Глава 4
Режим предварительного просмотра

Глава 4
Режим предварительного просмотра....1
Основные интерфейсы, используемые для реализации режима предварительного просмотра....3
Метод RenderStream....4
Отображающие фильтры DirectShow....8
Компоненты фильтров VMR....9
Режимы работы фильтров....10
Режимы работы фильтров VMR....11
Интерфейсы отображающих и оверлейных фильтров....13
Интерфейсы, управляющие графом....14
Инициализация режима предварительного просмотра....15
Достройка графа и запуск режима предварительного просмотра....17
Интерфейс IVideoWindow....21
Установка видеоокна в качестве дочернего окна приложения....24
Сообщения от окна фильтра Video Renderer....27
Управление параметрами видеопотока на фильтре сбора....27
Структура AM_MEDIA_TYPE....29
Источник и получатель изображения и их взаимодействие....31
Структура VIDEOINFOHEADER....34
Структура BITMAPINFOHEADER....35
Методы и структуры интерфейса IAMStreamConfig....37
Структура VIDEO_STREAM_CONFIG_CAPS....38
Параметры структуры, влияющие на формирование размеров изображения....41
Метод IAMStreamConfig::GetFormat и его применение....44
Метод IAMStreamConfig::GetNumberOfCapabilities....46
Метод IAMStreamConfig::GetStreamCaps и его применение....47
Метод IAMStreamConfig::SetFormat....48
Класс данных медиа формата....49
Определение свойств форматов на выходном контакте фильтра сбора....51
Метод для установки медиа формата....53
Интерфейс пользователя для отображения медиа форматов....59

Полнооконный режим отображения....62
Безоконный режим отображения фильтров VMR....65
Обработка событий в DirectShow....72
Обработка данных на контакте VBI фильтра сбора....76
Метод ICaptureGraphBuilder2::FindPin....81
Управление отображением телетекста....84
Регулировки изображения....87
Интерфейс IAMVideoProcAmp....88
Реализация регулировки яркости и контраста....92
Регулировки изображения на фильтре VMR9....97

Основные интерфейсы, используемые для реализации режима предварительного просмотра.

Прежде чем перейти непосредственно к построению ветви графа для реализации предварительного просмотра, оценим предстоящую задачу. Итак, мы имеем построенный граф до фильтра сбора включительно. Далее следует реализовать достройку графа от контакта предварительного просмотра фильтра сбора до фильтра отображения (Renderer). Определить наличие контакта VBI на фильтре сбора и построить ветвь графа для просмотра телетекста. Также необходимо оценить все возможные регулировки и настойки, которые предоставляются фильтром сбора и фильтром отображения, определить, какие из них следует предоставлять для управления пользователю, а какие можно оставить по умолчанию.

Достройка графа для предварительного просмотра имеет целый ряд особенностей, которые делают этот процесс многовариантным. Выбор способа реализации ветвей графа, определяется возможностями устройства, условиями задачи, текущим стандартом ТВ вещания. Для решения простой задачи отображения в минимальном варианте можно ограничиться применением методов "интеллектуального" построения ветвей графа. В остальных случаях необходимо комбинировать ручное построение графа с "интеллектуальным".

Начнем рассмотрение реализации предварительного просмотра с основных интерфейсов и их методов, необходимых для достройки графа.

Для построения отображающей ветви графа используются те же интерфейсы менеджера графа и построителя графа сбора, которые рассматривались ранее.

- Интерфейс построителя графа сбора ICaptureGraphBuilder2.
- Интерфейс менеджера графа IFilterGraph,.
- Интерфейс IFilterGraph2, расширяющий возможности IFilterGraph.
- Интерфейс построителя графа IGraphBuilder, производный от IFilterGraph.

Описание методов перечисленных интерфейсов приводится в главе 2. Здесь, прежде всего, рассмотрим методы этих интерфейсов, которые используют алгоритм "интеллектуального" соединения, позволяющий строить целые участки графа. Для реализации предварительного просмотра вполне возможно обойтись без трудоемкой работы по ручному построению графа. Или же максимально упростить задачу, используя эти методы. Из всех методов, указанных интерфейсов, для построения ветви графа предварительного просмотра можно использовать два метода, которые существенно влияют на реа-

лизацию собственных методов приложения. Конкретный выбор методов зависит от поставленной задачи и от особенностей поведения этих методов. Рассмотрим эти методы.

Метод IGraphBuilder::Render и IFilterGraph2::RenderEx подробно описан в главе 3. Он не имеет никаких параметров, которые могли бы управлять его поведением. Если используется этот метод, то Вы лишены любой возможности повлиять на процесс построения графа. Все решает алгоритм "интеллектуального" соединения.

Второй метод ICaptureGraphBuilder2::RenderStream имеет большие и разнообразные возможности по построению ветвей графа. Данный метод имеет несколько параметров, которые настраивают поведение метода при построении ветви графа. Он имеет возможности настройки своего поведения для специфических ветвей графа, таких как ветви, начинающиеся с выходных контактов фильтра сбора.

Метод RenderStream

Метод RenderStream производит построение участка графа от исходного фильтра до фильтра-приемника включительно. В типовом случае в качестве фильтра-приемника может быть отображающий или смешивающий фильтр. В процессе соединения метод может устанавливать дополнительные фильтры необходимые для отображения потока, а также промежуточный фильтр, указанный в отдельном параметре, например компрессор. Фильтр-источник (pSource) и промежуточный фильтр (pIntermediate*)* должны быть установлены в граф с помощью метода AddFilter до вызова метода RenderStream. Исключение сделано для фильтра отображения потока (pSink), который может устанавливаться по умолчанию. По умолчанию метод устанавливает в качестве фильтра-приемника отображающий фильтр Video Renderer, который может использоваться для всех платформ и в настоящее время считается устаревшим. Для канала звука устанавливается по умолчанию фильтр DirectSound Renderer. Если параметр pSink указывает на фильтр, то он также должен быть предварительно установлен в граф с помощью метода AddFilter. Метод RenderStream обрабатывает многие дополнительные детали построения графа сбора.

Некоторые фильтры сбора не имеют контакта предварительного просмотра, а только контакт сбора (capture). В этом случае метод производит установку фильтра Smart Tee. Фильтр разделяет поток на две части, каждая часть общего потока обеспечивает характеристики и поведение в соответствии с имитируемым контактом. При установке Smart Tee метод возвращает значение VFW_S_NOPREVIEWPIN. В данном случае при решении стандартных задач можно не обращать внимания на отсутствие отдельного контакта предварительного просмотра, метод обрабатывает эту ситуацию автоматически, скрывая все детали. Следует отметить, что метод не делает различия между звуковым фильтром сбора и фильтром сбора видео. Он также устанав-

ливает фильтр Smart Tee и для звукового фильтра сбора, что часто является излишним. Учитывая, что фильтр звука обычно имеет только контакт Capture, применять метод для построения ветви записи звука не следует.

Также некоторые фильтры сбора могут обрабатывать данные, передаваемые на обратном ходе кадровой развертки (VBI). Данные, передаваемые на обратном ходе кадровой развертки, могут быть двух стандартов, американского и европейского. Для обработки данных американского стандарта метод должен вызываться дважды, один раз для контакта категории PIN_CATEGORY_VBI и второй раз для контакта категории PIN_CATEGORY_CC. Метод устанавливает в граф любые фильтры необходимые для использования данных VBI. Для европейского стандарта передачи телетекста контакт VBI на фильтре сбора не обрабатывается автоматически, метод не устанавливает необходимые фильтры, их нужно установить в граф вручную.

Существуют фильтры сбора, имеющие контакт видеопорта категории PIN_CATEGORY_VIDEOPORT вместо контакта предварительного просмотра. Для таких случаев метод устанавливает смешивающий оверлейный фильтр (Overlay Mixer), который обеспечивает как предварительный просмотр, так и запись файла. Категория контакта в этом случае указывается PIN_CATEGORY_PREVIEW или PIN_CATEGORY_CAPTURE, но не PIN_CATEGORY_VIDEOPORT.

Таким же образом некоторые фильтры формируют данные VBI, используя контакт видеопорта (PIN_CATEGORY_VIDEOPORT_VBI). В этом случае следует указывать категорию PIN_CATEGORY_VIDEOPORT, детали порт обрабатывает самостоятельно.

```
HRESULT   RenderStream(
  const GUID      *pCategory,
  const GUID      *pType,
  IUnknown        *pSource,
  IBaseFilter     *pIntermediate,
  IBaseFilter     *pSink
);
```

Параметры

pCategory
[in]
Указатель на категорию контакта, являющийся членом структуры AMPROPERTY_PIN_CATEGORY или NULL при использовании контакта любого типа. Типовые значения величины следующие.
PIN_CATEGORY_CAPTURE
PIN_CATEGORY_PREVIEW
PIN_CATEGORY_CC
Все значения величин приводятся в приложении в разделе "Категории контактов" (Pin Property Set).

pType

[in]
Указатель на GUID главного медиа формата выходного контакта или NULL при использовании контакта любого медиа формата. Список значений приводится в приложении Главные типы (Major Types).

pSource
[in]
Указатель на стартовый фильтр, с которого начинается соединение или указатель на выходной контакт стартового фильтра. Если параметр pSource является указателем на фильтр, то производится поиск контакта на данном фильтре с использованием параметров pType и pCategory для ограничения поиска. Если параметр pSource является указателем на контакт, то параметры pType и pCategory не используются и должны быть установлены в NULL. Поиск производится для не соединенных контактов. Если не соединенных контактов, соответствующих критерию поиска несколько, то подключается первый найденный.

pIntermediate
[in]
Указатель на промежуточный фильтр, например компрессор. Может быть NULL.

pSink
[in]
Указатель на конечный фильтр. Для большинства применений параметр может указывать на отображающий или смешивающий фильтр. Если pSink равен NULL, тогда метод пытается установить в граф отображающий фильтр Video Renderer, обеспечивающий воспроизведение видеоданных для любых условий. Для звука устанавливается фильтр DirectSound Renderer. Описание фильтра Video Renderer приводится в приложении.

Возвращаемые значения

Возвращаемый код	Описание
S_OK	Успешное завершение.
VFW_S_NOPREVIEWPIN	Предварительный просмотр осуществляется через фильтр Smart Tee.
E_FAIL	Ошибка.
E_INVALIDARG	Неправильные аргументы.
E_POINTER	Значение указателя NULL.
VFW_E_NOT_IN_GRAPH	Фильтра нет в графе. Эта ошибка может произойти, если не был вызван метод AddFilter или метод SetFiltergraph для подключения графа к объекту построителя графа сбора (Capture Graph Builder). В этом случае объект построителя графа сбора автоматически создает

	собственный граф.

Метод RenderStream соединяет два или три фильтра в цепочку. Фильтр лучше всего работает при условии, что каждый фильтр имеет не более одного входного или выходного контакта определенного типа. Рассмотрим поведение фильтра при соединении трех фильтров A, B и C, где символы A, B и C означают указатели на фильтры (IBaseFilter). Первые два параметра пока не участвуют и равны нулю (NULL) и фильтры имеют один входной и один выходной контакт.

Допустим, что производится вызов метода в виде RenderStream(NULL, NULL, A, B, C) . Метод соединит все три фильтра последовательно.

Если указатель на фильтр B установить равным NULL RenderStream (NULL, NULL, A, NULL, C), то используя алгоритм "интеллектуального" соединения метод добавит в граф фильтр B и соединит их в цепочку. Длинную цепочку фильтров можно создать, вызвав метод два раза.

RenderStream(NULL, NULL, A, B, C)
RenderStream(NULL, NULL, C, D, E)

При установке последнего параметра в NULL, метод автоматически установит отображающий фильтр по умолчанию. Для видеоканала это будет фильтр Video Renderer, для звукового канала фильтр DirectSound Renderer. Для подключения отображающего фильтра Video Mixing Renderer (VMR), он должен быть указан явно.

Если в третьем параметре передается указатель на фильтр, и он имеет не один контакт, а несколько с разными медиа типами, то тогда следует использовать первые два параметра, которые помогают локализовать выходной контакт на фильтре. Первый параметр используется только для фильтров сбора. Он указывает выходного категорию контакта (GUID). Список категорий контактов приводится в справочном разделе. Для любых фильтров сбора две категории контактом всегда справедливы:
PIN_CATEGORY_CAPTURE
PIN_CATEGORY_PREVIEW

Второй параметр идентифицирует медиа тип и для типовых случаев может иметь один из следующих типов:

- MEDIATYPE_Audio
- MEDIATYPE_Video
- MEDIATYPE_Interleaved (DV)

Этот параметр можно использовать, если выходные контакты фильтра поддерживают перечисление предопределенных медиа типов.

Отображающие фильтры DirectShow

Как уже упоминалось, DirectShow предоставляет два типа отображающих фильтров:

1. Фильтр Video Renderer. Он доступен на всех платформах, которые поддерживают DirectX и не имеет системных ограничений. Данный фильтр использует DirectDraw там, где это допустимо, в остальных случаях применяется GDI. Он используется по умолчанию на всех платформах более ранних, чем Windows XP. При вызове метода RenderStream для построения графа, по умолчанию всегда будет устанавливаться этот фильтр.

2. Фильтр Video Mixing Renderer 7 (VMR-7) и фильтр Video Mixing Renderer 9 (VMR-9).
   Фильтр VMR-7 доступен на платформе Windows XP, где он используется по умолчанию, но только при использовании метода другого, а не RenderStream, например метода Render или RenderEx. Фильтр VMR-7 всегда использует DirectDraw 7 и все его возможности для отображения.
   Фильтр Video Mixing Renderer 9 (VMR-9) является новой версией фильтров VMR и использует возможности отображения, предоставляемые Direct3D9. Он доступен на всех платформах, которые поддерживают DirectX, но не является фильтром по умолчанию, поскольку имеет более высокие требования к системе, чем Video Renderer. Перед использованием фильтр должен быть создан и установлен в граф. Фильтр VMR-9 не поддерживает видеопорт. По возможности следует предпочесть этот фильтр другим, поскольку он использует новое графическое API.

Фильтры VMR решают следующие задачи:

- Обрабатывают все временные соотношения для представляемых кадров видеоизображения.
- Обеспечивают приложение и остальную часть системы отображения информацией о качестве обработке данных.
- Предоставляют интерфейсы компонентам выше по потоку, которым не известно, что приложение выполняет распределение кадрового буфера и отображение.
- Выполняют смешивание видеопотоков перед отображением.

Компоненты фильтров VMR

Фильтры VMR имеют модульную структуру, что позволяет приложению конфигурировать их для различных вариантов применения. В зависимости от конфигурации фильтр может иметь от двух до пяти компонентов. Все компоненты являются объектами COM. Структурная схема фильтра представлена на рисунке 4.1. Краткое описание компонентов фильтра приводится ниже.

Смеситель (Mixer).
Компонент загружается фильтром, когда на входы поступает более одного потока данных. Смеситель сортирует входные потоки по порядку (Z-порядок). Он определяет время поступления медиа выборки по каждому входу и указывает порядок смешивания для сборщика изображений. Смеситель также вычисляет временные штампы для каждого выходного изображения. Если приложение предоставляет растровое изображение для отображения на верхнем уровне композитного изображения, смеситель отвечает за модификацию Z-порядка входных потоков.

Сборщик изображений (Image Compositor).
Сборщик изображений создает из входных потоков композитное изображение на поверхности DirectDraw или Direct3D. Поверхность предоставляется распределителем. Фильтр по умолчанию предоставляет сборщик изображений для выполнения эффектов смешивания 2-D альфа. Приложение может настроить сборщик для выполнения различных других 2-D и 3-D эффектов.

Распределитель-передатчик изображения (Allocator-Presenter)
Распределитель производит распределение памяти для объекта DirectDraw или Direct3D и передает изображение в видеокарту. Рисование может выполняться переброской данных (flip) или побитовым копированием памяти (blit). Можно установить собственный распределитель-передатчик при создании и управлении объектом DirectDraw (Direct3D) и/или получить доступ к видеобитам во время отображения.

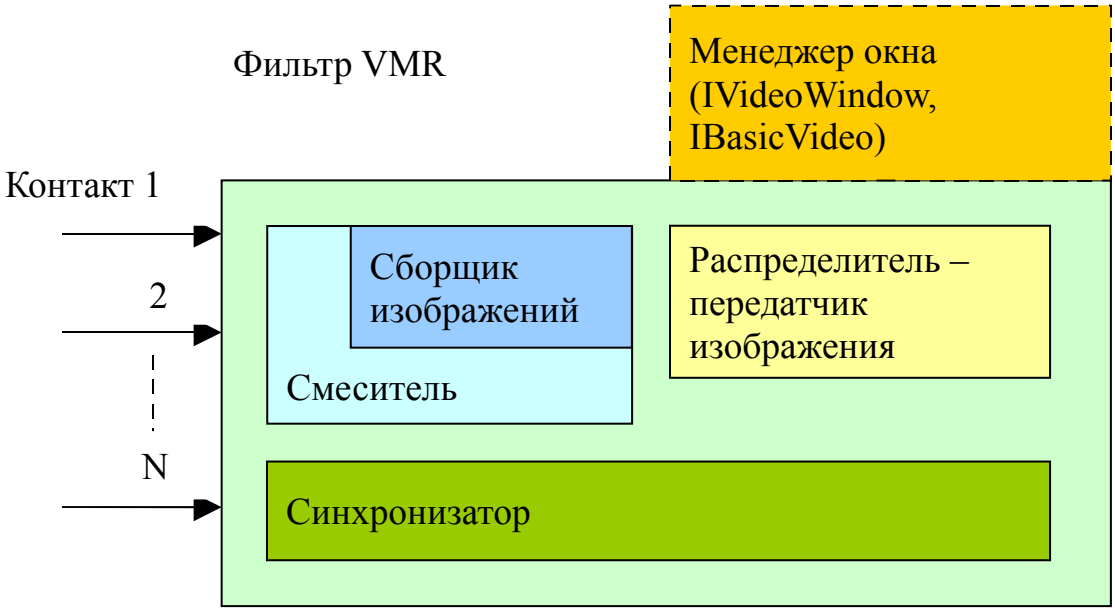

Рис.4. 1 Структурная схема фильтра VMR.

Синхронизатор (Core Synchronization Unit).
Синхронизатор отвечает за точную синхронизацию последовательности кадров. Он использует интерфейс IReferenceClock, предоставляемый менеджером графа для управления синхронизацией кадров.

Менеджер окна.
Используется только в режиме собственного окна. Менеджер окна реализован, как статическая библиотека и поддерживает старые интерфейсы DirectShow IVideoWindow и IBasicVideo для обеспечения обратной совместимости.

Режимы работы фильтров

Отображающие фильтры могут работать в двух режимах, с созданием собственного окна (windowed) и без создания собственного окна (windowless). Фильтр Video Renderer работает только в режиме windowed, то есть со своим окном, а фильтры VMR поддерживают оба режима. По умолчанию фильтры VMR работают в режиме своего окна для совместимости с фильтром Video Renderer.

В режиме своего окна фильтр создает собственное окно для отображения, которое обычно устанавливается как дочернее в окне приложения. Этот ре-

жим поддерживается с помощью интерфейса IVideoWindow. В безоконном режиме фильтр воспроизводит изображение прямо в окне приложения, не создавая собственного.

Оконный режим (windowed) имеет существенные ограничения, главными из которых являются.

- Потенциальная невозможность пересылки сообщений между потоками.
- Менеджер графа должен отправлять определенные опережающие сообщения отображающему фильтру, например WM_PAINT.
- Приложение обязано использовать реализацию интерфейса IVideoWindow для менеджера графа, а не отображающего фильтра, потому что менеджер графа сохраняет правильное внутреннее состояние.
- Приложение должно указать окно, которому будут передаваться сообщения от мыши или клавиатуры приходящие из окна отображающего фильтра.
- Для предотвращения проблем отсечения видеоокно должно иметь строго определенный стиль.

Все эти проблемы отсутствуют в безоконном режиме, поскольку рисование ведется прямо в клиентской области окна приложения. Необходимо только конфигурировать фильтр VMR для этого режима, это проще и удобней, чем использование оконного режима. Таким образом, универсальный фильтр Video Renderer является устаревшим и его использование оправдано в тех случаях, когда аппаратное обеспечение не способно реализовать возможности, заложенные в фильтрах VMR.

Для устройств с видеопортом следует использовать оверлейный фильтр смесителя (Overlay Mixer) для старых платформ, или VMR-7 для Windows XP. Если позволяет аппаратное обеспечение следует использовать фильтр VMR-9, который работает на всех платформах поддерживающих DirectX.

Режимы работы фильтров VMR

Модульная структура фильтров VMR позволяет конфигурировать компоненты фильтра в различных вариантах. В таблице показана конфигурация фильтра для трех режимов отображения и двух режимов смешивания.

Режим	Состав фильтра для одного входного потока.	Состав фильтра для нескольких входных потоков (режим смешивания)
Собственное окно.	1. Распределитель-передатчик изображения	1. Смеситель (Mixer). 2. Сборщик изображения (Com-

	(Allocator-presenter). 2. Синхронизатор (Core Synchronization Unit). 3. Менеджер окна (Window Manager).	positor). 3. Распределитель-передатчик изображения (Allocator-presenter). 4. Синхронизатор(Core Synchronization Unit). 5. Менеджер окна (Window Manager)
Безоконный режим.	1. Распределитель-передатчик изображения (Allocator-presenter). 2. Синхронизатор (Core Synchronization Unit).	1. Смеситель (Mixer). 2. Сборщик изображения (Compositor). 3. Распределитель-передатчик изображения (Allocator-presenter). 4. Синхронизатор(Core Synchronization Unit).
Без отображения.	1. Распределитель-передатчик изображения, который предоставляет приложение (Allocator-presenter). 2. Синхронизатор(Core Synchronization Unit)	1. Смеситель (Mixer). 2. Сборщик изображения (Compositor). 3. Распределитель-передатчик изображения, который предоставляет приложение (Allocator-presenter). 4. Синхронизатор(Core Synchronization Unit)

Таблица 1. Режимы работы фильтров VMR.

Фильтр VMR всегда требует конфигурирования перед подключением контактов, за исключением конфигурации по умолчанию для оконного режима с одиночным входным потоком. Тогда фильтр работает в совместимом режиме с менеджером окна, который поддерживает интерфейсы IVideoWindow и IBasicVideo.

Во всех конфигурациях фильтр не допускает динамического изменения числа контактов в подключенном состоянии, но допускает подключение и отключение. По умолчанию фильтр создается и устанавливается с четырьмя контактами. Не подключенные контакты не снижают производительности фильтра. Выходного контакта фильтр не имеет, поскольку не требует отдельного фильтра для управления окном.

Режим без отображения применяется для игр или приложений, которые создают сложные видеоэффекты. В этом режиме применяется заказной распределитель. Приложение создает и управляет собственной поверхностью DirectDraw (VMR-7) или Direct3D (VMR-9), с доступом к видеобитам во вре-

мя представления. В режиме без отображения приложение выполняет следующие задачи:

1. Управляет окном отображения.
2. Распределяет объект DirectDraw или Direct3D и конечный буфер.
3. Уведомляет остальную часть системы отображения объекта о происходящих событиях.
4. Представляет кадровый буфер в требуемое время.
5. Обрабатывает все изменения режима дисплея, разрешения экрана, потери поверхности.

Интерфейсы отображающих и оверлейных фильтров

Интерфейсы, которые показаны в таблице ниже, позволяют приложению управлять отображением. В таблице не указаны интерфейсы, которые были отменены с появлением фильтров VMR.

Интерфейс	Описание
IAMLine21Decoder	Обеспечивает отображение телетекста американского стандарта (closed-captioned).
IBasicVideo2 (IBasicVideo)	Устанавливает видеосвойства изображения. Интерфейс реализован только на отображающем фильтре Video Renderer.
IMixerOCX	Интерфейс реализован на оверлейном смесителе. Позволяет безоконному клиенту (элементу ActiveX) получать и устанавливать свойства видеопрямоугольника и уведомлять фильтр о событиях.
IMixerOCXNotify	Реализуется безоконными клиентами и вызывается оверлейным смесителем для посылки клиенту уведомлений о событиях в отображающем прямоугольнике.
IQualProp	Возвращает информацию о производительности из отображающего фильтра, например о частоте кадров.
IVideoWindow	Устанавливает свойства видеоокна.
IVMRAspectRatioControl9, IVMRDeinterlaceControl9, IVMRFilterConfig9, IVMRImageCompositor9, IVMRImagePresenter9, IVMRImagePresenterConfig9, IVMRMixer-	Интерфейсы VMR9

Bitmap9, IVMRMixerControl9, IVMRMonitorConfig9, IVMRSurface9, IVMRSurfaceAllocator9, IVMR-SurfaceAllocatorNotify9, IVMR-WindowlessControl9	
IVMRAspectRatioControl, IVM-RDeinterlaceControl, IVMRFilterConfig, IVMRImageCompositor, IVM-RImagePresenter, IVMRImageP-resenterConfig, IVMRMixerBit-map, IVMRMixerControl, IVMRSurfaceAllocator, IVMR-SurfaceAllocatorNotify, IVMR-WindowlessControl	Интерфейсы VMR7

Таблица 2 Список интерфейсов отображающих фильтров

Интерфейсы, управляющие графом

Имеется ряд интерфейсов, которые управляют графом в процессе работы. Для реализации режима предварительного просмотра понадобятся некоторые из них. Ниже приводится краткая характеристика всех интерфейсов этой группы.

Интерфейс	Описание
IAMClockAdjust	Регулировка ссылочного времени графа.
IAMGraphStreams	Методы интерфейса обеспечивают управление некоторыми временными параметрами графа при работе с источниками реального времени (живыми) и поиск интерфейсов вверх по потоку.
IFilterChain	Управление цепочками фильтров в графе.
IMediaControl	Запуск, пауза, остановка графа. Реализация некоторых методов для Visual Basic.
IMediaEventEx	Обработка событий в графе.
IMediaSeeking	Передвижение по потоку и установка скорости воспроизведения файла.
IQueueCommand	Очередь команд (методов) для выполнение в указанное время.
IVideoFrameStep	Покадровый просмотр видеопотока.

Таблица 3 Интерфейсы управления графом

Из всех представленных интерфейсов этой группы в первую очередь понадобится интерфейс IMediaControl и возможно IFilterChain. Интерфейс IMediaControl предоставляет методы для управления данными, проходящими через граф. Он имеет несколько методов, краткое описание которых приводятся в таблице. Реализует данный интерфейс графический менеджер.

Метод	Описание
Run	Запускает все фильтры в графе.
Pause	Устанавливает паузу.
Stop	Останавливает все фильтры в графе.
StopWhenReady	Устанавливает паузу, позволяющую установить данные в очередь, и затем останавливает граф.
GetState	Возвращает состояние графа.

Таблица 4 Методы интерфейса IMediaControl.

Инициализация режима предварительного просмотра

Для инициализации предварительного просмотра будем использовать метод CCapture::InitCaptureVideo. Продолжать работу будем с очередной версией проекта ProjectState5, которую создаем копированием предыдущей версии проекта. Предварительно следует определиться с интерфейсом IMediaControl. Для управления графом необходимо, как минимум два метода, один для пуска графа и второй для его останова. Поскольку интерфейс IMediaControl будет использоваться часто, то можно создать переменную класса для указателя на интерфейс, например IMediaControl* m_pControl. В таком случае следует получить указатель на него в методе CCapture::InitCaptureVideo

```
m_pGraph->QueryInterface(&m_pControl);
```

Можно не создавать переменную, а запрашивать интерфейс IMediaControl в каждом методе, когда это потребуется. Встречаются оба подхода к данной проблеме, следует только заметить, что хранение некоторых часто используемых указателей в виде членов класса, заметно уменьшает объем кода.

Если задача отображения простая и не требуется свойств отображающих фильтров VMR, то можно ограничиться только фильтром Video Renderer. В этом случае не требуется анализа версии ОС, в противном случае следует

проанализировать версию ОС и установить нужный фильтр. Далее напишем код, который будет создавать, и устанавливать в граф один из отображающих фильтров в зависимости от версии системы. В общем случае анализу следует подвергнуть большее число факторов, но если не учитывать устройства с видеопортом, то можно ограничиться двумя фильтрами Video Renderer и VMR-9, а для этого достаточно анализа версии системы. Текст метода CCapture::VerifyOSVersion здесь не рассматриваем, поскольку он имеет косвенное отношение к теме. При установке фильтра в граф зададим ему свое имя для того, чтобы можно было по имени идентифицировать в дальнейшем тип фильтра. Имена фильтров создадим в таблице строк.

```
//Инициализация режима предварительного просмотра
WCHAR nameF[32]; //Имя фильтра

//Выбор визуализирующего фильтра
if(VerifyOSVersion(5,1)) //Версия XP
{
  //Создать фильтр VMR9
  if(S_OK != (hr = CoCreateInstance(
            CLSID_VideoMixingRenderer9, NULL
            , CLSCTX_INPROC,IID_IBaseFilter
            , reinterpret_cast<void **>
                        (&m_pVRender))) )
    return FALSE;

  //Установить фильтр VMR9 в граф
  ::LoadStringW(NULL,IDS_VMR9,nameF,32);
  if(FAILED(hr =
          m_pGraph2->AddFilter(m_pVRender,nameF)))
    throw hr;

  //Инициализация фильтра VMR9. Не использовать
  //в совместимом режиме!
  //if(!InitWindowlessVMR(hwndView))
  //  return FALSE;
}
else
{
  //Создать старый фильтр
  if(S_OK != (hr = CoCreateInstance(
                CLSID_VideoRenderer, NULL
                , CLSCTX_INPROC,IID_IBaseFilter
                , reinterpret_cast<void **>
                          (&m_pVRender))) )
    throw hr;

  //Установить старый фильтр в граф
  ::LoadStringW(NULL,IDS_VROLD,nameF,32);
  if(FAILED(hr =
        m_pGraph2->AddFilter(m_pVRender,nameF)))
```

```
    throw hr;
  }
```

Теперь при инициализации в граф будет устанавливаться отображающий фильтр. Переменная m_pVRender сохраняет указатель на фильтр. Метод InitWindowlessVMR, который производит инициализацию фильтра VMR в безоконном режиме, описан в разделе " Безоконный режим фильтров VMR". Пока вызов этого метода не используется.

Если в граф устанавливается фильтр VMR, то по умолчанию он будет работать в совместимом режиме, полностью выполняя функции устаревшего фильтра Video Renderer, не потребляя системных ресурсов больше старого фильтра. Дополнительного конфигурирования фильтр VMR при этом не требует, и он полностью управляется с помощью интерфейсов IVideoWindow и IBasicVideo. Подключение в совместимом режиме производится только к одному, нулевому контакту фильтра. По умолчанию в совместимом режиме режим смешивания запрещен и если он требуется, то необходимо конфигурировать фильтр перед подключением.

Достройка графа и запуск режима предварительного просмотра

Имея в графе отображающий фильтр можно достроить граф и запустить предварительный просмотр. Для достройки графа напишем отдельный метод CCapture::SetModePreview.

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

  string sErr = _T("SetModePreview.");

  //Если граф построен, выход
  if(m_bPreview)
    return TRUE;

  m_bSmartTee = FALSE;

  //Если PIN_CATEGORY_PREVIEW отсутствует,  будет вставлен
  //фильтр Smart Tee.
  hr = m_pBuilder->RenderStream(&PIN_CATEGORY_PREVIEW
        ,&MEDIATYPE_Video,m_pCapV,NULL,m_pVRender);

  //Вставлен фильтр Smart Tee
  if(hr == VFW_S_NOPREVIEWPIN)
    m_bSmartTee = TRUE;
```

```
  if(FAILED(hr))
  {
    m_bPreview = FALSE;
    return FALSE;
  }

  m_bPreview = TRUE;

  return TRUE;
}
```

Здесь проще использовать метод RenderStream, с указанием отображающего фильтра, поскольку при использовании метода Render (RenderEx) следует найти указатель на контакт предварительного просмотра. Кроме того метод RenderStream установит тот отображающий фильтр, который указан в последнем параметре, а метод Render в системе под управлением Windows XP установит фильтр по умолчанию VMR7.

Существует особенность установки отображающего фильтра Video Renderer с помощью метода RenderStream. По умолчанию, на фильтре сбора при его установке в граф, устанавливается цветовой формат YUY2. Если затем устанавливается отображающий фильтр Video Renderer, тогда метод изменит цветовой формат фильтра сбора с YUY2 на формат RGB555 и соединит фильтры без установки промежуточных трансформирующих фильтров. Если же перед вызовом метода RenderStream, на фильтре сбора принудительно установить формат YUY2, тогда метод вставит дополнительный трансформирующий фильтр перед фильтром Video Renderer. Фильтр VMR9 (7) не нуждается в трансформирующих фильтрах и подключается прямо к фильтру сбора. Установку цветовых форматов рассмотрим позже.

Для запуска режима предварительного просмотра напишем метод CCapture::StartPreview, который будет производить запуск графа с контролем его состояния. На данном этапе можно было бы ограничиться простым вызовом метода IMediaControl::Run, который запускает весь граф, поскольку контроль состояния графа для данной ситуации не имеет большого значения. Контроль состояния графа важен в случаях, когда необходима блокировка элементов интерфейса. Особенности поведения метода IMediaControl::GetState позволяют синхронизировать метод StartPreview с состоянием графа.

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

  //Состояние фильтрового графа
  OAFilterState state = State_Stopped;
```

```
  //Пуск всех фильтров графа
  CComQIPtr<IMediaControl> pCtrl(m_pGraph2);
  ASSERT(pCtrl);

  if(S_FALSE ==  pCtrl->Run())
  {
    //Не все фильтры запустились, проверка состояния
  //графа
    if(FAILED(pCtrl->GetState(10,&state)))
      return FALSE;

    if(state != State_Running)
      return FALSE;
  }

  m_bStartGraph = TRUE;

  return TRUE;
}
```

Метод IMediaControl::Run может возвратиться с кодом S_FALSE, который означает, что не все фильтры успели перейти в запущенное состояние. Для уточнения состояния вызывается метод IMediaControl::GetState, который позволяет точно определить состояние графа. Ниже приведено описание метода IMediaControl::GetState.

Метод GetState возвращает состояние графа, определенное в перечислении

```
typedef enum _FilterState
{
    State_Stopped,
    State_Paused,
    State_Running
} FILTER_STATE;
```

Передача состояния графа не является синхронным процессом. Для определения состояния графа необходимо вызывать метод GetState. Метод блокируется до тех пор, пока граф не перейдет в запущенное состояние или пока не пройдет указанное время. Не следует указывать большое значение времени, поскольку в состоянии блокировки метода потоки не обрабатывают сообщения, например пользовательского ввода. Данный метод полезен для управления интерфейсом пользователя.

```
HRESULT GetState(
  LONG msTimeout,
  OAFilterState *pfs
);
```

Параметры.

msTimeout
[in]
Время, отводимое на определение состояния графа, в течение которого метод блокируется. Может быть равно INFINITE.

pfs
[out]
Указатель на переменную, которая принимает состояние графа.
Возвращаемые величины.

Величина.	Описание.
S_OK	Выполнено.
VFW_S_STATE_INTERMEDIATE	Граф находится в промежуточном состоянии.
VFW_S_CANT_CUE	Граф в состоянии паузы, но очереди данных нет.
E_FAIL	Ошибка.

Метод возвращает ошибку, если другой поток изменяет состояние графа, когда метод в состоянии блокировки.

Определение состояния графа достаточно частая операция, поэтому неплохо иметь отдельный метод, возвращающий состояние графа.

```
OAFilterState CCapture::GetStateGraph(void)
{
  OAFilterState state; //Состояние фильтрового графа
  CComQIPtr<IMediaControl> pCtrl(m_pGraph2);
  ASSERT(pCtrl);

  HRESULT hr;
  if(FAILED(hr = pCtrl->GetState(30,&state)))
        CTVshowApp::myMessageError(
                      hr,"Error GetStateGraph");

  return state;
}
```

Методов SetModePreview и StartPreview достаточно для запуска режима предварительного просмотра в собственном окне отображающего фильтра Video Renderer, но в большинстве случаев окно фильтра устанавливается как дочернее в окне приложения. Для запуска режима предварительного просмотра в обработчике CFView1::OnMyMessageEndInit можем добавить вызов

```
if(!pCap->SetModePreview()|| !pCap->StartPreview())
      AfxMessageBox(_T("Невозможно произвести запуск
                  режима предварительного просмотра"));
```

и посмотреть на результат, который показан на рисунке 4.2. Предварительно необходимо изменить условие установки отображающего фильтра в граф в методе CCapture::InitCaptureVideo, таким образом, чтобы при любой ОС устанавливался фильтр Video Renderer. Фильтр VMR пока не инициализирован и картинки мы не увидим. Для этого можно изменить условие установки таким образом.

```
if(!VerifyOSVersion(5,1))
```

Как видно на рисунке отображение происходит в собственном окне фильтра Video Renderer, которое не связано с окном приложения. Размеры окна так же установлены фильтром по умолчанию и не соответствуют текущему стандарту SECAM (352x 288).

Для управления окном отображающего фильтра и установки его параметров служит интерфейс IVideoWindow. Рассмотрим основные свойства и методы этого интерфейса, а затем напишем методы для корректировки отображения.

Интерфейс IVideoWindow

Данный интерфейс предоставляют все типы отображающих фильтров и менеджер графа, который имеет свою версию. Именно версию менеджера графа должно использовать приложение для управления окном отображающего фильтра. Фильтры VMR используют этот интерфейс по умолчанию. Менеджер графа перехватывает все обращения к отображающему фильтру, а также и все сообщения окна фильтра, например, такие как WM_DISPLAYCHANGE, которое требуется для его обновления.

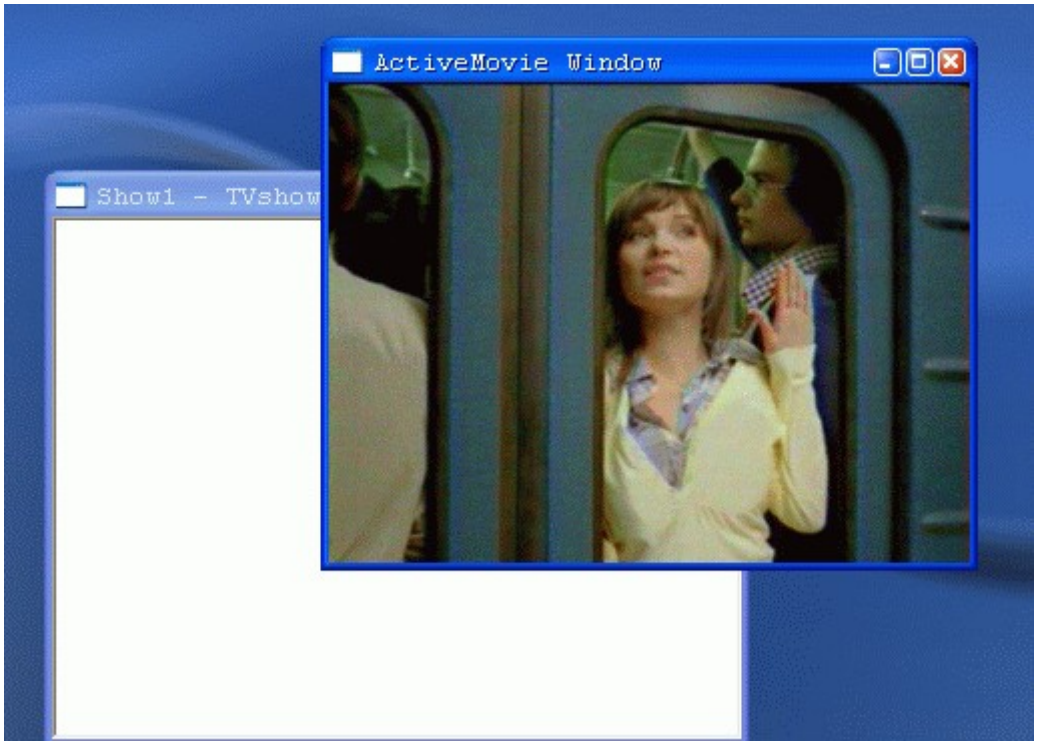

Рис.4. 2 Предварительный просмотр с фильтром Video Renderer

Если в графе есть больше одного отображающего фильтра, то менеджер графа связывается только с одним фильтром. Таким образом, при работе с несколькими окнами приложение должно использовать интерфейс IVideoWindow самого фильтра. В этом случае, для каждого окна отображающего фильтра следует отправлять сообщения, используя метод IVideoWindow::NotifyOwnerMessage.

Приложение MFC, устанавливающее видеоокно, в дочернем окне должно определить пустой обработчик для сообщения WM_ERASEBKGND для обеспечения правильной перерисовки видеоокна.

Если отображающий фильтр не подключен, то все методы интерфейса возвращают VFW_E_NOT_CONNECTED. Свойства установленные на фильтре не изменяются при подключении или отключении фильтра. Поскольку этот фильтр совместим с OLE автоматизацией, то все булевские величины имеют следующие значения OAFALSE (0) и OATRUE (–1).

Методы IVideoWindow	Описание
get_AutoShow	Определяет, может ли отображающий фильтр автоматически отображать видеоокно, когда он принимает данные.
get_BackgroundPalette	Определяет, отображает ли видеоокно палитру на заднем плане.
get_BorderColor	Возвращает цвет границ (рамки) прямоугольника

	изображения.
get_Caption	Возвращает заголовок видеоокна.
get_FullScreenMode	Определяет, способен ли фильтр работать в полно-экранном режиме.
get_Height	Возвращает высоту видеоокна.
get_Left	Возвращает координату X видеоокна.
get_MessageDrain	Возвращает окно, принимающее сообщения от мыши и клавиатуры из видеоокна.
get_Owner	Возвращает родительское окно.
get_Top	Возвращает координату Y видеоокна.
get_Visible	Определяет, является ли видеоокно видимым.
get_Width	Возвращает ширину видеоокна.
get_WindowState	Возвращает состояние окна, видимое, скрытое, мини-мизированное или максимизированное.
get_WindowStyle	Возвращает стиль видеоокна.
get_WindowStyleEx	Возвращает расширенный стиль видеоокна.
GetMaxIdealImageSize	Возвращает идеальный максимальный размер изоб-ражения.
GetMinIdealImageSize	Возвращает идеальный минимальный размер изобра-жения.
GetRestorePosition	Возвращает предыдущую позицию окна.
GetWindowPosition	Возвращает текущую позицию окна.
HideCursor	Скрывает курсор.
IsCursorHidden	Определяет, скрыт ли курсор.
NotifyOwnerMessage	Устанавливает окно, которое будет перехватывать со-общение, поступающее в видеоокно.
put_AutoShow	Устанавливает режим автоматического отображения, когда отображающий фильтр принимает данные.
put_BackgroundPalette	Указывает, что видеоокно должно отображать палит-ру на заднем плане.
put_BorderColor	Устанавливает цвет границ (рамки) прямоугольника изображения.
put_Caption	Устанавливает заголовок видеоокна.
put_FullScreenMode	Запрещает или разрешает полноэкранный режим.
put_Height	Устанавливает высоту видеоокна.
put_Left	Устанавливает координату X окна.
put_MessageDrain	Устанавливает окно, принимающее сообщения от мыши и клавиатуры из видеоокна.
put_Owner	Устанавливает родительское окно для видеоокна.

put_Top	Устанавливает координату Y окна.
put_Visible	Показывает или скрывает видеоокно.
put_Width	Устанавливает ширину видеоокна
put_WindowState	Устанавливает состояние видеоокна.
put_WindowStyle	Устанавливает стиль видеоокна
put_WindowStyleEx	Устанавливает расширенный стиль видеоокна
SetWindowForeground	Устанавливает видеоокно на передний план.
SetWindowPosition	Устанавливает позицию видеоокна.

Таблица 5. Интерфейс IVideoWindow.

Установка видеоокна в качестве дочернего окна приложения

Для установки видеоокна в качестве родительского напишем метод SetWndPreview, который в качестве первого параметра будет принимать дескриптор нашего окна отображения (класс CShowFrame). Дескриптор возвращается методом CTVshowApp::myGetHandleWndShow. В классе приложения сохраняются указатели на все окна приложения. Логика метода проста и комментариев в тексте достаточно для ее понимания.

```
BOOL CCapture::SetWndPreview(HWND hwnd)
{
  ASSERT(hwnd);

  CComQIPtr<IVideoWindow> pVW(m_pGraph2);
  if(!pVW)
    return FALSE;

  //Установка родительского окна для видео окна
  if(S_OK != pVW->put_Owner(
                   reinterpret_cast<OAHWND>(hwnd)))
    return FALSE;

  //Установка стиля для видео окна
  if(S_OK != pVW->put_WindowStyle(
                       WS_CHILD | WS_CLIPSIBLINGS))
    return FALSE;

  //Подгонка изображения под размеры клиентской области //окна
  HWND hwndP = ::GetParent(hwnd);
  RECT grc;
  ::GetClientRect(hwndP,&grc);

  //Установка видео окна фильтра в клиентской области //роди-
  тельского окна
```

```
  if(S_OK != pVW->SetWindowPosition(0, 0, grc.right
                                          , grc.bottom))
    return FALSE;

  //Обработка сообщений в классе CShowView, удалить //коммента-
  рии при необходимости
  //if(S_OK != pVW->put_MessageDrain(
  //                      reinterpret_cast<OAHWND>(hwnd)))
  //  return FALSE;

  return TRUE;
}
```

При смене родительского окна или при его закрытии необходимо освободить видеоокно и затем можно назначить ему новое родительское окно. Освобождение видеоокна производится в методе CCapture::DeleteVW.

```
void CCapture::DeleteVW(void)
{
  if(!m_bPreview) return;

  FILTER_INFO info;
  if(FAILED(m_pVRender->QueryFilterInfo(&info)))
    return;

  if(wcscmp(L"VR_OLD",info.achName))
    return;

  HRESULT hr= S_OK;
  CComQIPtr<IVideoWindow> pVW(m_pGraph2);

  if(pVW)
  {
    //Удалить дескриптор старого окна
    OAHWND hWnd;
    if(S_OK == pVW->get_Owner(&hWnd))
    {
      hr = pVW->put_Visible(OAFALSE);
      hr = pVW->put_Owner(NULL);
    }
  }
}
```

Вызов метода можно произвести, например, в обработчике CShowFrame::OnClose(), который вызывается при закрытии родительского окна.

```
void  CShowFrame::OnClose()
{
  CTVshowApp* pApp = reinterpret_cast<CTVshowApp*>(AfxGetApp());
```

```
  //Удалить окно IVideoWindow, очистить дескриптор
  pApp->myGetPtrCap()->DeleteVW();

  //Обнулить указатель на окно отображения
  pApp->myClearPtrWndShow();
  CFrameWnd::OnClose();
}
```

Теперь можно модернизировать запуск режима предварительного просмотра таким образом, чтобы окно отображения фильтра Video Renderer стало дочерним в окне CShowView. Для этого изменим вызовы методов в обработчике CFView1::OnMyMessageEndInit следующим образом.

```
//проверить существование родительского окна для
//предварительного просмотра
CTVshowApp* pApp = static_cast<CTVshowApp*>(AfxGetApp());

HWND hwnd = pApp->myGetHandleWndShow();

if(!hwnd)
{
  sErr += _T("Установите окно для предварительного
                                            просмотра");
  throw FALSE;
}

//установить окно предварительного просмотра
if(!pCap->SetModePreview())
  throw FALSE;

if(!pCap->SetWndPreview(hwnd) || !pCap->StartPreview())
{
  sErr += _T("Невозможно произвести запуск режима предваритель-
  ного просмотра");
  throw FALSE;
}
```

В результате окно фильтра Video Renderer будет вписано в родительское окно любых размеров. Необходимо только установить размеры родительского окна до вызовов указанных методов. Для изменения размеров родительского окна написан метод CCapture::ResizeWndParentPreview, ознакомиться с текстом, которого можно в проекте ProjectState5. Пока он не используется. На рисунке 4.3 показан результат установки окна фильтра в качестве дочернего в окне приложения.

Для проверки работы с отображающим фильтром VMR9 необходимо восстановить условие его установки. Фильтр будет работать по умолчанию в режиме фильтра Video Renderer. Применение фильтра VMR в безоконном режиме описано в разделе " Безоконный режим фильтров VMR".

Рис.4. 3. Установка окна Video Renderer.

Сообщения от окна фильтра Video Renderer

Для обработки сообщений от клавиатуры и мыши, которые могут происходить в видеоокне, достаточно в методе SetWndPreview добавить вызов метода put_MessageDrain.

```
if(S_OK !=
    pVW->put_MessageDrain(reinterpret_cast<OAHWND>(hwnd)))

    return FALSE;
```

Окно обработки не обязательно должно быть родительским, можно указать дескриптор другого окна. Обработчик события устанавливается обычным образом. Описание обработки событий приводится в разделе " Обработка событий в DirectShow". Проект в данном состоянии находится в папке ProjectState5. Далее продолжать работу над проектом будем новой версии в папке ProjectState6.

Управление параметрами видеопотока на фильтре сбора

Видеоданные на контактах фильтра сбора могут быть представлены в различных цветовых форматах, размеры изображения, частота кадров и другие параметры могут изменяться. Все параметры изображения определяются медиа форматом, который представлен на контакте фильтра.

Медиа формат описывается структурой AM_MEDIA_TYPE, которая кратко упоминалась ранее и будет подробно рассмотрена далее. Для WDM устройств управление медиа форматом производится с помощью интерфейса

IAMStreamConfig. При установке нового медиа формата выходной контакт фильтра может быть, как подключен к следующему фильтру, так и отключен.

Если контакт отключен и фильтр поддерживает устанавливаемый формат, то метод IAMStreamConfig::SetFormat выполняется успешно. Установка формата происходит при последующем соединении фильтров. При этом производится согласование медиа форматов с подключаемым входным контактом следующего фильтра. Если медиа форматы не могут быть согласованы фильтрами, соединение не происходит. Для соединения фильтров может применяться метод, использующий алгоритм интеллектуального соединения. Тогда для согласования форматов автоматически устанавливается промежуточный трансформирующий фильтр. При других способах соединения установку промежуточного фильтра следует делать вручную.

Если контакт подключен, метод IAMStreamConfig::SetFormat пытается установить формат и произвести повторное соединение фильтров. Поскольку метод SetFormat не производит установку промежуточных фильтров при несогласованности форматов, то он заканчивает выполнение с ошибкой. В остальных случаях установка формата происходит успешно.

Структура AM_MEDIA_TYPE описывает формат медиа выборки, для видеоданных это один кадр изображения. Для изучения установки параметров изображения можно использовать изменение медиа формата на контакте предварительного просмотра (Preview). Контакт предварительного просмотра позволяет производить изменение медиа формата в более широких пределах, чем контакт записи и не нужно дополнительно организовывать запись файла.

Изменение медиа формата допустимо только в определенных пределах, которые определяются возможностями фильтра и описаны в соответствующих структурах для каждого медиа формата. Все поддерживаемые фильтром медиа форматы можно определить с помощью метода IAMStreamConfig::GetStreamCaps, применение, которого рассмотрим далее. Большое количество поддерживаемых фильтром медиа форматов и сложность структур, используемых при этом, делает процесс изменения медиа формата достаточно сложным.

Установку медиа формата можно производить раздельно для контактов предварительного просмотра и контакта записи (capture). Если в режиме предварительного просмотра размер изображения можно установить любым, вплоть до полного экрана, то при записи файлов предварительный просмотр может быть на экране не более чем номинального размера 352x288 (SEKAM). Это вполне понятно, поскольку нет смысла производить запись растянутого изображения, это можно сделать при воспроизведении.

Если для установки формата достаточно предопределенного формата, то можно в структуре указать GUID главного типа и подтипа при нулевых остальных данных. Формат будет дополнен остальными данными автоматически.

Для правильного использования структуры AM_MEDIA_TYPE и других, участвующих в установке медиа формата следует предварительно ознакомиться с ними и методами интерфейса IAMStreamConfig, с помощью которых производится изменение медиа формата. Также далее ознакомимся с принципами изменения размеров изображения и преобразования входного изображения в выходное изображение.

Дальнейшая работа будет производиться с проектом в папке ProjectState6.

Структура AM_MEDIA_TYPE

```
typedef struct  _MediaType {
    GUID          majortype;
    GUID          subtype;
    BOOL         bFixedSizeSamples;
    BOOL        bTemporalCompression;
    ULONG       lSampleSize;
    GUID         formattype;
    IUnknown    *pUnk;
    ULONG       cbFormat;
    [size_is(cbFormat)] BYTE *pbFormat;
} AM_MEDIA_TYPE;
```

majortype
Главный тип (GUID), описывающий медиа выборку. Для нашего примера это MEDIATYPE_Video.

Subtype
Подтип (GUID), описывающий медиа выборку. Для видеоизображения определяет цветовые форматы, RGB, например, и другие. Из всего множества подтипов фильтр сбора может поддерживать ограниченное число подтипов. Определение поддерживаемых подтипов производится с помощью метода IAMStreamConfig::GetStreamCaps. Для некоторых форматов величина может иметь значение MEDIASUBTYPE_None, это означает, что подтип не нужен.

bFixedSizeSamples

Если значение равно TRUE, то выборка имеет фиксированный размер. Эта величина предназначена только для чтения. Для звука она всегда установлена в TRUE. Для не сжатого видеоизображения она также равна TRUE, а для сжатого FALSE.

bTemporalCompression
Значение TRUE указывает на применение межкадрового кодирования (временное сжатие). Эта величина предназначена только для чтения.

lSampleSize
Размер выборки в байтах. Для сжатых данных величина может быть равна нулю.

formattype
GUID, который указывает на тип блока форматирования, то есть указывается тип структуры, описывающей формат выборки.

Тип формата	Тип структуры
FORMAT_None	None.
FORMAT_DvInfo	DVINFO
FORMAT_MPEGVideo	MPEG1VIDEOINFO
FORMAT_MPEG2Video	MPEG2VIDEOINFO
FORMAT_VideoInfo	VIDEOINFOHEADER
FORMAT_VideoInfo2	VIDEOINFOHEADER2
FORMAT_WaveFormatEx	WAVEFORMATEX
GUID_NULL	None

Для фильтра сбора аналогового телевидения видеоизображение описывается структурой VIDEOINFOHEADER, а звук структурой WAVEFORMATEX.

pUnk
Не применяется.

cbFormat
Размер блока форматирования в байтах.

pbFormat
Указатель на блок форматирования. Структура формата должна быть представлена, если тип формата не GUID_NULL или не FORMAT_None. Может быть NULL. Для доступа к структуре определенного типа указатель должен быть преобразован в тип соответствующий типу структуры. В некоторых случаях можно установить указатель на блок формата в NULL и тип формата в GUID_NULL. Это позволит указать диапазон возможных форматов, например, указав MEDIASUBTYPE_RGB24, можно не указывать точно ширину и высоту изображения.

На рисунке 4.5 показано строение структуры AM_MEDIA_TYPE для типа формата FORMAT_VideoInfo.

Источник и получатель изображения и их взаимодействие

В структурах VIDEOINFO, VIDEOINFOHEADER (VIDEOINFOHEADER2) определены три размера, характеризующие прямоугольник изображения. Рассмотрим, как эти размеры связаны между собой, и каким образом фильтры формируют конечное изображение, используя данные этих структур. Первый размер biWidth и biHeight описан в структуре BITMAPINFOHEADER, которая является членом bmiHeader структуры VIDEOINFOHEADER (рис.5). Второй размер rcSource описан в структуре VIDEOINFOHEADER. Третий размер rcTarget также описан в структуре VIDEOINFOHEADER.

Предположим, что имеется два фильтра A и B соединенных с определенным медиа форматом (Рис.4.4).

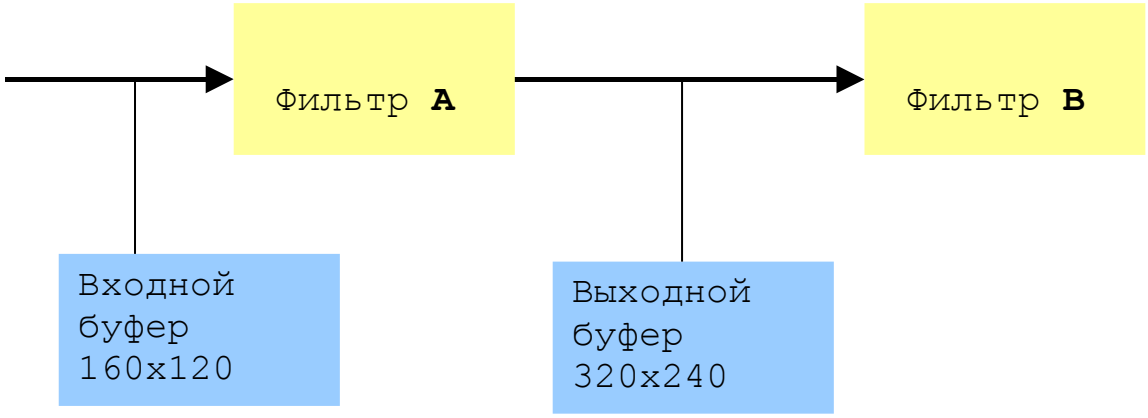

Рис.4. 4

Между этими фильтрами передается буфер изображения с размером bmiHeader.biWidth, bmiHeader.biHeight. Фильтр "A" должен взять из входного буфера изображение или его часть и растянуть (сжать) до размеров определенных прямоугольником rcTarget. Фильтр "A" берет из входного буфера часть изображения, которая определяется как результат сравнения прямоугольника rcSource с размером прямоугольника передаваемого между фильтрами буфера.

Сравнение прямоугольника rcSource производится со значениями bmiHeader.biWidth и bmiHeader.biHeight, установленными при соединении фильтров. Результат сравнения определяет, какую долю входного буфера использовать как источник изображения. Сравнение прямоугольника rcTarget производится со значениями доли входного буфера и результат сравнения определяет коэффициенты растяжения или сжатия. Если прямоугольник rcSource пустой,

то используется весь входной буфер. Если прямоугольник rcTarget пустой, то заполняется весь выходной буфер.
Пример, приведенный ниже, поясняет, как это происходит.

Например, пусть на вход поступает видеопоток с размером 160 x 120 пикселей, и фильтры соединены со следующим медиа форматом.

```
(biWidth, biHeight): 320, 240
rcSource: (0, 0, 0, 0)
rcTarget: (0, 0, 0, 0)
```

Фильтр должен будет растянуть входной прямоугольник в два раза и заполнить выходной буфер 320 x 240.

Второй пример. Для того же входного потока фильтры соединены с таким медиа форматом.

```
(biWidth, biHeight): 320, 240
rcSource: (0, 0, 160, 240)
rcTarget: (0, 0, 0, 0)
```

Прямоугольник rcSource представляет собой левую половину буфера (biWidth, biHeight). Фильтр "A" должен будет взять из входного потока его левую половину (0, 0, 80, 120) и растянуть до размеров 320 x 240 пикселей, то есть в четыре раза.

Третий пример. Предположим, что фильтр "A" вызвал метод, определяющий медиа формат IMediaSample::GetMediaType, который определил, что новый медиа формат имеет такие параметры.

```
(biWidth, biHeight): 640, 480
rcSource: (0, 0, 160, 120)
rcTarget: (0, 0, 80, 60)
```

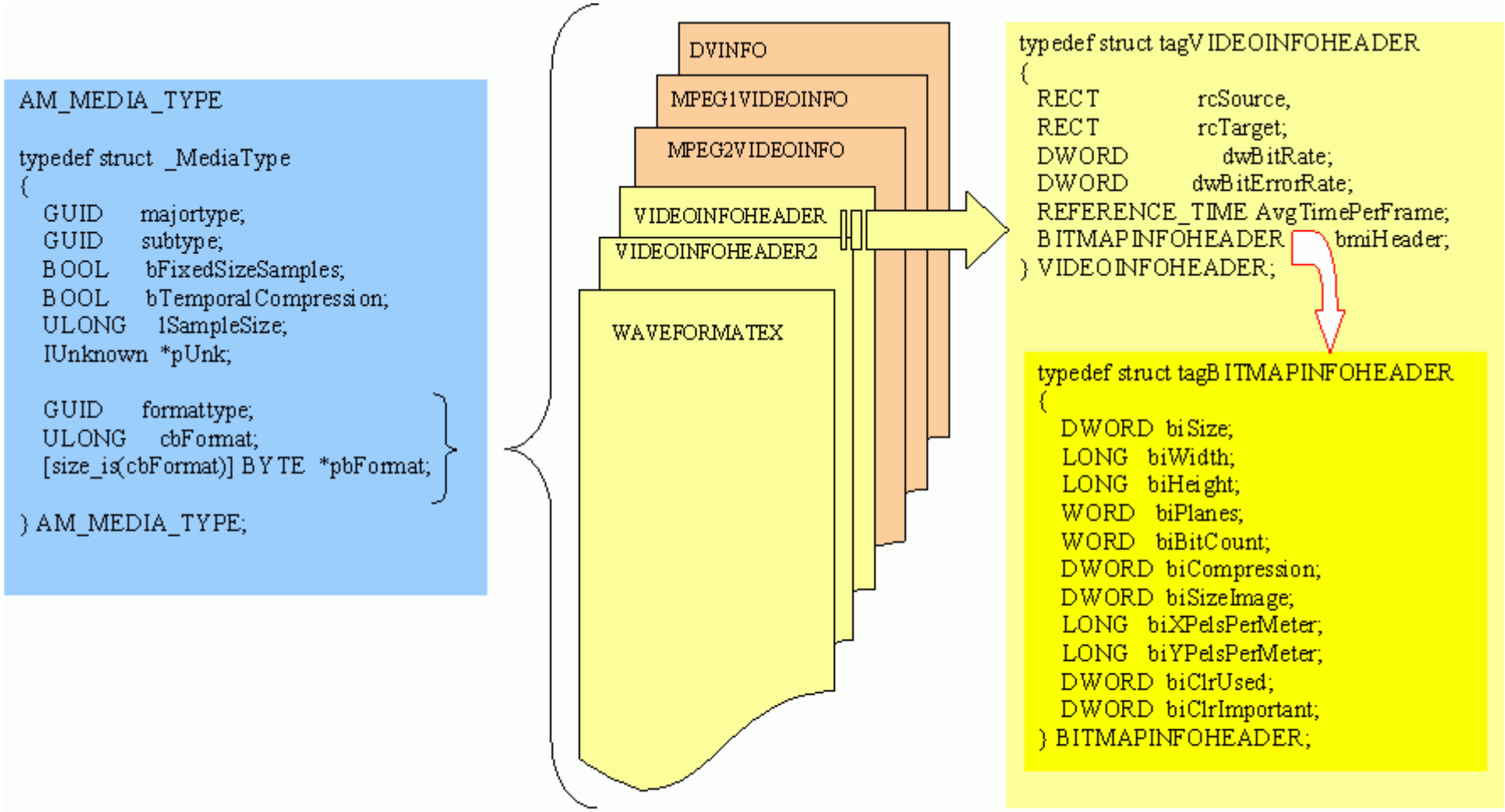

Рис.4. 1 Строение структуры AM_MEDIA_TYPE

Это означает, что фильтр "B" желает изменить изображение по сравнению с первоначально установленным медиа форматом при соединении фильтров. Буфер теперь будет иметь размер 640 x 480, прямоугольник rcSource сравнивается с исходным значением 320 x 240 пикселей и представляет собой левую половину буфера (biWidth, biHeight), а не 25% как можно было подумать. Левая половина входного потока (120,60) как раз и будет иметь значение (0, 0, 80, 60). Таким образом, изображение не будет растягиваться и разместится в буфере 640 x 480 в левом верхнем углу (80,60). Таким образом, прямоугольник rcSource ограничен размерами 320 x 240, а прямоугольник rcTarget размерами 640 x 480.
Фильтр "A" не изменяет данных за пределами прямоугольника rcSource.

Структура VIDEOINFOHEADER

Структура VIDEOINFOHEADER описывает растр и цветовую информацию видеоизображения.

```
typedef struct tagVIDEOINFOHEADER {
    RECT                    rcSource,
    RECT                     rcTarget;
    DWORD                   dwBitRate;
    DWORD                   dwBitErrorRate;
    REFERENCE_TIME        AvgTimePerFrame;
    BITMAPINFOHEADER      bmiHeader;
} VIDEOINFOHEADER;
```

rcSource
Координаты отсекающего прямоугольника, который определяет выборку части сигнала из входного видеопотока. Нулевые значения указывают на то, что параметр не используется.

rcTarget
Координаты прямоугольника в выходном окне, куда помещается часть изображения определенная в параметре *rcSource*. Нулевые значения указывают на то, что параметр не используется.

dwBitRate
Усредненная скорость видеопотока в битах за секунду.

dwBitErrorRate
Скорость потока ошибок в данных в битах за секунду.

AvgTimePerFrame
Среднее время отображения кадра, измеряемое в 100 наносекундных единицах. Действительное значение может быть больше. Фильтр может удалять часть кадров из потока, если производительность системы не позволяет обработать поток без потерь или если частота кадров дисплея не позволяет отображать видеопоток без интерференции. Точное значение можно определить на отображающем фильтре

Video Renderer с помощью метода IQualProp::get_AvgFrameRate или можно вызвать метод IBasicVideo::get_AvgTimePerFrame. Для фильтров VMR можно вызвать метод _IPin::ConnectionMediaType на входном контакте, поскольку фильтры VMR поддерживают несколько входных потоков, которые могут иметь разные значения этого параметра.

bmiHeader
Структура BITMAPINFOHEADER, которая содержит информацию о цветах и размерах растрового (DIB) изображения. Если блок формата содержит таблицу цветов или цветовую маску, то таблица размещается сразу за параметром bmiHeader. Указатель на первый вход таблицы можно получить, преобразовав адрес bmiHeader в указатель на BITMAPINFO. Подробней ознакомиться с получением указателя можно в описании структуры BITMAPINFOHEADER.

Структура BITMAPINFOHEADER

Структура BITMAPINFOHEADER содержит информацию о размере и цветовом формате растра DIB.

```
typedef struct tagBITMAPINFOHEADER {
    DWORD   biSize;
    LONG      biWidth;
    LONG      biHeight;
    WORD      biPlanes;
    WORD      biBitCount;
    DWORD   biCompression;
    DWORD   biSizeImage;
    LONG      biXPelsPerMeter;
    LONG      biYPelsPerMeter;
    DWORD   biClrUsed;
    DWORD   biClrImportant;
} BITMAPINFOHEADER;
```

biSize
Размер структуры в байтах.

biWidth
Ширина растра. Для формата RGB ширина указана в пикселях. Для форматов YUV, где битовая глубина является степенью двойки, ширина также указывается в пикселях. Для остальных вариантов YUV формата ширина указывается в байтах.
Декодеры и видеоисточники должны представлять форматы, где biWidth является шириной изображения. Если отображающий фильтр использует DirectDraw, то он модифицирует формат так, что бы величина biWidth была кратна шагу поверхности, и член структуры VIDEOINFOHEADER rcTarget указывал ширину изображения.

Для форматов RGB и YUV с битовой глубиной равной степени двойки и если отображающий фильтр не указывает шага, то ширина изображения устанавливается на границу DWORD.

biHeight
Высота растра в пикселях. Для не сжатого RGB формата, если величина
biHeight положительная, начало координат располагается в левом нижнем углу и растр увеличивается вверх. Если величина biHeight отрицательная, начало координат располагается в левом верхнем углу и растр увеличивается вниз. Для форматов YUV начало координат располагается в левом верхнем углу и растр всегда увеличивается вниз независимо от знака biHeight. Декодеры должны предлагать форматы YUV с положительной величиной biHeight, но для обратной совместимости они должны принимать форматы YUV с любым знаком biHeight.
Для сжатых форматов величина biHeight должна быть всегда положительной независимо от ориентации изображения.

biPlanes
Указывает количество планов для устройства потребителя. Эта величина должна установлена в единицу.

biBitCount
Указывает число бит на пиксель.

biCompression
Если растр сжат, то это код FOURCC, указывающий тип сжатого растра. Для не сжатых форматов возможны следующие величины.

Величина	Описание
BI_RGB	Не сжатый формат RGB.
BI_BITFIELDS	Не сжатый формат RGB с цветовой маской. Используется для 16-ти и 32 битных растров.

Если величина biCompression равна BI_BITFIELDS, растр использует цветовую маску из трех слов DWORD (красный, зеленый, синий). Для 16-ти разрядного растра при величине biCompression равной BI_RGB формат должен быть RGB 555, при величине BITFIELDS формат может быть RGB 555 или RGB 565.
Для несжатых форматов точно определить тип формата, используя эту величину, невозможно.
Для сжатых форматов или форматов YUV тип формата указывается четырех символьной величиной FOURCC. Например, формат YUYV представляется кодом 0x56595559. Подробней с кодами FOURCC можно ознакомиться в приложении.

biSizeImage
Указывает размер изображения в байтах. Может быть равным 0 для не сжатого RGB растра.

```
biXPelsPerMeter
Указывает горизонтальное разрешение в пикселях на метр растра
устройства потребителя.

biYPelsPerMeter
Указывает горизонтальное разрешение в пикселях на метр растра
устройства потребителя.

biClrUsed
Указывает номер индекса цвета в цветовой таблице, который ис-
пользуется растром.

biClrImportant
Указывает номер индекса цвета, который считается важным для
отображения растра. Если величина равна нулю, все цвета важны.
```

Примечание.
Для восьмибитовых и менее растров растр использует цветовую таблицу, которая следует сразу за структурой BITMAPINFOHEADER. Цветовая таблица представляет собой массив величин типа RGBQUAD. Размер массива определен в члене biClrUsed. Если он равен нулю, массив содержит максимальное число цветов для данной глубины цвета, что равно двум в степени biBitCount. Если имеется цветовая таблица или маски, можно использовать структуру BITMAPINFO для доступа к таблице. Структура определена следующим образом.

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER    bmiHeader;
    RGBQUAD             bmiColors[1];
} BITMAPINFO;
```

Если преобразовать структуру BITMAPINFOHEADER в BITMAPINFO, то член bmiColors укажет на первый вход таблицы или цветовую маску. Размер цветовой таблицы не соответствует величине sizeof(BITMAPINFOHEADER) или sizeof(BITMAPINFO) и требуется вычисление размера таблицы для каждого экземпляра.

Методы и структуры интерфейса IAMStreamConfig

Данный интерфейс позволяет устанавливать выходной формат на фильтре сбора и на сжимающих фильтрах, как для видео, так и для звуковых данных. Фильтры представляют этот интерфейс на своих выходных контактах. Установка формата на фильтре сбора производится при остановленном фильтре и отключенном выходном контакте. Если контакт подключен, и следующий фильтр может принять формат, то затем производится повторное соединение

фильтров, как указывалось ранее. Для некоторых фильтров сжатия может потребоваться, чтобы входной контакт фильтра сжатия был подключен. Подробности процесса установки формата будут рассмотрены далее.

Метод	Описание
GetFormat	Возвращает текущий или предустановленный формат.
GetNumberOfCapabilities	Возвращает количество форматов, которые этот контакт поддерживает.
GetStreamCaps	Возвращает свойства формата.
SetFormat	Устанавливает выходной формат на контакте.

Таблица 6. Интерфейс IAMStreamConfig.

Интерфейс IAMStreamConfig кроме структуры AM_MEDIA_TYPE оперирует еще с двумя структурами. Структура VIDEO_STREAM_CONFIG_CAPS описывает диапазон свойств видеоформата, а структура AUDIO_STREAM_CONFIG_CAPS описывает диапазон свойств звукового формата. Метод GetStreamCaps заполняет данными структуру AM_MEDIA_TYPE и одну из структур VIDEO_STREAM_CONFIG_CAPS или AUDIO_STREAM_CONFIG_CAPS . Количество возвращаемых структур определяется числом форматов, которые поддерживает фильтр, это число возвращает метод GetNumberOfCapabilities.

Структура VIDEO_STREAM_CONFIG_CAPS

Структура описывает следующие параметры изображения:

- пределы изменения размеров изображения.
- дискретность изменения размеров изображения.
- свойства растяжки и сжатия изображения.
- период следования кадров.
- скорость потока и другие параметры.

```
typedef struct _VIDEO_STREAM_CONFIG_CAPS {
    GUID guid;
    ULONG VideoStandard;
    SIZE InputSize;
    SIZE MinCroppingSize;
    SIZE MaxCroppingSize;
    int CropGranularityX;
    int CropGranularityY;
    int CropAlignX;
    int CropAlignY;
    SIZE MinOutputSize;
```

```
    SIZE MaxOutputSize;
    int OutputGranularityX;
    int OutputGranularityY;
    int StretchTapsX;
    int StretchTapsY;
    int ShrinkTapsX;
    int ShrinkTapsY;
    LONGLONG MinFrameInterval;
    LONGLONG MaxFrameInterval;
    LONG MinBitsPerSecond;
    LONG MaxBitsPerSecond;
} VIDEO_STREAM_CONFIG_CAPS;
```

guid
Указывает тип описываемого формата, например FORMAT_VideoInfo.

VideoStandard
Указывает поддерживаемые ТВ стандарты (PAL, SECAM, NTSC), определенные в перечислении AnalogVideoStandard или NULL.

InputSize
Размер входного видеосигнала. Для фильтра компрессора размер дается на входном контакте, для фильтра сбора это наибольший размер сигнала, который может оцифровать фильтр с точностью до пикселя.

MinCroppingSize
Наименьший разрешенный прямоугольник источника сигнала. Прямоугольник источника сигнала определен в структуре VIDEOINFOHEADER или VIDEOINFOHEADER2 (член rcSource).

MaxCroppingSize
Наибольший разрешенный прямоугольник источника сигнала.

CropGranularityX
Дискретность (зернистость) изменения размера прямоугольника сигнала по горизонтали. Диапазон размеров прямоугольника определен членами
MinCroppingSize и MaxCroppingSize.

CropGranularityY
Дискретность (зернистость) изменения размера прямоугольника сигнала по вертикали. Диапазон размеров прямоугольника определен членами
MinCroppingSize и MaxCroppingSize.

CropAlignX
Заданное горизонтальное выравнивание прямоугольника источника.

CropAlignY
Заданное вертикальное выравнивание прямоугольника источника.

MinOutputSize
Минимальный выходной размер.

MaxOutputSize
Максимальный выходной размер.

OutputGranularityX
Дискретность изменения размера ширины выходного прямоугольника. Значение величины лежит в диапазоне между значениями MinOutputSize и MaxOutputSize

OutputGranularityY

Дискретность изменения размера высоты выходного прямоугольника. Значение величины лежит в диапазоне между значениями MinOutputSize и MaxOutputSize

StretchTapsX
Указывает возможности фильтра по растяжке изображения для координаты X.

StretchTapsY
Указывает возможности фильтра по растяжке изображения для координаты Y.

ShrinkTapsX
Указывает возможности фильтра по сжатию изображения для координаты X.

ShrinkTapsY
Указывает возможности фильтра по сжатию изображения для координаты Y.

Предыдущие четыре члена структуры могут иметь следующие значения:

Величина	Описание
0	Не поддерживается сжатие растяжка изображения
1	Используется удвоение каждого пикселя при растяжке или исключением пикселей при сжатии.
2	Интерполяция по двум точкам.
3 и более	Интерполяция по трем и более точкам

MinFrameInterval
Минимальная длительность периода кадра в 100 наносекундных единицах. Используется только для фильтров сбора.

MaxFrameInterval
Максимальная длительность периода кадра в 100 наносекундных единицах. Используется только для фильтров сбора.

```
MinBitsPerSecond
Минимальная скорость передачи данных через контакт.

MaxBitsPerSecond
Максимальная скорость передачи данных через контакт.
```

Параметры структуры, влияющие на формирование размеров изображения

Предположим, что структура описывает параметры прямоугольника изображения для формата RGB 24 стандартов SECAM и PAL. В этом случае реальные значения параметров структуры могут быть следующие.

```
guid  {CLSID_Обработчик типа данных WDM Streaming Capture
                                        VideoInfoHeader}
VideoStandard         1046000
InputSize             {cx=768   cy=576}
MinCroppingSize       {cx=88    cy=72}
MaxCroppingSize       {cx=768   cy=576}
CropGranularityX      2
CropGranularityY      2
CropAlignX            2
CropAlignY            2
MinOutputSize         {cx = 96,   cy = 72}
MaxOutputSize         {cx = 768,  cy = 576}
OutputGranularityX    16
OutputGranularityY    4
StretchTapsX          0
StretchTapsY          0
ShrinkTapsX           2
ShrinkTapsY           2
MinFrameInterval      400000
MaxFrameInterval      400000
MinBitsPerSecond      178200
MaxBitsPerSecond      12441600
```

Рассмотрим, что означают и как участвуют в установке медиа формата следующие параметры:

- MinCroppingSize {cx=88, cy=72}
- MaxCroppingSize {cx=768, cy=576}
- CropGranularityX 2
- CropGranularityY 2
- CropAlignX 2
- CropAlignY 2

Эти параметры определяют правильные значения для прямоугольника источника сигнала rcSource, определенного в структуре VIDEOINFOHEADER или VIDEOINFOHEADER2. Источник сигнала это отсекающий прямоугольник, то есть выборка части изображения из входного сигнала. На рисунке 4.6 показано изменение размеров прямоугольника для источника сигнала rcSource.

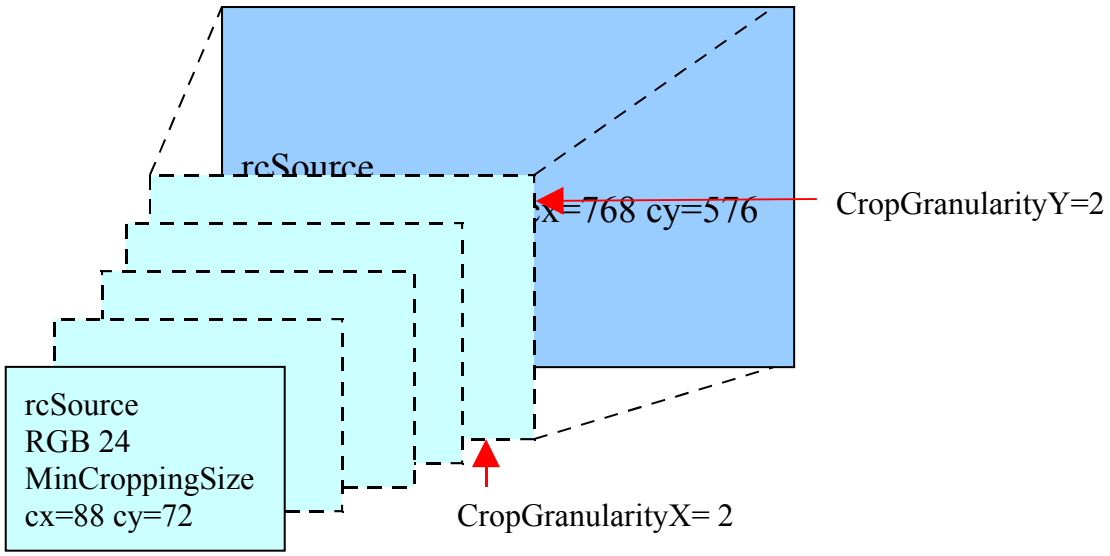

Рис.4. 5

Таким образом, источник может иметь размеры от минимального размера до максимального с дискретностью в два пикселя. Например, правильными будут следующие размеры:
88 x 72
90 x 72
90 x 74
92 x 74
92 x 76
…….
промежуточные размеры
……..
768 x 576

Далее на примере рассмотрим, каким образом фильтр производит вырезку источника из входного сигнала, растяжку (сжатие) и размещение результирующего изображения в выходном сигнале, а также участие в этом процессе остальных параметров структур. Принципы взаимодействия параметров rcSource и rcTarget рассматривались ранее.

Параметры CropAlignX и CropAlignY определяют дискретность установки координат источника rcSource во входном сигнале. На рисунке 4.7 показан пример источника сигнала и его координаты во входном сигнале.

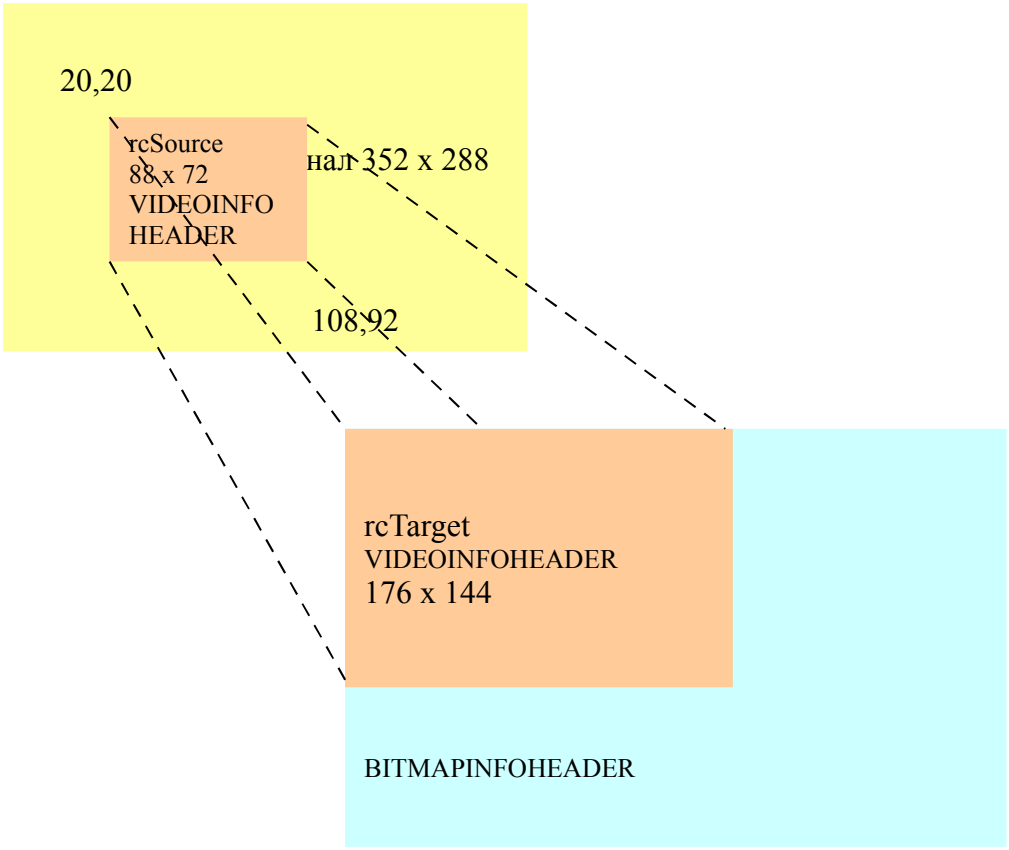

Рис.4. 6. Отображение входного сигнала на выходную плоскость.

Соответственно параметры MinOutputSize, MaxOutputSize, OutputGranularityX, и OutputGranularityY описывают параметры выходного прямоугольника (biHeight, biWidth) в структуре BITMAPINFOHEADER. Для данного сценария фильтр должен произвести растяжку источника и поместить результирующее изображение в выходном сигнале, как показано на рисунке, не изменяя остальную часть изображения. Если фильтр не поддерживает операции с источником сигнала, то соединение фильтра с не нулевыми значениями rcSource и rcTarget не произойдет и соответствующий метод возвратиться с ошибкой.

Фильтр сбора проверяемого устройства не поддерживает операции с rcSource и rcTarget в полном объеме. Некоторые методы не используют указанные параметры вообще, а в некоторых случаях растяжка изображения производится, но некорректно. Например, метод SetFormat не учитывает эти параметры, возвращается с S_OK, и после установки формата никаких изменений в изображении не происходит. Для этого конкретного устройства нельзя использовать операции с rcSource и rcTarget.

Параметры MinFrameInterval и MaxFrameInterval для фильтра сбора определяют период следования кадров также как и член структуры VIDEOINFOHEADER (VIDEOINFOHEADER2) AvgTimePerFrame. Фильтр может не поддерживать все промежуточные значения в указанном диапазоне. В таком случае метод IAMStreamConfig::SetFormat установит ближайшее поддерживаемое значение периода следования кадров.

Метод IAMStreamConfig::GetFormat и его применение

Метод возвращает текущий или привилегированный выходной формат. Фильтр устанавливается в граф с привилегированным форматом, например, фильтр сбора устанавливается с форматом YUY2.

```
HRESULT GetFormat(
  AM_MEDIA_TYPE **pmt
);

Параметры

pmt[out]
Адрес указателя на структуру AM_MEDIA_TYPE.

Коды возврата.
```

Код	Описание
S_OK	Успешное завершение.
E_OUTOFMEMORY	Недостаточно памяти.
E_POINTER	Указатель равен NULL .
VFW_E_NOT_CONNECTED	Вход фильтра не подключен.

Примечание.

При подключенном контакте метод возвращает текущий формат. В противном случае возвращается привилегированный формат, который будет установлен при подключении контакта. Если перед вызовом GetFormat не было вызова IAMStreamConfig::SetFormat, метод возвратит первый привилегированный формат из списка привилегированных форматов, как определено для метода IPin::EnumMediaTypes. На некоторых фильтрах сжатия метод может возвращаться с ошибкой, если входной контакт не подключен.

Метод распределяет память для структуры AM_MEDIA_TYPE и возвращает указатель на заполненную структуру. После того, как работа со структурой будет завершена необходимо освободить память, включая блок формата.

Это может представлять определенное неудобство, так как при запросе медиа формата из другого класса, необходимо всякий раз производить вызов метода для освобождения памяти. Освобождение памяти для структуры произ-

водится в два приема, сначала освобождается блок формата, а затем вся структура. Можно поступить другим образом, сделав указатель на структуру AM_MEDIA_TYPE членом класса CCapture, и использовать этот указатель при возврате текущего медиа формата для определенного контакта. Все операции с использованием структуры AM_MEDIA_TYPE тогда будут производиться с этим указателем внутри класса. Освобождение памяти при этом производиться также внутри класса CCapture. При возврате медиа типа указатель проверяется на нуль и если он не нулевой, то производится освобождение памяти и возврат новых данных. После освобождения памяти указатель обнуляется. Последняя проверка указателя и освобождение памяти производится в деструкторе. Таким образом, все заботы об указателе на структуру скрываются внутри класса.

Документация DirectShow рекомендует для освобождения структуры использовать метод из библиотеки базовых классов. Поскольку библиотека базовых классов не используется, то для освобождения памяти следует написать отдельный метод в классе CCapture, просто позаимствовав его из библиотеки классов.

```
void CCapture::DeleteMediaTypeAndFormat(AM_MEDIA_TYPE * pmt)
{
    if (pmt == NULL)   return;

    //Освободить формат
    if (pmt->cbFormat != 0)
    {
        CoTaskMemFree((PVOID)pmt->pbFormat);

        //Ненужно, но на всякий случай
        pmt->cbFormat = 0;
        pmt->pbFormat = NULL;
    }

    //Освободить интерфейс
    if (pmt->pUnk != NULL)
    {
        pmt->pUnk->Release();
        pmt->pUnk = NULL;
    }

    //Освободить медиа формат
    CoTaskMemFree((PVOID)pmt);
}
```

Примечание.
Поскольку выше были упомянуты базовые классы DirectShow, то следует сказать, по какой причине они не используются. Первая причина состоит в том, что применение базовых классов требует хорошего знания основ функционирования DirectShow. Вторая причина состоит в том, что некоторые базовые

классы могут быть недостаточно проработаны, и ввести в заблуждение, без хороших знаний основ DirectShow. Так случилось, например, с библиотечным классом переключателя CCrossbar, который в предыдущей версии DirectShow 8.1 имел серьезные ошибки в реализации и был не пригоден для применения. В следующей версии он был переработан.

Для возврата текущего медиа формата на определенном контакте фильтра сбора напишем метод, который будет заполнять структуру и возвращать указатель член класса.

```
AM_MEDIA_TYPE * CCapture::GetMediaFmt(GUID  gPin)
{
  CComPtr<IAMStreamConfig> pSC;

  string sErr = "Ошибка CCapture::GetMediaFmt ";

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

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

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

  return m_pMediaType;
}
```

Метод IAMStreamConfig::GetNumberOfCapabilities

Метод возвращает количество форматных свойств, поддерживаемое контактом. Он вызывается перед вызовом IAMStreamConfig::GetStreamCaps.

```
HRESULT GetNumberOfCapabilities(
  int *piCount,
  int *piSize
);

Параметры.
```

piCount
[out]
Указатель на переменную, принимающую число форматных свойств.

piSize
[out]
Указатель на переменную, принимающую размер структуры в байтах. Поскольку метод IAMStreamConfig::GetStreamCaps может возвращать одну из двух структур VIDEO_STREAM_CONFIG_CAPS или AUDIO_STREAM_CONFIG_CAPS, то параметр принимает размер одной из этих структур.

Возвращаемые коды.

Код	Описание
S_OK	Успешное выполнение.
E_POINTER	Указатель равен NULL.
VFW_E_NOT_CONNECTED	Входной контакт не подключен.

На некоторых фильтрах сжатия метод может возвращаться с ошибкой, если входной контакт не подключен.

Метод IAMStreamConfig::GetStreamCaps и его применение

Метод возвращает форматные свойства, которые поддерживаются фильтром на определенном контакте. Он возвращает структуру AM_MEDIA_TYPE и одну из двух структур, VIDEO_STREAM_CONFIG_CAPS для форматов видеоизображения или AUDIO_STREAM_CONFIG_CAPS для форматов звука.

```
HRESULT GetStreamCaps(
  int iIndex,
  AM_MEDIA_TYPE **pmt,
  BYTE *pSCC
);
```

Параметры.

iIndex
[in]
Индекс форматных свойств из множества форматных свойств поддерживаемых контактом. Общее число форматных свойств указывает метод IAMStreamConfig::GetNumberOfCapabilities.

pmt[out]
Адрес указателя на структуру AM_MEDIA_TYPE. Метод заполняет структуру данными.

pSCC[out]
Указатель на массив байт. Массив может быть структурой VIDEO_STREAM_CONFIG_CAPS или AUDIO_STREAM_CONFIG_CAPS. Распределение памяти производится до вызова метода, размер определяется при вызове метода GetNumberOfCapabilities.

Возвращаемые коды.

Код	Описание
S_FALSE	Указанный индекс вышел за предел.
S_OK	Успешное окончание.
E_INVALIDARG	Неправильный индекс.
E_OUTOFMEMORY	Недостаточно памяти.
E_POINTER	Нулевое значение указателя.
VFW_E_NOT_CONNECTED	Вход фильтра не подключен.

Данные, полученные с помощью этого метода, необходимо сохранить с тем, чтобы в дальнейшем использовать их для выбора и установки нового формата. Обычно основная часть данных формата остается неизменной. Пользователю для текущей работы достаточно иметь возможность изменять размер изображения, частоту кадров и цветовой формат. В простых случаях нет нужды и в изменении цветового формата, достаточно использовать привилегированный формат. Изменение частоты кадров может быть полезно для систем документирования информации, когда важен конечный размер записываемого файла. От выбора числа используемых параметров зависит способ организации данных. Для целей изучения и проведения экспериментов, желательно сохранить максимум полученной информации, с тем, чтобы не заниматься затем корректировкой способа хранения данных. Данные лучше всего организовать в класс, подобно тому, как делалось ранее.

Метод IAMStreamConfig::SetFormat

Метод устанавливает формат на выходном контакте.

```
HRESULT SetFormat(
  AM_MEDIA_TYPE *pmt
);
```

Параметры.

pmt
[in]
Указатель на структуру AM_MEDIA_TYPE, содержащую новый формат.

Возвращаемые коды.

Код	Описание
S_OK	Успешное окончание.
E_OUTOFMEMORY	Недостаточно памяти.
E_POINTER	Нулевой указатель.
VFW_E_INVALIDMEDIATYPE	Медиа формат неправильный.
VFW_E_NOT_CONNECTED	Входной контакт не подключен.
VFW_E_NOT_STOPPED	Невозможно установить тип, фильтр не остановлен.
VFW_E_WRONG_STATE	Невозможно установить тип, фильтр не остановлен.

Примечание.

Если выходной контакт не подключен, медиа формат устанавливается при следующем подключении. Если контакт подключен, производится попытка произвести повторное соединение с новым медиа форматом. Метод возвращает ошибку, если другой контакт не принимает новый медиа формат.
На практике это выглядит следующим образом. При соединенных контактах новый медиа тип устанавливается при условии, что цветовой формат не изменился. Если цветовой формат изменяется, метод может возвратить VFW_E_INVALIDMEDIATYPE, но может и произвести установку нового формата. Для принятия решения на отключение фильтра сбора следует анализировать код VFW_E_INVALIDMEDIATYPE.

Для большинства фильтров метод возвращает ошибку, если фильтр запущен или в состоянии паузы. Для некоторых фильтров сжатия метод возвращает ошибку, если входной контакт фильтра не подключен.

Класс данных медиа формата

Поскольку медиа формат для видеоданных описывается сложными структурами и содержит большое количество данных, лучше всего создать класс данных, для его описания. Назовем класс CMediaFmtV и поместим его определение в файле Capture.h. Структура AM_MEDIA_TYPE содержит избыточные данные, поэтому в классе сохраняем только данные необходимые для правильного формирования структуры и дополнительно символьное имя цветового формата.

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

```
//Символьное имя цветового формата
string    m_nameColorFmt;

//Данные из структуры AM_MEDIA_TYPE
GUID* m_pSubType;
BOOL  m_bFixedSizeSamples;
BOOL  m_bTemporalCompression;

//Структуры
VIDEOINFOHEADER       m_vih;
VIDEO_STREAM_CONFIG_CAPS    m_vscc;

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

  //Оператор присваивания
  const CMediaFmtV & operator =
                      (const CMediaFmtV & mediaFmt)
  {
    if(this != &mediaFmt)
    {
      m_nameColorFmt = mediaFmt.m_nameColorFmt;
      m_pSubType = mediaFmt.m_pSubType;
      m_bFixedSizeSamples =
                          mediaFmt.m_bFixedSizeSamples;
      m_bTemporalCompression =
                       mediaFmt.m_bTemporalCompression;
      m_vscc = mediaFmt.m_vscc;
      m_vih = mediaFmt.m_vih;
    }
    return *this;
  };


  //Оператор <
  //(сортировка по умолчанию по цветовому формату)
  BOOL operator < (const CMediaFmtV & mediaFmt) const
  {
    return m_pSubType < mediaFmt.m_pSubType;
  };

  BOOL operator == (const CMediaFmtV& mediaFmt) const
  {
    return m_pSubType == mediaFmt.m_pSubType;
  };
};
```

В класс также включена структура VIDEOINFOHEADER и структура VIDEO_STREAM_CONFIG_CAPS, возвращаемая методом IAMStreamConfig::GetStreamCaps. Этих данных достаточно для формирования структуры AM_MEDIA_TYPE с измененными данными для установки

нового формата. Если фильтр сбора использует структуру VIDEOINFOHEADER2, то следует произвести простую корректировку класса. Можно изначально использовать структуру VIDEOINFOHEADER2, поскольку она является расширением VIDEOINFOHEADER и легко преобразуется в нее. Фильтры VMR осуществляют прямую поддержку структуры VIDEOINFOHEADER2. При необходимости можно написать метод, преобразующий одну структуру в другую. Класс содержит только два перегруженных оператора, которых достаточно для большинства алгоритмов библиотеки STL.

Определение свойств форматов на выходном контакте фильтра сбора

После ознакомления с методами интерфейса IAMStreamConfig можно написать метод для определения свойств поддерживаемых форматов на выходном контакте фильтра сбора. Свойства форматов будем сохранять в векторе vector<CMediaFmtV> m_vMediaTVideo, что позволит иметь оперативный доступ к данным форматов. Для уменьшения объема данных будем производить отбор только тех медиа форматов, которые применимы для текущего ТВ стандарта, например для SECAM.
Первый параметр метода это GUID выходного контакта, а второй ссылка на вектор данных.

```
BOOL CCapture::GetVideoCaps(GUID gPin
                      , vector<CMediaFmtV>& vMediaTVideo)
{
  BOOL bret =TRUE;
  HRESULT hr = S_OK;
  CComPtr<IAMStreamConfig> pSC;
  AM_MEDIA_TYPE *pmt = 0;
  VIDEO_STREAM_CONFIG_CAPS vscc;
  int count =0; int size = 0;

  string sErr = "Ошибка CCapture:: GetVideoCaps ";

  //Возвращаем текущий ТВ стандарт из видеодекодера
  LONG standardTV = GetTVStandard();

  try
  {
  //Найти IAMStreamConfig для выходного контакта
  //фильтра захвата
    if(S_OK != (hr = m_pBuilder->FindInterface(&gPin
            ,&MEDIATYPE_Video,m_pCapV,IID_IAMStreamConfig
            ,reinterpret_cast<void**>(&pSC))))
      throw hr;

  //Определяем общее количество свойств форматов
```

```
if(S_OK != (hr =
        pSC->GetNumberOfCapabilities(&count, &size)))
  throw hr;

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

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

  //Заполняем список доступных медиа форматов
  //для текущего стандарта TV
  if (pmt->formattype == FORMAT_VideoInfo
                && (vscc.VideoStandard & standardTV))
  {
    //Заполняем данные
    CMediaFmtV mfmt;
    mfmt.m_pSubType = &(pmt->subtype);

    //Вернуть символьное имя формата
    mfmt.m_nameColorFmt =
                GetNameFromSubtype(mfmt.m_pSubType);

    mfmt.m_bFixedSizeSamples =
                  pmt->bFixedSizeSamples;
    mfmt.m_bTemporalCompression =
                  pmt->bTemporalCompression;

    //Копировать структуру VIDEO_STREAM_CONFIG_CAPS
    mfmt.m_vscc = vscc;

    //Проверить наличие блока формата
    if (pmt->cbFormat >= sizeof(VIDEOINFOHEADER))
    {
      //Копировать структуру VIDEOINFOHEADER
      mfmt.m_vih =
      *(reinterpret_cast<VIDEOINFOHEADER*>
                                    (pmt->pbFormat));
    }
    else

    ZeroMemory(&mfmt.m_vih,sizeof(VIDEOINFOHEADER));

    //Запись в вектор
    vMediaTVideo.push_back(mfmt);
  }
}//for
```

```
    //Установить текущий формат в начало вектора
    if(GetMediaFmt(gPin)!= NULL)
    {
      CMediaFmtV mfmt;

      //Константа поиска
      mfmt.m_pSubType = &m_pMediaType->subtype;

      rotate(vMediaTVideo.begin()
              ,find(vMediaTVideo.begin()
              ,vMediaTVideo.end(),mfmt)
              ,vMediaTVideo.end());
    }

  }
  catch(HRESULT h)
  {
    bret= 0;
    CTVshowApp::myMessageError(h,sErr);
  }

  DeleteMediaTypeAndFormat(pmt);
  return bret;
}
```

Метод GetNameFromSubtype определен в файле Constants.h и возвращает имя формата. Для форматов YUV символьное имя можно получить из структуры в виде кода FOURCC, как упоминалось ранее в описании этой структуры. Для других форматов коды неопределенны и поэтому было принято решение определить символьные имена всех форматов в отдельной структуре.

Метод для установки медиа формата

Метод CCapture::SetMediaFmtPreview будет производить установку медиа формата, изменяя три параметра.

- Цветовой формат.
- Размер изображения.
- Частоту кадров.

Первое действие в методе это остановка фильтра, если он был в запущенном состоянии или состоянии паузы. Затем производится поиск контакта, на котором устанавливается медиа формат и поиск управляющего интерфейса IAMStreamConfig для фильтра сбора.

Если формат устанавливается на фильтре сбора, который не имеет отдельного контакта предварительного просмотра, то в графе присутствует фильтр SmartTee. В такой ситуации без экспериментальной проверки нельзя точно определить, какой контакт следует отключать при установке формата. То ли

это будет контакт Capture на фильтре сбора, или достаточно отключить контакт Preview на фильтре SmartTee, оставив соединенными контакты фильтров сбора и SmartTee. Можно перестраховаться и всегда отключать контакт Capture на фильтре сбора и удалять SmartTee, который затем будет повторно устанавливаться. Если использовать для перестройки метод RenderStream, то можно не удалять из графа SmartTee, указав его в предпоследнем параметре, что должно повысить скорость работы метода. Поскольку данная ситуация оставляет место сомнениям и не может быть проверена, то она оставлена без обработки.

Далее проверяется, подключен обрабатываемый контакт к следующему фильтру или нет. Ситуацию фиксируем булевской переменной.

Для установки медиа формата формируется структуру AM_MEDIA_TYPE. Для этого сначала отыскиваются в списке свойств форматов фильтра те свойства формата, которые соответствуют устанавливаемому цветовому формату. В структуру AM_MEDIA_TYPE копируются все свойства, кроме размеров изображения, цветового формата и частоты кадров, которые берутся из входных параметров.

Если контакт отключен, метод IAMStreamConfig::SetFormat должен установить формат и успешно завершиться. В случае соединенных контактов возможны несколько вариантов установки формата.

Первый вариант. Удалить часть графа от фильтра сбора и до конца графа, включая фильтр отображения, установить формат и достроить новый участок графа. Может возникнуть вопрос, а почему просто не произвести разъединение, установку формата и повторное соединение без удаления фильтров? Ответ состоит в том, что при установке некоторых форматов, в зависимости от типа отображающего фильтра могут устанавливаться промежуточные трансформирующие фильтры. Большое количество факторов, которые необходимо учитывать при этом, приводят к решению, что удалить участок графа и построить новый проще. Если бы отображающий фильтр подпадал под классификацию интерфейса IFilterChain, то удаление участка графа можно было бы сделать очень просто. Например, таким образом.

```
//Запросить данные на фильтр, подключенный фильтру
//сбора
PIN_INFO info;
 if(S_OK != (hr = pConnect->QueryPinInfo(&info)))
      throw hr;

//Удалить все фильтры начиная с фильтра не
//принимающего формат
IFilterChain* pChain = 0;
if(S_OK != (hr = m_pGraph->QueryInterface(&pChain)))
    throw hr;

//Удаляет часть графа начиная с указанного фильтра и
//до конечного(pFilterEnd). Если pFilterEnd=NULL, то
удаляются //все фильтры графа начиная с указанного.
```

```
 if(S_OK != (hr = pChain->RemoveChain(info.pFilter,0)))
      throw hr;
```

Интересно, что этот вариант работает, не взирая на то, что отображающий фильтр не подпадает под классификацию фильтров интерфейса IFilterChain, поскольку имеет несколько входных контактов и не имеет выходного контакта. Поэтому его нельзя считать, в данном случае, надежным способом удаления части графа.

Если часть графа удалена этим или иным способом, то затем следует установить формат на фильтре сбора и достроить граф. При этом каждый раз будут удаляться все фильтры, как промежуточные, так и фильтр отображения. Удаление фильтра отображения VMR потребует его повторной инициализации, что вызывает дополнительные затраты. Чтобы не удалять фильтр отображения, следует определить предшествующий ему фильтр, который будет удаляться последним и это не должен быть фильтр сбора. Тогда можно надежно использовать интерфейс IFilterChain. Определение фильтра предшествующего фильтру отображения сразу вызывает усложнение кода, сводя на нет преимущества от использования интерфейса IFilterChain.

Второй вариант. Этот вариант предполагает разъединение фильтров, начиная с контакта фильтра сбора и до входного контакта фильтра отображения включительно. Затем следует установка формата и достройка графа. При этом промежуточные фильтры остаются в графе, и при последующих сменах формата их установка должна происходить быстрее. Однако промежуточные фильтры могут остаться в графе неподключенными до конца работы приложения, или же они могут включаться в цепочку тогда, когда они не нужны для обработки. И такое происходит.

В методе при подключенном фильтре сбора не делается попытка установить новый формат, хотя это возможно в некоторых ситуациях. Например, при переходе от формата YVU9 к формату YUY2 установка происходит правильно. При этом из графа не удаляются подключенные промежуточные фильтры, они подключаются также как и для формата YVU9, хотя это совершенно не нужно, например, при использовании фильтра VMR. Таким образом, возникает проблема поиска ненужных фильтров и их удаления, что является дополнительным усложнением кода. Если сразу не удалять промежуточные фильтры, то большое число трудно учитываемых факторов может привести к неустойчивой установке формата и сбоям в программе.

Учитывая все сказанное, в данном методе при смене формата при подключенном контакте фильтра сбора будем удалять все промежуточные фильтры без удаления фильтра отображения и затем достраивать граф с прежним фильтром отображения.

```
BOOL CCapture::SetMediaFmtPreview(GUID* psubtype
                ,DWORD_PTR dwSize,LONGLONG frameInterval)
{
```

```
HRESULT hr = S_OK;

BOOL bConnect = FALSE;
BOOL bSmart = FALSE;

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

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

string sErr="SetMediaFmtPreview";


//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_PREVIEW,0,0,0,&pPinOut))
  return FALSE;

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

if(m_bSmartTee)
{
  //здесь следует обработать ситуацию при установленном фильтре
  //SmartTee.
  //……
}

//Проверка подключения контакта
if(S_OK == pPinOut->ConnectedTo(&pConnect))
  bConnect = TRUE;//Контакт подключен

//Найти устанавливаемый медиа тип в векторе доступных  //типов
```

```
CMediaFmtV mfmt;
mfmt.m_pSubType = psubtype;

if(!FindStreamCaps(PIN_CATEGORY_PREVIEW,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, и 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;


if(!bConnect)
{
  //установка нового формата при отключенном контакте
  //фильтра
  if(S_OK != pConfig->SetFormat(&amt))
    return FALSE;
  else
    return TRUE;
}
```

```
//Запросить данные на фильтр, подключенный к фильтру захвата
PIN_INFO info;
if(S_OK != pConnect->QueryPinInfo(&info))
  return FALSE;

//Отключить выходной контакт и входной контакт
//следующего фильтра. Вызов Disconnect отключает только
один //контакт, полностью соединение разрывается при отключе-
нии
//и сопряженного контакта


if(S_OK != m_pGraph2->Disconnect(pPinOut))
  return FALSE;

if(S_OK != m_pGraph2->Disconnect(pConnect))
  return FALSE;


pConnect = NULL;
pPinOut = NULL;

//Удаляем все промежуточные фильтры вниз по потоку
while(info.pFilter != m_pVRender)
{
  //Это не Renderer фильтр. Получаем указатель на
  //промежуточный фильтр
  CComPtr<IBaseFilter> pF(info.pFilter);
  info.pFilter->Release();

  //Ищем его выходной контакт
  if(S_OK != m_pBuilder->FindPin(pF,PINDIR_OUTPUT
                        ,NULL,NULL,FALSE,0,&pPinOut))
    return FALSE;

  //Проверяем, к какому фильтру он подключен
  if(S_OK == pPinOut->ConnectedTo(&pConnect))
  {
    //Контакт подключен, получить данные на входной
    //контакт следующего фильтра
    if(S_OK != pConnect->QueryPinInfo(&info))
      return FALSE;

    //Удаляем промежуточный фильтр
    if(S_OK != m_pGraph2->RemoveFilter(pF))
      return FALSE;

    pF = NULL;
  }
  pConnect = NULL;
  pPinOut = NULL;
```

```
  }//while

  m_bPreview = FALSE; //Граф не достроен

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

  //Достройка графа от контакта Preview
  if(S_OK != m_pBuilder->RenderStream(
              &PIN_CATEGORY_PREVIEW
              ,NULL,m_pCapV,NULL,m_pVRender))
    return FALSE;

  m_bPreview = TRUE; //Граф достроен

  return TRUE;
}
```

Вызов метода производится в обработчике CFView1::OnBnClickedButSet-format.

Замечание.
Для режима предварительного просмотра изменение цветового формата излишне, достаточно использовать формат YUY2, устанавливаемый на фильтре сбора по умолчанию. Установка цветового формата важна при записи файла, когда нужно уменьшить объем записываемой информации. В некоторых приложениях также можно значительно снизить частоту кадров при записи особенно это касается приложений для систем наблюдения. Если установка цветового формата не нужна, то для управления размерами изображения можно воспользоваться интерфейсом IBasicVideo, который предоставляется менеджером графа. Тогда управлять структурой AM_MEDIA_TYPE вообще не нужно.

Интерфейс пользователя для отображения медиа форматов

Для изменения медиа формата на выходном контакте предварительного просмотра фильтра сбора выберем несколько основных параметров.

- Цветовой формат.
- Размер изображения.
- Частоту кадров.

Следует отметить, что изменение частоты кадров для режима предварительного просмотра не имеет значения и не изменяется. Этот параметр важен при записи файла. Если производительность системы позволяет производить

запись файла с установленной частотой кадров, то предварительный просмотр осуществляется по остаточному принципу. То есть частота кадров предварительного просмотра может быть в пределах от номинальной для данного стандарта ТВ до меньшей величины при низкой производительности системы.

Для установки цветового формата и размера изображения на второй вкладке (класс CFView1) установлены два элемента управления CComboBox, для установки частоты кадров установлен элемент CEdit. Ввод данных производится при нажатии кнопки и обрабатывается в обработчике CFView1::OnBnClickedButSetformat. На рисунке показан вид вкладки с элементами управления.

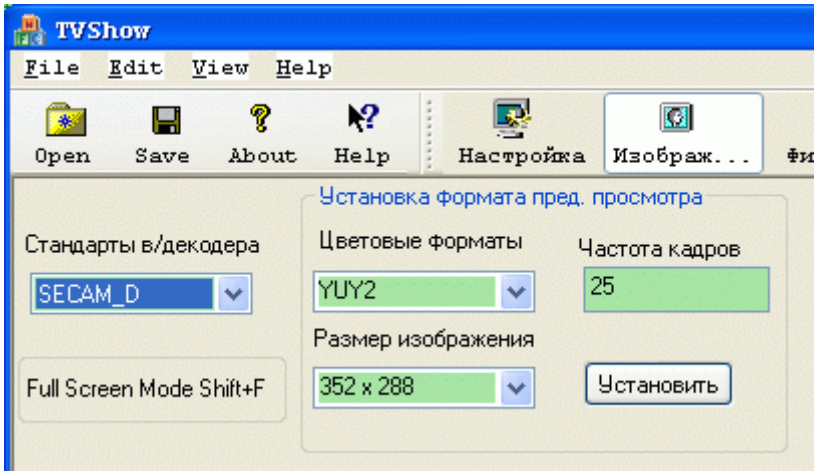

Рис.4. 7 Управление медиа форматом фильтра сбора

Инициализация элементов управления выполняется в обработчике CFView1::OnMyMessageEndInit. После заполнения списка доступных цветовых форматов производится отображение текущего цветового формата фильтра и для него заполняется список доступных размеров изображения, а также отображается стандартное значение частоты кадров. Отображение текущего формата на фильтре производится после нажатия кнопки установки формата. Начиная с данного момента, все проверки будут проводиться только для фильтра VMR9 в безоконном режиме. Описание инициализации фильтра VMR и установка безоконного режима приводится далее в разделе "Безоконный режим фильтров VMR".

В методе CFView1::OnMyMessageEndInit также производится установка медиа формата. Это необходимо потому, что по умолчанию фильтр сбора устанавливает размер изображения для стандарта NTSC (320x240), несмотря на то, что установлен стандарт SECAM (352x288).

Метод CCapture::FindStreamCaps производит поиск свойств формата для контакта по указателю на подтип формата. Указатель на подтип формата должен быть установлен во втором параметре, как константа для поиска, перед

вызовом метода. Метод заполняет свойствами формата объект класса данных, передаваемый по ссылке во втором параметре.

```
BOOL CCapture::FindStreamCaps(GUID gPin
                        , CMediaFmtV& mfmt)
{
  //Список всех видеосвойств
  vector<CMediaFmtV> vmfmt;
  GetStreamCaps(gPin,vmfmt);

  //Найти данные на новый цветовой формат
  vector<CMediaFmtV>::iterator pos;
  pos = find(vmfmt.begin(),vmfmt.end(),mfmt);

  if(pos == vmfmt.end())
      return FALSE;

  //Возврат данных
  mfmt = *pos;

  return TRUE;
}
```

Метод CFView1::myFillListSizePicture заполняет список допустимых для установки размеров изображения. Пользователь может выбрать любой размер для изменения исходного изображения. Поскольку дискретность изменения размера изображения очень небольшая, список допустимых размеров может быть чрезмерно велик и его следует ограничить, используя подходящий критерий. Например, в данном методе отбираются те размеры изображения, соотношения сторон, которых соответствует соотношению сторон максимально допустимого прямоугольника изображения (3x4). В простых вариантах изменения размеров изображения можно задать несколько типовых размеров, выбрав из множества допустимых, или произведя их расчет. При необходимости, можно даже установить ползунки для изменения каждой координаты прямоугольника изображения.

Выбранный шаг изменения размера изображения в 64 пикселя удовлетворяет параметрам всех цветовых форматов. Список размеров изображения будет одинаков для всех форматов, но, для страховки, неплохо пересчитать список при смене цветового формата. Это сделано в обработчике CFView1::OnCbnSelchangeComboColorformat подобно тому, как делается при установке формата.

Установка нового формата производится в обработчике CFView1::OnBnClickedButSetformat, который обрабатывает нажатие кнопки установки формата. В обработчике сначала производится сбор данных из элементов управления, затем установка нового формата.

Метод CCapture::ResizeWndParentPreview производит регулировку размеров окна приложения таким образом, что бы размер клиентской области окна соответствовал установленному размеру изображения. Этот метод и некоторые упомянутые выше не используют никаких средств DirectShow и могут быть просмотрен в архиве проектов. После корректировки размеров окна вызывается метод CCapture::SetMediaFmtPreview для установки медиа формата и затем производится запуск режима предварительного просмотра.

Полнооконный режим отображения

Полноэкранный режим отображения обеспечивает интерфейс IVideoWindow. В этом режиме может работать фильтр Video Renderer и фильтр VMR, когда он находится в совмещенном режиме. Переключение в полноэкранный режим и обратно обеспечивает метод IVideoWindow::put_FullScreenMode.

```
HRESULT put_FullScreenMode(
    long FullScreenMode
);
```

Параметры.

FullScreenMode
[in]
Указывает, какой режим установить.

Величина	Описание
OATRUE	Полноэкранный режим
OAFALSE	Отмена полноэкранного режима (по умолчанию).

Коды возврата.

Величина	Описание
E_NOTIMPL	Фильтр не поддерживает полноэкранный режим.
S_FALSE	Режим уже установлен.
S_OK	Успешное выполнение.
VFW_E_NO_FULLSCREEN	Нет ни одного фильтра поддерживающего полноэкранный режим.

Полноэкранный режим может быть невидим до тех пор, пока граф не будет запущен или установлен в паузу. При снятии полноэкранного режима графический менеджер посылает сообщение EC_FULLSCREEN_LOST.

Напишем метод CCapture::SetFullScreenMode в предположении, что видеокарта поддерживает полноэкранный режим. Все современные карты его поддерживают, если же старая видеокарта не поддерживает полноэкранный режим, метод вернется с E_NOTIMPL. Тогда необходимо вручную установить в граф фильтр полноэкранного режима Full Screen Renderer, который в настоящее время не рекомендуется применять.

```
void CCapture::SetFullScreenMode()
{
  if(!m_bPreview) return;

  HRESULT hr = S_OK;
  string sErr = "Error SetFullScreenMode";

  IVideoWindow* pVW=0;
  try
  {
    //Получить указатель на интерфейс
    if(S_OK != (hr = m_pGraph->QueryInterface(&pVW)))
      throw hr;

    //Опросить состояние
    LONG fullScreen;
    if(FAILED(hr =
              pVW->get_FullScreenMode(&fullScreen)))
      throw hr;

    //Переключить режим
    if(hr == S_OK)
    {
      if(FAILED(hr =
              pVW->put_FullScreenMode(~fullScreen)))
        throw hr;
    }

  }
  catch(HRESULT hr)
  {
    CTVshowApp::MessageError(hr,sErr);
  }

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

Вызов метода осуществим в обработчике горячей (hot key) клавиши. Реализовать горячую клавишу можно с помощью класса MFC CHotKeyCtrl либо с помощью функции Win32 API RegisterHotKey. Второй способ проще. Для

его реализации в классе окна (CFView1), которое будет обрабатывать сообщение, объявляем обработчик.

```
  afx_msg LRESULT OnHotKey(WPARAM,LPARAM);
```

В карте сообщений добавляем соответствующий вход для сообщения WM_HOTKEY. Делаем все вручную, поскольку соответствующего сообщения в списке для автоматической установки нет.

```
  ON_MESSAGE(WM_HOTKEY,OnHotKey)
```

В обработчике CFView1::OnInitialUpdate регистрируем сообщение для своего окна. Горячую клавишу назначим Shift+F.

```
void CFView1::OnInitialUpdate()
{
  CFormView::OnInitialUpdate();

  //Регистрация горячей клавиши
  ::RegisterHotKey(
        this->m_hWnd,m_hotKeyID,MOD_SHIFT,m_hotKey);

}
```

При закрытии приложения горячую клавишу следует снять с регистрации. Это действие выполним в обработчике CFView1::OnDestroy(), когда окно будет разрушаться.

```
void CFView1::OnDestroy()
{
  //Удаление регистрации горячей клавиши
  ::UnregisterHotKey(this->m_hWnd,m_hotKeyID);

  CFormView::OnDestroy();
}
```

И, наконец, произведем вызов метода CCapture::SetFullScreenMode() в обработчике события WM_HOTKEY.

```
LRESULT CFView1::OnHotKey(WPARAM wp,LPARAM lp)
{
  CCapture* pCap = myGetPtrCap();

  //Переключить режим
  pCap->SetFullScreenMode();
  return 1;
}
```

Далее проверяем выполнение. Результат положительный, остается подсказать каким-либо образом пользователю об этой возможности. Например, просто текстом на вкладке. Полноэкранный режим для фильтра VMR, работающего в безоконном режиме, устанавливается иначе.

Безоконный режим отображения фильтров VMR

Безоконный режим фильтров VMR является наиболее предпочтительным способом их использования. В безоконном режиме фильтр VMR не загружает компонент менеджера окна (Window Manager) и таким образом не поддерживает интерфейсы управления окном IVideoWindow и IBasicVideo. Вместо этого используется окно приложения для установки выходного прямоугольника, в котором отображается видеопоток. Фильтр VMR не использует субклассирование окна или установку системной ловушки (hook). Он использует отсекающий объект DirectDraw (Direct3D) для расположения выходного прямоугольника в клиентской области окна приложения. В оставшейся части клиентской области возможно рисование произвольной картинки.

В безоконном режиме фильтр требует конфигурирования после его создания и установки в граф, но перед подключением к фильтру выше по потоку. Для этого необходимо выполнить следующие действия.

- Вызвать метод IVMRFilterConfig::SetRenderingMode для установки флага безоконного режима VMRMode_Windowless.
- При необходимости конфигурировать смеситель для работы с несколькими потоками, следует вызвать метод IVMRFilterConfig::SetNumberOfStreams. Для установки Z-порядка входных потоков и других параметров потоков используется интерфейс IVMRMixerControl. Если смеситель не конфигурируется, то по умолчанию используется только один входной контакт.
- Вызвать метод IVMRWindowlessControl::SetVideoClippingWindow для указания окна, в котором будет отображаться видеопоток.
- Подключить фильтр.
- Установить позицию изображения в окне приложения, вызвав метод IVMRWindowlessControl::SetVideoPosition.

При воспроизведении изображения, фильтр должен уведомляться о некоторых сообщениях Windows. К таким сообщениям относится WM_PAINT. Приложение должно иметь обработчик этого сообщения в классе окна, в котором располагается изображение, создаваемое фильтром. В обработчике вызывается метод IVMRWindowlessControl::RepaintVideo для выполнения перерисовки. Безоконный режим позволяет производить операции рисования в окне, совмещая их с отображением видеопотока. Рисование следует производить в той части окна, которая не занята отображением видеопотока, в про-

тивном случае фильтр наложит свое изображение поверх рисованного, что приведет к мерцанию. По этой же причине не следует устанавливать кисть для фона в окне отображения. В приложениях MFC рисование можно производить в методе OnDraw, или в обработчике сообщения WM_PAINT. Для приложений MFC рекомендуется производить рисование в методе OnDraw.

Второе сообщение, которое должно обрабатываться в приложении это WM_DISPLAYCHANGE. В обработчике сообщения необходимо вызвать метод IVMRWindowlessControl::DisplayModeChanged. Фильтр выполнит все действия по учету новых параметров дисплея.

Третье сообщение, которое должно обрабатываться в приложении это WM_SIZE. В обработчике сообщения необходимо пересчитать позицию изображения и вызвать IVMRWindowlessControl::SetVideoPosition для установки положения изображения. Если не обрабатывать это сообщение, тогда при изменении размера окна положение и размер изображения останется неизменным .

Четвертое сообщение, которое должно обрабатываться в приложении это WM_ERASEBKGND. Приложение MFC должно определить пустой обработчик сообщения WM_ERASEBKGND для правильной перерисовки фона.

Установка безоконного режима производится с помощью двух интерфейсов фильтра VMR9 IVMRFilterConfig9 и IVMRWindowlessControl9. Такие же интерфейсы используются и для фильтра VMR7 с цифрой 7 в имени интерфейса. Кратко рассмотрим назначение методов этих интерфейсов.

Интерфейс IVMRFilterConfig9.

Метод	Описание
GetNumberOfStreams	Возвращает число входных потоков, которые смешиваются.
GetRenderingMode	Возвращает текущий режим отображения фильтра.
GetRenderingPrefs	Возвращает текущие предпочтительные установки отображения для VMR.
SetImageCompositor	Устанавливает сборщик изображений, предоставленный приложением.
SetNumberOfStreams	Устанавливает число потоков для смешивания и устанавливает режим смешивания для фильтра.
SetRenderingMode	Устанавливает режим отображения.
SetRenderingPrefs	Устанавливает текущие предпочтительные установки отображения для VMR.

Таблица 7. Интерфейс IVMRFilterConfig9.

Интерфейс IVMRWindowlessControl9.

Методы	Описание
DisplayModeChanged	Информирует фильтр VMR, что сообщение WM_DISPLAYCHANGE было принято приложением.
GetAspectRatioMode	Возвращает текущий коэффициент сжатия для режима отображения.
GetBorderColor	Возвращает текущий цвет рамки, используемый фильтром VMR.
GetCurrentImage	Возвращает копию текущего изображения отображаемого фильтром VMR.
GetMaxIdealVideoSize	Возвращает максимальный размер изображения, который может быть отображен фильтром без снижения производительности и без значимых искажений изображения.
GetMinIdealVideoSize	Возвращает минимальный размер изображения, который может быть отображен фильтром без снижения производительности и без значимых искажений изображения.
GetNativeVideoSize	Возвращает не сжатый размер изображения и коэффициент сжатия.
GetVideoPosition	Возвращает текущие прямоугольники источника и получателя изображения.
RepaintVideo	Вызывает перерисовку изображения.
SetAspectRatioMode	Устанавливает текущий коэффициент сжатия.
SetBorderColor	Устанавливает цвет рамки, используемый фильтром VMR.
SetVideoClippingWindow	Указывает окну контейнеру, что видеоизображение рисуется в отсекающем регионе.
SetVideoPosition	Устанавливает текущие прямоугольники источника и получателя изображения.

Таблица 8. IVMRWindowlessControl9.

С учетом выше приведенных замечаний, напишем метод CCapture::InitWindowlessVMR для инициализации безоконного режима фильтра.

```
BOOL CCapture::InitWindowlessVMR(HWND hwnd)
{
  if(!hwnd) return FALSE;

  HRESULT hr = S_OK;
```

```
  //запрос интерфейса на фильтре
  CComQIPtr<IVMRFilterConfig9> pConfig(m_pVRender);
  if(!pConfig)
    return FALSE;

  //Конфигурирование фильтра VMR, установка
  // безоконного режима.
  //Установка числа входных контактов (по умолчанию 4)
  hr = pConfig->SetNumberOfStreams(2);
  if(hr != S_OK)
    return FALSE;

  //Установка безоконного режима
  hr = pConfig->SetRenderingMode(VMRMode_Windowless);
  if(hr != S_OK)
    return FALSE;

  //Запрос этого интерфейса на фильтре производится
  //после установки безоконного режима
  CComQIPtr<IVMRWindowlessControl9> pWc(m_pVRender);
  if(!pWc)
    return FALSE;

  //Установка окна
  hr= pWc->SetVideoClippingWindow(hwnd);
  if(hr != S_OK)
    return FALSE;

  return TRUE;
}
```

Для проверки в каком режиме находится фильтр, напишем метод Capture::IsWindowlessMode.

```
BOOL CCapture::IsWindowlessMode(void)
{
  DWORD mode;
  CComQIPtr<IVMRFilterConfig9> pConfig(m_pVRender);
  if(!pConfig)
    return FALSE;

  if(FAILED(pConfig->GetRenderingMode(&mode)))
    return FALSE;

  if(mode == VMR9Mode_Windowless)
    return TRUE;

  return TRUE;
}
```

Теперь напишем метод для установки позиции изображения в окне приложения. Метод IVMRWindowlessControl9::GetNativeVideoSize возвращает размеры изображения, установленные на фильтре сбора при смене медиа формата. Это входной прямоугольник для фильтра отображения. Метод IVMRWindowlessControl9 ::SetVideoPosition устанавливает изображение в соответствии с принципами, рассмотренными ранее в разделе " Параметры структуры, влияющие на формирование размеров изображения". Для упрощения, метод устанавливает изображение без каких либо преобразований.

```
BOOL CCapture::SetVideoPositionVMR()
{
  if(!IsWindowlessMode())
    return FALSE;

  string sErr="Error SetVideoPositionVMR";

  CComQIPtr<IVMRWindowlessControl9> pWc(m_pVRender);
  if(!pWc)
    return FALSE;

  //Возврат размеров входного прямоугольника
  //изменяется при установке формата на фильтре захвата
  long lWidth,lHeight;
  if(S_OK != pWc->GetNativeVideoSize(&lWidth
                      , &lHeight, NULL, NULL))
    return FALSE;

  RECT rcSrc, rcDest;
  ::SetRect(&rcSrc,0,0,lWidth,lHeight);
  ::SetRect(&rcDest,0,0,lWidth,lHeight);

  if(S_OK != pWc->SetVideoPosition(&rcSrc,&rcDest))
    return FALSE;

  return TRUE;
}
```

Первый вызов метода SetVideoPositionVMR можно расположить в конце метода CCapture::SetModePreview(). Повторно метод вызывается в обработчике CFView1::OnBnClickedButSetformat. При необходимости манипуляции изображением в окне, метод может вызываться в соответствующем обработчике. Для этого его необходимо откорректировать и ввести входные параметры для прямоугольников.

Для работы в безоконном режиме, необходимо откорректировать еще полнооконный режим отображения, который для фильтра VMR выполняется иначе. Полноэкранный режим в безоконном режиме можно заменить отображением в прямоугольнике с идеальным наибольшим размером, который возвра-

щается методом IVMRWindowlessControl9:: GetMaxIdealVideoSize. При этом отображается рамка окна и необходимо принимать дополнительные меры по ее удалению и восстановлению, которые здесь не рассматриваются.

Модернизированный метод CCapture::SetFullScreenMode() устанавливает максимальный размер окна и затем восстанавливает предыдущие размеры.

```
void CCapture::SetFullScreenMode()
{
  if(!m_bPreview)
    return;
  if(!IsWindowlessMode())
    return;

  HRESULT hr = S_OK;
  string sErr = "Error SetFullScreenMode";

  //Запрос интерфейса управления
  CComQIPtr<IVMRWindowlessControl9> pWc(m_pVRender);

  try
  {
    if(!pWc)
      throw hr = E_NOINTERFACE;

    //Определить текущее положение прямоугольников
    RECT rcSrc,rcDest;
    if(S_OK != (hr =
                  pWc->GetVideoPosition(&rcSrc,&rcDest)))
      throw hr;

    //определить максимальный идеальный размер
    LONG widthMax,heightMax;
    if(S_OK != (hr = pWc->GetMaxIdealVideoSize(
                                &widthMax,&heightMax)))
      throw hr;


    LONG width,height;
    width = rcDest.right -rcDest.left;
    height = rcDest.bottom -rcDest.top;

    if(width < widthMax && height < heightMax)
    {
      //Установить максимальный идеальный размер
      ResizeWndParentPreview(widthMax,heightMax);

      ::SetRect(&rcDest,0,0,widthMax,heightMax);
      if(S_OK != (hr = pWc->SetVideoPosition(
                                  &rcSrc,&rcDest)))
        throw hr;
```

```
    }
    else
    {
      //Восстановление размеров окна.
      //Установить размер источника
      ResizeWndParentPreview(rcSrc.right -rcSrc.left
                             ,rcSrc.bottom -rcSrc.top);

      if(S_OK != (hr = pWc->SetVideoPosition(
                                        &rcSrc,&rcSrc)))
        throw hr;

    }

  }
  catch(HRESULT hr)
  {
    CTVshowApp::myMessageError(hr,sErr);
  }
}
```

Для правильной перерисовки окна следует создать обработчик CShowView::OnPaint(). Обработчик нужен в любом случае, даже если нет дополнительного рисования в клиентской области окна. Например, для восстановления изображения, после выхода системы из режима заставки. Остальные обработчики создаются при необходимости, обязателен только пустой обработчик CShowView::OnEraseBkgnd. Следует отметить, что его влияния на изображение не замечалось.

```
void CShowView::OnPaint()
{
  CPaintDC dc(this); // device context for painting

  CCapture* pCap =
  static_cast<CTVshowApp*>(AfxGetApp())->myGetPtrCap();

  if(pCap)
    pCap->RepaintVideoVMR(this->m_hWnd,dc.m_hDC);

}
```

В заключение следует отметить, что VMR9 не работает с устаревшим форматом I420. В приложении он используется для тестирования реакции фильтра на установку "неправильного" формата.

Обработка событий в DirectShow

В процессе работы приложения DirectShow в фильтровом графе происходят события, например фильтр, может подсчитывать число ошибок в цифровом потоке и сообщать об этом, отправляя событие. Событие содержит код типа события и два параметра. Часть событий обрабатываются менеджером графа, и приложение не уведомляется о них. Остальные события располагаются в очереди для обработки их приложением.

Менеджер графа предоставляет три интерфейса, которые поддерживают систему событий DirectShow. Первый из них IMediaEventSink, который содержит только один метод Notify для посылки события. Он не используется приложением и предназначен для применения фильтрами. Приложение использует два оставшихся интерфейса, которые рассмотрим более подробно.

Интерфейс IMediaEvent содержит методы для возврата событий из очереди и для перегрузки обработки события, которая производится менеджером графа по умолчанию.

Метод	Описание
CancelDefaultHandling	Отменяет обработку события по умолчанию менеджером графа и отправляет его для обработки приложением.
FreeEventParams	Освобождает ресурсы, связанные с событием.
GetEvent	Возвращает следующее событие из очереди.
GetEventHandle	Возвращает дескриптор события ручной установки (manual-reset event). Объяснение приводится далее в примере.
RestoreDefaultHandling	Восстанавливает обработку события по умолчанию менеджером графа.
WaitForCompletion	Ожидает, пока граф закончит обработку всех доступных данных.

Таблица 9. Интерфейс IMediaEvent .

Интерфейс IMediaEventEx является производным от IMediaEvent и расширяет его возможности. Интерфейс предоставляет методы для указания окна, которое будет принимать сообщения.

Методы	Описание
SetNotifyWindow	Регистрирует окно для обработки сообщения.
SetNotifyFlags	Разрешает или запрещает обработку сообщений приложением.

GetNotifyFlags	Определяет состояние флага, управляющего обработкой сообщений в приложении.

Таблица 10. Интерфейс IMediaEventEx.

Обработка событий в DirectShow может производиться двумя способами:

- Менеджер графа посылает пользовательское сообщение приложению, когда происходит событие.
- Менеджер графа устанавливает системное событие (Windows event), если в очереди событий DirectShow есть событие и сбрасывает системное событие, если очередь пустая.

Приложение может использовать оба способа обработки событий DirectShow, но первый способ проще в реализации. Для реализации первого способа в приложении определяется частное сообщение WM_GRAPHNOTIFY с кодом от WM_APP до 0xBFFF.

```
#define WM_GRAPHNOTIFY WM_APP + 1.
```

Затем следует вызвать метод, который установит окно для приема сообщений.

```
pEvent->SetNotifyWindow((OAHWND) hwnd
                          , WM_GRAPHNOTIFY, 0)
```

Когда менеджер графа располагает событие в своей очереди, он посылает частное сообщение окну. В классе этого окна устанавливается стандартный обработчик для частного сообщения. Таким образом, событие DirectShow внедряется в стандартный цикл обработки сообщений Windows, без существенных усилий со стороны программиста. Поскольку очередь событий DirectShow и цикл обработки событий Windows не синхронизированы, то каждый раз при обработке поступившего частного сообщения следует просматривать всю очередь DirectShow для поиска нужного сообщения. Для просмотра очереди в цикле вызывается метод pEvent->GetEvent. Перед освобождением указателя на интерфейс IMediaEventEx необходимо вызвать метод SetNotifyWindow с нулевым дескриптором окна, для прекращения поступления событий в это окно.

Реализуем первый способ обработки событий DirectShow. Для этого назначим окно для приема события в методе CCapture::InitCaptureVideo. В конце метода добавляем код для установки окна обработки события.

```
hr = m_pGraph2->QueryInterface(&m_pEvent);
if(hr == S_OK && m_pEvent)
{
  CView* pfV = GetPtrApp()->myGetPtrView1();
```

```
      hr = m_pEvent->SetNotifyWindow(
      reinterpret_cast<OAHWND>(pfV->m_hWnd),
                        WM_GRAPHNOTIFY, 0);

      #ifdef _DEBUG
            if(hr != S_OK)
            {
              sErr = "Error SetNotifyWindow";
              CTVshowApp::myMessageError(hr,sErr);
            }
      #endif //_DEBUG
      }
```

Обработчик событий DirectShow располагаем, в соответствии с назначенным окном, в классе CFView1.

```
LRESULT CFView1::OnGraphNotify(WPARAM wp,LPARAM lp)
{
  long    evCode = 0, param1, param2;
  HRESULT hr;

  CCapture* pCap = myGetPtrCap();
  if(!pCap)
    return 0;

  //Получаем интерфейс
  IMediaEventEx * pEvent = pCap->GetPtrEvent();
  if(!pEvent)
    return 0;

  //Просматриваем очередь событий
    while (SUCCEEDED(pEvent->GetEvent(&evCode, &param1,
                                      &param2, 0)))
  {
    switch(evCode)
    {
      case EC_STREAM_CONTROL_STARTED:
        //Обработка события
        break;
      case EC_STREAM_CONTROL_STOPPED:
        //Обработка события
        break;
      case EC_COMPLETE:
        //Обработка события
        break;
      default : ;
    }

    //освободить параметры
    hr = pEvent->FreeEventParams(evCode, param1,
                                     param2);
  }
```

```
  return 0;
}
```

В качестве примера, используются несколько произвольных событий. Для того чтобы выбранные события EC_STREAM_CONTROL_STARTED и EC_STREAM_CONTROL_STOPPED можно было перехватить, пуск графа и его остановку следует производить с вызовом метода ICaptureGraphBuilder2::ControlStream, подробное описание, которого приводилось в главе " Запись файлов". Ознакомиться со всем списком событий можно в справочном разделе.

Второй способ обработки событий DirectShow использует событие ручной установки, которое поддерживается графическим менеджером. Это событие отражает состояние очереди. Здесь может возникнуть путаница из-за неудачного совпадения терминов, событие ручной установки это событие создаваемое функцией API CreateEvent и его не нужно путать с событиями DirectShow. Если в очереди имеется событие DirectShow, то менеджер графа устанавливает событие ручной установки. Если очередь пустая, вызов IMediaEvent::GetEvent сбрасывает событие ручной установки. Приложение может использовать событие ручной установки для периодического опроса очереди DirectShow. Ниже приведен пример кода для опроса очереди с периодом в сто миллисекунд. Следует отметить, что этот способ хуже еще и потому, что он отнимает дополнительные системные ресурсы для организации периодического опроса.

```
HANDLE  hEvent;
long    evCode, param1, param2;
BOOL bDone = FALSE;
HRESULT hr = S_OK;

//Вернуть дескриптор события ручной установки
hr = pEvent->GetEventHandle((OAEVENT*)&hEvent);

if (FAILED(hr)
{
//Обработать ошибку
}

while(!bDone)
{
     if (WAIT_OBJECT_0 == WaitForSingleObject(
                                        hEvent, 100))
    {
        while (hr = pEvent->GetEvent(&evCode
            , &param1, &param2, 0),SUCCEEDED(hr))
        {
            //Обработка кодов события
            //  ..................................
            //Освободить ресурсы
```

```
            pEvent->FreeEventParams(evCode, param1
                                          , param2);

            //Выход по окончанию воспроизведения файла
            bDone = (EC_COMPLETE == evCode);
        }
    }
}
```

Фильтровый граф полностью управляет событием ручной установки, приложение не должно освобождать дескриптор события по окончанию работы с ним.

Обработка данных на контакте VBI фильтра сбора.

Для обработки данных, поступающих с контакта VBI фильтра сбора, в составе DirectShow имеются фильтры режима ядра в категории WDM Streaming VBI Codecs. Для обработки данных IP интерактивного телевидения в этой категории находится фильтр NABTS/FEC VBI кодек, а для обработки данных телетекста фильтры CC декодер и фильтр WST кодек. Декодеры телетекста, осуществляющие отображение, находятся в категории фильтров DirectShow. К ним относятся фильтры декодеров Line 21 и WST. Фильтры, обрабатывающие телетекст, подразделяются на две группы для двух стандартов телевидения американского NTSC и западноевропейского PAL и SECAM. Телетекст американского стандарта обрабатывается фильтрами CC декодер и Line 21 декодер, а западноевропейского стандарта фильтрами WST.

Следует отметить, что отображающий фильтр VMR не работает с текущей реализацией фильтра декодера WST, а с фильтрами телетекста американского стандарта работает при условии использования фильтра Line 21 декодер 2. Схемы построения графа для обработки телетекста каждого стандарта показаны ниже на рисунках.

Провести экспериментальную проверку отображения телетекста стандарта NTSC не представляется возможным. Поэтому ограничимся некоторыми теоретическими сведениями о фильтрах. Технические данные, необходимые для применения фильтров приведены в справочном разделе.

Фильтр Tee/Sink-to-Sink разделяет поток данных на несколько одинаковых потоков данных для организации обработки телетекста в одном потоке и данных IP в другом потоке. Фильтр работает в режиме ядра и устанавливается в граф с использованием системного перечислителя. Он представлен в категории фильтров "WDM Streaming Tee/Splitter Devices" (AM_KSCATEGORY_SPLITTER).

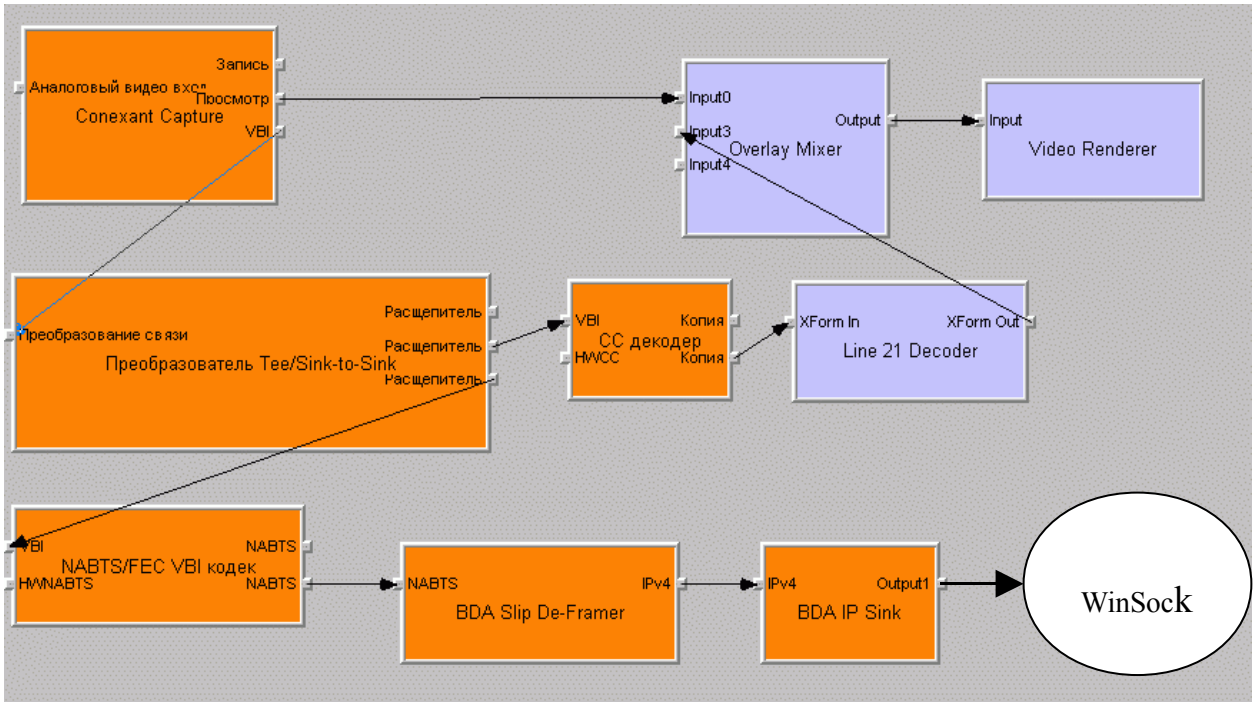

Рис.4. 8. Обработка данных VBI для стандарта NTSC.

Фильтр CC декодер принимает оцифрованный поток аналоговых данных с контакта VBI и формирует данные телетекста, расположенные на 21 телевизионной строке. Второй вход фильтра HWCC предназначен для приема данных аппаратного декодера телетекста. В этом случае фильтр выполняет простую передачу данных. Если фильтр сбора выполняет аппаратное декодирование телетекста, то фильтр CC подключается непосредственно к контакту фильтра сбора, который имеет в этом случае категорию PINNAME_VIDEO_CC_CAPTURE. Фильтр имеет до восьми выходных линий, каждая из которых декодирует свою строку. Первая выходная линия подключается к фильтру декодера Line 21. Фильтр работает в режиме ядра и устанавливается в граф с использованием системного перечислителя. Он представлен в категории фильтров "WDM Streaming VBI Codecs".

Фильтр Line 21 производит формирование растрового изображения телетекста по данным фильтра CC. Фильтр имеет две разновидности. На платформах Win98 и Win2000 используется фильтр Line 21 Decoder, и он подключается к фильтру микшера (Overlay Mixer) и далее к отображающему фильтру. Для платформ XP с использованием фильтров отображения VMR применяется фильтр Line 21 Decoder 2.

Построение графа для отображения телетекста американского стандарта может производиться автоматически при вызове метода RenderStream. Следует учесть, что контакт VBI может иметь категорию PIN_CATEGORY_VBI, PIN_CATEGORY_CC или PINNAME_VIDEO_CC_CAPTURE. Поэтому этот метод нужно будет вызывать три раза или точно определить категорию контакта. Управление отображением телетекста производится с помощью основного интерфейса IAMLine21Decoder, который управляет включением и выключением телетекста, а также всеми остальными сервисами. Для передачи

IP данных интерактивного телевидения строится дополнительный участок графа, который передает данные непосредственно в WinSock. Поскольку провести экспериментальную проверку для отображения телетекста американского стандарта не представляется возможным, то и построение графа для этого случая не рассматривается.

Отображение телетекста для условий телетрансляции в зоне стран СНГ можно увидеть для мирового стандарта телетекста (WST), так как некоторые телеканалы производят такую передачу. При построении графа для просмотра телетекста стандарта WST следует учесть некоторые особенности этого процесса. Автоматическое построение этого участка графа не поддерживается. Также не поддерживается соединение с фильтром VMR, что приводит к дополнительным осложнениям. К счастью, эти сложности даже дают неожиданные преимущества, поскольку появляется возможность отобразить окно телетекста на фоне основного окна и управлять им отдельно. Для построения участка графа следует добавить в граф фильтр WST кодек, затем произвести его соединение с фильтром сбора. После этого уже допускается автоматическое построение графа с выходного контакта фильтра кодек WST с помощью метода RenderStream. В этом методе не следует указывать фильтр VMR, если он уже имеется в графе. Тогда будет достроен дополнительный участок графа с фильтром отображения Video Renderer, как показано на рисунке 4.10. Подобное построение графа значительно расширяет возможности отображения, поскольку окном отображения телетекста можно манипулировать отдельно.

Следует отметить, что при построении графа с помощью графического редактора, разрешается соединение WST декодера с фильтром VMR любым способом. Отображение страницы телетекста в этом случае выполняется. В тоже время программное соединение WST декодера с фильтром VMR не удалось произвести, хотя использовались все возможные методы DirectShow для соединения фильтра WST Decoder с фильтром VMR. Объяснить этот казус могут только разработчики графического редактора. Плохо только то, что подобное поведение графического редактора может заставить усомниться прикладного программиста в правильности своего кода.

Для построения участка графа, отображающего телетекст, напишем метод RenderVBI. В методе перечисляются контакты фильтра сбора, и определяется их категория. Если контакт с нужной категорией найден, тогда строиться граф. Вместо перечисления контактов можно использовать метод ICaptureGraphBuilder2::FindPin, который ищет контакт, указанной категории. Тогда этот метод придется вызывать несколько раз для разных категорий, что по объему написанного кода не будет проще, а с учетом того, что метод сам использует перечисление контактов то и эффективность будет ниже.

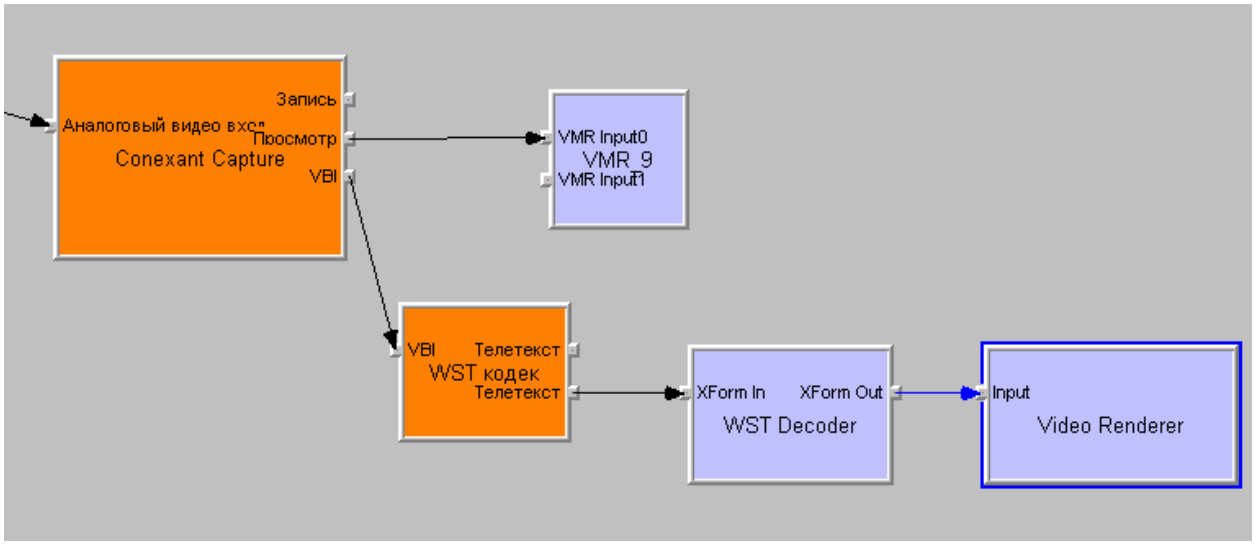

Рис.4. 9. Обработка данных VBI для стандарта WST.

Следует сказать, что если приложение часто перестраивает часть графа ответственного за отображение, то лучшим вариантом будет хранение указателей на выходные контакты фильтра сбора как членов класса. Это можно сделать на стадии инициализации фильтра сбора. Такой подход значительно упростит код приложения.

При перечислении контактов, для каждого найденного контакта запрашивается интерфейс IKsPropertySet. Интерфейс с таким же именем определен и в файле dsound.h. Они не совместимы! Интерфейс, определенный в DirectShow, используется для приема и передачи свойств в WDM драйвер при вызовах методов COM. Фильтр KxProxy осуществляет трансформирование вызовов с помощью потоковых класс драйверов. Этот интерфейс также используется для передачи информации между программными компонентами. Интерфейс имеет только три метода, Set, Get и QuerySupported, который запрашивает объект о поддержке определенного свойства.

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

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

  CComPtr<IBaseFilter> pWSTCodec;

  //Найти на фильтре захвата контакты для отображения
  // телетекста
  if (FAILED(hr = m_pCapV->EnumPins(&pEnum)))
    return FALSE;

  //Перечисляем контакты
```

```
while(pEnum->Next(1, &pPin, 0) == S_OK)
{
  PIN_DIRECTION PinDirThis;
  pPin->QueryDirection(&PinDirThis);

  if (PinDirThis != PINDIR_OUTPUT)
  {
    pPin = NULL;
    continue;
  }

  //Запросить интерфейс IKsPropertySet
  //для определения категории контакта
  CComQIPtr<IKsPropertySet> pKs(pPin);
  if(!pKs)
  {
    pPin = NULL;
    continue;
  }

  //Определить категорию контакта
  DWORD cbReturned;

  if(S_OK == pKs->Get(
        AMPROPSETID_Pin,  //GUID   свойств
        AMPROPERTY_PIN_CATEGORY, //ID свойства
        NULL, 0,&guid, sizeof(GUID), &cbReturned))
  {
    //Проверка категории
    if(guid == PIN_CATEGORY_VBI
                || guid == PIN_CATEGORY_CC
                ||guid == PINNAME_VIDEO_CC_CAPTURE)
    {
      //Проверка автоматического построения графа для
      //просмотра данных 21 строки. Здесь следует
      //построить участок графа полностью
      //для телетекста американского стандарта
      if(S_OK == m_pBuilder->RenderStream(&guid
                            ,0,m_pCapV,0,NULL))

      {
        break;
      }
      else if(guid == PIN_CATEGORY_VBI)
      {
        //Попытаться построить участок графа для
        //телетекста с фильтрами WST
        if(S_OK != AddKernelFilter(
                     AM_KSCATEGORY_VBICODEC
                     ,&pWSTCodec,L"WST кодек"))
          return FALSE;
```

```
          //Подключить контакт VBI к кодеку WST
          if(S_OK != ConnectFilters(pPin,NULL,pWSTCodec))
            return FALSE;

          //Вызов этого метода с фильтром VMR9
          //возвращает ошибку, поскольку декодер WST
          //не поддерживает соединение
          if(S_OK != m_pBuilder->RenderStream(
                              &PIN_CATEGORY_TELETEXT,0
                              ,pWSTCodec,0,NULL))
            return FALSE;
        }
      }
    }

    pPin = NULL;

  }//while

  return TRUE;
}
```

В методе CCapture::RenderVBI используется метод CCapture::ConnectFilters, который производит ручное соединение фильтров. В методе ConnectFilters применяется метод ICaptureGraphBuilder2::FindPin, который рекомендуется документацией DirectShow для решения задачи ручного соединения фильтров. Метод FindPin имеет некоторые особенности применения и его следует рассмотреть более детально.

Метод ICaptureGraphBuilder2::FindPin.

Метод возвращает указатель на контакт, найденный на фильтре по указанным критериям.

```
HRESULT FindPin(
  IUnknown            *pSource,
  PIN_DIRECTION     pindir,
  const GUID          *pCategory,
  const GUID          *pType,
  BOOL               fUnconnected,
  int                  num,
  IPin                **ppPin
);

Параметры.

pSource
[in]
```

Указатель на фильтр или указатель на контакт. Если это указатель на контакт, то метод тестирует его на соответствие критериям.

pindir
[in]
Направление контакта.

pCategory
[in]
Указатель на категорию контакта. Член структуры AMPROPERTY_PIN_CATEGORY. Может быть NULL для использования любой категории.

pType
[in]
Указатель на главный медиа тип. Может быть NULL для использования любого типа.

fUnconnected
[in]
Указывает состояние контакта. Если TRUE, то ищется только не подключенный контакт. Если FALSE, то ищется контакт с любым состоянием, подключенный или не подключенный.

num
[in]
Индекс (номер) возвращаемого контакта. Если pSource указатель на фильтр и имеется несколько контактов соответствующих набору параметров, индекс указывает, для какого контакта вернуть указатель. Если pSource указатель на контакт, параметр игнорируется.
Кроме того, при проверке было обнаружено что, индексация контактов зависит от числа подключенных контактов на фильтре и параметра fUnconnected. Если этот параметр установлен в TRUE, то индексация контактов ограничивается только не подключенными контактами и не соответствует общей индексации контактов на фильтре. Например, допустим, что фильтр VMR имеет подключенный нулевой контакт и оставшиеся три не подключенные. Нужно найти контакт номер два на фильтре. Для неподключенных контактов индексация начинается с нуля и для того, чтобы вернуть указатель на контакт номер два нужно указать индекс равный единице. Такое поведение метода требует повышенного внимания к индексации контактов на фильтре.

ppPin
[out]
Указатель на найденный контакт.

Метод возвращает S_OK при нахождении контакта или E_FAIL в противном случае.

Метод CCapture::ConnectFilters производит соединение двух фильтров. Фильтр источник указывается либо своим выходным контактом, либо указа-

телем на фильтр. Третий параметр указывает на фильтр приемник. Четвертый параметр является индексом входного контакта для фильтров, которые имеют несколько входных контактов. Замечание по поводу индекса входного контакта приведено в описании метода FindPin.

```
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;

  CComPtr<IPin> pOut;

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

  //Указан контакт на фильтре источнике
  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 прямым соединением без "интеллектуальных" //вста-
  вок фильтров
  hr = m_pGraph2->ConnectDirect(pOut,pPinIn,NULL);

  //если контакт уже соединен, тогда S_OK
  if(hr == VFW_E_ALREADY_CONNECTED)
    hr = S_OK;

    return hr;
```

```
}
```

Для проверки работы метода RenderVBI достаточно произвести его вызов в конце метода CCapture::SetModePreview. Необходимо предусмотреть элементы интерфейса управления окном отображения телетекста для вызова метода по требованию пользователя.

```
……................................................................
if(!RenderVBI())
{
  sErr += " Просмотр телетекста невозможен";
  CTVshowApp::MessageError(hr = E_FAIL,sErr);
}

return TRUE;
```

При удачном выполнении метода будет построен участок графа для просмотра телетекста. При условии передачи вещательным каналом информации телетекста можно сразу наблюдать картинку приблизительно такого содержания, как показано на рисунке 4.11. Отображение производится в отдельном окне фильтра Video Renderer. Таким образом, можно одновременно просматривать программу ТВ в окне фильтра VMR и телетекст в своем окне.

Управление отображением телетекста.

Управление отображением телетекста производится на фильтре WST Decoder, свойства которого приведены в справочном разделе. Фильтр предоставляет два интерфейса ISpecifyPropertyPages и IAMWstDecoder. Первый интерфейс позволяет вызвать страничный диалог свойств фильтра, а второй интерфейс позволяет приложению управлять свойствами отображения телетекста. Сводная таблица методов интерфейса IAMWstDecoder демонстрирует возможности управления отображением.

Method	Description
GetAnswerMode	Возвращает состояние отображения скрытого текста (ответа), будет он отображаться или нет.
GetBackgroundColor	Возвращает текущие физические цвета фона в формате RGB.
GetCurrentPage	Возвращает текущую страницу телетекста.
GetCurrentService	Возвращает текущий сервис телетекста.
GetDecoderLevel	Возвращает текущий уровень декодирования. Метод не реализован и всегда возвращает одно значение.

GetDrawBackgroundMode	Определяет прозрачный фон или нет.
GetHoldPage	Определяет, удерживается ли текущая страница или нет.
GetOutputFormat	Возвращает текущий видеоформат.
GetRedrawAlways	Определяет, будет ли перерисовываться растр для каждой выборки или нет.
GetServiceState	Определяет состояние сервиса, разрешено отображение телетекста или нет.
SetAnswerMode	Устанавливает отображение скрытого текста (ответа).
SetBackgroundColor	Устанавливает текущие физические цвета для фона.
SetCurrentPage	Устанавливает текущую страницу телетекста.
SetDrawBackgroundMode	Устанавливает прозрачный или не прозрачный фон для телетекста.
SetHoldPage	Устанавливает режим удержания текущей страницы.
SetOutputFormat	Устанавливает выходной видеоформат.
SetRedrawAlways	Назначает перерисовку растра для каждой выборки.
SetServiceState	Включает и выключает отображение телетекста.

Не все методы, приведенные в таблице, реализованы в текущей версии фильтра. Для быстрого определения текущей реализации методов можно вызвать диалог установки свойств фильтра, который отобразит все устанавливаемые свойства фильтра. Для вызова диалога следует найти указатель на фильтр, который был установлен автоматически (метод RenderStream), и вызвать диалог.

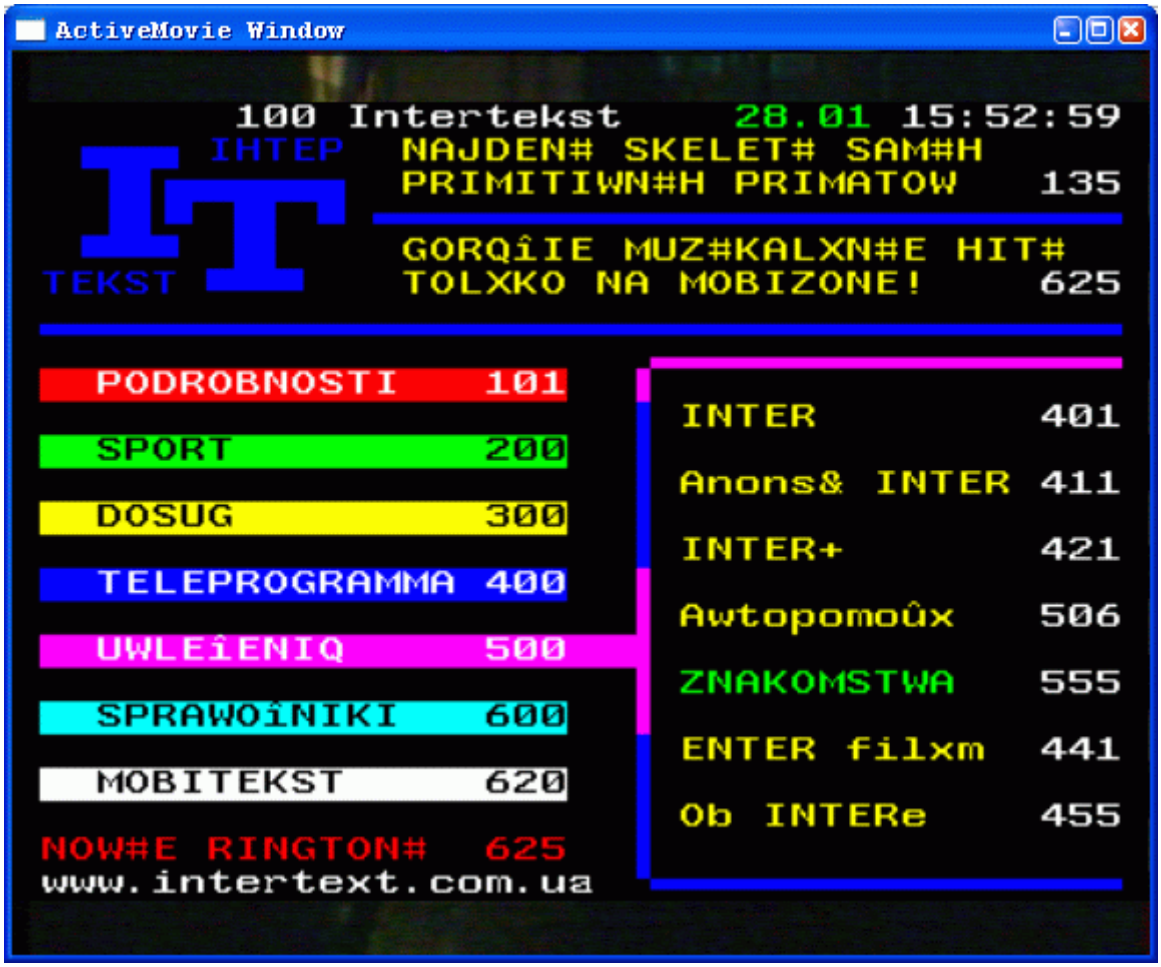

Рис.4. 10 Отображение телетекста.

```
CComPtr<IBaseFilter> pWSTDecoder;

if(S_OK!=m_pGraph2->FindFilterByName(
                    L"WST Decoder",&pWSTDecoder))
  return 0;

IBaseFilter* pF = pWSTDecoder;
DisplayPropertyPages  (pF);
```

На рисунке 4.12 показан диалог свойств телетекста.

По числу устанавливаемых свойств можно определить, что четыре метода точно используются для установки свойств телетекста. Остальные методы следует проверять дополнительно. Например, метод SetServiceState и GetServiceState работают, а метод SetBackgroundColor не работает. Следует отметить, что при запуске приложения в режиме отладки с отладочной информацией страница телетекста отображалась некорректно.

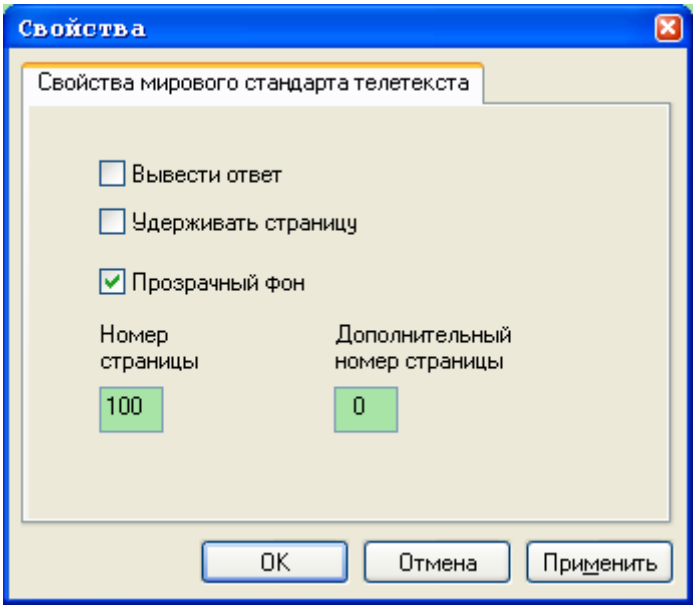

Рис.4. 11. Страница свойств телетекста стандарта WST.

Регулировки изображения

Регулировки изображения, такие как яркость, контрастность, цветовой тон, насыщенность и некоторые другие осуществляются устройством на входном видеосигнале и доступны на фильтре сбора посредством интерфейса IAMVideoProcAmp.

Фильтр отображения VMR9 также позволяет производить основные регулировки изображения, причем он использует те же самые регулировки, которые предоставляет драйвер устройства, но в ограниченном наборе. Для этого предоставляется интерфейс IVMRMixerControl9, в котором имеется два метода GetProcAmpControl и SetProcAmpControl, осуществляющие регулировки яркости, контрастности, цветового тона и насыщенности. Методы интерфейса пригодны больше для оперативных регулировок, когда в приложении не требуется регулировок других параметров.

Фильтр Video Renderer не имеет интерфейса для выполнения регулировок изображения. При его применении можно использовать только интерфейс IAMVideoProcAmp фильтра сбора.

Интерфейс фильтра сбора IAMVideoProcAmp предоставляет больше возможностей для регулировки изображения по сравнению с фильтром VMR. Он более универсален, поскольку при его использовании отсутствует зависимость регулировок от типа отображающего фильтра. Интерфейс содержит три метода, показанные в таблице. Метод GetRange позволяет определить набор регулируемых параметров предоставляемых устройством и включает все возможные регулировки параметров изображения.

Интерфейс IAMVideoProcAmp

Интерфейс предоставляется на фильтре сбора, если устройство поддерживает регулировки изображения.

Метод	Описание
GetRange	Возвращает максимальное, минимальное и значение по умолчанию для устанавливаемого параметра.
Set	Устанавливает значение указанного параметра.
Get	Возвращает значение указанного параметра.

Метод IAMVideoProcAmp::GetRange

```
HRESULT GetRange(
  long Property,
  long *pMin,
  long *pMax,
  long *pSteppingDelta,
  long *pDefault,
  long *pCapsFlags
);
```

Параметры.

Property
[in]
Указывает запрашиваемый параметр, который определен в перечислении
VideoProcAmpProperty.

pMin
[out]
Указатель на переменную, которая принимает значение минимальной величины.

pMax
[out]
Указатель на переменную, которая принимает значение максимальной величины.

pSteppingDelta
[out]
Указатель на переменную, которая принимает значение шага изменения величины.

pDefault
[out]

Указатель на переменную, которая принимает значение величины устанавливаемой по умолчанию.

pCapsFlags
[out]
Указатель на переменную, которая принимает значение флага, определенного в перечислении VideoProcAmpFlags. Флаг указывает, каким образом регулируется параметр автоматически или вручную.

Возвращаемые коды.

Коды возврата	Описание
E_INVALIDARG	Неправильный аргумент.
E_POINTER	Нулевой указатель.
E_PROP_ID_UNSUPPORTED	Устройство не поддерживает регулировку параметра.
S_OK	Нет ошибок.

Метод IAMVideoProcAmp::Set

```
HRESULT Set(
  long Property,
  long lValue,
  long Flags
);
```

Параметры

Property
[in]
Указывает устанавливаемый параметр, который определен в перечислении
VideoProcAmpProperty.

lValue
[in]
Новое значение величины.

Flags
[in]
Флаг указывает, каким образом регулируется параметр автоматически или вручную. Флаги определены в перечислении VideoProcAmpFlags. Если флаг указывает на автоматическую регулировку, то новое значение величины игнорируется.

Коды возврата.
Если метод выполнился без ошибок, возвращается S_OK, в противном случае возвращается код ошибки HRESULT.

Метод IAMVideoProcAmp::Get

```
Syntax
HRESULT Get(
  long Property,
  long *lValue,
  long *Flags
);
```

Параметры

Property
[in]
Указывает запрашиваемый параметр, который определен в перечислении
VideoProcAmpProperty.

lValue [out]
Указатель на переменную, которая принимает текущее значение параметра.

Flags [out]
Указатель на переменную, которая принимает значение флага.
Флаг указывает, каким образом регулируется параметр автоматически или вручную. Флаги определены в перечислении VideoProcAmpFlags.

Коды возврата.

Коды возврата	Описание
S_OK	Без ошибок.
E_POINTER	Нулевой указатель аргумента.

Перечисление VideoProcAmpProperty.

Перечисление определяет константы для всех регулируемых параметров на устройстве.

```
typedef enum tagVideoProcAmpProperty
{
    VideoProcAmp_Brightness,
    VideoProcAmp_Contrast,
    VideoProcAmp_Hue,
    VideoProcAmp_Saturation,
    VideoProcAmp_Sharpness,
    VideoProcAmp_Gamma,
    VideoProcAmp_ColorEnable,
    VideoProcAmp_WhiteBalance,
    VideoProcAmp_BacklightCompensation,
    VideoProcAmp_Gain
} VideoProcAmpProperty;
```

Элементы.

VideoProcAmp_Brightness
Яркость или уровень черного. Для NTSC устанавливается в единицах

IRE * 100. Для других стандартов ноль определяется уровнем черного на строчном гасящем импульсе и величиной 10000, которая соответствует уровню белого на строчном гасящем импульсе, то есть максимальной яркости.

VideoProcAmp_Contrast
Контрастность определяется коэффициентом усиления, умноженным на 100, и имеет диапазон от нуля до 10000.

VideoProcAmp_Hue
Цветовой тон указывается в градусах умноженный на 100, и изменяется в диапазоне от – 180000 до 180000.

VideoProcAmp_Saturation
Насыщенность цвета изменяется в диапазоне от 0 до 10000.

VideoProcAmp_Sharpness
Четкость изображения изменяется в диапазоне от 0 до 100.

VideoProcAmp_Gamma
Гамма или коэффициент контрастности указывается как гамма, умноженная на 100. Изменяется в диапазоне от 1 до 500.

VideoProcAmp_ColorEnable
Включение и выключения канала цветности. Нуль выключает цвет, единица включает цвет.

VideoProcAmp_WhiteBalance
Баланс белого или цветовая температура в градусах Кельвина. Диапазон зависит от устройства.

VideoProcAmp_BacklightCompensation
Включение компенсации подсветки. Величина 0 выключение, 1 включение.

VideoProcAmp_Gain
Регулировка усиления. Ноль означает нормальное усиление. Значения по обе стороны от нуля означают уменьшение или увеличение усиления. Диапазон величин зависит от устройства.

Перечисление VideoProcAmpFlags.

Определяет способ управления параметрами, ручной или автоматический.

```
typedef enum tagVideoProcAmpFlags
```

```
{
VideoProcAmp_Flags_Auto   = 0x0001,
VideoProcAmp_Flags_Manual = 0x0002,
} VideoProcAmpFlags;
```

Реализация регулировки яркости и контраста

Рассмотрим реализацию регулировок изображения на примере двух наиболее часто используемых регулировок яркости и контрастности. Для регулировки необходимы два регулятора, которые установим на второй вкладке в классе CFView1. Для реализации регуляторов используем класс регулятора CFader собственной разработки. Он используется для создания регуляторов канала изображения и звука. Описание класса регулятора и его применение приводится в главе " Звуковой микшер".

Для использования регуляторов объявляем в классе CFView1 две переменные.

```
  //Указатель на регулятор яркости
  auto_ptr<CFader> m_pBright;

  //Указатель на регулятор контрастности
  auto_ptr<CFader> m_pContrast;
```

Добавляем в редакторе ресурсов ресурсный символ для регуляторов IDC_FADER произвольной величины (2000), который является базой для формирования ID регуляторов. Для этого открываете вкладку Resource View и делаете правый щелчок на папке TVshow.rc. Открывается контекстное меню, в котором выбираете строку ID= Resource symbols. В результате открывается окно со списком всех ресурсных идентификаторов. Нажимаем кнопку New и добавляем новый идентификатор. На рисунке 4.13 показан результат этих действий.

В проект включаем папку с именем Mixer, в которой находятся файлы класса CFader. Создаем ограничивающее окно в редакторе ресурсов, класса CStatic. На вкладке ToolBox редактора ресурсов этот элемент называется Group Box. Первоначальную инициализацию регуляторов произведем в методе CFView1::OnInitialUpdate(), добавив в конце строки, создающие регуляторы в ограничивающем окне.

```
.................................................................
//Установка регулятора яркости
//указатель на ограничивающее окно
  CWnd* pWnd = GetDlgItem(IDC_BRIGHTNESS);
  auto_ptr<CFader> pBright(
                  new CFader(pWnd,STYLE_FADER_HORZ,this));
  m_pBright = pBright;
```

```
//Имя окна это имя регулятора
pWnd->SetWindowText("Яркость");

//Установка регулятора контрастности
pWnd = GetDlgItem(IDC_CONTRAST);
auto_ptr<CFader> pContrast(
              new CFader(pWnd,STYLE_FADER_HORZ,this));
m_pContrast = pContrast;

//Имя окна это имя регулятора
pWnd->SetWindowText("Контраст");
```

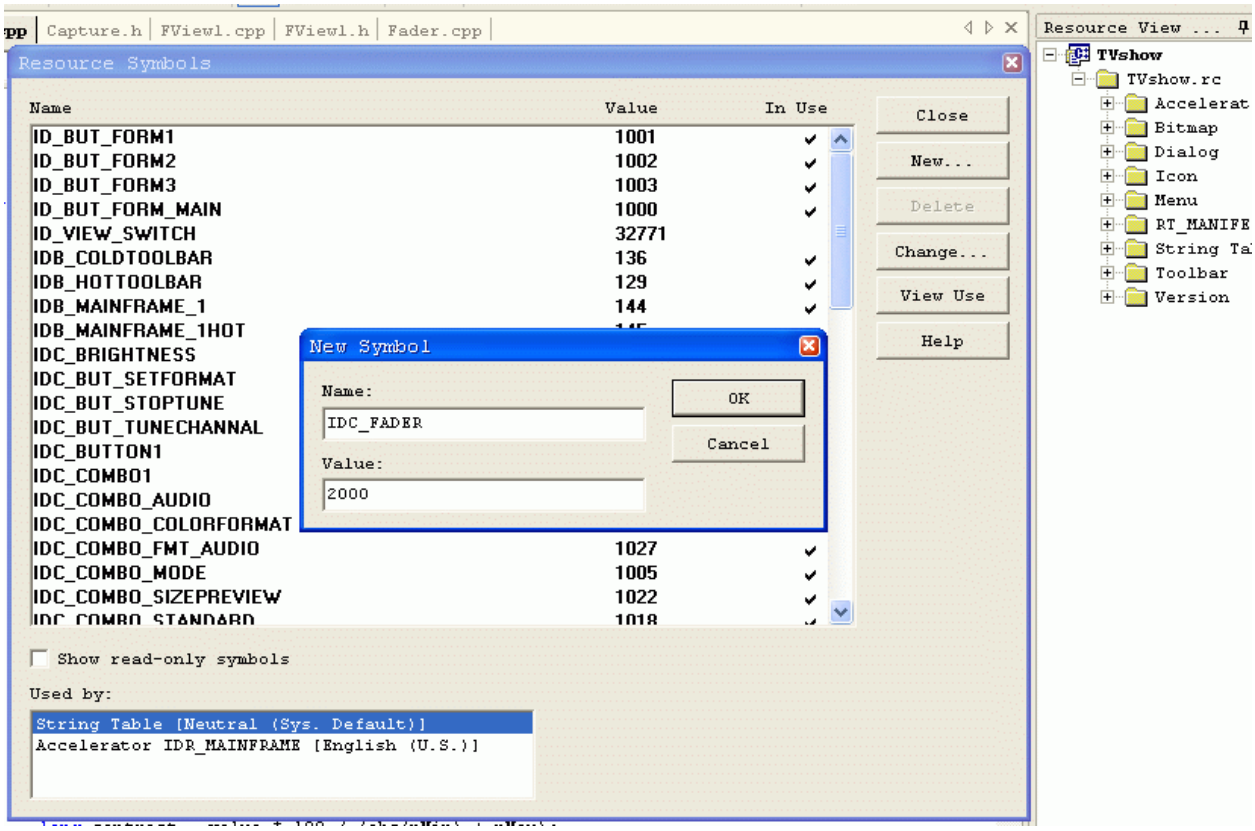

Рис.4. 12 Ввод нового идентификатора

В классе CCapture напишем два метода для установки параметров яркости и контрастности и два метода для возврата значений этих параметров. Величина параметра всегда указывается в процентах от полной шкалы регулятора, который имеет диапазон регулировки от нуля до 100 единиц. Задачей метода Set является преобразование значения шкалы регулятора в значение определенного параметра и его установка. Метод Get выполняет обратное преобразование и возвращает результат в процентах от полной шкалы. Для ознакомления с ними ниже приводится пара методов для установки и возврата яркости.

```
void CCapture::SetBrightness(int nProc)
{
  if(!m_pCapV)
      return;

  HRESULT hr = S_OK;

  //Запрос интерфейса видеоусилителя на фильтре захвата
  CComQIPtr<IAMVideoProcAmp> pAmp(m_pCapV);
  if(!pAmp)
      return;

  long nMin,nMax,nSteingDelta,nDefault,nCasFlags;

  //Возвращаем параметры регулировки яркости,
  //флаг возвращает режим управления параметром,
  //ручной или автоматический
  hr = pAmp->GetRange(VideoProcAmp_Brightness,
    &nMin,&nMax,&nSteingDelta,&nDefault,&nCasFlags);

  if(hr == E_PROP_ID_UNSUPPORTED)
  {
    //Если нет поддержки, регулятор заблокировать
    //или удалить. Можно сделать возврат HRESULT для
  //анализа в вызывающем методе.

    return;
  }

  if(hr != S_OK)
      return;

  //Расчет величины для установки, которая кратна шагу
  long value =
  ((((abs(nMin) + nMax) *nProc / 100) - nMin) /
                      nSteingDelta) *nSteingDelta;

  //установить значение
  hr = pAmp->Set(VideoProcAmp_Brightness
                   ,value,VideoProcAmp_Flags_Manual);

}


int CCapture::GetBrightness()
{
  if(!m_pCapV)
    return FALSE;

  HRESULT hr = S_OK;

  //Запрос интерфейса видеоусилителя на фильтре захвата
```

```
  CComQIPtr<IAMVideoProcAmp> pAmp(m_pCapV);
  if(!pAmp)
    return FALSE;

  long nMin,nMax,nSteingDelta,nDefault,nCasFlags;

  //Определяем возможность регулировки и пределы
  //параметров
  hr = pAmp->GetRange(VideoProcAmp_Brightness,
    &nMin,&nMax,&nSteingDelta,&nDefault,&nCasFlags);

  if(hr == E_PROP_ID_UNSUPPORTED)
  {
    //Если нет поддержки, регулятор заблокировать
    //или удалить

    return FALSE;
  }

  if(hr != S_OK)
      return FALSE;

  //Возврат значения параметра
  long value,flag;
  hr = pAmp->Get(VideoProcAmp_Brightness,&value,&flag);

  //Определить в процентах
  int bright =  value * 100 / (abs(nMin) + nMax);

  return bright;

}
```

В обработчике CFView1::OnMyMessageEndInit, в конце метода, добавляем строки для установки текущих значений регуляторов, которые определены устройством по умолчанию.

```
    //Установка текущих значений яркости и контрастности
    if(m_pBright.get() && m_pContrast.get())
    {
      int nProc = pCap->GetBrightness();
      m_pBright->mySetFaderValueProc(nProc);

      nProc = pCap->GetContrast();
      m_pContrast->mySetFaderValueProc(nProc);
    }
```

После запуска приложения на вкладке появляются два регулятора, которые полностью готовы к работе. Поскольку обработка прокрутки регуляторов

производится в классе CFader, то в обработчике CFader::HScroll добавим вызовы методов для передачи позиции регулятора в процентах от полной шкалы в объект класса CFView1.

```
//Проверяем, что регулятор, который прокручивается,  //принадле-
жит классу владельцу регулятора
CFView1* pv1 = static_cast<CFView1*>(m_pOwner);

if(pv1->IsKindOf(RUNTIME_CLASS(CFView1)))
{
    //установка яркости или контрастности
    //возвращаем показания регулятора в процентах
    int nProc = myGetFaderValueProc();

    //передаем данные объекту создавшему регулятор
    pv1->mySetFaderValue(this,nProc);

    //установка текстовых показаний регулятора
    mySetWndBuddyText(nProc);
}
```

Метод CFView1::mySetFaderValue устанавливает значение для всех регуляторов на вкладке. Он передает в параметрах указатель на регулятор (на себя), который прокручивается и свое текущее значение в процентах. Данный метод нельзя реализовать прямо в классе CCapture, поскольку он не является производным от CObject и не связан с библиотекой MFC. Поэтому он реализован в классе CFView1 и уже из этого класса будет вызываться методы класса CCapture для установки значений яркости и контрастности.

В методе CFView1::mySetFaderValue по указателю на регулятор, переданному как параметр, определяется его принадлежность к определенному органу управления, установленному на вкладке. Затем вызывается метод для установки величины параметра на фильтре сбора.

```
void CFView1::mySetFaderValue(CFader* pFader, int nProc)
{
  CCapture* pCap = myGetPtrCap();
  if(!pCap) return;

  //Проверка указателя и установка яркости
  if(pFader == m_pBright.get())
      pCap->SetBrightness(nProc);

  //Проверка указателя и установка контрастности
  if(pFader == m_pContrast.get())
      pCap->SetContrast(nProc);

}
```

Регулировки изображения на фильтре VMR9

Для реализации регулировок на фильтре VMR9 следует использовать интерфейс фильтра IVMRMixerControl9. Этот интерфейс управляет входным видеопотоком, он предназначен для использования только в приложении и не может использоваться фильтром, расположенным выше по потоку. Методы интерфейса приведены в таблице.

Метод	Описание
GetAlpha	Возвращает величину коэффициента прозрачности альфа, которая установлена для видеопотока.
GetBackgroundClr	Возвращает цвет фона для выходного прямоугольника.
GetMixingPrefs	Возвращает параметры поверхности Direct3D во входном потоке, которые определены в перечислении VMR9MixerPrefs.
GetOutputRect	Возвращает позицию прямоугольника потока в сборном (выходном) прямоугольнике.
GetProcAmpControl	Возвращает текущие значения регулировок изображения, такие как яркость, контрастность, цветовой тон и насыщенность.
GetProcAmpControlRange	Возвращает диапазон регулировок изображения.
GetZOrder	Возвращает положение видеопотока в Z порядке потоков.
SetAlpha	Устанавливает величину коэффициента прозрачности альфа, которая установлена для видеопотока.
SetBackgroundClr	Устанавливает цвет фона для выходного прямоугольника.
SetMixingPrefs	Устанавливает параметры поверхности Direct3D во входном потоке, которые определены в перечислении VMR9MixerPrefs.
SetOutputRect	Устанавливает позицию прямоугольника потока в сборном (выходном) прямоугольнике.
SetProcAmpControl	Устанавливает текущие значения регулировок изображения.
SetZOrder	Устанавливает положение видеопотока в Z порядке потоков.

Из всех методов интерфейса для регулировок изображения используются методы GetProcAmpControl, SetProcAmpControl и GetProcAmpControlRange, которые рассмотрим более подробно.

Метод GetProcAmpControlRange возвращает диапазон регулировок изображения, таких как яркость, контрастность, цветовой тон и насыщенность (brightness, contrast, hue, saturation). Регулировки выполняются устройством, и их диапазон определяется драйвером устройства.

```
HRESULT GetProcAmpControlRange(
  DWORD  dwStreamID,
  VMR9ProcAmpControlRange*  lpClrControl
);
```

Параметры.

dwStreamID
[in]
Идентификатор входного потока.

pRect
[in, out]
Указатель на структуру VMR9ProcAmpControlRange, которая принимает данные. Перед вызовом метода необходимо заполнить поля dwSize и dwProperty. Поле dwProperty определяется в перечислении VMR9ProcAmpControlFlags.

Возвращаемые величины.

Величина	Описание
E_INVALIDARG	Неправильный номер потока.
E_POINTER	Нулевой указатель.
S_OK	Успешное выполнение.
VFW_E_NOT_CONNECTED	Контакт не подключен.

Метод GetProcAmpControl возвращает значения текущих регулировок изображения.

Syntax
```
HRESULT GetProcAmpControl(
  DWORD  dwStreamID,
  VMR9ProcAmpControl*  lpClrControl
);
```

Параметры.

dwStreamID
[in]
Specifies the input stream.

lpClrControl
[in, out]
Указатель на структуру VMR9ProcAmpControl, которая принимает данные. Перед вызовом метода необходимо заполнить поля dwSize и dwFlags для структуры VMR9ProcAmpControl. Поле флага определяется в перечислении VMR9ProcAmpControlFlags.

Возвращаемые значения.

Величина	Описание
E_INVALIDARG	Неправильный номер потока.
E_POINTER	Нулевой указатель.
S_OK	Успешное выполнение.
VFW_E_NOT_CONNECTED	Контакт не подключен.

Метод SetProcAmpControl устанавливает значения текущих регулировок изображения на фильтре VMR-9.

```
HRESULT SetProcAmpControl(
  DWORD  dwStreamID,
  VMR9ProcAmpControl*  lpClrControl
);
```

Параметры

dwStreamID
[in] Specifies the input stream.

lpClrControl
[in]
Указатель на структуру VMR9ProcAmpControl, которая содержит устанавливаемые данные. Перед вызовом метода необходимо заполнить поля dwSize и dwFlags для структуры VMR9ProcAmpControl. Поле флага определяется в перечислении VMR9ProcAmpControlFlags.

Возвращаемые значения.

Величина	Описание
E_INVALIDARG	Неправильный номер потока.
E_POINTER	Нулевой указатель.
S_OK	Успешное выполнение.
VFW_E_NOT_CONNECTED	Контакт не подключен.

Структуры

Методы используют две структуры для операций с данными и одно перечисление, определяющее тип данных.

Структура VMR9ProcAmpControl содержит регулируемые параметры изображения. От аналогичных параметров, описанных в интерфейсе IAMVideoProcAmp, они отличаются только типом данных и уменьшенным набором регулируемых параметров.

```
typedef struct _VMR9ProcAmpControl
{
    DWORD  dwSize;
    DWORD  dwFlags;
    float  Contrast;
    float  Brightness;
    float  Hue;
    float  Saturation;
} VMR9ProcAmpControl;
```

Член структуры dwFlags определяет, какие параметры устанавливаются или возвращаются.

Вторая структура VMR9ProcAmpControlRange заполняется данными, определяющими пределы регулировок для определенного параметра, указанного членом структуры dwProperty. От аналогичных данных, описанных в интерфейсе IAMVideoProcAmp, они отличаются только типом и уменьшенным набором данных.

```
typedef struct _VMR9ProcAmpControlRange
{
    DWORD                    dwSize;
    VMR9ProcAmpControlFlags  dwProperty;
    float                    MinValue;
    float                    MaxValue;
    float                    DefaultValue;
    float                    StepSize;
} VMR9ProcAmpControlRange;
```

Перечисление VMR9ProcAmpControlFlags содержит значения флагов, определяющих тип параметров, которые возвращают или устанавливают методы.

```
typedef enum {
    ProcAmpControl9_Contrast       = 0x00000001,
    ProcAmpControl9_Brightness   = 0x00000002,
    ProcAmpControl9_Hue            = 0x00000004,
    ProcAmpControl9_Saturation   = 0x00000008,
    ProcAmpControl9_Mask           = 0x0000000F
} VMR9ProcAmpControlFlags;
```

Результаты проверки.

Для проверки функционирования описанных методов интерфейса IVMR-MixerControl9 использовался следующий код.

```
//Запрос интерфейса
  CComQIPtr<IVMRMixerControl9> pMc(m_pVRender);
  if(!pMc)
    return ;

//Структура для возврата и установки параметров
//изображения.

  VMR9ProcAmpControl ampC;

  ampC.dwSize = sizeof(VMR9ProcAmpControl);
  ampC.dwFlags = ProcAmpControl9_Brightness;

//Структура для возврата диапазона  параметров
//изображения.

  VMR9ProcAmpControlRange ampR;

  ampR.dwSize = sizeof(VMR9ProcAmpControlRange);
  ampR.dwProperty = ProcAmpControl9_Brightness;


//Возврат значений параметров
  HRESULT hr = pMc->GetProcAmpControl(0,&ampC);

//Возврат диапазона значений параметров
  hr = pMc->GetProcAmpControlRange(0,&ampR);
```

Код можно разместить в методе CCapture::SetBrightness, тогда удобно производить проверку при изменении яркости изображения.

При проверке оба метода работали некорректно. Функционирование методов было таким, что требовало тщательного изучения отклонений от описанного поведения. Подробно описывать неправильную работу методов не имеет смысла, поскольку их функционирование определяется драйвером устройства и может изменяться от версии к версии драйвера или конкретного типа устройства. Намного проще и надежней воспользоваться интерфейсом IAMVideoProcAmp фильтра сбора.