Meraka RTVC Library

Table of Contents

1. Introduction

The objective of the Real-Time Video Coding group is to research and develop intelligent multimedia software components and delivery systems that adapt to congested and low infrastructure network environments. The emphasis is on jointly-optimised or co-operative behaviour between the delivery context, as in real-time network conditions, and the video/audio encoder rate control mechanisms. Innovative solutions are sought to enhance the Internet video experience in a developing world context where bandwidth is a scarce resource.

Current research within the group is concentrating on optimal and weighted context-based bit allocation techniques and implementations within international standard video codecs. A scalable and bandwidth adaptive multimedia broadcasting platform is under development to encompass the novelty of the bit allocation research.

To this end some basic DirectShow filters such as scale, crop and colour conversion filters as well as base classes and interfaces have been developed which make filter development somewhat easier.

A part of the objective of the Real-Time Video Coding group is to utilise open source frameworks and to contribute to the open source community as well as to provide source code that is available for general usage. The filters and source code released are freely available under a BSD license.

2. Interfaces

The status and settings COM interfaces provide a uniform way of interaction with custom DirectShow filters.

The status interface is used for filter-main application communication.

The settings interface is used to configure filter parameters and to obtain the current configuration.

2.1. ISettingsInterface

The settings interface is used to configure filters before they are connected downstream. The downstream connection finalises the media type and often, such as in the scale and crop filter, the media type is modified when configuring the filter.

The interface defines the following methods:

- STDMETHOD GetParameter(const char* szParamName, char* szResult, int& nResultLength)

  Use this method to retrieve tha value of a certain parameter where szParamName is the name of the parameter, szResult is memory that has been allocated to store the result and nResultLength stores the length of the result if the call was successful and -1 if the call failed.

- STDMETHOD SetParameter(const char* szParamName, const char* szValue)

  This method is used to set filter parameters where szParamName is the name of the parameter you want to set and szValue is the value you want to set it to. One should always verify that the call was successful after callng these methods. These methods will return S_OK if successful and E_FAIL if the call failed.

Some example code follows on how to use the settings interface:

```
ISettingsInterfacePtr pSettingsInterface = pScaler;
char szParamValue[255];
int nLength = 0;
```

```
HRESULT hrTemp = pSettingsInterface->GetParameter("targetwidth", szParamValue, nLength);
hrTemp = pSettingsInterface->GetParameter("targetheight", szParamValue, nLength);

hrTemp = pSettingsInterface->SetParameter("targetwidth", "800");
hrTemp = pSettingsInterface->SetParameter("targetheight", "600");
```

2.2. IStatusInterface

The status interface stores errors that have occured. It also has a hook to the the update window

- SetFriendlyID(long lId)

  This method should be called on each custom filter in the application so that we can identify filters in the main application thread. The default value is -1.

- GetFriendlyID(long& lID)

  This method can be called to obtain the ID of the filter provided it has been set previously.

- SetLastError(std::string sError, bool bNotifyApplication) – this method should be called from inside the filter. If an error occurrs when applying somekind of transform to the input data, this method should be called with a helpful error message so that the main application can query the filter about the error.

- If bNotifyApplication is true and an event sink has been set on the filter, the filter will notify the main application with event code WM_TM_GRAPHNOTIFY (EC_USER + 5). The main application must provide a handler and the friendly ID of the filter should have been set previously such that the application can identify which filter the notification came from.

- GetLastError(std::string& sError)

  This method retrieves the last error that occurred in the filter

- SetMediaEventSink(IMediaEventSink* pEventSink)

  Set the event sink of a filter using this method so that the filter can notify the main application thread about occurrences in the filter.

3. Base Classes

3.1. Using the CCustomBaseFilter class

The CCustomBaseFilter class just wraps the settings and status interfaces to create a base Filter class. The two pure virtual methods InitialiseInputTypes and ApplyTransform must be be overridden when using this filter as a base class.

- **Implement InitialiseInputTypes()**

  The InitialiseInputTypes should be called inside your sub class constructor and is used to add input types to the filter. Input types can be added to a filter using the AddInputType method:

```
AddInputType(&MEDIATYPE_Video, &MEDIASUBTYPE_RGB24, &FORMAT_VideoInfo);
```

- **Implement ApplyTransform(BYTE* pBufferIn, BYTE* pBufferOut)**

  This method receives an input buffer and the transformed media should be written to the output buffer. The size of the transformed data in bytes should be returned. Parameters such as the dimensions of the input image can be accessed via protected member variables of the CCustomBaseFilter class such as m_nInWidth and m_nInHeight.

- **The GetMediaType method must be overridden from CBaseFilter in the usual fashion**

- **The DecideBufferSize should also be overridden in the usual fashion**

3.2. Using the CMultiIOBaseFilter class

3.2.1. Overview

The CMultiIOBaseFilter class follows the same design as the CTransformFilter class. The only difference is that it allows the implementer of the subclass to easily create a filter with multiple input and output pins and takes care of all the base work needed. This should make it easy for anyone used to writing standard transform filters to reuse this class as a basis for writing transform filters with *x* input pins and *y* output pins.

These base classes take care of:

- Memory management of input and output pins (Covering initial and subsequent requirements)
- Acceptable media type management on a per pin basis
- Providing an easy way to write transform filters with more than one input/output pin

Similarly to the CTransformFilter class, CBaseInputPin and CBaseOutputPin have been subclassed and further responsibility has been delegated to the CMultiIOBaseFilter class.

In the last release output queues have been added to each output pin to avoid concurrency issues.

This base class does **NOT** take care of any application logic pertaining to the multiplexing, etc of the input streams, etc. The code to do this remains the responsibility of the application developer using the super class, it simply allows one to create a basic filter with multiple input/output pins effortlessly.

3.2.2. Usage:

On the development the following steps are necessary to use this base class

- **Create subclass of CMultiIOBaseFilter**

  Extend the base class as is illustrated in the example project.

- **Call Initialise in your subclass constructor**

  This method calls the virtual method needed to initialise the acceptable types, subtypes and formats on the input and output pins of the filter, as well as initial number of input and output pins

- **Override InitialNumberOfInputPins and InitialNumberOfOutputPins**

  The default number of input and output pins is 1. Override these methods if this is not suitable for your application

- **Override InitialiseInputTypes and InitialiseOutputTypes**

  Call the AddInputType and AddOutputType method with acceptable media types for your filter's input and output pins.

- **Override OnFullCreateMoreInputs and OnFullCreateMoreOutputs**

  These methods determine whether new inputs/outputs are created once all available ones have been used.

- **Override DecideBufferSize**

  This method is very similar to the CTransformFilter method which needs to be overridden. The extra parameter denotes which output pin the buffer size is being decided for.

- **Override GetMediaType**

  This method again is very similar to the CTransformFilter equivalent. The extra parameter again denotes which output pin is being queried for its media type.

3.3. Building

The base classes and interfaces are all contained in the DSCustomFilterBase project which builds a static library. Be sure to link the DirectShow base class library into the projects. Once you've built the static DSCustomFilterBase project lib, be sure to link this into your own custom filter DLL.

4. Filters

4.1. Libraries

At present only parts of the image utils library have been released.

4.1.1. ImageUtils Library

The image utils project is a static library which needs to be linked in to the scale, crop and RGB conversion filters when building them.

4.2. Using the ScaleFilter

4.2.1. Description

The scale filter is a standard DirectShow filter that scales images to the specified target width and height.

Target width and height can be configured/obtained programmatically by getting the COM ISettingsInterface (IID_ISettingsInterface) from the filter and setting "targetwidth" and "targetheight" using the GetParameter and SetParameter methods.

	Type	Subtype	Format
1)	MEDIATYPE_Video	MEDIASUBTYPE_RGB24	FORMAT_VideoInfo
2)	MEDIATYPE_Video	MEDIASUBTYPE_YUV420P (Custom)	FORMAT_VideoInfo

Table 1: Scale Filter Input Types

4.2.2. Limitiations:

The target dimensions can not be greater than twice the original image dimensions.

At present the scale filter only supports RGB24 media and out custom YUV4:2:0 Planar type.

4.3. Using the CropFilter

4.3.1.Description

The crop filter is a standard DirectShow filter that crops images of types RGB24 and RGB32 to the specified target width and height.

The Crop filter can be configured via the ISettingsInterface using the following parameters:

Target dimensions:

- targetwidth: width of the output image
- targetheight: height of the output image

Ratios:

The two grouped values determine a cropping ratio. A ratio of 2:1 (leftcropratio:rightcropratio) would result in the CropFilter cropping twice as much on the left as on the right side of the image.

One of the grouped values is allowed to be zero, in which case nothing is cropped from that side of the image.

- topcropratio and bottomcropratio:
- leftcropratio and rightcropratio

	Type	Subtype	Format
1)	MEDIATYPE_Video	MEDIASUBTYPE_RGB24	FORMAT_VideoInfo
2)	MEDIATYPE_Video	MEDIASUBTYPE_RGB32	FORMAT_VideoInfo

Table 2: Crop Filter Input Types

4.4. Using the RGB YUV420P converter filters

4.4.1. Description

The RGBtoYUV420P conversion filter converts input images of type RGB24 and RGB32 to a custom YUV 4:2:0 Planar format in which the Luminance and Chrominance values are stored in separate planes.

This filter is currently being used to convert video streams to a YUV format to prepare the stream for H263 encoding.

Note that if you build the filters with the _BUILD_FOR_SHORT preprocessor directive, each luminance or chrominance value is stored in 16-bits. If the directive is undefined, 8 bits are used.

The two converters as well as the ImageUtils library should always use the same setting in this regard, else the conversion back to RGB24 will fail.

	Type	Subtype	Format
1)	MEDIATYPE_Video	MEDIASUBTYPE_RGB24	FORMAT_VideoInfo
2)	MEDIATYPE_Video	MEDIASUBTYPE_RGB32	FORMAT_VideoInfo

Table 3: RGB to YUV420P Filter Input Types

	Type	Subtype	Format
1)	MEDIATYPE_Video	MEDIASUBTYPE_YUV420P (Custom)	FORMAT_VideoInfo

Table 4: YUV420P to RGB Filter Input Types

5. Live RTSP Audio Streaming Server (Beta)

5.1. Description

The RTSP audio streaming server demonstrates the use of integrating the live555 library with DirectShow. The live555 library is an open-source RTP/RTCP/RTSP library. The application simply uses DirectShow to capture live audio and then streams this using RTSP/RTP. The stream can then be listened to via the VLC client. Unfortunately Windows Media Player is not standard compliant and so can not be used as a client.

This project shows:

- How to integrate DirectShow and the live555 libraries.
- How to configure the ACMWrapper audio compression filter.
- RTSP aspects of the live555 library.

5.1.1. Build instructions

1. Download the latest version of the LiveMedia library from http://www.live555.com
2. Build the library and copy the lib file into the lib directory of the solution
3. Build the DirectShowAudioStreamingServer project.
4. Launch the executable
5. The stream can be listened using VLC http://www.videolan.org/vlc/

5.1.2. Limitations

For testing purposes the audio capture device of a webcam was used. The output was compressed in order to obtain small packet sizes that are suitable for streaming using basic provided live555 classes. Modifications need to be done to the code to cater for audio data of a higher data rates.

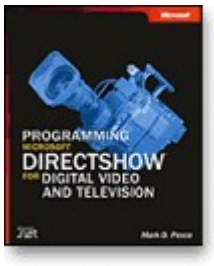
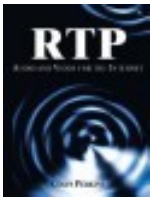
This project really just serves as an example on how to get started integrating DirectShow with the LiveMedia library.

6. Contact:

Please send comments, feedback, suggestions to rtvc (at) meraka (dot) org (dot) za

7. Links:

DirectShow		
MSDN		http://msdn2.microsoft.com/en-us/library/ms783323(VS.85).aspx
DirectShow forum		http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=129&SiteID=1
DirectShow News Group		http://groups.google.com/group/microsoft.public.win32.programmer.directx.video/topics?start=0&sa=N
Geraint Davies' Page	GMFBridge	http://www.gdcl.co.uk/
Alessandro Angeli's Programming FAQ	General	http://www.riseoftheants.com/mmx/faq.htm
The March Hare's FAQ	General	http://tmhare.mvps.org/faqs.htm
ChrisNet	Audio	http://www.chrisnet.net/code.htm

DirectShow training	Training	http://www.roujansky.com/monBlog/pivot/entry.php?id=11#body
Programming Microsoft DirectShow for Digital Video and Television 		http://www.microsoft.com/MSPress/books/6381.aspx
RTP/RTCP/RTSP		
Live555 Streaming Media library	RTP/RTCP/RTSP – Extensive streaming library with good support via mailing list. Slightly higher entry level to get started. Emphasis on compliance with standards.	http://www.live555.com
JRTPLIB	RTP/RTCP Lower entry level to get started	http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jrtplib
RTP RFC		http://rfc.dotsrc.org/rfc/rfc3550.html
RTSP RFC		http://rfc.dotsrc.org/rfc/rfc2326.html
RTP – Audio and Video for the Internet 		http://csperkins.org/rtp-book.html

8. Credits:

Thanks to dume777 from the MS DirectShow forum for picking up a bug in the Scale filter.