlags	Устанавливает на фильтре флаг конфигурации. Флаг имеет одно значение, которое говорит, что фильтр удаляется в процессе реконфигурации.
GetFilterFlags	Возвращает флаг конфигурации.
RemoveFilterEx	Удаляет фильтр из графа.

Таблица 1. Интерфейс IGraphConfig.

В списке методов интерфейса IGraphConfig отсутствует метод для явного извлечения фильтра из кеша. В документации DirectShow указывается, что методы IGraphBuilder::Render, IGraphBuilder::RenderFile, и IGraphBuilder::Connect пытаются использовать фильтры из кеша до поиска фильтров в регистре. В действительности метод IGraphBuilder::Connect не работает с кешем. Если во втором параметре метода указан входной контакт фильтра, находящегося в кеше, то метод возвращается с ошибкой. В этом можно убедиться, проверив его на практике с помощью такого кода.

```
  CComPtr<IPin> pP, pPCapV, pPCnt;
  pPCapV = GetPinCapture(m_pCapV);

//Находим контакт фильтра, который находится в кеше
  hr = m_pBuilder->FindPin(m_pFSinkASF
, PINDIR_INPUT,NULL,  NULL,TRUE,0,&pP);

  if(pP && pPCapV)
  {
    hr = m_pGraph2->Connect(pPCapV,pP);
    if(hr != S_OK)
      CTVshowApp::myMessageError(hr,"");
```

```
    hr = pP->ConnectedTo(&pPCnt);
    if(hr != S_OK)
      CTVshowApp::myMessageError(hr,"");
}
```

Метод Connect возвращает код VFW_E_NOT_IN_GRAPH, если фильтр с указанным контактом не находится в графе и не производит никаких действий по подключению фильтра. Неоднократные проверки в различных условиях не привели к положительному результату.

Метод RenderFile предназначен для работы с файлами и в данной ситуации не применим. Только метод Render, действительно автоматически извлекает фильтр из кеша в граф и подключает его. Остается определить, каким образом метод различит два или более фильтров в кеше предназначенных для решения одинаковой задачи записи файлов. А он никак их не различает, кеш оказался стековым, о чем документация умалчивает. Метод Render устанавливает в граф и подключает фильтр, находящийся на верхушке стека. Дополнительным препятствием при работе с кешем является то, что при помещении в кеш имя фильтра, назначенное при установке в граф, очищается и при возврате из кеша назначается имя, генерируемое автоматически. Возможности интерфейса IGraphConfig очень ограничены в части работы с кешем из-за неработающего с кешем метода Connect. Другого метода для простого извлечения фильтра из кеша нет. В примерах DirectShow также нет кода, работающего с кешем.

Таким образом, простым способом извлечь нужный фильтр записи файла из кеша не удастся, если в нем одновременно находятся фильтры записи файлов AVI и ASF. Из сложившейся ситуации есть выход, который позволяет решить задачу. Дело в том, что фильтр мультиплексора и фильтр записи файла AVI, находясь в графе и будучи не подключенными, к фильтрам захвата видео и звука, никак не влияют на состояние графа, даже если они соединены между собой. В результате можно постоянно держать в графе неподключенную ветвь фильтров AVI, а фильтр ASF хранить в кеше и извлекать его только при записи файлов ASF. После записи файла фильтр ASF опять отправляется в кеш и запускается режим предварительного просмотра.

Есть и второй способ извлечения фильтра из кеша. После некоторых дополнительных экспериментов удалось извлечь фильтр из кеша не совсем очевидным способом. Ниже приведен пример кода.

```
//Добавляем в граф фильтр, который находится в кеше.
//Фильтр при этом не удаляется автоматически из кеша и
//таким образом, указатель m_pFSinkASF указывает на
//фильтр в кеше и на фильтр в графе одновременно, в чем
//можно удостовериться при перечислении фильтров в кеше.

  HRESULT hr = m_pGraph2->AddFilter(m_pFSinkASF,L"ASF");
```

```
  CComQIPtr<IGraphConfig> pCnf(m_pGraph2);

//Теперь удаляем фильтр из кеша. В графе фильтр
//остается.
  if(pCnf)
  {
    hr = pCnf->RemoveFilterFromCache(m_pFSinkASF);
  }
```

Далее с фильтром можно работать обычным способом. Возможно, что именно так и предполагается извлекать фильтр из кеша, просто в документации DirectShow об этом забыли упомянуть. Данный способ извлечения фильтра из кеша не используется, но его применение может быть полезно. Ниже приводится код метода, извлекающий фильтр из кеша с назначением ему имени.

```
HRESULT CCapture::AddFilterFromCache(IBaseFilter* pF,
                            wstring nameFilter)
{
  HRESULT hr = S_OK;
  CComPtr<IEnumFilters> pEnum;

  CComQIPtr<IGraphConfig> pCnf(m_pGraph2);
  if(pCnf)
  {
    hr = pCnf->EnumCacheFilter(&pEnum);
    if(FAILED(hr))
      return hr;
  }

  //найти фильтр в кеше
  CComPtr<IBaseFilter> pFCache;
  while (S_OK == pEnum->Next(1, &pFCache, NULL))
  {
    if(pFCache == pF)
    {
      //добавляем фильтр в граф
      hr = m_pGraph2->AddFilter(pF,nameFilter.c_str());
      if(FAILED(hr))
        return hr;

      //удаляем фильтр из кеша
      hr = pCnf->RemoveFilterFromCache(pF);
      if(FAILED(hr))
        return hr;

      //фильтр найден
      break;
    }
```

```
    pFCache = NULL;
  }

  return hr;
}
```

Для проверки всех решений напишем несколько методов, которые будут реализовывать следующие задачи:

- Осуществлять первоначальную установку фильтра ASF в кеш и фильтра AVI в граф.
- Производить установку нового имени файла и конфигурировать ветвь графа для записи файла определенного типа. Определять тип файла по его расширению.
- Производить остановку графа и отключение фильтра записи файла. При этом если записывался файл ASF, то фильтр отправляется в кеш и запускается режим предварительного просмотра. Если же записывался файл AVI, то отключается ветвь фильтров из мультиплексора и фильтра записи файла, после чего запускается режим предварительного просмотра.
- Производить перезапись последнего записанного файла.

Реализация перечисленных задач позволит выполнять запись файлов в двух режимах, последовательная запись нескольких файлов, или перезапись текущего файла.

При обращении к записанному файлу следует учесть, что фильтр ASF освобождает файл для доступа сразу после останова графа, в отличие от фильтра AVI, который требует смены имени файла и только после этого освобождает файл. Перед началом работы необходимо модернизировать проект и установить Windows Media Format SDK. Модернизированный проект находится в папке под именем ProjectState7. Установочный пакет Windows Media Format SDK следует загрузить с сайта Microsoft.

Реализация расширенного способа записи файлов AVI и ASF

Первоначальную установку фильтров в граф произведем в методе CCapture::InitWriteFile, который можно вызвать в конце метода CCapture::InitCaptureVideo. Фрагмент кода InitCaptureVideo показан ниже.

```
  ……….
  //установка фильтров записи файлов AVI и ASF
  if(!InitWriteFile())
    return FALSE;
  }
  catch(...)
  {
    DestroyGraph();
    sErr = "Неизвестное исключение (InitCaptureVideo)";
    CTVshowApp::myMessageError(hr= E_FAIL,sErr);
  }
  return hr;
}
```

Метод выполняет первую из перечисленных задач, установку фильтров записи файлов AVI в граф и загрузку в кеш фильтра записи ASF. Для фильтра ASF производится определение профиля кодирования по умолчанию, который устанавливается при добавлении фильтра в граф. Профиль фильтра фиксируется в переменной класса m_profileGuid, и устанавливается при записи нового файла. Работа с профилями кодирования будет рассматриваться позже. Указатели на фильтры записи являются членами класса.

```
BOOL CCapture::InitWriteFile()
{
  HRESULT hr = S_OK;

  //Если граф запущен, то остановить
  if(m_bStartGraph && !StopGraph())
    return FALSE;

  //Устанавливаем в граф фильтр ASF
  if(!m_pFSinkASF)
  {
    if(S_OK != AddFilterByCLSID(CLSID_WMAsfWriter
                    ,L"FileWriteASF",m_pFSinkASF))
      return FALSE;

    //Возвращаем GUID профиля по умолчанию
    CComQIPtr<IConfigAsfWriter> pCnf(m_pFSinkASF);
```

```
    if(!pCnf)
      return FALSE;

    //Фиксируем профиль фильтра, установленный при
    //создании
    hr = pCnf->GetCurrentProfileGuid(&m_profileGuid);
    if(S_OK != hr)
      return FALSE;

    //Запрашиваем интерфейс и устанавливаем временное имя
    файла. //Можно не устанавливать временное имя для файла ASF.
    CComQIPtr<IFileSinkFilter2> pFSink(m_pFSinkASF);

    //Переводим фильтр в кеш. Фильтр после установки в кеш
    //теряет имя,полученное при установке в граф
    if(pFSink &&
        (S_OK == pFSink->SetFileName(L"C:\\Temp.asf",NULL)))
    {
      if(S_OK != PushFilterToCache(m_pFSinkASF))
      return FALSE;
    }
    else
      return FALSE;
  }

  //Устанавливаем в граф фильтры AVI с временным именем
  if(!m_pFMux && !m_pFSink)
  {
    CComPtr<IFileSinkFilter> pSink;
    if(S_OK != m_pBuilder->SetOutputFileName(&MEDIASUBTYPE_Avi
                        ,L"C:\\TempAVI.avi",&m_pFMux,&pSink))
      return FALSE;

    //Запрашиваем указатель на фильтр записи файла
    CComQIPtr<IBaseFilter> pFSink(pSink);

    //Присваиваем его члену класса
    m_pFSink = pFSink;

    if(!m_pFSink)
      return FALSE;
  }

  if(!StartPreview())
    return FALSE;

  return TRUE;
}
```

В методе CCapture::InitWriteFile используется метод AddFilterByCLSID, который производит установку в граф фильтра по его идентификатору

CLSID с присвоением ему имени. Метод возвращает указатель на установленный фильтр в последнем параметре, который передается по ссылке.

```
HRESULT CCapture::AddFilterByCLSID(
            const GUID& clsid,       // CLSID фильтра
            wstring wszName,        // Имя фильтра
            CComPtr<IBaseFilter>& ppF)
{
  ASSERT(m_pGraph2);
  CComPtr<IBaseFilter> pF ;

  HRESULT hr = CoCreateInstance(clsid, 0, CLSCTX_INPROC_SERVER
              ,IID_IBaseFilter, reinterpret_cast<void**>(&pF));


  if (SUCCEEDED(hr))
  {
    hr = m_pGraph2->AddFilter(pF, wszName.c_str());
    if (SUCCEEDED(hr))
    {
       ppF = pF;
    }
  }
    return hr;
}
```

Для управления процессом записи файлов в редакторе ресурсов установим три кнопки под именами "Новый файл", "Перезапись" и "Стоп". Каждую кнопку снабдим своим обработчиком в классе CFView2, соответственно с именами OnBnClickedButSetFilename, OnBnClickedButWritefile и OnBnClickedButStop. Для управления именем файла и директорией записи введем два поля класса CEdit, которым назначим переменные CString m_nameFile и CString m_dirFile. Также создадим соответствующие обработчики, в которых будет производиться только обновление строки имени и строки директории

В обработчике OnBnClickedButSetFilename при нажатии кнопки "Новый файл" будет вызываться метод CCapture::SetNameFile, который произведет все необходимые действия для подготовки процесса записи файла.

Для отображения производительности фильтра записи в обработчике производится установка таймера с периодом срабатывания в одну секунду. Подробней производительность фильтра записи будет рассмотрена позже. Метод CCapture::SetNameFile по расширению файла находит необходимый фильтр записи и производит его подключение к фильтрам захвата изображения и звука. Метод Render всегда подключает только фильтр ASF, если он находится в кеше. Он не обращает внимания на фильтр AVI, находящийся в графе и не подключает его. Поэтому подключение фильтра при записи файлов AVI следует производить либо вручную, либо используя метод

RenderStream, который не столь "интеллектуальный". Для определения контактов на фильтрах сбора написан метод CCapture::GetPinCapture, который возвращает указатель на контакт записи фильтра захвата. Он представляет собой обертку вокруг метода ICaptureGraphBuilder2::FindPin. Если контакты фильтров захвата используются часто, то лучше их определить как переменные класса на стадии установки фильтра в граф, и затем использовать на протяжении жизни объекта сбора.

```
BOOL CCapture::SetNameFile(wstring nameFile,DWORD mode)
{
 HRESULT hr;
 //Остановка графа если запущен
 if(m_bStartGraph && !StopGraph())
    return FALSE;

 //Извлечение расширения имени файла
 wstring wext = GetExtFileName(nameFile);
 if(wext == L"")
  {
    StartPreview();
    return FALSE;
  }

 //Находим указатель на выходной контакт фильтра
 //захвата видео
 CComPtr<IPin> pPinOutV;
 pPinOutV = GetPinCapture(m_pCapV);

 if(!pPinOutV)
    return FALSE;

 //Находим указатель на выходной контакт фильтра звука
 CComPtr<IPin> pPinOutA;
 pPinOutA = GetPinCapture(m_pCapA);

 if(!pPinOutA)
    return FALSE;
```

```
//Запрашиваем интерфейс динамической реконфигурации
CComQIPtr<IGraphConfig> pCnf(m_pGraph2);
if(wext == L".asf" )
{
    if(IsFilterCache(m_pFSinkASF))
    {
      //Подключаем фильтр ASF из кэша
      if((S_OK != m_pGraph2->Render(pPinOutA))
                || (S_OK !=m_pGraph2->Render(pPinOutV)))
        return FALSE;
    }

  //Устанавливаем профиль фильтра
  CComQIPtr<IConfigAsfWriter> pCnfASF(m_pFSinkASF);
  if(!pCnf)
    return FALSE;

  hr = pCnfASF->ConfigureFilterUsingProfileGuid(m_profileGuid);
  if(S_OK != hr)
    return FALSE;

  //Устанавливаем имя файла и режим записи файла
  CComQIPtr<IFileSinkFilter2> pSink2(m_pFSinkASF);
  hr = pSink2->SetFileName(nameFile.c_str(),NULL);

  if(hr != S_OK)
    return FALSE;
  m_wstrNameFile = nameFile;

  //Флаг AM_FILE_OVERWRITE указывает на перезапись
  //файла. NULL указывает на запись  поверх старого //файла
  hr = pSink2->SetMode(mode);

  if(hr != S_OK)
    return FALSE;
}
else if(wext == L".avi")
```

```
{
  //Подключаем AVI
  if((S_OK != m_pBuilder->RenderStream(NULL,NULL,pPinOutV
          ,NULL,m_pFMux))||(S_OK != m_pBuilder->RenderStream(
          NULL,NULL,pPinOutA,NULL,m_pFMux)))
    return FALSE;

  //Или подключаем AVI прямым подключением
  //if((S_OK != ConnectFilters(pPinOutV,NULL,m_pFMux)) ||
  //(S_OK != ConnectFilters(pPinOutA,NULL,m_pFMux)))
  //    return FALSE;

  //Устанавливаем имя файла и режим записи файла
  CComQIPtr<IFileSinkFilter2> pSink2(m_pFSink);
  if(pSink2 && S_OK!=
           pSink2->SetFileName(nameFile.c_str(),NULL))
    return FALSE;

  m_wstrNameFile = nameFile;

  //Флаг AM_FILE_OVERWRITE указывает на перезапись
  //файла. NULL продолжает запись  поверх старого
  //файла
  hr = pSink2->SetMode(mode);
  if(hr != S_OK)
    return FALSE;
      }
  else //Расширение имени не найдено, фильтры не подключены
  {
    StartPreview();
    return FALSE;
  }

  return TRUE;
}
```

В методе SetNameFile может использоваться метод для прямого соединения фильтров CCapture::ConnectFilters. Метод позволяет производить прямое соединение двух фильтров. Фильтр источник может быть указан своим выходным контактом или указателем на фильтр. Фильтр приемник указывается указателем на фильтр. Если фильтр приемник имеет несколько входных кон-

тактов, то для указания подключаемого контакта в четвертом параметре передается его индекс, который по умолчанию равен нулю. При использовании индекса контакта, следует помнить особенность метода FindPin, которая состоит в том, что метод все неподключенные контакты индексирует с нуля, не учитывая подключенные контакты.

```
HRESULT CCapture::ConnectFilters(IPin *pPinSrc,IBaseFilter
            *pFSrc,IBaseFilter *pFDest, int nPin/* = 0*/ )
{
  HRESULT hr;
  //Если граф запущен, остановить
  if(m_bStartGraph && !StopGraph())
    return FALSE;

  if(!pPinSrc && !pFSrc || !pFDest)
    return E_FAIL;

  //Указан фильтр источник
  CComPtr<IPin> pOut;
  if(!pPinSrc && pFSrc)
  {
    //Ищем выходной неподключенный контакт на фильтре источнике
    if(S_OK != m_pBuilder->FindPin(pFSrc,PINDIR_OUTPUT,0 ,NULL
                                        ,TRUE,0,&pOut))
        return E_FAIL;
  }

  if(pPinSrc && !pFSrc)
    pOut = pPinSrc;

  //Ищем входной неподключенный контакт на фильтре потребителе
  CComPtr<IPin> pPinIn;
  hr = m_pBuilder->FindPin(pFDest,PINDIR_INPUT,0,NULL,TRUE
                                              ,nPin,&pPinIn);

  if(hr != S_OK || !pPinIn)
    return E_FAIL;
```

```
  //Соединяем
  hr = m_pGraph2->ConnectDirect(pOut, pPinIn, NULL);

  //Если контакт уже соединен
  if(hr == VFW_E_ALREADY_CONNECTED)
    hr = S_OK;

    return hr;
}
```

Остановка записи и перестройка графа для предварительного просмотра осуществляется в методе CCapture::StopCapture(), который вызывается в обработчике CFView2::OnBnClickedButStop(). При остановке графа производится отключение записывающего фильтра. Если записывался файл ASF, то фильтр возвращается в кеш. При записи файла AVI фильтр только отключается и устанавливается временное имя файла для того, чтобы фильтр освободил записанный файл, иначе доступ к нему будет заблокирован. При корректном окончании работы приложения временный файл удаляется самостоятельно, поскольку он имеет нулевой размер.

```
BOOL CCapture::StopCapture()
{
//Отключаем любые фильтры записи
  if(!DisconnectFilters(GetPinCapture(m_pCapV))
              || !DisconnectFilters(GetPinCapture(m_pCapA)))
    return FALSE;

 if(!IsFilterCache(m_pFSinkASF))
  {
    //Если записывался файл ASF возвращаем его в кеш
    if(S_OK != PushFilterToCache(m_pFSinkASF))
       return FALSE;
  }
 else //Файл AVI
  {
    //Запрос интерфейса
    CComQIPtr<IFileSinkFilter> pSink(m_pFSink);
    if(!pSink)
    return FALSE;
```

```
    //Установка временного имени файла для
    //освобождения записанного файла
    if(S_OK != pSink->SetFileName(L"C:\\Temp.avi",NULL))
       return FALSE;
  }

  //Пуск графа
  if(!StartPreview())
      return FALSE;

  return TRUE;
}
```

Метод DisconnectFilters производит разъединение двух фильтров. В параметрах метода следует указать выходной контакт первого фильтра либо сам фильтр.

```
BOOL CCapture::DisconnectFilters(IPin* pinOut,
                                 IBaseFilter* pFilter/*=NULL*/)
{
  //Если граф запущен, остановить
  if(m_bStartGraph && !StopGraph())
    return FALSE;

  //Проверка параметров
  if(!pinOut && !pFilter)
    return FALSE;

  //Если не указан контакт, ищем его на фильтре
  CComPtr<IPin> pOut;
  if(!pinOut && pFilter)
  {
    //Ищем выходной контакт на фильтре
    if(S_OK != m_pBuilder->FindPin(pFilter,PINDIR_OUTPUT
                              ,NULL,NULL,FALSE,0,&pOut))
       return FALSE;
  }
```

```
  //Если контакт указан, используем его
  if(pinOut && !pFilter)
  pOut = pinOut;

  //Проверяем его на подключение к следующему фильтру
  CComPtr<IPin> pConnect;
  if(S_OK == pOut->ConnectedTo(&pConnect))
  {
    //Контакт подключен, отключить его и входной контакт
    // следующего фильтра
    if(FAILED(m_pGraph2->Disconnect(pOut)))
      return FALSE;

    if(FAILED(m_pGraph2->Disconnect(pConnect)))
      return FALSE;
  }
  return TRUE;
}
```

Если пользователь после остановки записи нажимает кнопку "Запись", то производится перезапись последнего записанного файла. Реализация перезаписи записанного (текущего) файла выполняется в методе CCapture::StartCapture.

```
BOOL CCapture::StartCapture()
{
  //Определяем выходной контакт фильтра захвата видео
  CComPtr<IPin> pPinOutV;
  pPinOutV = GetPinCapture(m_pCapV);
  if(!pPinOutV)
    return FALSE;

  //Определяем выходной контакт фильтра звука
  CComPtr<IPin> pPinOutA;
  pPinOutA = GetPinCapture(m_pCapA);
  if(!pPinOutA)
```

```
    return FALSE;

  //Выделяем расширение имени текущего файла
  wstring wext = GetExtFileName(m_wstrNameFile);
  if(wext == L".asf")
  {
    //Записывается файл ASF, подключаем фильтр ASF
    if(S_OK != m_pGraph2->Render(pPinOutA))
      return FALSE;

    //Подключаем контакт видео
    if(S_OK != m_pGraph2->Render(pPinOutV))
      return FALSE;

    //Установка имени не требуется, фильтр сохраняет
    //имя текущего файла. Пуск графа.
    if(StartPreview())
      return FALSE;

    return TRUE;
  }

  if(wext == L".avi")
  {
    //Подключаем фильтры AVI
    if(S_OK != ConnectFilters(pPinOutV,NULL,m_pFMux))
      return FALSE;

    //Подключаем канал звука
    if(S_OK != ConnectFilters(pPinOutA,NULL,m_pFMux))
      return FALSE;

    CComQIPtr<IFileSinkFilter> pS(m_pFSink);
    if(!pS)
      return FALSE;
```

```
  //Устанавливаем имя текущего файла
  if(S_OK != pS->SetFileName(m_wstrNameFile.c_str(), NULL))
    return FALSE;

  //Пуск графа
  if(StartPreview())
    return FALSE;

    return TRUE;
  }
  return FALSE;
}
```

Чтобы проверить результат работы при записи файлов в методе CFView2::OnInitialUpdate допишем строки, инициализирующие директорию и имя файла. После запуска программы посмотрим на созданный граф с помощью утилиты GraphEdit. В графе должны быть соединенные между собой фильтр мультиплексора и фильтр записи, как показано на рисунке 7.1.

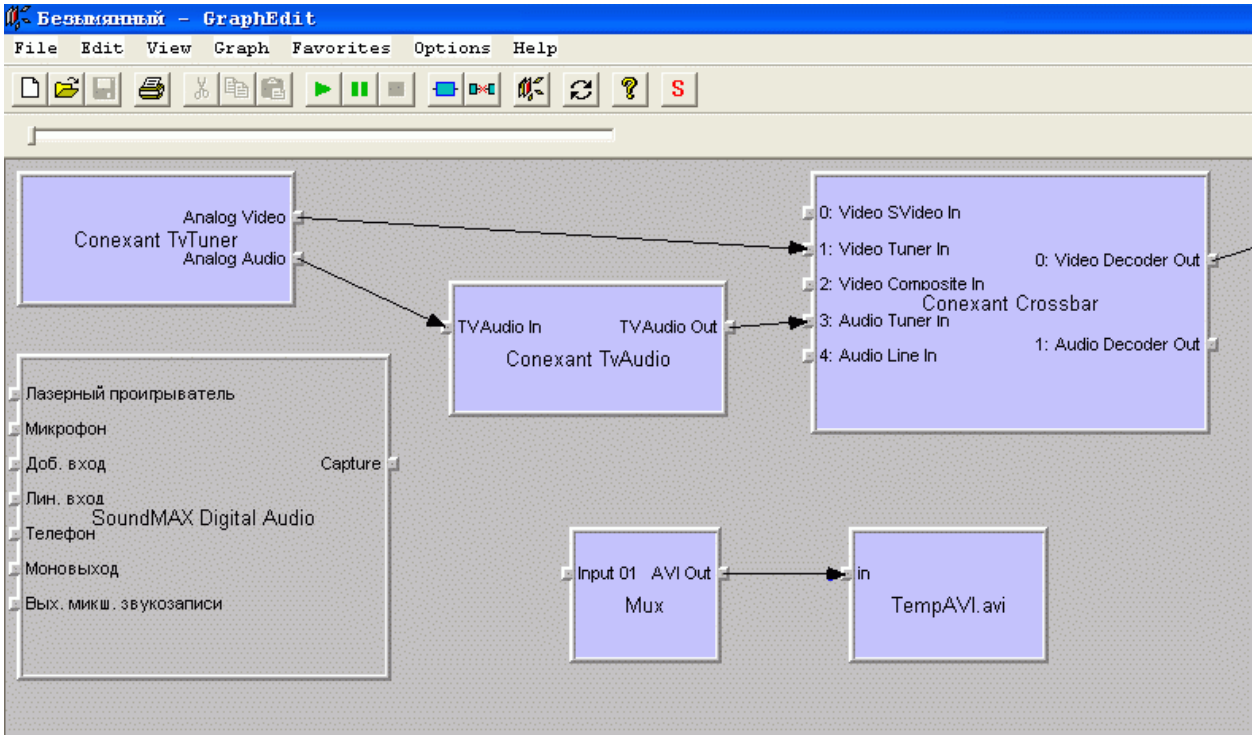

Рис.7. 1. Установка фильтров записи.

К сожалению, утилита GraphEdit не позволяет посмотреть, какие фильтры находятся в кеше, но можно увидеть в динамике изменение графа после включения записи файла. Для этого нажимаем кнопку "Новый файл", затем,

не закрывая утилиту, повторно подключаемся к графу (Ctrl+G). Для обновления состояния экрана утилиты удобней использовать кнопку . Утилита обновляет свой экран, и мы увидим граф, выполняющий запись файла (рис.2).

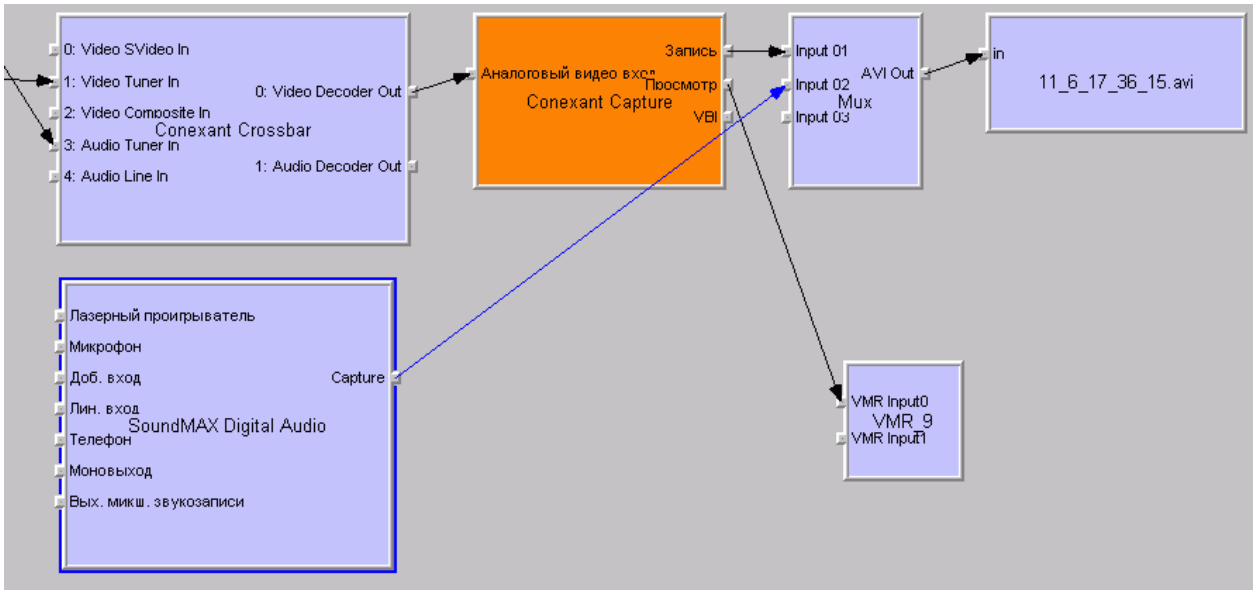

Рис.7. 2. Запись файла AVI.

Точно также можно увидеть изменения графа при переходе к другим режимам записи файла. Даже без измерения времени перестройки графа можно заметить, что использование кеша существенно сокращает переход от одного типа записи файла к другому. Хранение фильтра записи в графе или кеше также значительно ускоряет начало записи файла, что может быть важным в некоторых применениях.

Проект, выполняющий запись файлов находится в папке под именем ProjectState7. Дальнейшая работа будет проводиться с проектом в папке ProjectState8.

Управление профилями фильтра ASF

Если профиль по умолчанию не удовлетворяет требованиям, фильтр ASF должен конфигурироваться после установки в граф и до подключения входных контактов. Фильтр также позволяет изменять профиль при подключенных контактах. Профиль по умолчанию обеспечивает невысокое качество записи, которое может не удовлетворить пользователя. Обычно устанавливается профиль для локальных сетей со скоростью передачи 256 kb за секунду. Без возможности выбора и установки профиля из набора системных профилей фильтр ASF может стать не лучшим решением при записи сжатых файлов.

Конфигурирование фильтра осуществляется с помощью интерфейса IConfigAsfWriter . Использование этого интерфейса требует установки Windows Media Format SDK и включения основных заголовочных файлов dshowasf.h и wmsdk.h из состава Windows Media Format SDK, а также файла библиотеки Wmvcore.lib.

Интерфейс IConfigAsfWriter.

Методы	Описание
ConfigureFilterUsingProfileGuid	Конфигурирует фильтр, основываясь на предопределенных профилях Windows Media Format SDK, которые указываются их идентификаторами GUID.
ConfigureFilterUsingProfile	Конфигурирует фильтр, основываясь на профилях определенных приложением.
ConfigureFilterUsingProfileId	Конфигурирует фильтр, основываясь на системных профилях, которые указываются их идентификаторами профиля (ID). (Профили только для Windows Media Format 4.0)
GetCurrentProfileGuid	Возвращает GUID текущего профиля, определенного Windows Media Format SDK.
GetCurrentProfile	Возвращает текущий профиль, определенный приложением.
GetCurrentProfileId	Возвращает идентификатор текущего системного профиля. (Только для Windows Media Format 4.0)
GetIndexMode	Возвращает режим индексации файла.
SetIndexMode	Устанавливает режим индексации файла.

Таблица 2. Интерфейс IConfigAsfWriter.

Системные профили могут быть четырех версий 4.0, 7.0, 8.0, 9.0. Профиль версии 4.0 на платформе Windows XP уже отсутствует, а профиля версии 9.0 еще нет. Существует утилита в составе DirectX 9.0, позволяющая просмотреть профили, имеющиеся в системе. Она расположена по адресу [SDK root]\ bin\DXUtils и называется Profile Enumerator.

Для изменения профиля фильтра существует несколько способов:

- Первый способ это предоставление пользователю страницы свойств фильтра, которая выводит список системных профилей с текстовым описанием свойств профиля. Способ простой и неудобный, поскольку страница свойств выводится на английском языке, список профилей

большой. Описание профиля приводится в одной строке, которая требует горизонтальной прокрутки и содержит лишнюю информацию не связанную с содержанием профиля. Для пользователя не знающего английского языка она просто бесполезна.

- Второй способ предполагает предоставление пользователю списка профилей, который может быть предварительно отредактирован и адаптирован. Этот способ потребует использования методов интерфейса IConfigAsfWriter, что возможно только при условии установки Windows Media Format SDK. В этом случае предоставляются широкие возможности по управлению профилями вплоть до создания собственного специфического профиля.

Рассмотрим реализацию управления системными профилями фильтра ASF вторым способом. Управление системными профилями фильтра записи ASF разместим в классе CFView2. На вкладке этого класса будем создавать органы управления профилями. При смене профиля, в классе CCapture переменной m_profileGuid будет присваиваться новое значение.

Управление профилями в Windows Media Format SDK осуществляет объект называемый менеджером профилей (profile manager object). При создании этого объекта возвращается интерфейс IWMProfileManager, который позволяет запросить еще несколько интерфейсов для управления профилями.

Методы интерфейса IWMProfileManager приведены в таблице.

Метод	Описание
CreateEmptyProfile	Создает пустой профиль.
GetSystemProfileCount	Возвращает число системных профилей.
LoadProfileByData	Создает объект профиля и заполняет его данными из существующего профиля, сохраненного в виде строки. Применяется для управления заказными профилями.
LoadProfileByID	Создает объект профиля и заполняет его данными из системного профиля. Использует GUID для поиска данных профиля.
LoadSystemProfile	Создает объект профиля и заполняет его данными из системного профиля. Использует индекс профиля для поиска его данных.
SaveProfile	Сохраняет профиль в строке XML. Можно сохранить профиль на диске, записав строку в файл .prx.

Таблица 3. Интерфейс IWMProfileManager.

Из интерфейса IWMProfileManager можно запросить другие интерфейсы для управления профилями. Список этих интерфейсов приведен в таблице.

Интерфейс	IID
IWMCodecInfo	IID_IWMCodecInfo
IWMCodecInfo2	IID_IWMCodecInfo2
IWMCodecInfo3	IID_IWMCodecInfo3
IWMProfileManager2	IID_IWMProfileManager2
IWMProfileManagerLanguage	IID_IWMProfileManagerLanguage

Таблица 4. Список интерфейсов.

Из перечисленных дополнительных интерфейсов для управления системными профилями будут нужны IWMProfileManager2 и IWMProfileManagerLanguage. Методы этих интерфейсов приведены в таблицах.

Интерфейс IWMProfileManager2.

Метод	Описание
GetSystemProfileVersion	Возвращает номер версии используемого системного профиля.
SetSystemProfileVersion	Устанавливает номер версии используемого системного профиля.

Таблица 5.Интерфейс IWMProfileManager2.

Интерфейс IWMProfileManagerLanguage.

Метод	Описание
GetUserLanguageID	Возвращает идентификатор языка текущего системного профиля.
SetUserLanguageID	Указывает, какой идентификатор языка использовать для текущего системного профиля.

Таблица 6. Интерфейс IWMProfileManagerLanguage.

Объект профиля предоставляет интерфейс IWMProfile, который служит для настройки заказных профилей. Для управления системными профилями он необходим в небольшой степени для получения имени профиля и его описания. Из этого интерфейса можно запросить производные интерфейсы IWMProfile2 и IWMProfile3. В таблице приведены методы интерфейса IWMProfile и их краткое описание.

Методы	Описание
AddMutualExclusion	Добавляет объект взаимного исключения к профилю. Объект взаимного исключения определяет, какой из потоков в установленное время может проходить на выход.
AddStream	Добавляет поток к профилю.
CreateNewMutualExclusion	Создает объект взаимного исключения для профиля.
CreateNewStream	Создает конфигурирующий объект для профиля.
GetDescription	Возвращает описание профиля.
GetMutualExclusion	Возвращает объект взаимного исключения из профиля.
GetMutualExclusionCount	Возвращает число объектов взаимного исключения из профиля.
GetName	Возвращает имя профиля.
GetStream	Возвращает из профиля поток по его индексу.
GetStreamByNumber	Возвращает из профиля поток по его номеру.
GetStreamCount	Возвращает число потоков в профиле.
GetVersion	Возвращает номер версии профиля.
ReconfigStream	Позволяет изменить конфигурацию потока включенного в профиль.
RemoveMutualExclusion	Удаляет из профиля объект взаимного исключения.
RemoveStream	Удаляет поток из профиля.
RemoveStreamByNumber	Удаляет поток из профиля по его номеру.
SetDescription	Устанавливает описание профиля.
SetName	Устанавливает имя профиля.

Таблица 7.Интерфейс Интерфейс IWMProfile.

Интерфейс IWMProfile2 предоставляет один метод.

Метод	Описание
GetProfileID	Возвращает идентификатор (GUID) профиля.

Таблица 8. Интерфейс IWMProfile2.

Интерфейс IWMProfile3 предоставляет методы, ни один из которых не нужен для управления системными профилями. Этот интерфейс вводит управление двумя новыми объектами профиля. Один объект управляет полосой пропускания потоков, а второй вводит приоритетность потоков. Ознакомиться с методами интерфейса можно в документации MSDN.

Для управления системными профилями в редакторе ресурсов на вкладке класса CFView2 создадим три элемента управления, которым будут соответствовать три переменные класса.

```
//Список версий WM системных профилей
CListBox m_versionProfile;

//Имя системного WM профиля
CComboBox m_nameProfile;

//Описание системного WM профиля
CEdit m_descriptionProfile;
```

Для хранения информации о профиле в заголовочном файле класса CFView2объявим структуру WM_PROFILE и вектор структур vector<WM_PROFILE> m_vPrf.

```
//Структура для хранения данных профилей
struct WM_PROFILE
{
  //Имя профиля
  string sName;

  //Описание профиля
  string sDescription;

  //Указатель на интерфейс профиля
  CComPtr<IWMProfile> pProfile;
};
```

Данные можно было бы сохранять и в элементах управления m_nameProfile и m_descriptionProfile, но в элементе управления m_nameProfile указатель на профиль IWMProfile* можно сохранить только в виде указателя на void*. Преобразование указателей CComPtr к void* и обратно не допускается. Это приводит к тому, что возникает необходимость, заботится об освобождении интерфейсов. Проще сохранять данные в векторе, а для установки соответствия данных между вектором и элементом управления хранить индексы вектора в данных элемента управления m_nameProfile. Такой подход также позволит при необходимости организовать поиск и сортировку данных с использованием алгоритмов STL. Хранение указателей в векторе потребует отдельного метода для его очистки, чтобы сначала удалить объекты CComPtr, а затем сам вектор.

После выбора номера версии профилей из списка версий все данные этой версии записываются в вектор для последующего отображения в списке имен профилей и для отображения описания выбранного профиля в поле описа-

ния. Код метода CFView2::myGetListSystemProfile, выполняющего запись данных всех профилей выбранной версии в вектор можно посмотреть в архиве проектов.

Для вывода описания профиля на определенном (русском) языке в системной папке, например C:\WINDOWS\, должен быть файл wmprfRUS.prx, кроме него там же должен быть файл WMSysPr9.prx. Оба файла записываются в системную папку автоматически при установке WMSDK. Если файлы были случайно удалены, их нужно скопировать из папки установки WMSDK.

Файл wmprfRUS.prx содержит описание профилей на русском языке. Если для определенной версии профилей отсутствует такое описание, можно откорректировать файл в любом текстовом редакторе, сохраняя его структуру. Описание профиля на английском языке и остальные данные можно взять из файла WMSysPr9.prx. Например, начало описания профиля версии 8.0 выглядит следующим образом. Версия указана в десятичном формате.

```
<profile version="524288"
 name="Windows Media Video 8 for Dial-up Modems or Single-channel ISDN (28.8 to 56 Kbps)"
 guid="{D66920C4-C21F-4ec8-A0B4-95CF2BD57FC4}"
 description="Use this multiple bit rate profile for target audiences with dial-up modems or single-channel ISDN connections (bandwidth is between 28.8 Kbps and 56 Kbps).">
 <streamconfig majortype="{73647561-0000-0010-8000-00AA00389B71}"
……..
```

Произвольная строка описания одного профиля на русском языке в файле wmprfRUS.prx выглядит следующим образом.

```
<profile-local version="458752" name="Видео для цветных PDA-устройств (150 Кбит/с)" guid="{0F472967-E3C6-4797-9694-F0304C5E2F17}" description="Используйте этот профиль при создании файлов видео, воспроизводимых на большинстве карманных ПК.">
</profile-local>
```

В качестве примера создадим описание профиля на русском языке. Для этого из описания профиля на английском языке (WMSysPr9.prx) берем номер версии и GUID без изменений, а имя и описании переводим на русский язык. Эту строку добавляем в конце файла wmprfRUS.prx.

```
<profile-local version="524288" name="Пример перевода профиля" guid="{D66920C4-C21F-4ec8-A0B4-95CF2BD57FC4}" description="Используйте этот профиль для модемов 28.8."> </profile-local>
```

Теперь в описании профиля будет отображаться строка на русском языке. Таким образом, можно адаптировать описание списка профилей и упростить его понимание пользователем.

Заполнение элементов управления данными профилей первоначально производится в методе CFView2::OnInitialUpdate().При первоначальном заполнении списков профилей текущей версией назначается версия 7.0. Метод myFillListVersionWMProfile заполняет общий список всех возможных версий профилей, в том числе и отсутствующих в настоящее время. Для заполнения списка имен профилей текущей версии написан метод myFillListNameWMProfile. Метод myFillDescriptionProfile заполняет описание профиля для выбранного текущего профиля. Он заполняет поле CEdit.

Смена текущего профиля производится в обработчике CFView2::OnCbnSelchangeComboNameProfile при выборе пользователем нового имени профиля. При этом обновляется идентификатор профиля (GUID) в классе CCapture (m_profileGuid). Идентификатор используется для установки профиля при записи файла в методе CCapture::SetNameFile.

Новый идентификатор профиля используется при смене имени файла. Если производится перезапись старого файла, то имя файла не изменяется и перезапись производится со старым профилем.

Дополнительные способы управления записью файлов

Определенный интерес для управления потоком данных при записи и предварительном просмотре предоставляет метод ICaptureGraphBuilder2::ControlStream. Этот метод определяет выходные контакты на всех фильтрах захвата в графе с учетом критериев поиска и затем вызывает методы интерфейса IAMStreamControl StartAt и StopAt для установки времени включения и выключения потока данных на этих контактах. Время исчисляется от момента пуска графа. Интерфейс IAMStreamControl предоставляется на выходных контактах записи фильтра захвата видео и звука и на входном контакте фильтра мультиплексора AVI Mux. Этот интерфейс также можно использовать, но метод ControlStream избавляет от необходимости поиска фильтров и контактов.

Метод позволяет производить фиксировать начало и конец записи с точностью до одного кадра изображения. Он блокирует передачу потока данных, но он никак не влияет на фильтр записи файлов, запись файла продолжается в режиме стоп кадра. Метод работает только при запущенном графе, при остановленном графе он не имеет эффекта до запуска графа. Для кон-

такта предварительного просмотра можно использовать метод только для немедленного пуска и остановки просмотра, поскольку поток на этом контакте не имеет временного штампа. Метод не поддерживает предварительный просмотр для контакта видеопорта, потому что тот посылает поток прямо в видеокарту.

Каждый контакт при старте посылает сообщение EC_STREAM_CONTROL_STARTED

и при остановке сообщение
EC_STREAM_CONTROL_STOPPED.

Когда метод находит соответствующий контакт, он ищет ниже по потоку второй контакт, поддерживающий интерфейс IAMStreamControl, обычно это мультиплексор. Если он его находит, то также запускает или останавливает поток на входном контакте. В результате генерируется две пары сообщений по одной паре на каждом фильтре. В первом параметре сообщения посылается указатель на контакт, а во втором параметре данные пользователя. Данные пользователя используются только в сообщении об остановке от нижнего фильтра. Ожидание поступление этого сообщения гарантирует, что нижний фильтр принял последнюю выборку данных.

Если нижний фильтр не поддерживает интерфейс IAMStreamControl, то метод возвращает S_FALSE и сообщение об остановке может быть принято до отображения последней выборки. Обработка сообщений позволяет предпринять необходимые действия по сигналу остановки или пуска.

Наибольшее возможное значение времени REFERENCE_TIME, которое можно установить, равно константе MAXLONGLONG.

```
HRESULT ControlStream(
  const GUID *pCategory,
  const GUID *pType,
  IBaseFilter *pFilter,
  REFERENCE_TIME *pstart,
  REFERENCE_TIME *pstop,
  WORD wStartCookie,
  WORD wStopCookie
);
```

Параметры.

pCategory
[in]
Указатель на категорию контакта (член структуры AMPROPERTY_PIN_CATEGORY). Не может иметь значение NULL.

pType

[in]
Указатель на GUID главного медиа типа, или NULL. Если параметр устанавливается в NULL, то и параметр pFilter должен быть установлен в NULL.

pFilter
[in]
Указатель на интерфейс IBaseFilter фильтра, который будет управляться. Для управления всеми фильтрами параметр должен быть установлен в NULL. Если один из фильтров не поддерживает интерфейс IAMStreamControl, метод возвратится с ошибкой.

pstart
[in]
Указатель на переменную, которая содержит стартовое время. Если величина равна MAXLONGLONG (0x7FFFFFFFFFFFFFFF), метод закрывает предыдущий запрос на старт. Если величина времени равна нулю, контакт немедленно запускается, если запущен граф.

pstop
[in]
Указатель на переменную, которая содержит время остановки. Если величина равна MAXLONGLONG (0x7FFFFFFFFFFFFFFF), метод закрывает предыдущий запрос на остановку. Если величина времени равна нулю, контакт немедленно останавливается, если запущен граф.

wStartCookie
[in]
Величина посылаемая, как второй параметр в сообщении EC_STREAM_CONTROL_STARTED.

wStopCookie

[in]
Величина посылаемая, как второй параметр в сообщении EC_STREAM_CONTROL_STOPPED.

Коды возврата.

Код	Описание
S_FALSE	Один из нижних фильтров не послал сообщение об остановке.
S_OK	Без ошибок.
E_FAIL	Не найден контакт или контакт не поддерживает управление.
E_POINTER	Указатель равен NULL.

Дополнительно метод может возвращать ошибку E_NOINTERFACE, если один из фильтров не поддерживает интерфейс IAMStreamControl.

Код, приведенный ниже, производит остановку контакта записи на фильтре захвата видео с посылкой сообщения.

```
HRESULT hr;
```

```
  REFERENCE_TIME start = MAXLONGLONG;
  REFERENCE_TIME stop = 0;

 hr= m_pBuilder->ControlStream(
        &PIN_CATEGORY_CAPTURE,
        &MEDIATYPE_Video
        ,m_pCapV    //управляемый фильтр
        ,&start     // время старта = MAXLONGLONG
        ,&stop      //=NULL немедленная остановка
        ,0
        ,0x0077     //посылается сообщение  останова,
                    //0x0077 это пример
  );
```

Для запуска контакта следует поменять местами значение переменных start и stop.

Примечание.

В приводимом коде следует обратить внимание на то, что параметры времени start и stop не должны устанавливаться в NULL или нуль (0). В документации нечетко сформулировано это условие, в описании параметра сказано " If the value is NULL…", что воспринимается как значение указателя. В действительности это относится к значению величины, на которую указывает указатель, то есть к значению времени. Даже в примере, который включен в документацию DirectX, есть эта ошибка, что приводит к полной неработоспособности метода при его успешном завершении. Если присвоить указателю pstart или pstop значение NULL, то в действительности значение времени устанавливается в MAXLONGLONG.

Для управления стартом и остановкой потока данных написан метод CCapture::StartAtTime. Метод вызывается перед запуском графа и открывает поток данных в указанное время после запуска графа. Запись файла начинается сразу после пуска графа. Таким образом, некоторое время производится запись в режиме стоп-кадра. После остановки потока данных в указанное время запись файла не прекращается. Для остановки записи файла следует обработать сообщение об остановке потока данных и остановить граф, либо перевести его в режим предварительного просмотра.

```
HRESULT CCapture::StartAtTime(DWORD timeStart, DWORD timeStop)
{
  if(timeStart >= timeStop)
    return E_FAIL;
```

```
  HRESULT hr;
  REFERENCE_TIME start = timeStart * 10000;
  REFERENCE_TIME stop = timeStop * 10000;

  hr= m_pBuilder->ControlStream(&PIN_CATEGORY_CAPTURE,
          &MEDIATYPE_Video, m_pCapV, &start,&stop,0
          ,0x0077 //Посылается сообщение, 0x0077 это пример
    );
  return hr;
}
```

Обработка сообщений EC_STREAM_CONTROL_STARTED и EC_STREAM_CONTROL_STOPPED имеет некоторые особенности обнаруженные в процессе экспериментальной проверки. Каждое из этих сообщений по данным документации имеет два параметра:

- Указатель на контакт (IPin), который производит данные. Это параметр pSender в сообщении.
- Пользовательские данные, посылаемые при условии, что параметры wStartCookie и wStopCookie не равны нулю. Это параметр dwCookie в сообщении.

В документации DirectShow указано, что каждое сообщение посылается дважды, одно от контакта фильтра сбора и второе от входного контакта ниже лежащего фильтра (мультиплексора). Параметр wStopCookie используется только в сообщении от входного контакта нижележащего фильтра. В действительности сообщения передаются попарно, только если параметры wStartCookie и wStopCookie не равны нулю, в противном случае каждое из сообщений передается один раз.

Например, при нулевых параметрах wStartCookie и wStopCookie принимались такие параметры сообщений:

- Первый параметр при старте pSender был равен величине 0x012e70a0.
- Второй параметр при старте dwCookie был равен величине 0x000f4240.
- Первый параметр при остановке pSender был равен величине 0x012f9ef0.
- Второй параметр при остановке dwCookie был равен величине 0x000f4240.

Во втором случае при значениях wStartCookie и wStopCookie равных, соответственно, случайным кодам 0x76 и 0x77, сообщения принимались попарно и их параметры были равны.

- Первое сообщение при старте. Первый параметр pSender был равен величине 0x012f9ef0.
- Первое сообщение при старте. Второй параметр dwCookie был равен величине 0x00000076.
- Второе сообщение при старте. Первый параметр pSender был равен величине 0x012d2488.
- Второе сообщение при старте. Второй параметр dwCookie был равен величине 0x000f42b6.

- Первое сообщение при остановке. Первый параметр pSender был равен величине 0x012f9ef0.
- Первое сообщение при остановке. Второй параметр dwCookie был равен величине 0x000f42b7.
- Второе сообщение при остановке. Первый параметр pSender был равен величине 0x012d2488.
- Второе сообщение при остановке. Второй параметр dwCookie был равен величине 0x00000077.

Из приведенных данных видно, что в параметре dwCookie всегда передается определенный код значение, которого не выяснено, и только при парной передаче сообщений можно уверенно выделить пользовательские данные. Для этого нужно вводить подсчет сообщений.

В первом параметре передается указатель на контакт, который производит данные. Попытка определить, какому фильтру принадлежит контакт или найти данные самого контакта, не увенчалась успехом. При проверке параметра pSender вообще не удалось использовать его как указатель интерфейса IPin. Для целей поиска и проверки в проекте написан метод для сравнения параметра pSender с контактами всех фильтров графа EnumFiltersAndPins(long p).

Синхронизация потоков при записи файлов AVI

При записи файлов AVI следует обратить внимание на интерфейс IConfigAviMux, представленный на фильтре мультиплексора. Он управляет

относительной синхронизацией звукового и видео потоков, а также обеспечивает совместимость старого формата файлов и нового формата AVI 2.0, обеспечивающего запись файлов размером более 1 GB.

При установке синхронизирующего потока рекомендуется выбирать в качестве ведущего (master) потока звуковой поток. Таким образом, обеспечивается более высокое качество записи, поскольку вырезка из звукового потока данных при синхронизации более заметна для пользователя, чем удаление нескольких кадров изображения. Синхронизировать источники следует в том случае, если запись звука производится от постороннего источника, например, производится наложение звука при записи изображения. Если записывается телевизионная программа, то в синхронизации потока нет нужды.

Метод	Описание
GetMasterStream	Определяет индекс потока, относительно которого синхронизируются другие потоки. Индекс потока соответствует номеру подключенного контакта на фильтре.
GetOutputCompatibilityIndex	Определяет, создает ли фильтр индекс старого формата в дополнение к новому индексу формата.
SetMasterStream	Устанавливает индекс потока, относительно которого будут синхронизироваться другие потоки.
SetOutputCompatibilityIndex	Создает или не создает старый индекс формата в дополнение к новому индексу формата.

Метод, представленный ниже, производит установку ведущего синхронизирующего потока.

```
BOOL CCapture::SetMasterStream(long ind)
{
 HRESULT hr;
  //Возвращаем интерфейс на фильтре мультиплексора
  CComQIPtr<IConfigAviMux> pCnf(m_pFMux);
  if(!pCnf)
  return FALSE;
  //Установка ведущего потока
  hr = pCnf->SetMasterStream(ind);

  if(hr !=S_OK)
   return FALSE;
```

```
  return TRUE;
}
```

Управление звуком и видео при записи файлов AVI

Для управления процессом записи на диск видео и звуковых данных фильтр AVI Mux предоставляет интерфейс IConfigInterleaving. Данный интерфейс управляет двумя параметрами. Первый параметр "interleaving" определяет, каким образом осуществлять чередование видео и звуковых выборок при записи файла. Второй параметр, которым управляет интерфейс это "preroll". Параметр "preroll" определяет начальную порцию данных, которая записывается или воспроизводится, чтобы обеспечить вхождение источника сигнала (устройства) в режим стабильной работы. Возможный перевод параметра "предварительная прокрутка". В различных системах регистрации параметров существует термин (команда) "протяжка", по которой проводится включение всех устройств и некоторое время производится запись тестовых или нулевых параметров. После выхода устройств из пускового режима и начала стабильной работы, на регистрацию по команде "запись" поступают истинные параметры. Очевидно, что параметр "preroll" близок по смыслу к термину "протяжка".

Интерфейс использует перечисление InterleavingMode, константы которого определяют режим чередования потоков видео и звука. Например, константа INTERLEAVE_CAPTURE определяет режим, при котором производится перекрытая не буферизированная операция записи на диск. Если параметры выбраны правильно, то запись будет производиться без потерь. В противном случае, если например, выбрать звуковой буфер чрезмерно большим, часть кадров видео сигнала будет выброшена.

Интерфейс предоставляет следующие методы.

Метод	Описание
get_Interleaving	Возвращает время "preroll" для звука и частоту чередования потоков.
get_Mode	Возвращает режим чередования.
put_Interleaving	Устанавливает время "preroll" для звука и частоту чередования потоков.
put_Mode	Устанавливает режим чередования.

По умолчанию режим устанавливается без чередования (INTERLEAVE_NONE), то есть потоки записываются по мере их поступле-

ния, что обеспечивает максимальную производительность. Время "preroll" по умолчанию устанавливается равным нулю, а период чередования 1000 миллисекунд. Для обычных условий записи файлов нет необходимости в использовании данного интерфейса.

Установка видеоформата на контакте Capture фильтра захвата

Запись файлов возможна при установленном одинаковом видео формате на контакте предварительного просмотра (Preview) и контакте записи (Capture) фильтра захвата. Размер изображения для стандарта SECAM, PAL не может превышать 352x288 пикселей. Для стандарта NTSC размер не может превышать 320x240 пикселей. Запись файлов при установке размера изображения на контакте предварительного просмотра больше указанного значения невозможна, даже если на контакте записи установлен правильный размер. Запись файлов с размером изображения меньше 352x288 пикселей допускается. Для установки медиа формата на контакте записи фильтра захвата напишем метод CCapture::SetMediaFmtCapture, который является упрощенным вариантом метода, устанавливающего формат на контакте предварительного просмотра.

```
BOOL CCapture::SetMediaFmtCapture(GUID* psubtype
                  ,DWORD_PTR dwSize,LONGLONG frameInterval)
{
 HRESULT hr = S_OK;
  BOOL bSmart = FALSE;

  //Выходной контакт фильтра, на котором
  //устанавливается медиа формат
  CComPtr<IPin> pPinOut;

  CComPtr<IPin> pConnect;
  CComPtr<IAMStreamConfig> pConfig;

 string sErr="SetMediaFmtCapture";

  //Проверить состояние графа и остановить, если был запущен
  CComQIPtr<IMediaControl> pCtrl(m_pGraph2);
  ASSERT(pCtrl);
  OAFilterState stateF = GetStateGraph();
  if(stateF == State_Running || stateF == State_Paused)
  {
   if(S_OK != pCtrl->Stop())
      return FALSE;
  }
```

```
//Найти выходной контакт фильтра, на котором устанавливается
// медиа формат
if(S_OK != m_pBuilder->FindPin(m_pCapV,PINDIR_OUTPUT,
                &PIN_CATEGORY_CAPTURE,0,0,0,&pPinOut))
  return FALSE;

//Найти IAMStreamConfig для выходного контакта
if(S_OK != m_pBuilder->FindInterface(&PIN_CATEGORY_CAPTURE
            ,&MEDIATYPE_Video,m_pCapV,IID_IAMStreamConfig
            ,reinterpret_cast<void**>(&pConfig)))
  return FALSE;

//Отключить, если контакты подключены
if(S_OK == pPinOut->ConnectedTo(&pConnect))
{
 if(FAILED(m_pGraph2->Disconnect(pPinOut)))
    return FALSE;

 if(FAILED(m_pGraph2->Disconnect(pConnect)))
    return FALSE;
}
//Найти устанавливаемый медиа тип в векторе доступных типов
CMediaFmtV mfmt;
mfmt.m_pSubType = psubtype;
if(!FindStreamCaps(PIN_CATEGORY_CAPTURE,mfmt))
  return FALSE; //Тип не найден

//Заполнить структуру AM_MEDIA_TYPE найденными
//данными
AM_MEDIA_TYPE amt;
amt.majortype             = MEDIATYPE_Video;
amt.subtype               = *psubtype;
amt.bFixedSizeSamples     = mfmt.m_bFixedSizeSamples;
amt.bTemporalCompression  = mfmt.m_bTemporalCompression;
amt.formattype            = FORMAT_VideoInfo;
amt.pUnk                  = NULL;
amt.cbFormat              = sizeof(mfmt.m_vih);
```

```
  //Адрес структуры VIDEOINFOHEADER
  amt.pbFormat  = reinterpret_cast<BYTE*>(&mfmt.m_vih);

  //Установка частоты кадров
  mfmt.m_vih.AvgTimePerFrame = frameInterval;

  //Размеры изображения
  DWORD width = HIWORD(dwSize);
  DWORD height = LOWORD(dwSize);

  //Установка размеров изображения в структуре BITMAPINFOHEADER
  mfmt.m_vih.bmiHeader.biWidth = width;
  mfmt.m_vih.bmiHeader.biHeight = height;

  //Установить в структуре AM_MEDIA_TYPE размер
  //изображения в байтах, выровнять на границу DWORD.
  // Для форматов RGB размер в BITMAPINFOHEADER может быть 0
  amt.lSampleSize = mfmt.m_vih.bmiHeader.biSizeImage
                  = (((width + 3) & ~3) * height *
                    mfmt.m_vih.bmiHeader.biBitCount) /8;

  //Установка нового формата при отключенном контакте фильтра
  if(S_OK != pConfig->SetFormat(&amt))
    return FALSE;

  return TRUE;
}
```

Вызовы этого метода произведем после вызовов метода SetMediaFmtPreview. При первоначальной инициализации в обработчике CFView1::OnMyMessageEndInit метод будет устанавливать медиа формат по умолчанию и такой же, как на контакте предварительного просмотра.

При вызове в обработчике CFView1::OnBnClickedButSetformat можно произвести несколько экспериментов с различными комбинациями форматов на контактах.

```
    szimage = MAKELONG(288,352);
    if(!pCap->SetMediaFmtCapture(pguid,szimage,interval))
        throw serr;
```

Результат должен быть таким, как указывалось выше. Для полноценной работы следует предусмотреть автоматическую смену формата при переходе от режима предварительного просмотра к записи и восстановление предварительного просмотра с предыдущим размером изображения.

Использование кодеров при записи файлов AVI

Для записи сжатых файлов AVI можно использовать различные кодеры, установленные в системе. Кодеры могут быть как поставляемые с операционной системой, так и сторонних производителей. Кодеры сторонних производителей должны быть зарегистрированы в системе, что обычно происходит автоматически при их установке. Для управления фильтрами в системе существуют удобные утилиты менеджеров, доступные для свободной закачки, например по адресу www.softella.com.

Для реализации записи файлов AVI с применением кодеров произведем модернизацию вкладки класса CFView2, на которой установим два элемента CComboBox. Они будут предоставлять пользователю списки кодеров для видео канала и звукового канала. Заполнение списков кодеров производится с помощью системного перечислителя аналогично перечислению устройств. Перечисление фильтров производится для категории видео компрессоров CLSID_VideoCompressorCategory и для категории звуковых компрессоров CLSID_AudioCompressorCategory. Для хранения моникеров кодеров используем массив класса CInterfaceArray из коллекции массивов ATL 7, который предназначен для хранения интерфейсных указателей и обеспечивает необходимое управление памятью. Хранение интерфейсных указателей на моникеры (IMoniker*) непосредственно в элементе CComboBox вполне возможно, но более хлопотно, так как требует преобразования указателей к типу void* и обратно, а также освобождения их после использования и подсчета ссылок на моникер. Намного проще и безопасней хранить в элементе CComboBox индексы массива CInterfaceArray. Для хранения моникеров кодеров в заголовочном файле FView2.h объявим две переменные класса CInterfaceArray.

```
  //Массив интерфейсных указателей IMoniker видео кодеков
  CInterfaceArray<IMoniker,&IID_IMoniker> m_arIMonCodec_V;

  //Массив интерфейсных указателей IMoniker звуковых кодеков
  CInterfaceArray<IMoniker,&IID_IMoniker> m_arIMonCodec_A;
```

Перечисление и заполнение элементов CComboBox производится в методе myEnumFilter, который вызывается в методе CFView2::OnInitialUpdate(). Часть метода myEnumFilter, заполняющая список видео компрессоров приводится ниже. Полностью метод можно просмотреть в проекте.

```
void CFView2::myEnumFilter(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 системный перечислитель для фильтров
    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 видео. Категорию
    //компрессора выбираем из таблицы категорий.

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

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

    if(FAILED(hr))
          throw hr;

    if(hr== S_OK)
    {
      //Нулевая строка без кодера
      m_arIMonCodec_V.Add(NULL);
      m_codecV.InsertString(0,"Без кодера");
      m_codecV.SetItemData(0,0);

      //Перечисление  фильтров компрессоров 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  фильтра компрессора
          // из хранилища
          if(S_OK == pPropBag->Read(L"FriendlyName"
                                  , &varName, 0))
          {
            //Преобразуем строку из BSTR (макрос ATL7)
            CString sName = COLE2T(varName.bstrVal);

            //Сохраняем строку, сохраняем указатель
            //моникера
            int idx =
            static_cast<int>(m_arIMonCodec_V.Add(pMoniker));
            m_codecV.InsertString(idx,sName);
            m_codecV.SetItemData(idx,idx);
          }
          pPropBag = NULL;
        }
        pMoniker = NULL;
      }
    }
    pEnumCat = NULL;

    //Далее создается класс перечислителя для Audio компрессора

    ……...

    //Установить нулевую строку списка видимой
    m_codecV.SetCurSel(0);
    m_codecA.SetCurSel(0);
}
  catch(HRESULT h)
  {
  sErr += "Метод myEnumDevice  \n\r";
  CTVshowApp::MessageError(h, sErr);
  }

}
```

При выборе пользователем строки с именем определенного компрессора в обработчике CFView2::OnCbnSelchangeCodecVideo(), или для звукового компрессора в обработчике CFView2::OnCbnSelchangeCodecAudio(), производится вызов метода из класса CCapture для установки в граф соответствующего кодера. В классе CCapture создаются две переменные для хранения указателей на используемые кодеры m_pCodecV и m_pCodecA.

Метод AddCodecVForMoniker должен выполнять установку кодера только при записи файлов AVI. При установке нового кодера следует удалить из графа предыдущий кодер. Удаление предыдущего кодера может быть сопряжено с некоторыми проблемами, поскольку при подключении некоторых видеокодеров с помощью метода ICaptureGraphBuilder2::RenderStream вместе с кодером может быть установлен трансформирующий фильтр выше по потоку. Для канала звука корректировка методов значительно проще, поскольку трансформирующих фильтров в канал звука не устанавливается. В результате возникает задача корректного управления в графе не одним фильтром кодера, а цепочкой фильтров, которую следует подключать и удалять при переключении режимов записи и выборе нового кодера. Для удаления цепочки фильтров написан метод RemoveFilterChain, который рассмотрим далее.

```
BOOL CCapture::AddCodecVForMoniker(
CComPtr<IMoniker> pMoniker)
{
  //Если запись, возврат
  if(m_bCapture)
    return FALSE;

  //Не подключать для файлов ASF
  wstring wsext = GetExtFileName(m_wstrNameFile);
  if(wsext == L".asf")
    return FALSE;

  //Удалить старый кодер из графа с предшествующими фильтрами
  if(m_pCodecV)
  {
    if(S_OK ==
            RemoveFilterChain(GetPinCapture(m_pCapV),m_pCodecV))
      m_pCodecV = NULL;
    else
      return FALSE;
  }

  //Нет кодера
  if(!pMoniker)
  {
    StartPreview();
      return TRUE;
  }

  //Устанавливаем кодер в граф
  if(S_OK != m_pGraph2->AddSourceFilterForMoniker(pMoniker,NULL
        ,GetNameFilterFromMoniker(pMoniker).c_str(),&m_pCodecV))
    return FALSE;

  //Возвращение свойств компрессора
  //LONG capabilities;
```

```
  //GetInfoCompressor(capabilities);

  return TRUE;
}
```

Корректное удаление цепочки фильтров производится в методе RemoveFilterChain, который производит удаление нескольких соединенных фильтров. Поскольку при установке кодера метод ICaptureGraphBuilder2::RenderStream не возвращает никакой информации об установленных промежуточных фильтрах, то необходимо определить указатель на первый удаляемый фильтр в цепочке. Конечный удаляемый фильтр известен, это кодер. Для определения первого фильтра в цепочке в первом параметре метода передается указатель на выходной контакт фильтра, к которому подключается цепочка фильтров с кодером, а во втором параметре указатель на последний удаляемый фильтр, которым будет кодер. Если первый удаляемый фильтр в цепочке подключен, то после определения указателя на него производится удаление всей цепочки. В противном случае, производится поиск первого неподключенного фильтра в цепочке, начиная снизу по потоку от кодера и до первого неподключенного фильтра, который и будет первым удаляемым фильтром.

```
HRESULT CCapture::RemoveFilterChain(IPin* pPOut
                            , IBaseFilter* pFEnd)
{
  HRESULT hr;
  PIN_INFO info;

  //Запрашиваем интерфейс
  CComQIPtr<IFilterChain> pFCh(m_pGraph2);
  if(!pFCh)
    return E_FAIL;

  //Найти указатель на первый фильтр, подключенный к
  //выходному контакту
  CComPtr<IPin> pPConnect;
  hr = pPOut->ConnectedTo(&pPConnect);

  //Удаляем цепочку, если контакт соединен
  if(S_OK == hr)
  {
    if(S_OK != (hr = pPConnect->QueryPinInfo(&info)))
      return hr;

    //Удаляем цепочку
    hr = pFCh->RemoveChain(info.pFilter,pFEnd);
    info.pFilter->Release();

    return hr;
```

```
  }

  //Если контакт был не подключен, то ищем цепочку, начиная с
  //последнего фильтра и до неподключенного входного контакта
  if(hr == VFW_E_NOT_CONNECTED)
  {
    //Возвращаем подключенный  выше по потоку фильтр
    CComPtr<IBaseFilter> pFUp , pF;
    pF = pFEnd;

    //Ищем первый в цепочке неподключенный фильтр
    while(S_OK == GetUpFilter(pF,&pFUp))
    {
      pF = pFUp;
      pFUp = NULL;
    }

    //Удаляем цепочку
    hr = pFCh->RemoveChain(pF,pFEnd);
  }

  return hr;
}
```

Для поиска указателя на подключенный фильтр выше по потоку написан вспомогательный метод CCapture::GetUpFilter, который возвращает указатель на искомый фильтр во втором параметре. В первом параметре указывается фильтр, с которого начинается поиск.

```
HRESULT CCapture::GetUpFilter(IBaseFilter* pFEnd,
                              IBaseFilter** pFUp)
{
  HRESULT hr;
  CComPtr<IPin> pPIn;

  //Ищем входной  контакт на фильтре
  if(S_OK != (hr = m_pBuilder->FindPin(pFEnd,PINDIR_INPUT
                           ,NULL,NULL,FALSE,0,&pPIn)))
    return hr;

  //Ищем выходной подключенный контакт на фильтре выше по потоку
  CComPtr<IPin> pPConnect;
  PIN_INFO info;

  if(S_OK != (hr = pPIn->ConnectedTo(&pPConnect)))
    return hr;

  //Возвращаем указатель на фильтр выше по потоку
  if(S_OK != (hr = pPConnect->QueryPinInfo(&info)))
    return hr;
```

```
  *pFUp = info.pFilter;

  //Не освобождать (info.pFilter->Release();), если фильтр
  //используется, иначе, при удалении фильтра в DestroyGraph
  // получим исключение, без каких либо комментариев

  return hr;
}
```

Для работы с перечисленными методами требуется откорректировать методы CCapture::SetNameFile, CCapture::StopCapture(),CCapture::StartCapture().

Первым следует откорректировать метод StopCapture с тем, чтобы при остановке записи файла отключить фильтр кодера и возможный предшествующий ему трансформирующий фильтр. В случае отсутствия трансформирующего фильтра отключается только кодер. Часть метода, подвергшаяся изменениям, показана ниже.

```
BOOL CCapture::StopCapture()
{
  //Отключаем любые фильтры, подключенные к фильтрам
// сбора
  if(!DisconnectFilters(GetPinCapture(m_pCapV))
          || !DisconnectFilters(GetPinCapture(m_pCapA)))
    return FALSE;

  //Отключаем фильтры кодеров
  if(m_pCodecV)
  {
    if(!DisconnectFilters(NULL,m_pCodecV))
      return FALSE;
  }

  if(m_pCodecA)
  {
    if(!DisconnectFilters(NULL,m_pCodecA))
      return FALSE;
  }
……….

}
```

Метод SetNameFile должен быть откорректирован потому, что метод RenderStream не может правильно подключить цепочку фильтров кодера, находящуюся в графе. Следует либо удалять фильтр кодера полностью из графа при остановке записи, либо предусмотреть подключение цепочки фильтров ручным способом. В откорректированном методе используется второй способ. Часть метода, подвергшаяся изменениям, показана ниже.

```
……….
  else if(wext == L".avi")
  {
    CComPtr<IBaseFilter> pFUp;
    if(m_pCodecV && S_OK == GetUpFilter(m_pCodecV,&pFUp))
    {
      //Есть фильтр выше по потоку, подключенный к кодеру
      if(S_OK != ConnectFilters(pPinOutV,NULL,pFUp))
        return FALSE;

      if(S_OK != ConnectFilters(NULL,m_pCodecV,m_pFMux))
        return FALSE;
    }
    else
    {
      //Нет фильтра выше по потоку или нет кодера
      if((S_OK != m_pBuilder->RenderStream(
                  NULL,NULL,pPinOutV,m_pCodecV,m_pFMux)))
        return FALSE;
    }

    //Делаем попытку подключить звуковой кодер
    if((S_OK != m_pBuilder->RenderStream(NULL, NULL, pPinOutA
                         ,m_pCodecA, m_pFMux)) && m_pCodecA)
    {
      m_pGraph2->RemoveFilter(m_pCodecA);
      m_pCodecA = NULL;

      //Подключить звук без кодера
      hr= m_pBuilder->RenderStream(NULL,NULL,pPinOutA
                                 , m_pCodecA,m_pFMux);
    }

    //Устанавливаем имя файла и режим записи файла
    ……
```

Последний метод StartCapture, должен в результате корректировки подключать цепочку фильтров кодера, которая уже есть в графе. Часть метода, подвергшаяся изменениям, показана ниже.

```
  if(wext == L".avi")
  {

    CComPtr<IBaseFilter> pFUp;
    if(m_pCodecV && S_OK == GetUpFilter(m_pCodecV,&pFUp))
    {
      //Есть выше по потоку фильтр, подключенный к кодеру
      if(S_OK != ConnectFilters(pPinOutV,NULL,pFUp))
        return FALSE;
```

```
      if(S_OK != ConnectFilters(NULL,m_pCodecV,m_pFMux))
        return FALSE;

    }
    else
    {
      //Нет фильтра выше по потоку или нет кодера
      if((S_OK != m_pBuilder->RenderStream(NULL
                      ,NULL,pPinOutV,   m_pCodecV,m_pFMux)))
        return FALSE;
    }

    //Устанавливаем имя текущего файла
    ……….
```

Управление установкой и сменой видео и звуковых кодеров производится в обработчиках CFView2::OnCbnSelchangeCodecVideo и CFView2::OnCbn-SelchangeCodecAudio. В обработчиках происходит поиск моникера соответствующего кодера и вызов метода для установки его в граф. Обработчик для звука реализован по такому же принципу, как и для видео.

После проведенной корректировки проекта фильтры кодеров могут участвовать в процессе записи файлов AVI. В заключение следует отметить, что из всего множества фильтров кодеров, работать в графе аналогового телевидения могут только некоторые, например фильтры кодеров Microsoft Video 1 и MJPEG, которые всегда есть в системе. Эти кодеры работают без проблем и отладку лучше всего производить с ними. Фильтр Microsoft Video 1 устанавливается при необходимости с трансформирующим фильтром AVI Decompressor (для формата YUY2). Остальные фильтры нуждаются либо в экспериментальной проверке, либо необходимо иметь технические данные на каждый фильтр для определения возможности его работы в графе. Большое количество фильтров, имеющихся в системе, может существенно затруднить выбор пользователя, поэтому желательно производить предварительный отбор только необходимых для проектируемой программы видеокодеров. Звуковые компрессоры не нуждаются в таком отборе.

Отображение свойств компрессора

Каждый кодер для настройки предоставляет свою страницу свойств. Это могут быть страницы свойств на базе интерфейса ISpecifyPropertyPages, который предоставляют все настраиваемые фильтры. Для вызова этого диалога ранее был написан метод CCapture::DisplayPropertyPages. Кроме этого интерфейса существует интерфейс IAMVfwCompressDialogs, реализуемый компрессорами AVI. Многие компрессоры сторонних производителей также широко используют страницы свойств на основе интерфейса IAMVfwCompressDialogs. Диалог отображает страницы конфигурации компрессора и страницу "About". Интерфейс предоставляет следующие методы.

Метод	Описание
ShowDialog	Отображает диалог.
GetState	Возвращает текущие установки кодека.
SetState	Устанавливает текущие установки кодека.
SendDriverMessage	Посылает сообщение драйверу. Содержание сообщения определяется разработчиком кодека.

Для вызова диалога, отображающего страницы свойств IAMVfwCompressDialogs напишем отдельный метод.

```
BOOL CCapture::DisplayVfwDialogs(IBaseFilter* pFilter)
{
  CComQIPtr<IAMVfwCompressDialogs> pDlg(pFilter);
  if(!pDlg)
    return FALSE;

  if(FAILED(pDlg->ShowDialog(VfwCompressDialog_Config,NULL)))
    return FALSE;

  return TRUE;
}
```

Обычно кодеры предоставляют свои свойства для одного из указанных интерфейсов. Поэтому можно совместить в одном методе вызовы диалогов двух типов.

```
void CCapture::DisplayCodecDialogVideo(void)
{
  if(!m_pCodecV)
    return;
```

```
  if(!DisplayVfwDialogs(m_pCodecV))
    DisplayPropertyPages(m_pCodecV);
}
```

Кодеры звукового канала редко имеют страницы свойств, но некоторые кодеры сторонних производителей могут их иметь. Для отображения страницы свойств звукового компрессора напишем свой метод.

```
void CCapture::DisplayCodecDialogAudio(void)
{
  if(!m_pCodecA)
    return;

  if(!DisplayVfwDialogs(m_pCodecA))
      DisplayPropertyPages(m_pCodecA);
}
```

Для вызова диалогов на вкладке класса CFView2 установим кнопку под именем "Свойства кодека" и переключатель для выбора каналов видео и звука. В обработчике кнопки будут вызываться диалоги свойств кодеров в зависимости от состояния переключателя..

```
void CFView2::OnBnClickedButCodecProp()
{
  CCapture* pCap = myGetPtrCap();
  if(!pCap)
    return;
 if(!m_bCodecDlg)
    pCap->DisplayCodecDialogVideo();
 else
    pCap->DisplayCodecDialogAudio();
}
```

Кроме описанных интерфейсов на фильтрах кодеров может поддерживаться интерфейс IAMVideoCompression, который позволяет управлять параметрами компрессора. На некоторых WDM фильтрах захвата, которые имеют аппаратные компрессоры, этот интерфейс также может поддерживаться. Фильтры, поддерживающие данный интерфейс предоставляют его на выходном контакте. Фильтры не обязаны поддерживать все методы интерфейса.

Метод	Описание
put_KeyFrameRate	Устанавливает частоту ключевых кадров. Ключевой кадр используется как эталонный для построения последующих сжатых кадров изображения.
get_KeyFrameRate	Возвращает частоту ключевых кадров.

put_PFramesPerKeyFrame	Устанавливает частоту предсказанных кадров изображения, которые располагаются равномерно между ключевыми кадрами. Используется в компрессорах MPEG.
get_PFramesPerKeyFrame	Возвращает частоту предсказанных кадров изображения.
put_Quality	Устанавливает качество сжатия в диапазоне от 0.0 до 1.0.
get_Quality	Возвращает качество сжатия.
put_WindowSize	Устанавливает размер окна, выраженный в числе кадров изображения, для которого компрессор должен обеспечить среднюю скорость потока данных.
get_WindowSize	Возвращает размер окна, выраженный в числе кадров изображения, для которого компрессор должен обеспечить среднюю скорость потока данных.
GetInfo	Возвращает информацию о свойствах компрессора.
OverrideKeyFrame	Форсированная установка частоты ключевых кадров.
OverrideFrameSize	Форсированная установка максимального размера кадра.

Использование интерфейса IAMVideoCompression требуется в редких случаях, обычно достаточно установок по умолчанию. Если все же есть необходимость в его применении, то в первую очередь следует вызвать метод GetInfo, который возвращает информацию о поддержке компрессором свойств и методов интерфейса. Поддерживаемые свойства определяют методы интерфейса, которые можно вызывать для их установки. Описание возвращаемых флагов свойств компрессора приводится ниже.

```
typedef enum {
    CompressionCaps_CanQuality  =    0x01,
    CompressionCaps_CanCrunch   =    0x02,
    CompressionCaps_CanKeyFrame =    0x04,
    CompressionCaps_CanBFrame   =    0x08,
    CompressionCaps_CanWindow   =    0x10
} CompressionCaps;
```

Элементы перечисления.

CompressionCaps_CanQuality

Видео компрессор поддерживает методы put_Quality и get_Quality, управляющие качеством сжатия.

CompressionCaps_CanCrunch
Видео компрессор может производить сжатие на указанной скорости. Если компрессор имеет это свойство, то тогда можно устанавливать скорость потока данных в медиа типе на выходном контакте. Это производиться установкой члена dwBitRate в структуре VIDEOINFOHEADER. Других способов установки скорости передачи данных нет.

CompressionCaps_CanKeyFrame
Видео компрессор поддерживает методы put_KeyFrameRate и get_KeyFrameRate, управляющие скоростью поступления ключевых кадров

CompressionCaps_CanBFrame
Видео компрессор поддерживает методы put_PFramesPerKeyFrame и get_PFramesPerKeyFrame, управляющие скоростью поступления предсказанных кадров. Используется в компрессорах MPEG.

CompressionCaps_CanWindow
Видео компрессор поддерживает методы put_WindowSize и get_WindowSize, управляющие числом кадров, которые должен обработать компрессор при установленной скорости потока данных.

В качестве примера напишем метод возвращающий флаги свойств видео компрессора. При необходимости можно легко расширить метод для возврата других данных. Для возврата интерфейса IAMVideoCompression проще всего использовать метод FindInterface, поскольку фильтр захвата может не иметь компрессора и тогда его нужно искать на выходном контакте фильтра компрессора. Метод FindInterface производит поиск интерфейса в графе на соединенных фильтрах. Если компрессор не подключен, то поиск нужно производить отдельно на фильтре компрессора. Вызов метода GetInfoCompressor для тестирования можно произвести в конце метода AddCodecVForMoniker.

```
BOOL CCapture::GetInfoCompressor(LONG & capabilities)
{
  if(!m_pCodecV)
  return FALSE;

  //Находим интерфейс компрессора IAMVideoCompression,
  // если он подключен или есть на фильтре сбора
  CComPtr<IAMVideoCompression> pComp;
```

```
  m_pBuilder->FindInterface(&PIN_CATEGORY_CAPTURE,
            &MEDIATYPE_Video,m_pCapV,IID_IAMVideoCompression
            ,reinterpret_cast<void**>(&pComp));

    //Если интерфейс не найден ищем его на фильтре компрессора
  if(!pComp && m_pCodecV)
  {
    if(S_OK != m_pBuilder->FindInterface(NULL,&MEDIATYPE_Video
                   ,m_pCodecV, IID_IAMVideoCompression
                 , reinterpret_cast<void**>(&pComp)))
      return FALSE;
  }


  int   cbVersion = 0;    //Размер строки версии

  //Размер строки имени компрессора
  int   cbDescription = 0;

  //Частота ключевых кадров
  long  defaultKeyFrameRate = 0;
  long  defaultPFramesPerKey = 0; //Количество P кадров
  double  defaultQuality = 0;     //Качество


  //Вызываем первый раз для определения размеров строк
  if(S_OK != pComp->GetInfo(0,&cbVersion
                      ,0,&cbDescription,0,0,0,0))
    return FALSE;

  //Строка номера версии
  CComBSTR bstrVers(cbVersion);

  //Строка имени компрессора
  CComBSTR bstrDesc(cbDescription);

    pComp->GetInfo(bstrVers,&cbVersion,bstrDesc,&cbDescription
              , &defaultKeyFrameRate  , &defaultPFramesPerKey
              ,&defaultQuality,&capabilities);

  return TRUE;
}
```

Используя данный интерфейс можно отображать и управлять свойствами компрессора без вызова страниц свойств компрессора, в диалоге собственной разработки, но в таком случае, можно потерять дополнительные свойства компрессора, определенные разработчиком фильтра.

Использование кодеров сторонних производителей

Использование кодеров сторонних производителей может понадобиться, прежде всего, при записи файлов в формате MPEG2, который не поддерживается Microsoft. Остальные потребности может обеспечить формат ASF. Существует множество коммерческих фильтров сжатия, которые продаются как отдельный продукт. Некоторые фильтры и кодеры DirectShow поставляются в комплекте с устройством в составе программы, сопровождающей приобретенное устройство. Некоторые фильтры можно получить при покупке системной платы в составе бесплатно предоставляемой программы. Например, программа WinDVD фирмы InterVideo, поставляемая при покупке системной платы ASUS содержит все необходимые фильтры для записи файлов формата MPEG2. Также своей программой комплектуется телевизионный приемник, которая содержит фильтры и кодеры DirectShow. Фильтры DirectShow регистрируются в системе при установке программ и при наличии лицензии на исходную программу могут использоваться на компьютере для "внутреннего употребления", если лицензия прямо не запрещает использование отдельных частей программного обеспечения.

Для проверки возможности использования кодеров MPEG2 из комплекта программного обеспечения устройств воспользуемся программой WinDVD и программой, прилагаемой к телевизионному устройству.

Прежде всего, нужно определить, какие фильтры нужны для выполнения задачи. Здесь поможет редактор графа, в котором в списке фильтров InterVideo можно увидеть фильтры "InterVideo Multiplexer", "InterVideo Video Encoder", "InterVideo Audio Encoder", "InterVideo File Writer". С помощью редактора графа устанавливаем указанные фильтры и проводим проверку записи файла MPEG2. Фильтры соединяются так же, как и при записи файлов AVI с использованием компрессора. Воспроизведение файла на проигрывателе Windows Media показывает полную работоспособность данного графа. Теперь следует проверить работоспособность графа в реальной программе. Как показывает практика, здесь могут обнаружиться некоторые проблемы и "странное поведение" фильтров. Сначала, для сведения, определим, какие интерфейсы поддерживаются фильтрами. Список поддерживаемых интерфейсов для каждого фильтра можно найти, запустив на выполнение утилиту OleView.exe, меню Tools \ Ole/Com Object Viewer. В разделе All Objects утилиты нужно найти требуемый фильтр и просмотреть все его интерфейсы и их данные. На рисунке показан вид окна утилиты для просмотра списка интерфейсов.

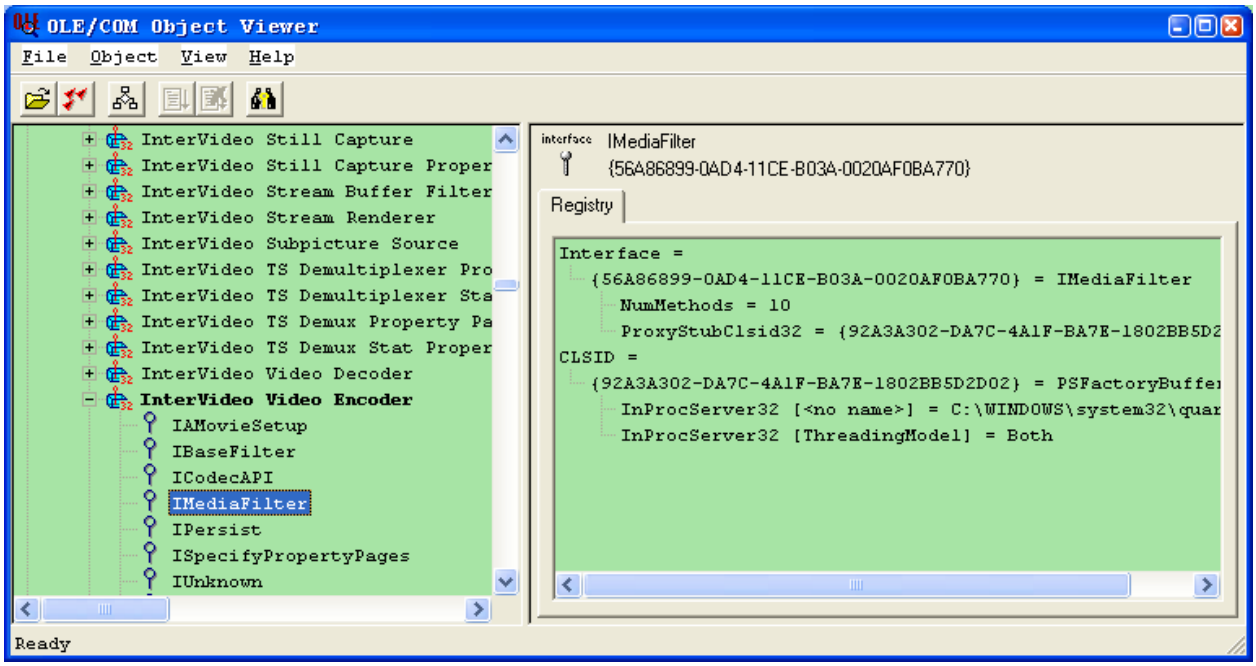

Рис.7. 3. Просмотр интерфейсов объектов.

Для проверки создадим отдельный проект из копии проекта "ProjectState8" и назовем его ProjectState_MPEG2". В данном проекте произведем модернизацию некоторых методов для записи только файлов формата MPEG2. Первый метод CCapture::InitWriteFile, который полностью перепишем. В методе произведем установку фильтров "InterVideo Multiplexer", "InterVideo Video Encoder", "InterVideo Audio Encoder", "InterVideo File Writer". Установку фильтров можно произвести с помощью системного перечислителя, распознавая фильтры по именам, а можно непосредственно установить фильтры в граф, используя для установки CLSID фильтра. Второй способ проще и позволяет установить фильтры под именами собственного назначения. После установки фильтры можно предварительно соединить. Фильтры кодеров подключить к фильтрам сбора и фильтр мультиплексора соединить с фильтром записи файлов.

Идентификатор фильтра (CLSID) можно получить, если он зарегистрирован в системе, например при установке программы, которая его использует. Для отдельной регистрации фильтров существуют различные менеджеры фильтров, которые свободно распространяются в Интернете, или можно использовать системную утилиту Regsrv32.

```
BOOL CCapture::InitWriteFile()
{
  HRESULT hr = S_OK;

  //Если граф запущен, то остановить
  if(m_bStartGraph && !StopGraph())
    return FALSE;
```

```
try
{
  m_nameCoderV =  L"IV CoderVideo";
  m_nameCoderA =  L"IV CoderAudio";
  m_nameFileWriter =  L"IV File Writer";
  m_nameMux = L"IV Mux";

  //Установка кодера видео
  LPOLESTR strGUID = L"{317DDB61-870E-11D3-9C32-00104B3801F6}";

  CLSID guid;
  if(S_OK == CLSIDFromString(strGUID,&guid))
  {
    hr = AddFilterByCLSID(&guid,m_nameCoderV,m_pCodecV);

    if(S_OK != hr)
      throw hr;
  }

  //установка мультиплексора
  strGUID = L"{317DDB63-870E-11D3-9C32-00104B3801F7}";
  if(S_OK == CLSIDFromString(strGUID,&guid))
  {
    hr = AddFilterByCLSID(&guid,m_nameMux,m_pFMux);
    if(S_OK != hr)
      throw hr;
  }

  //установка кодера звука
  strGUID = L"{0CD2E140-8D60-11D3-9C32-00104B3801F6}";
  if(S_OK == CLSIDFromString(strGUID,&guid))
  {
    hr = AddFilterByCLSID(&guid,m_nameCoderA,m_pCodecA);
    if(S_OK != hr)
      throw hr;
  }

  //установка фильтра записи
  strGUID = L"{D2288805-7D1E-49D4-9934-6D5B3728E155}";
  if(S_OK == CLSIDFromString(strGUID,&guid))
  {
    hr = AddFilterByCLSID(&guid,m_nameFileWriter,m_pFSink);
    if(S_OK != hr)
      throw hr;
  }

  //Соединяем мультиплексор и фильтр записи
  hr = ConnectFilters(NULL,m_pFMux,m_pFSink);
  if(S_OK != hr)
    throw hr;
```

```
    //Соединяем фильтр сбора и кодер
    hr = ConnectFilters(NULL,m_pCapV,m_pCodecV);
    if(S_OK != hr)
      throw hr;

    //Соединяем фильтры
    hr =ConnectFilters(NULL,m_pCapA,m_pCodecA);
    if(S_OK != hr)
      throw hr;

  }
  catch(HRESULT h)
  {
    string sErr = "Ошибка в InitWriteFile";
            CTVshowApp::myMessageError(h, sErr);
  }

  return TRUE;
}
```

Вызов метода следует производить после установки медиа формата, или производить повторное соединение кодера. Поскольку программа должна производить только запись файла MPEG2, то вызов метода сделаем в обработчике CFView1::OnMyMessageEndInit, что обеспечит решение поставленной задачи.

Далее произведем изменения в методе CCapture::SetNameFile для выполнения соединения кодеров видео и звука с мультиплексором и запуск графа для выполнения записи файла. В методе добавим участок кода, который будет производить соединение фильтров в записывающую секцию.

```
.................................................................................
else if(wext == L".mpg")
  {
    //Соединяем  кодер и мультиплексор
    if(S_OK != (hr=ConnectFilters(NULL,m_pCodecV,m_pFMux)))
      return FALSE;

    //Устанавливаем имя файла
    CComQIPtr<IFileSinkFilter> pSink(m_pFSink);
    if(pSink)
    {
      hr = pSink->SetFileName(nameFile.c_str(),NULL);
      if(hr != S_OK)
        return FALSE;

      m_wstrNameFile = nameFile;
    }
    else
```

```
      return FALSE;

    if(S_OK !=ConnectFilters(NULL,m_pCodecA,m_pFMux))
      return FALSE;

  }
  else
  {
    //Расширение имени не найдено, фильтры не подключены
    StartPreview();
      return FALSE;
  }

  //Запись подготовлена и будет производиться
  m_bCapture = TRUE;

  //Включить запись
  StartPreview();

  return TRUE;
}
```

После этого делаем попытку записать файл. Здесь нас поджидает неудача. Метод возвращает FALSE. Детальная локализация ошибки показывает, что при соединении кодера с мультиплексором возвращается код VFW_E_NOT_STOPPED, это означает, что фильтры не остановлены перед соединением. Это странно, поскольку граф останавливается при вызове метода SetNameFile. Для проверки состояния фильтров напишем метод CCapture::GetStateFilter, который возвращает его состояние. Для этого воспользуемся интерфейсом IMediaFilter, который поддерживают все фильтры DirectShow.

```
HRESULT CCapture::GetStateFilter(IBaseFilter* pF
                                        , FILTER_STATE& state)
{
  HRESULT hr = S_OK;
  CComQIPtr<IMediaFilter> pMF(pF);
  if(pMF)
    hr = pMF->GetState(30,&state);

  return hr;
}
```

Произведя последовательно вызовы этого метода для кодера, мультиплексора и фильтра записи перед соединением фильтров обнаруживаем, что мультиплексор находится в состоянии паузы. Таким образом, остановка графа всего лишь переводит этот мультиплексор в паузу, в отличие от всех остальных фильтров в графе, которые останавливаются. Попробуем остано-

вить этот фильтр с использованием методов интерфейса IMediaFilter. Для этого напишем метод CCapture::StopFilter.

```
HRESULT CCapture::StopFilter(IBaseFilter* pF)
{
  HRESULT hr = S_OK;
  CComQIPtr<IMediaFilter> pMF(pF);
  if(pMF)
    hr = pMF->Stop();

  return hr;
}
```

Перед соединением фильтра произведем вызов этого метода.

```
    hr = GetStateFilter(m_pFMux,st);
    if(hr == S_OK && st != State_Stopped)
      hr = StopFilter(m_pFMux);
```

Дополнительно модернизируем метод CCapture::StopCapture() .

```
BOOL CCapture::StopCapture()
{
  //Отключаем фильтр мультиплексора
  FILTER_STATE st;
  HRESULT hr = GetStateFilter(m_pFMux,st);
  if(hr == S_OK && st != State_Stopped)
    hr = StopFilter(m_pFMux);

  //Разрываем соединение кодера
  if(m_pCodecV)
  {
    if(!DisconnectFilters(NULL,m_pCodecV))
      return FALSE;
  }

  //Разрываем соединение кодера
  if(m_pCodecA)
  {
    if(!DisconnectFilters(NULL,m_pCodecA))
      return FALSE;
  }

  if(m_pFSinkASF && !IsFilterCache(m_pFSinkASF))
  {
    //Если записывался файл ASF возвращаем его в кеш
    if(S_OK != PushFilterToCache(m_pFSinkASF))
      return FALSE;
  }
  else if(m_pFSink)//Файл AVI или другой
```

```
  {
    //Запрос интерфейса
    CComQIPtr<IFileSinkFilter> pSink(m_pFSink);
    if(!pSink)
      return FALSE;

    //Установка временного имени файла для
    //освобождения записанного
    wstring sext = GetExtFileName(m_wstrNameFile);
    if(sext == L".avi")
    {
      if(S_OK != pSink->SetFileName(L"C:\\Temp.avi",NULL))
        return FALSE;
    }

    //Файл MPEG2
    if(sext == L".mpg")
    {
      if(S_OK != pSink->SetFileName(L"C:\\Temp.mpg",NULL))
        return FALSE;
    }
  }

  //Пуск графа
  if(!StartPreview())
    return FALSE;

  m_bCapture = FALSE;

  return TRUE;

}
```

После вызова модернизированного метода фильтр останавливается, и соединение происходит правильно, файл записывается, размер файла правильный, но при попытке его воспроизвести он не воспроизводится проигрывателем Windows Media. Если после этого файла записать следующий файл, не перезапуская программу, то второй файл будет воспроизводиться полностью, но проигрыватель после остановке воспроизведения сообщит об ошибке. Эта "странность" потребовала написания отдельной программы воспроизведения файлов DXRender, чтобы исключить проигрыватель Windows Media. Тема воспроизведения файлов здесь не рассматривается, программа достаточно простая и имеет ограниченное функционирование, с ней можно ознакомиться в архиве проектов. Она строит граф, который показан на рисунке 7.4. Проверка воспроизведения файла с помощью данной программы показала такой же результат, как и проигрыватель Windows Media. Поиск причин подобных "странностей" может быть очень длительным и непростым. В данном случае удалось достаточно быстро найти решение.

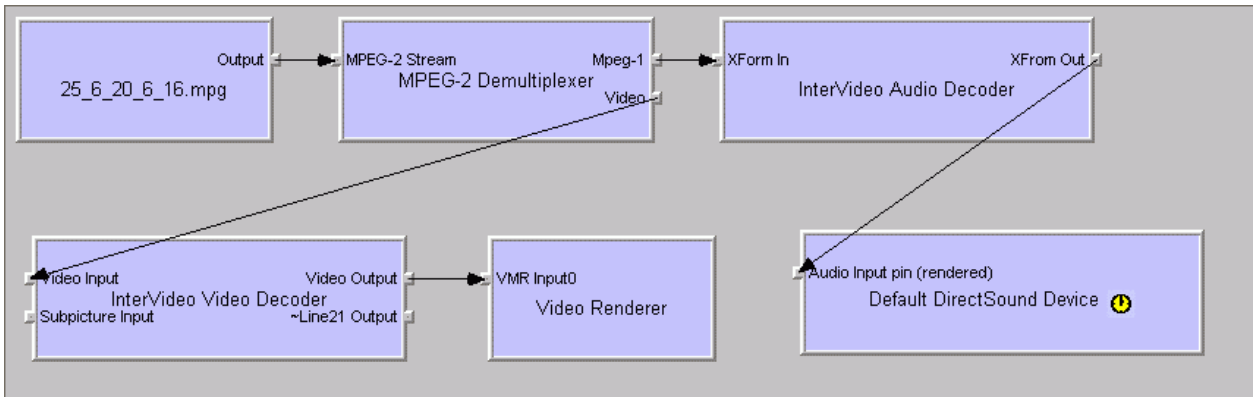

Рис.7. 4. Граф воспроизведения файла MPEG2.

Первый файл записывается правильно, если дополнительно остановить фильтр мультиплексора перед стартом записи, несмотря на то, что он находится в остановленном состоянии. Во всяком случае, при возврате состояния фильтра, получаем результат " State_Stopped", но в данном случае доверять результату нельзя.

Сюрпризы на этом не заканчиваются, файл воспроизводится с ускорением в два раза на проигрывателе Windows Media и программой DXRender также. Только "родная" программа WinDVD воспроизводит файл правильно, без ускорения. Таким образом, становится ясно, что граф построенный средствами DirectShow в автоматическом режиме не может правильно воспроизвести файл.

Для проверки воспользуемся редактором графа и проверим граф, который показан на рисунке 7.4, с другими демультиплексорами, не Microsoft, а InterVideo. Их два, InterVideo Demultiplexer и InterVideo Demux. Если использовать второй демультиплексор, то файл будет воспроизводиться нормально. Для правильного воспроизведения файла программой DXRender ее следует незначительно доработать. Для этого в процессе инициализации в граф сразу устанавливается фильтр InterVideo Demux, что заставляет метод Render использовать только этот демультиплексор.

Таким образом, использование кодера InterVideo вполне возможно, но только воспроизводить файл следует собственным проигрывателем. Воспроизведение файла проигрывателем Windows Media будет производиться в ускоренном темпе. Скорее всего, это способ предотвратить использование кодера вне собственной программы. Остальные "странности" больше похожи на недоделки, чем на защиту фильтра от несанкционированного использования. Возможно, что в другой версии фильтра этих "странностей" не будет, но могут появиться другие, не менее интересные.

Второй кодер, из состава программы сопровождающей устройство телевизионного приемника, производится компанией Honest Technology под именем honestech htvcdsvcdcvt2.ax. Для регистрации фильтра требуется файл фильтра htmpeg2enc.ax и библиотека mpeg2enc.dll. Кодер представляет собой многофункциональный конвертирующий фильтр, который включает в себя мультиплексор, кодеры MPEG1/2 и другие (типа VCDSVCD), а также фильтр запи-

си файлов. При записи файлов он не требует дополнительных фильтров, и записывает файл, который нормально воспроизводится проигрывателем Windows Media. Из проблем можно отметить только то, что графический редактор не может подключиться к графу. Тестовый проект для этого кодера находится в папке ProjectState_MPEG2_HT. Кодер имеет излишне большую и неряшливо сделанную станицу свойств. На рисунке 7.5 показана его страница свойств со списком кодеров.

Поскольку по умолчанию всегда установлен кодер MPEG1, то для правильной записи файла MPEG2 установку фильтра желательно производить при первом вызове страницы свойств в методе CCapture::DisplayCodecDialogVideo. Тогда запись будет возможна только после первого вызова страницы диалога, что автоматически напомнит пользователю о необходимости выбора кодера перед записью файла.

```
void CCapture::DisplayCodecDialogVideo()
{
  //Первичная инициализация
  if(!m_pCodecV)
    InitWriteFile();

  if(!DisplayVfwDialogs(m_pCodecV))
    DisplayPropertyPages(m_pCodecV);
}
```

При использовании этого кодера существенно упрощаются методы InitWriteFile и SetNameFile.

```
BOOL CCapture::InitWriteFile()
{
  HRESULT hr = S_OK;

  //Если граф запущен, то остановить
  if(m_bStartGraph && !StopGraph())
    return FALSE;

  try
  {
    m_nameFileWriter =  L"honestech Mpeg1/2 Recorder Filter";

    //Установка фильтра записи
    LPOLESTR strGUID = L"{5083B9E8-713F-40AA-B5F3-2AD35D3D7498}";

    CLSID guid;

    if(S_OK == CLSIDFromString(strGUID,&guid))
    {
      hr = AddFilterByCLSID(&guid,m_nameFileWriter,m_pCodecV);
      if(S_OK != hr)
        throw hr;
```

```
    }
  }
  catch(HRESULT h)
  {
    string sErr = "Ошибка в InitWriteFile";
    CTVshowApp::myMessageError(h, sErr);
  }
  return TRUE;
}
```

В методе SetNameFile участок кода для данного кодера становится значительно компактней.

```
.................................................................................
  else if(wext == L".mpg")
  {
    //Соединяем фильтр сбора и фильтр MPEG HT
    if(S_OK !=  (hr=ConnectFilters(GetPinCapture(m_pCapV),
                                        NULL,m_pCodecV)))
      return FALSE;

    //Устанавливаем имя файла
    CComQIPtr<IFileSinkFilter> pSink(m_pCodecV);
    if(pSink)
    {
      hr = pSink->SetFileName(nameFile.c_str(),NULL);
      if(hr != S_OK)
        return FALSE;

      m_wstrNameFile = nameFile;
    }
    else
      return FALSE;

    if(S_OK !=
        ConnectFilters(GetPinCapture(m_pCapA),NULL,m_pCodecV))

      return FALSE;

  }
..................................................................................
```

В данном варианте кода можно использовать для соединения метод Render вместо прямого соединения.

```
    hr = m_pGraph2->Render(GetPinCapture(m_pCapV));
    hr = m_pGraph2->Render(GetPinCapture(m_pCapA));
```

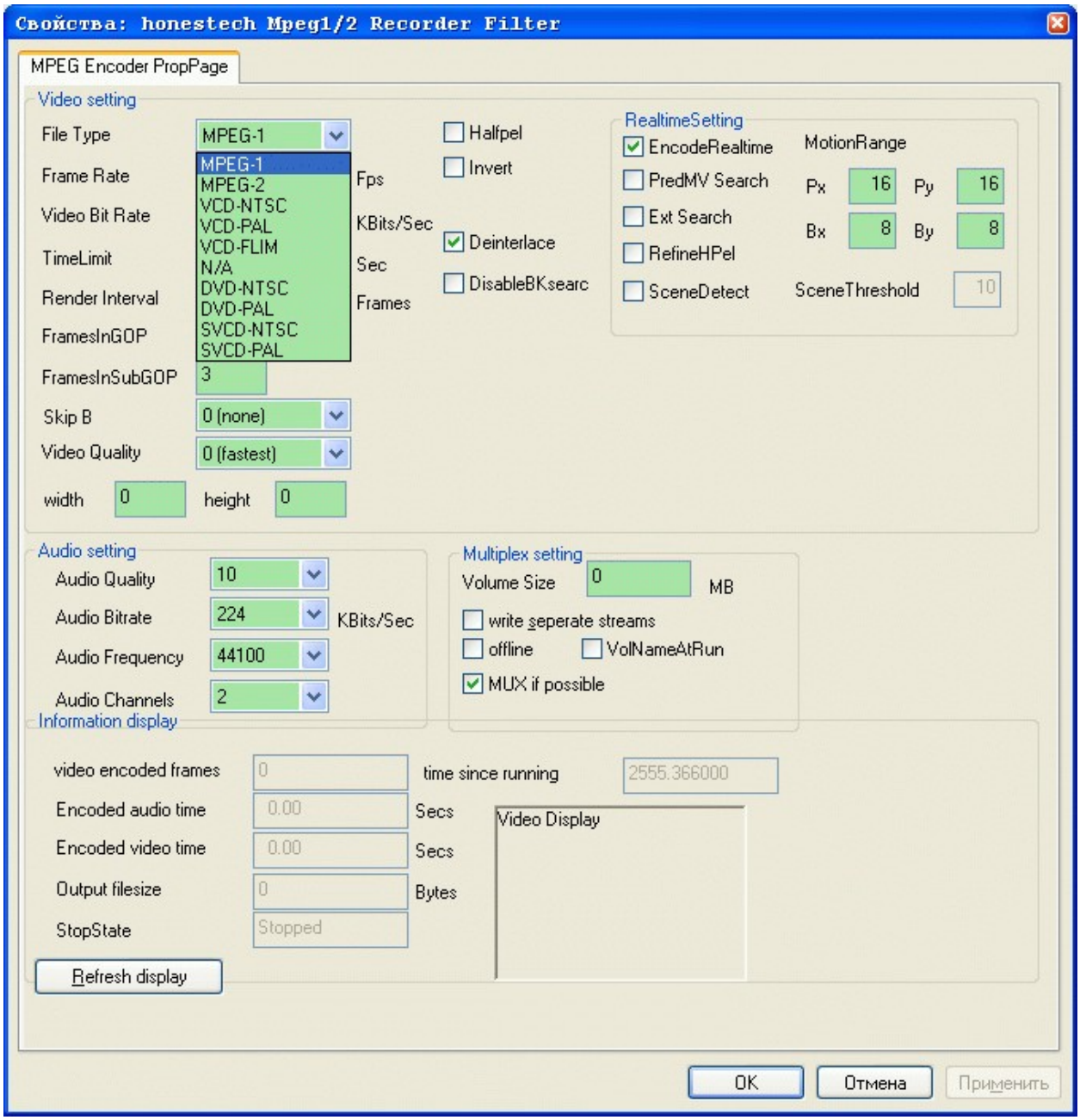

Рис.7. 5. Страница свойств кодера honestech.

Воспроизведение файлов записанных с помощью этого кодера можно произвести в двух вариантах графа:

- С использованием фильтров InterVideo. В этом случае для воспроизведения необходимо три фильтра InterVideo. На рисунке 7.6 показан граф для этого случая.
- Без использования фильтров InterVideo. Такой вариант потребует дополнительного использования еще двух фильтров HT, фильтра видеодекодера и звукового фильтра перекодировки потока MPEG2 в поток MPEG1. На рисунке 7.7 показан граф для этого случая.

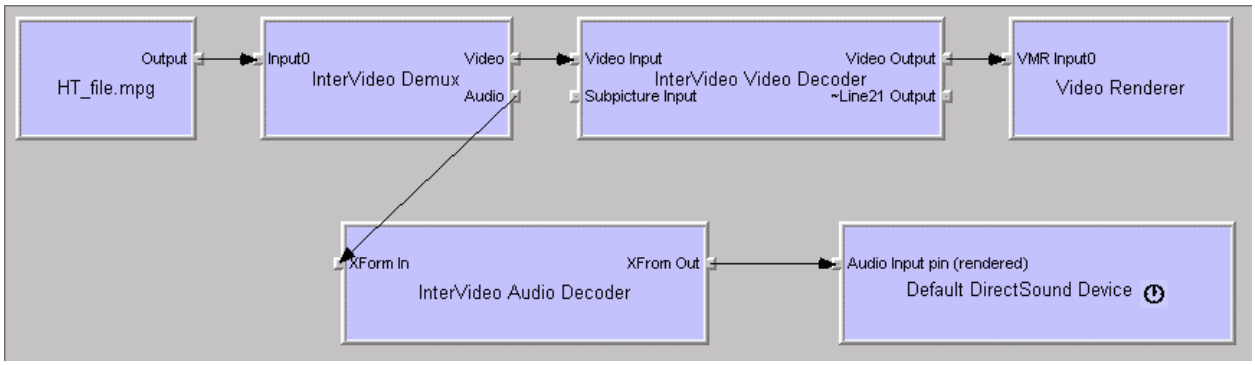

Рис.7. 6. Граф отображения файлов MPEG2. Вариант 1.

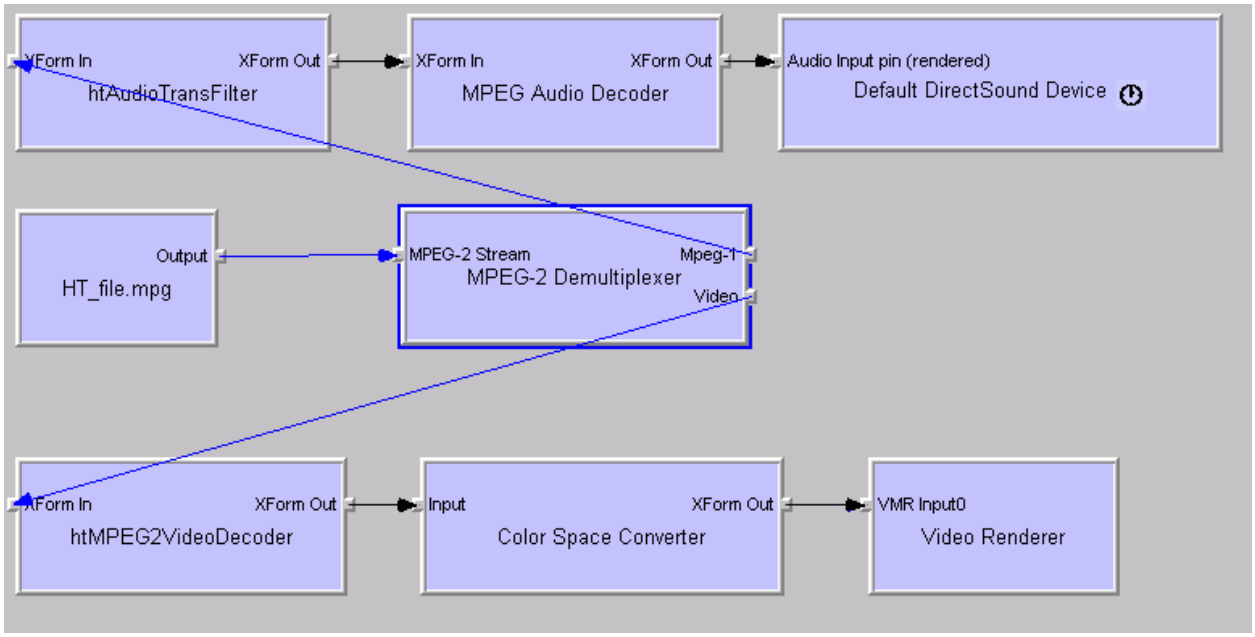

Рис.7. 7.Граф отображения файлов MPEG2. Вариант 2.

Возможен и третий вариант, но он потребует создания приложения с использованием SDK Windows Media, либо просто использовать проигрыватель Windows Media, если такой вариант устраивает разработчика.

Отображение производительности фильтра захвата

Для определения производительности фильтра захвата DirectShow предлагает интерфейс IAMDroppedFrames, позволяющий оценить производительность фильтра в масштабе реального времени. Интерфейс предоставляется на выходном контакте фильтра захвата после установки записывающей секции графа и ее подключения к фильтру захвата.

Методы интерфейса показаны в таблице.

Метод	Описание
GetAverageFrameSize	Возвращает средний размер кадра изображения, обрабатываемого фильтром.
GetDroppedInfo	Возвращает массив номеров кадров изображения, которые были удалены из обработки.
GetNumDropped	Возвращает общее количество кадров изображения, которые фильтр удалил из обработки с момента старта.
GetNumNotDropped	Возвращает общее количество кадров изображения, которые фильтр обработал и выдал на выход с момента старта

Таблица 9. Интерфейс IAMDroppedFrames.

Для оценки производительности фильтра напишем метод CCapture::GetDroppedFrames, возвращающий число удаленных из обработки кадров и общее число обработанных кадров. При достаточной производительности компьютера число удаленных кадров будет равно нулю.

```
void CCapture::GetDroppedFrames(long & drop, long & notDrop)
{
 CComPtr<IPin> pPin = GetPinCapture(m_pCapV);
  CComQIPtr<IAMDroppedFrames> pDrop(pPin);
  if(!pDrop)
    return;

  if(S_OK != pDrop->GetNumDropped(&drop))
    return;
 if(S_OK != pDrop->GetNumNotDropped(&notDrop))
    return;
}
```

Вызов этого метода удобней всего производить в обработчике системного таймера, поскольку высокой временной точности не требуется. Можно использовать и метод AdvisePeriodic интерфейса IReferenceClock, создающий периодический запрос временного интервала после старта графа, но этот путь сложней в реализации и, кроме того, для отображения производительности фильтра привязка к ссылочному времени не нужна. Для отображения возвращенных данных на вкладке класса CFView2 создадим два элемента CEdit со стилем только для чтения. В обработчике таймера будем производить вызов метода CCapture::GetDroppedFrames и отображение возвращен-

ных данных. Периодичность запроса не должна быть высокой, вполне достаточно одной секунды.

```
void CFView2::OnTimer(UINT nIDEvent)
{
  long drop=0,notDrop=0;
  myGetPtrCap()->GetDroppedFrames(drop,notDrop);
  CString str;
  if(drop)
  {
    str.Format("%.8d",drop);
    m_dropCtrl.SetSel(0,-1);
    m_dropCtrl.ReplaceSel(str);
  }
 else
  {
    m_dropCtrl.SetSel(0,-1);
    m_dropCtrl.ReplaceSel("0");
  }

  str.Format("%.8d",notDrop);
  m_notDropCtrl.SetSel(0,-1);
  m_notDropCtrl.ReplaceSel(str);

 CFormView::OnTimer(nIDEvent);
}
```

Установка таймера производится только при старте записи файла в методе CFView2::OnBnClickedButWritefile,

```
……..................................
    if(!m_timer)
    {
      m_dropCtrl.SetSel(0,-1);
      m_dropCtrl.ReplaceSel("0");
      m_notDropCtrl.SetSel(0,-1);
      m_notDropCtrl.ReplaceSel("0");

      m_timer = SetTimer(1, 1000, 0);
    }
……................................
```

Удаление таймера производится при остановке записи в методе CFView2::OnBnClickedButStop.

На рисунке 7.8 показано отображение производительности.

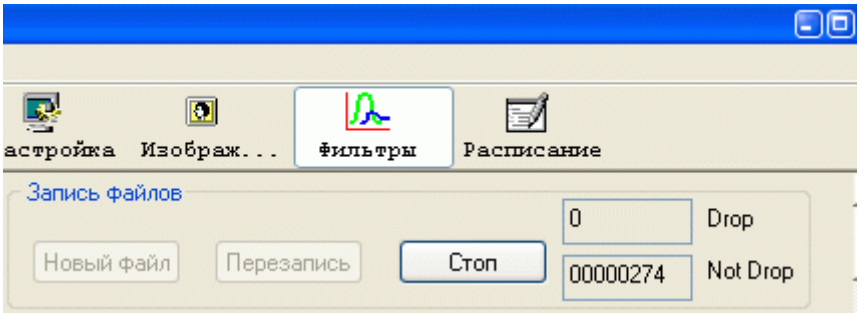

Рис. 7.8. Отображение производительности.

Отображение производительности кроме предоставления полезной информации, также является одновременным индикатором процесса записи. Это более эффективно, чем отображение полосы индикатора процесса.