.NET Wrapper of FFmpeg libraries

Author: Maxim Kartavenkov

Document History

22/06/2022 - Initial revision

Version History

1.0.2207.0201 - Initial release.

Introduction

There are lots of attempts to create .NET wrappers of FFmpeg libraries. I saw different implementations, mostly just performing calling ffmpeg.exe with prepared arguments and parsing outputs from it. More productive results have some basic classes which can perform just one of few tasks of ffmpeg and whose classes are exposed as COM objects or regular C# classes, but their functionality is very limited and does not fully perform in .NET what actually can be done with full access to ffmpeg libraries in C++. I also have seen a totally .NET "unsafe" code implementation of wrapper, so the C# code must be also marked as "unsafe" and the functions of such wrapper have lots of pointers like "void *" and other similar types. So, usage of such code is also problematic and I am just talking about C# and not take other languages such as VB.NET.
Current implementation of .NET wrapper is representation of all exported structures from ffmpeg libraries, along with APIs which can operate with those objects from managed code either from C# or VB.NET or even Delphi.NET. It is not just calls of certain APIs of a couple predefined objects - all API exposed as methods. You may ask a good question: "how is that possible and why wasn't that done before?", and the answers you can find in current documentation.

Objective

Article describes created .NET wrapper of FFmpeg libraries. It covers basic architecture aspects along with information: how that was done and what issues are solved during implementation. It contains a lot of code samples for different aims of usage of ffmpeg library. Each code is workable and has a description with the output results. It will be interesting for those who want to start learning how to work with the ffmpeg library or for those who plan to use it in production. Even for those who are sure that he knows the ffmpeg library from C++ side, this information also will be interesting. And if you are looking for a solution of how to integrate the whole ffmpeg libraries power directly into your .NET application - then you are on the right way.

How to read the documentation

The first part describes wrapper library architecture, main features, basic design of the classes, what kind of issues appear during implementation and how they were solved. Here is also introduced the base classes of the library with description of their fields and purposes. In the second part you can find core wrapper classes of the most ffmpeg libraries structures. Each class documentation contains basic description and one or few sample code of usage for this class. Part three introduces some advanced topics of the library. In additional documentation include description of the example C# projects of the .NET wrapper of native C++ ffmpeg examples. There are also a few of my own sample projects, which, I think, will be interesting to look for FFmpeg users. Last part contains a description about how to build the project, license information and distribution notes.

Supported FFmpeg versions

Current implementation supports FFmpeg libraries from 4.2.2 and higher. By saying higher I mean that it controls some aspects of newer versions with some definitions and/or dynamic library linking. For example native ffmpeg libraries have definitions of what APIs are exposed and what fields each structure contains and that is properly handled in the current structure.

Architecture

By saying that implementation of the current .NET wrapper isn't simple or not easy it means say nothing. The result of the ability of this library to exist and work properly is in its architecture. Project is written on C++/CLI and compiled as a standalone .NET DLL which can be added as reference into .NET applications written in any language. Answers why decisions were made in favor of C++/CLI you can find in this documentation.
Library contains couple h/cpp files which are mostly represent underlying imported FFmpeg dll library:

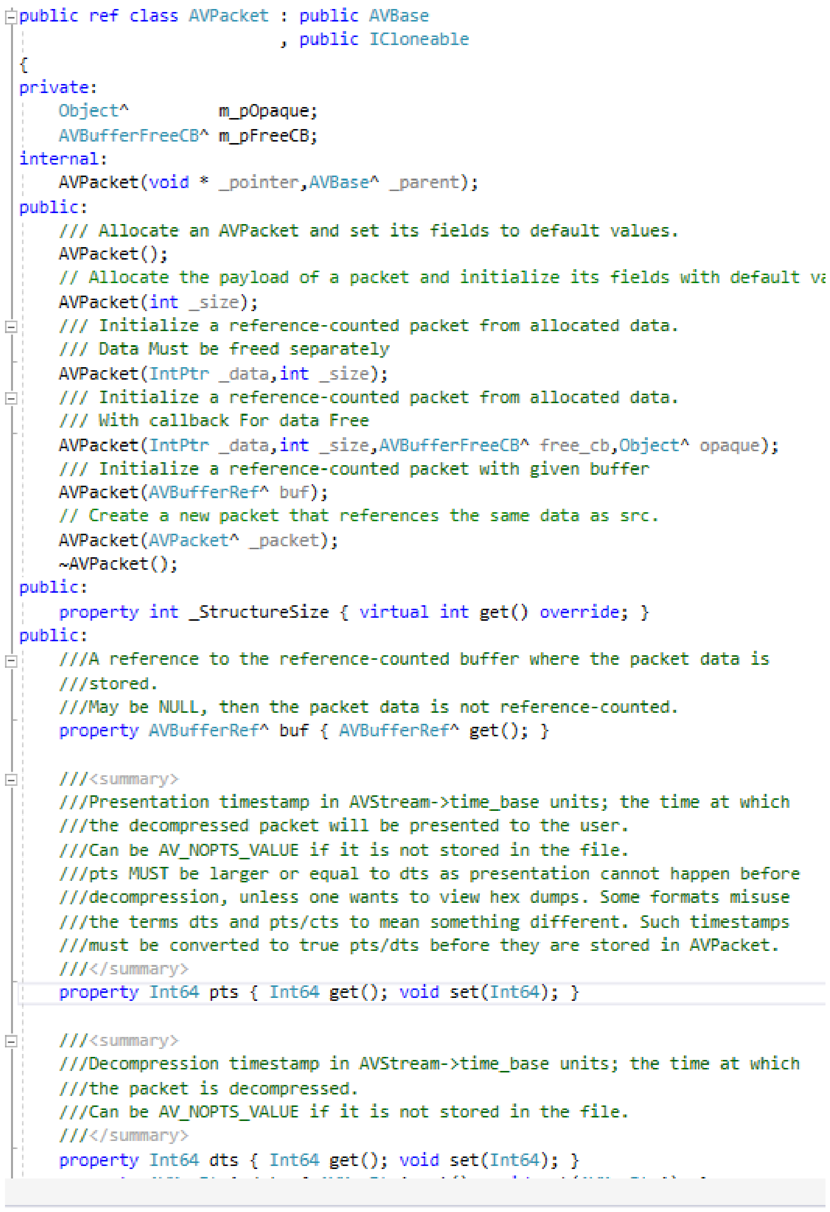
AVCore.h/cpp	Contains core classes enumerations and structures. Also contains imported classes and enumerations from AVUtil library
AVCodec.h/cpp	Contains classes and enumerations for wrapper from AVCodec Library
AVFormat.h/cpp	Contains classes and enumerations for wrapper from AVFormat Library
AVFilter.h/cpp	Contains classes and enumerations for wrapper from AVFilter Library
AVDevice.h/cpp	Contains classes and enumerations for wrapper from AVDevice Library
SWResample.h/cpp	Contains classes and enumerations for wrapper from SWResample Library
SWScale.h/cpp	Contains classes and enumerations for wrapper from SWScale Library
Postproc.h/cpp	Contains classes and enumerations for wrapper from Postproc Library

Core Architecture

This topic contains the description of most benefits and a design guide of major parts of library architecture from base classes design to core aspects and resolved issues. This part along with C# also contains code snippets from wrapper library implementation which is in C++/CLI.

Classes

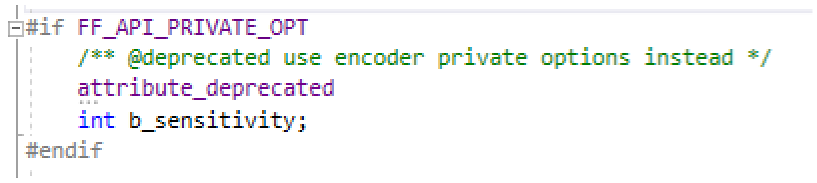
Most classes in the wrapper library implementation rely on structure or enumeration in ffmpeg libraries. Class has the same name as its native underlying structure. For example: the managed class ***AVPacket*** in the library represents the ***AVPacket*** structure of the libavcodec: it contains all fields of the structure along with exposed related methods. The fields implemented not as regular structure fields, but as managed properties:



```
public ref class AVPacket : public AVBase
                          , public ICloneable
{
private:
    Object^          m_pOpaque;
    AVBufferFreeCB^ m_pFreeCB;
internal:
    AVPacket(void * _pointer,AVBase^ _parent);
public:
    /// Allocate an AVPacket and set its fields to default values.
    AVPacket();
    // Allocate the payload of a packet and initialize its fields with default va
    AVPacket(int _size);
    /// Initialize a reference-counted packet from allocated data.
    /// Data Must be freed separately
    AVPacket(IntPtr _data,int _size);
    /// Initialize a reference-counted packet from allocated data.
    /// With callback For data Free
    AVPacket(IntPtr _data,int _size,AVBufferFreeCB^ free_cb,Object^ opaque);
    /// Initialize a reference-counted packet with given buffer
    AVPacket(AVBufferRef^ buf);
    // Create a new packet that references the same data as src.
    AVPacket(AVPacket^ _packet);
    ~AVPacket();
public:
    property int _StructureSize { virtual int get() override; }
public:
    ///A reference to the reference-counted buffer where the packet data is
    ///stored.
    ///May be NULL, then the packet data is not reference-counted.
    property AVBufferRef^ buf { AVBufferRef^ get(); }

    ///<summary>
    ///Presentation timestamp in AVStream->time_base units; the time at which
    ///the decompressed packet will be presented to the user.
    ///Can be AV_NOPTS_VALUE if it is not stored in the file.
    ///pts MUST be larger or equal to dts as presentation cannot happen before
    ///decompression, unless one wants to view hex dumps. Some formats misuse
    ///the terms dts and pts/cts to mean something different. Such timestamps
    ///must be converted to true pts/dts before they are stored in AVPacket.
    ///</summary>
    property Int64 pts { Int64 get(); void set(Int64); }

    ///<summary>
    ///Decompression timestamp in AVStream->time_base units; the time at which
    ///the packet is decompressed.
    ///Can be AV_NOPTS_VALUE if it is not stored in the file.
    ///</summary>
    property Int64 dts { Int64 get(); void set(Int64); }
```

Properties

The class which relies on native structure is inherited from the ***AVBase*** class. The classes represent the pointer to underlying structure. Exposing structure fields as properties gives the ability to hide whole internals and this allows to support different versions of ffmpeg by encapsulation implementations. For example: in certain ffmpeg versions some fields can be hidden in structure and accessed via Options APIs, like the "*b_sensitivity*" field of ***AVCodecContext*** it may be removed during build by having **FF_API_PRIVATE_OPT** definition set:

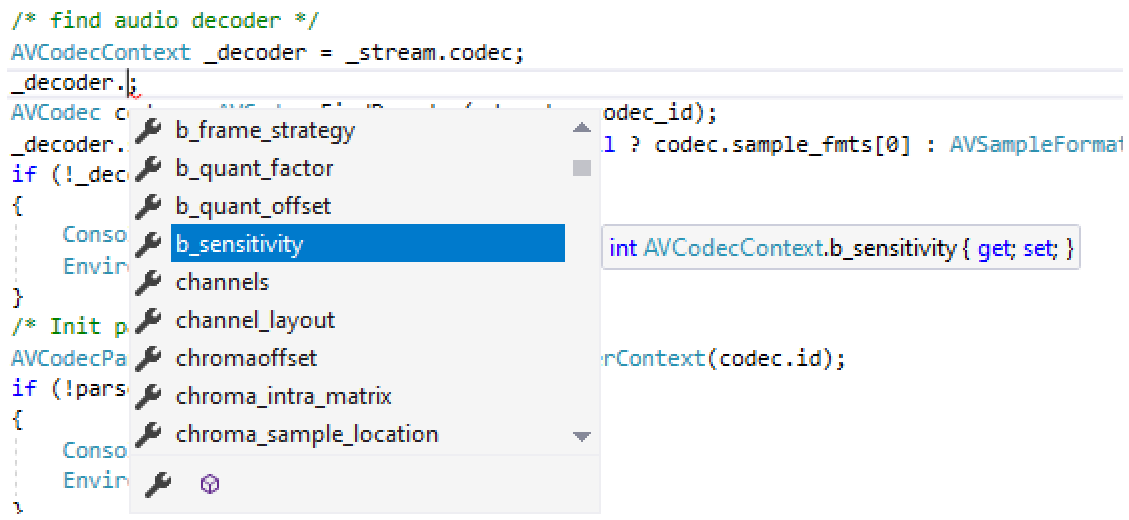


```
#if FF_API_PRIVATE_OPT
    /** @deprecated use encoder private options instead */
    attribute_deprecated
    int b_sensitivity;
#endif
```

In library you only see the property and internally it choose the way to access it:

```
int FFmpeg::AVCodecContext::b_sensitivity::get()
{
    __int64 val = 0;
    if (AVERROR_OPTION_NOT_FOUND == av_opt_get_int(m_pPointer, "b_sensitivity", 0, &val))
    {
#if FF_API_PRIVATE_OPT
        val = ((::AVCodecContext*)m_pPointer)->b_sensitivity;
#endif
    }
    return (int)val;
}
void FFmpeg::AVCodecContext::b_sensitivity::set(int value)
{
    if (AVERROR_OPTION_NOT_FOUND == av_opt_set_int(m_pPointer, "b_sensitivity", (int64_t)value, 0))
    {
#if FF_API_PRIVATE_OPT
        ((::AVCodecContext*)m_pPointer)->b_sensitivity = (int)value;
#endif
    }
}
```

In same time in C# code:



```
/* find audio decoder */
AVCodecContext _decoder = _stream.codec;
_decoder.;
AVCodec c                                    odec_id);
_decoder.                                    l ? codec.sample_fmts[0] : AVSampleFormat
if (!_dec
{
    Conso
    Envir
}
/* Init p
AVCodecPa                                    rContext(codec.id);
if (!pars
{
    Conso
    Envir
}
```

```
b_frame_strategy
b_quant_factor
b_quant_offset
b_sensitivity
channels
channel_layout
chromaoffset
chroma_intra_matrix
chroma_sample_location
```

```
int AVCodecContext.b_sensitivity { get; set; }
```

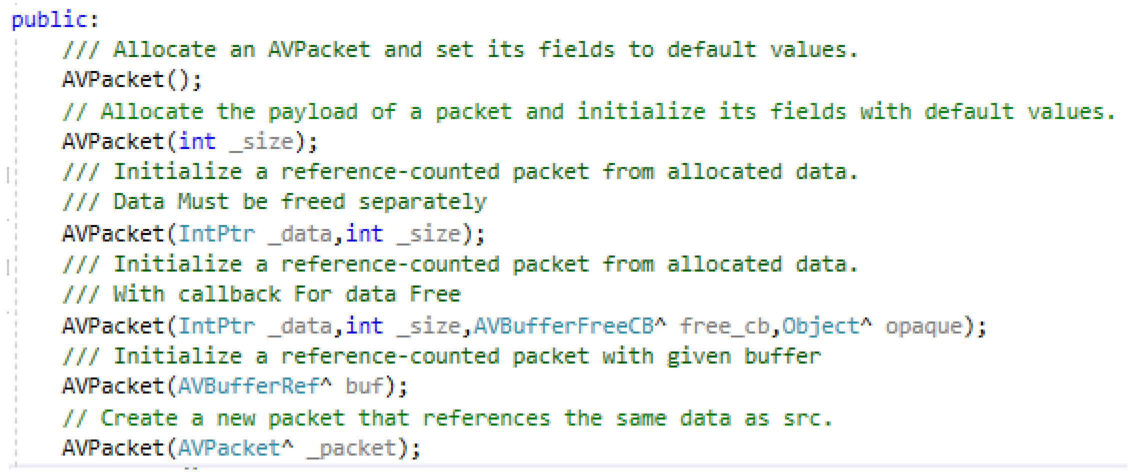
Or, in some versions of ffmpeg, fields may be not present at all. For example “*refcounted_frames*” of ***AVCodecContext***:

```
int FFmpeg::AVCodecContext::refcounted_frames::get()
{
    __int64 val = 0;
    if (AVERROR_OPTION_NOT_FOUND == av_opt_get_int(m_pPointer, "refcounted_frames", 0, &val))
    {
#if (LIBAVCODEC_VERSION_MAJOR < 59)
        val = ((::AVCodecContext*)m_pPointer)->refcounted_frames;
#endif
    }
    return (int)val;
}
void FFmpeg::AVCodecContext::refcounted_frames::set(int value)
{
    if (AVERROR_OPTION_NOT_FOUND == av_opt_set_int(m_pPointer, "refcounted_frames", (int64_t)value, 0))
    {
#if (LIBAVCODEC_VERSION_MAJOR < 59)
        ((::AVCodecContext*)m_pPointer)->refcounted_frames = (int)value;
#endif
    }
}
```

I’m sure you already think that it is very cool and no need to worry about the background. Such implementation is known as Facade pattern.

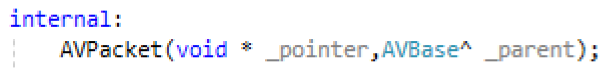
Constructors

If you know ffmpeg libraries API’s you can agree that it contains APIs for allocating structures. There can be different API’s for every structure for such purposes. In wrapper classes all objects have a constructor - so users no need to think about what API to use; and even more: the constructor can encapsulate a list of FFmpeg API calls to perform full object initialization. Different FFmpeg API as well as different object initialization are done as separate constructor with different arguments. As an example constructor of ***AVPacket*** object:



```
public:
    /// Allocate an AVPacket and set its fields to default values.
    AVPacket();
    // Allocate the payload of a packet and initialize its fields with default values.
    AVPacket(int _size);
    /// Initialize a reference-counted packet from allocated data.
    /// Data Must be freed separately
    AVPacket(IntPtr _data,int _size);
    /// Initialize a reference-counted packet from allocated data.
    /// With callback For data Free
    AVPacket(IntPtr _data,int _size,AVBufferFreeCB^ free_cb,Object^ opaque);
    /// Initialize a reference-counted packet with given buffer
    AVPacket(AVBufferRef^ buf);
    // Create a new packet that references the same data as src.
    AVPacket(AVPacket^ _packet);
```

There is a special case of an internal constructor which is not exposed outside of the library. Most objects contain it. This is done for internal object management which I describe later in this document:



```
internal:
    AVPacket(void * _pointer,AVBase^ _parent);
```

Destructors

Also for users no need to worry about releasing memory and free object data. This is all done in object destructors and internally in finalizers. In C# that is implemented automatically as an ***IDisposable*** interface of an object. So, once an object is not needed it is good to call *Dispose* method - to clear all dependencies and free allocated object memory. Users also have no need to think which ffmpeg API to call to free data and deallocate object resources as It is all handled by the wrapper library. In case the user forgot to call *Dispose* method: object and its memory will be freed by finalizer, but it is good practice to call *Dispose* directly in code.

```
// a wrapper around a single output AVStream
public class OutputStream
{
    public AVStream st;
    public AVCodecContext enc;

    /* pts of the next frame that will be generated */
    public long next_pts;
    public int samples_count;

    public AVFrame frame;
    public AVFrame tmp_frame;

    public float t, tincr, tincr2;

    public SwsContext sws_ctx;
    public SwrContext swr_ctx;
}

static void close_stream(OutputStream ost)
{
    if (ost.enc != null) ost.enc.Dispose();
    if (ost.frame != null) ost.frame.Dispose();
    if (ost.tmp_frame != null) ost.tmp_frame.Dispose();
    if (ost.sws_ctx != null) ost.sws_ctx.Dispose();
    if (ost.swr_ctx != null) ost.swr_ctx.Dispose();
}
```

Also it is necessary to keep in mind that such objects as ***AVPacket*** or ***AVFrame*** can contain buffers which are allocated internally by ffmpeg API and which should be freed separately see *av_packet_unref* and *av_frame_unref* APIs. For such purposes those objects expose methods named *Free*, so, don't forget to call it to avoid memory leaks.

Enumerations

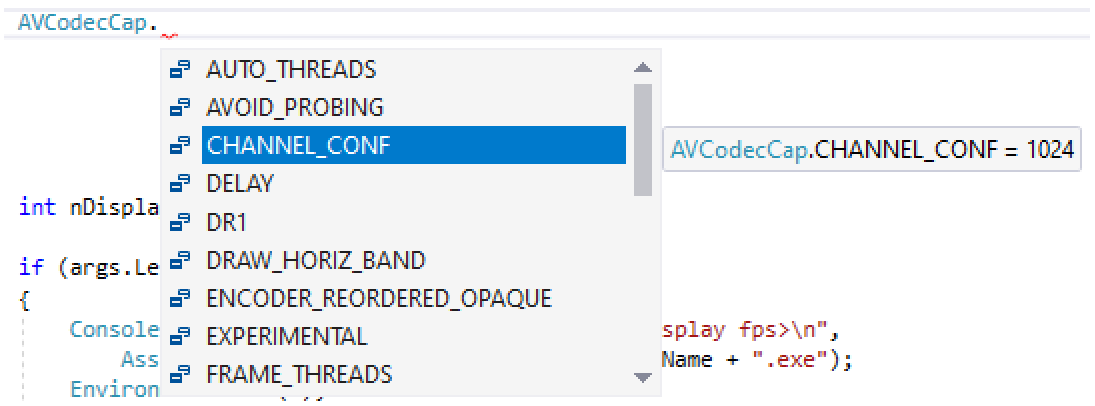
Some FFmpeg library definitions or enumerations are done as .NET enum classes - this way it is easy to find what value can be set in certain fields of the structures. For example **AV_CODEC_FLAG_***, **AV_CODEC_FLAG2_*** or **AV_CODEC_CAP_*** definitions:

```
#define AV_CODEC_CAP_DR1                    (1 <<  1)
#define AV_CODEC_CAP_TRUNCATED              (1 <<  3)
```

Designed as .NET enum for better access from code:

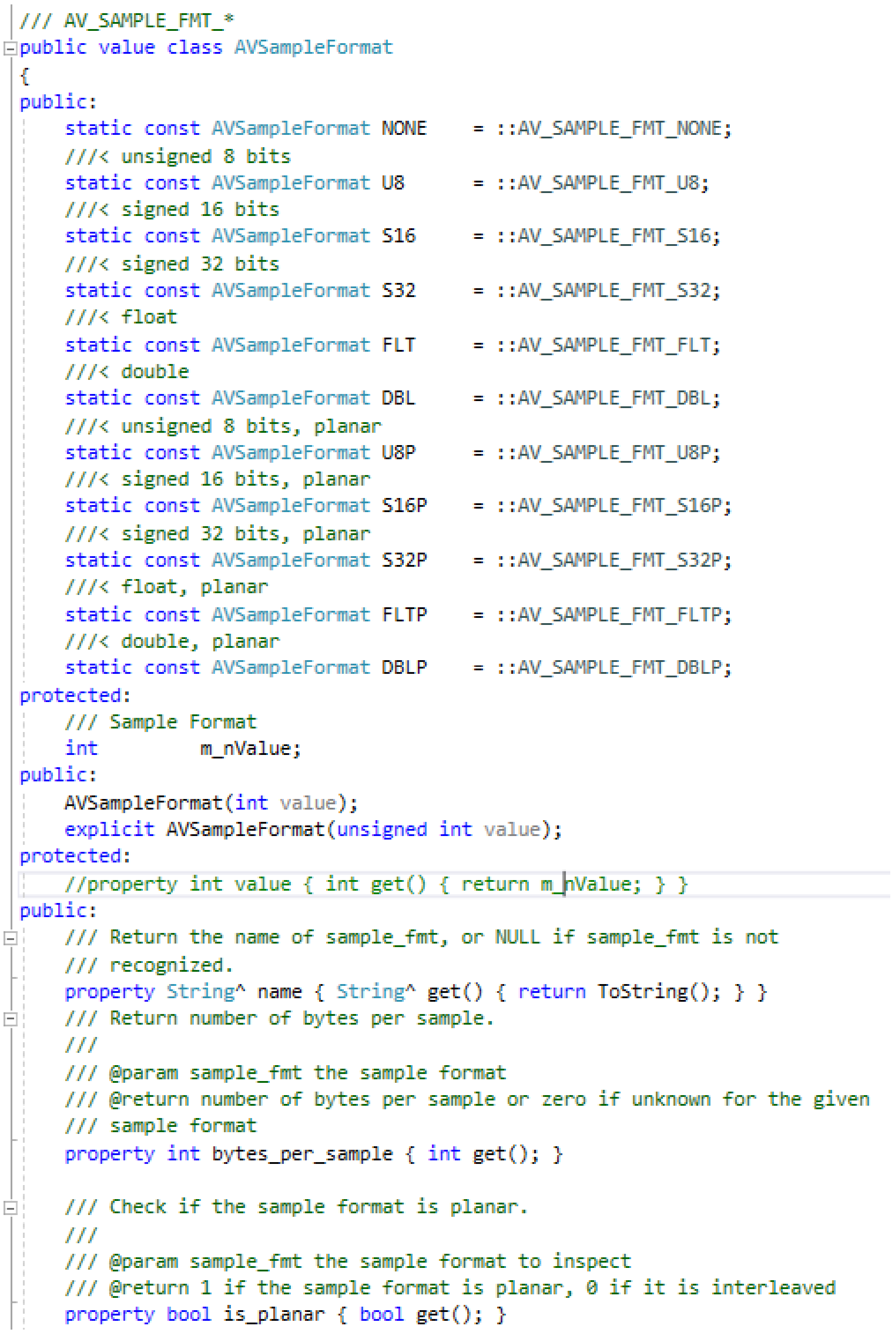
```
// AV_CODEC_CAP_*
[Flags]
public enum class AVCodecCap : UInt32
{
    ///< Decoder can use draw_horiz_band callback.
    DRAW_HORIZ_BAND = (1 <<  0),
    ///<summary>
    /// Codec uses get_buffer() for allocating buffers and supports custom allocators.
    /// If not set, it might not use get_buffer() at all or use operations that
    /// assume the buffer was allocated by avcodec_default_get_buffer.
    ///</summary>
    DR1 = (1 <<  1),
    TRUNCATED = (1 <<  3),
```

In C# this looks:



```
AVCodecCap.
            AUTO_THREADS
            AVOID_PROBING
            CHANNEL_CONF              AVCodecCap.CHANNEL_CONF = 1024
            DELAY
int nDispla DR1

if (args.Le DRAW_HORIZ_BAND
{           ENCODER_REORDERED_OPAQUE
    Console EXPERIMENTAL          splay fps>\n",
        Ass FRAME_THREADS         Name + ".exe");
    Environ
```

Some enumerations or definitions from FFmpeg are done as regular .NET classes. In that case class also expose some useful methods which rely to specified enumerator type:



```
/// AV_SAMPLE_FMT_*
public value class AVSampleFormat
{
public:
    static const AVSampleFormat NONE    = ::AV_SAMPLE_FMT_NONE;
    ///< unsigned 8 bits
    static const AVSampleFormat U8      = ::AV_SAMPLE_FMT_U8;
    ///< signed 16 bits
    static const AVSampleFormat S16     = ::AV_SAMPLE_FMT_S16;
    ///< signed 32 bits
    static const AVSampleFormat S32     = ::AV_SAMPLE_FMT_S32;
    ///< float
    static const AVSampleFormat FLT     = ::AV_SAMPLE_FMT_FLT;
    ///< double
    static const AVSampleFormat DBL     = ::AV_SAMPLE_FMT_DBL;
    ///< unsigned 8 bits, planar
    static const AVSampleFormat U8P     = ::AV_SAMPLE_FMT_U8P;
    ///< signed 16 bits, planar
    static const AVSampleFormat S16P    = ::AV_SAMPLE_FMT_S16P;
    ///< signed 32 bits, planar
    static const AVSampleFormat S32P    = ::AV_SAMPLE_FMT_S32P;
    ///< float, planar
    static const AVSampleFormat FLTP    = ::AV_SAMPLE_FMT_FLTP;
    ///< double, planar
    static const AVSampleFormat DBLP    = ::AV_SAMPLE_FMT_DBLP;
protected:
    /// Sample Format
    int         m_nValue;
public:
    AVSampleFormat(int value);
    explicit AVSampleFormat(unsigned int value);
protected:
    //property int value { int get() { return m_nValue; } }
public:
    /// Return the name of sample_fmt, or NULL if sample_fmt is not
    /// recognized.
    property String^ name { String^ get() { return ToString(); } }
    /// Return number of bytes per sample.
    ///
    /// @param sample_fmt the sample format
    /// @return number of bytes per sample or zero if unknown for the given
    /// sample format
    property int bytes_per_sample { int get(); }

    /// Check if the sample format is planar.
    ///
    /// @param sample_fmt the sample format to inspect
    /// @return 1 if the sample format is planar, 0 if it is interleaved
    property bool is_planar { bool get(); }
```

Such classes expose all required things to work with such classes as value types along with implicit types conversions:

```
public:
    static operator int(AVSampleFormat a) { return a.m_nValue; }
    static explicit operator unsigned int(AVSampleFormat a) { return (unsigned int)a.m_nValue; }
    static operator AVSampleFormat(int a) { return AVSampleFormat(a); }
    static explicit operator AVSampleFormat(unsigned int a) { return AVSampleFormat((int)a); }
internal:
    static operator ::AVSampleFormat(AVSampleFormat a) { return (::AVSampleFormat)a.m_nValue; }
    static operator AVSampleFormat(::AVSampleFormat a) { return AVSampleFormat((int)a); }
```

Libraries

Library contains special static classes which can identify used libraries - their version, build configuration and license:

```
// LibAVCodec
public ref class LibAVCodec
{
private:
    static bool s_bRegistered = false;
private:
    LibAVCodec();
public:
    // Return the LIBAVCODEC_VERSION_INT constant.
    static property UInt32 Version { UInt32 get(); }
    // Return the libavcodec build-time configuration.
    static property String^ Configuration { String^ get(); }
    // Return the libavcodec license.
    static property String^ License { String^ get(); }
public:
    // Register the codec codec and initialize libavcodec.
    static void Register(AVCodec^ _codec);
    // Register all the codecs, parsers and bitstream filters which were enabled at
    // configuration time. If you do not call this function you can select exactly
    // which formats you want to support, by using the individual registration
    // functions.
    static void RegisterAll();
};
```

Each FFmpeg imported library has a similar class with described fields.
Those library classes can contain static initialization methods which are relay to internal calls of such APIs as: *avcodec_register_all*, *av_register_all* and others. Those methods may be called manually by the user, but the library manages to call them internally if needed. The API which is deprecated and may not be present as exported from ffmpeg library - internally designed dynamically linking. How that is done is described later in this document.

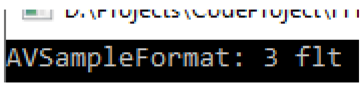
ToString() implementation

Some classes or enumerations have overridden the string conversion method: "*ToString()*". In such a case it returns the string value which is output by the related FFmpeg API to get a readable description of the type or structure content:

```
String^ FFmpeg::AVSampleFormat::ToString()
{
    auto p = av_get_sample_fmt_name((::AVSampleFormat)m_nValue);
    return p != nullptr ? gcnew String(p) : nullptr;
}
```

In example it calls that method to get default string representation of the sample format value:

```
var fmt = AVSampleFormat.FLT;
Console.WriteLine("AVSampleFormat: {0} {1}",(int)fmt,fmt);
```



```
AVSampleFormat: 3 flt
```

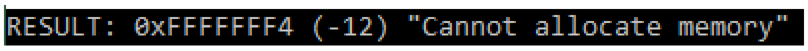
AVRESULT

As you may know, most of FFmpeg API returns integer value, which may mean error or number processed data or have any other meaning. For the good understanding of the returned value wrapper library have the class with the name ***AVRESULT***. It is the same as the **AVERROR** macro in FFmpeg, just designed as a class object with exposed useful fields and with .NET pluses. That class can be used as integer value without direct type casting as it is done as value type in its basis. By using the class it is possible to get an error description string from the returned API value as the class has overridden the *ToString()* method so it can be easily used in string conversion:

```
/* Write the stream header, if any. */
int ret = oc.WriteHeader(opt);
if (ret < 0) {
    Console.WriteLine("Error occurred when opening output file: {0}\n",(AVRESULT)ret);
    Environment.Exit(1);
}
```

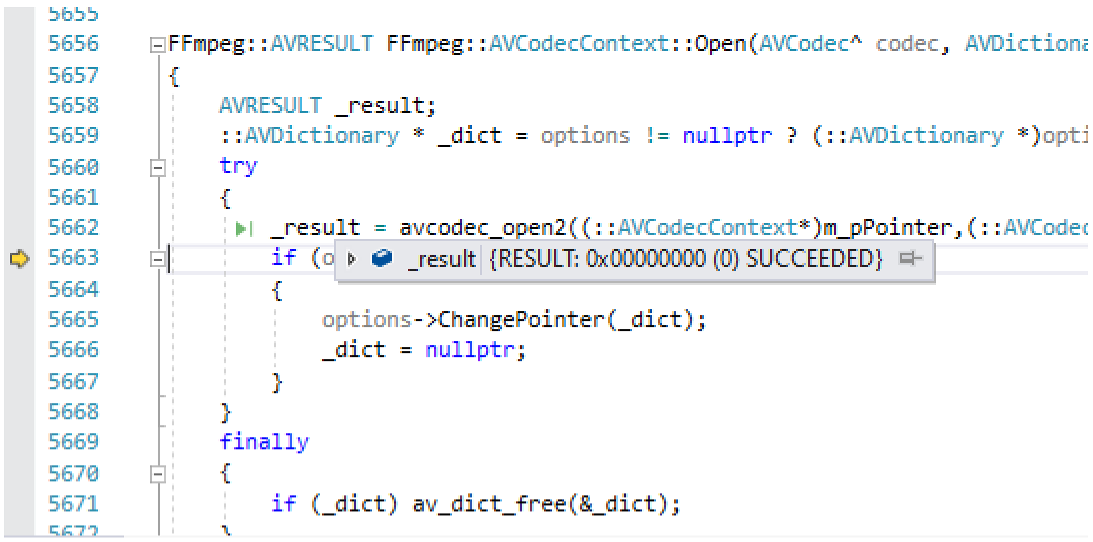
Given error description string is good for understanding what result have in execution of the method:

```
Console.WriteLine(AVRESULT.ENOMEM.ToString());
```



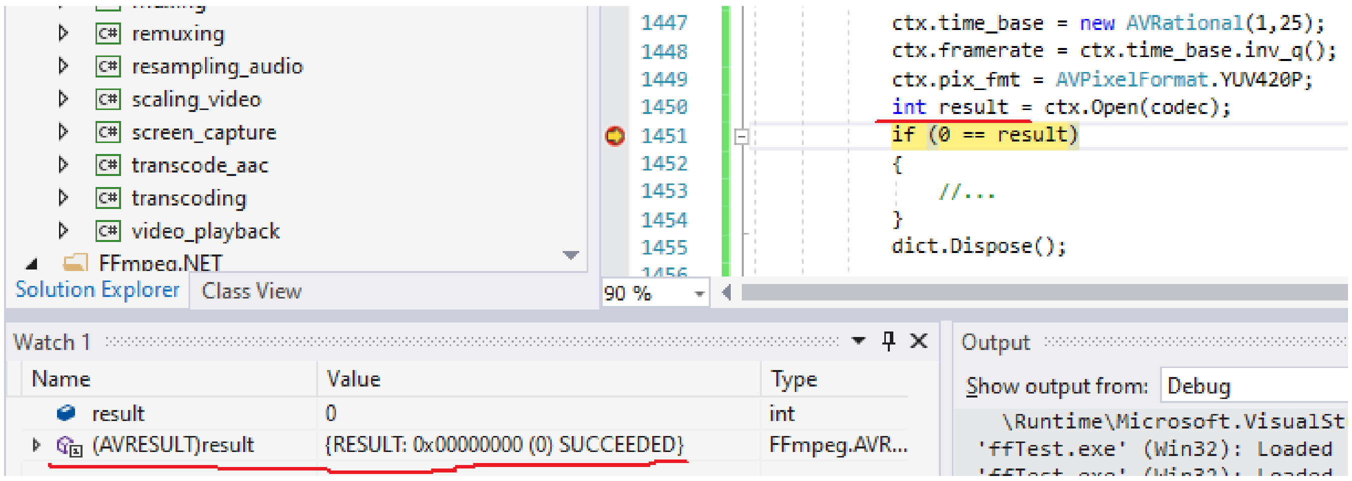
```
RESULT: 0xFFFFFFF4 (-12) "Cannot allocate memory"
```

If ***AVRESULT*** type used in code then possible to see result string under debugger:



```
FFmpeg::AVRESULT FFmpeg::AVCodecContext::Open(AVCodec^ codec, AVDictiona
{
    AVRESULT _result;
    ::AVDictionary * _dict = options != nullptr ? (::AVDictionary *)opti
    try
    {
        _result = avcodec_open2((::AVCodecContext*)m_pPointer,(::AVCode
        if (o   _result {RESULT: 0x00000000 (0) SUCCEEDED}
        {
            options->ChangePointer(_dict);
            _dict = nullptr;
        }
    }
    finally
    {
        if (_dict) av_dict_free(&_dict);
```

In case if we have integer result type and we know that it is an error then just cast to ***AVRESULT*** type to see error description in variable values view while debugging:



```
ctx.time_base = new AVRational(1,25);
ctx.framerate = ctx.time_base.inv_q();
ctx.pix_fmt = AVPixelFormat.YUV420P;
int result = ctx.Open(codec);
if (0 == result)
{
    //...
}
dict.Dispose();
```

Name	Value	Type
result	0	int
(AVRESULT)result	{RESULT: 0x00000000 (0) SUCCEEDED}	FFmpeg.AVR...

```
\Runtime\Microsoft.VisualSt
'ffTest.exe' (Win32): Loaded
```

This class also can be accessed with some predefined error values which can be used in code:

```
static AVRESULT CheckPointer(IntPtr p)
{
    if (p == IntPtr.Zero) return AVRESULT.ENOMEM;
    return AVRESULT.OK;
}
```

Some of the methods already designed to return ***AVRESULT*** objects, other types can be easily casted to it.

```
FFmpeg::AVRESULT FFmpeg::AVCodecParameters::FromContext(AVCodecContext^ codec)
{
    return avcodec_parameters_from_context(
        ((::AVCodecParameters*)m_pPointer),(::AVCodecContext*)codec->_Pointer.ToPointer());
}

FFmpeg::AVRESULT FFmpeg::AVCodecParameters::CopyTo(AVCodecContext^ context)
{
    return avcodec_parameters_to_context(
        (::AVCodecContext*)context->_Pointer.ToPointer(), (::AVCodecParameters*)m_pPointer);
}

FFmpeg::AVRESULT FFmpeg::AVCodecParameters::CopyTo(AVCodecParameters^ dst)
{
    return avcodec_parameters_copy(
        (::AVCodecParameters*)dst->_Pointer.ToPointer(),((::AVCodecParameters*)m_pPointer));
}
```

AVLog

The way to inform applications from your component or from the internal of FFmpeg library uses the *av_log* API. Access to the same functionality of the logging is done with a static ***AVLog*** class.

```
if ((ret = AVFormatContext.OpenInput(out ifmt_ctx, in_filename)) < 0) {
    AVLog.log(AVLogLevel.Error, string.Format("Could not open input file {0}", in_filename));
    goto end;
}
```

It also has the ability to set up your own callback for the receiving log messages:

```
static void LogCallback(AVBase avcl, int level, string fmt, IntPtr vl)
{
    Console.WriteLine(fmt);
}

static void Main(string[] args)
{
    AVLog.Callback = LogCallback;
```

AVOptions

Most of native FFmpeg structure objects have properties which are hidden and can be accessed only with *av_opt_** API's. Current implementations of the wrapper library also have such abilities. For that purpose the library contains class ***AVOptions***. Initially it was designed as a static object but later implemented as a regular class. It can be initialized with a constructor with ***AVBase*** or just with pointer type argument.

```
/* Set the filter options through the AVOptions API. */
AVOptions options = new AVOptions(abuffer_ctx);
options.set("channel_layout", INPUT_CHANNEL_LAYOUT.ToString(), AVOptSearch.CHILDREN);
options.set("sample_fmt", INPUT_FORMAT.name, AVOptSearch.CHILDREN);
options.set_q("time_base", new AVRational( 1, INPUT_SAMPLERATE ),  AVOptSearch.CHILDREN);
options.set_int("sample_rate", INPUT_SAMPLERATE, AVOptSearch.CHILDREN);
```

Class exposes all useful methods for manipulation with options API. Along with it there is an ability to enumerate options:

```
AVOptions options = new AVOptions(ifmt_ctx);
foreach (AVOption o in options)
{
    Console.WriteLine(o.name);
}
```

AVDictionary

Another most used structure in FFmpeg API is the ***AVDictionary***. It is a Key-Value collection class which can be an input or an output of certain functions. In such functions you pass some initialization options as a dictionary object and in return getting all available options or options which are not used by the function.The management of that structure pointer in such function methods is done in place there is such an API called. So, users just need to free the returned dictionary object in a regular way as it described earlier, and internally the structure pointer just replaced. For example we have an FFmpeg API:

```
int avcodec_open2(AVCodecContext *avctx, const AVCodec *codec, AVDictionary **options);
```

And related implementation in library:

```
AVRESULT AVCodecContext::Open(AVCodec^ codec, AVDictionary^ options);
AVRESULT AVCodecContext::Open(AVCodec^ codec);
```

So, according to FFmpeg API implementation details, structure designed to set options, and on returns there is a new dictionary object with options which are not found. But on implementation only the structure object is used which was passed as an argument. So this is working in a regular way and users have only one structure at all. Even if the call of the internal API failed, the user needed to free only the target structure object, which was created initially:

```
AVDictionary dct = new AVDictionary();
dct.SetValue("preset", "veryfast");
s.id = (int)_output.nb_streams - 1;
bFailed = !(s.codec.Open(_codec, dct));
dct.Dispose();
```

Extending functionality

As mentioned, structure classes can contain fields of the related structure and methods which generated from FFmpeg API and also related to functionality of that structure. But classes can contain helper methods which allows extending functionality of FFmpeg. For example class ***AVFrame*** has static methods for conversion from/to .NET ***Bitmap*** object into ***AVFrame*** structure:

```
public:
    System::Drawing::Bitmap^ ToBitmap();
public:
    static String^ GetColorspaceName(AVColorSpace val);
public:
    // Create Frame From Image
    static AVFrame^ FromImage(System::Drawing::Bitmap^ _image,AVPixelFormat _format);
    static AVFrame^ FromImage(System::Drawing::Bitmap^ _image);
    static AVFrame^ FromImage(System::Drawing::Bitmap^ _image, AVPixelFormat _format,bool bDeleteBitmap);
    static AVFrame^ FromImage(System::Drawing::Bitmap^ _image, bool bDeleteBitmap);
    // Convert Frame To Different Colorspace
    static AVFrame^ ConvertVideoFrame(AVFrame^ _source,AVPixelFormat _format);
    // Convert Frame To Bitmap
    static System::Drawing::Bitmap^ ToBitmap(AVFrame^ _frame);
```

Those methods encapsulate whole functionality for colorspace conversion and any other list of calls. More of that, there is an ability to wrap around existing ***Bitmap*** object without any copy or allocating new image data. In such a case an internal helper object is created, which is freed with the parent ***AVFrame*** class. How that mechanism is implemented is described later in the document.

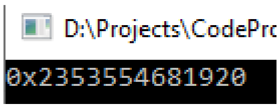
Class AVBase

All FFmpeg structure classes are inherited from the ***AVBase*** class. It contains common fields and helper methods. ***AVBase*** class is the root class of each FFmpeg object. Main tasks of the ***AVBase*** object are to handle object management and memory management.

Fields

As mentioned, ***AVBase*** class is the base for every exposed structure of FFmpeg library and It contains fields which are common to all objects. Those fields are: structure pointer and size of structure. *Size* of structure may be not initialized unless the structure is not allocated directly, so user no need to rely on that feld. *Pointers* can be casted to raw structure type, for access to any additional fields or raw data directly:

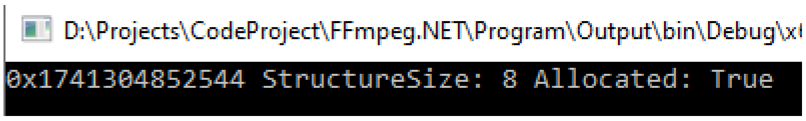
```
AVBase avbase = new AVPacket();
Console.WriteLine("0x{0:x}",avbase._Pointer);
```


D:\Projects\CodePrc

```
0x2353554681920
```

In case if structure can be allocated with a common allocation method it may contain structure size non zero value and in case if the structure allocated the related property is set to *true*:

```
AVBase avbase = new AVRational(1,1);
Console.WriteLine("0x{0:x} StructureSize: {1} Allocated: {2}",
    avbase._Pointer,
    avbase._StructureSize,
    avbase._IsAllocated);
```


D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x

```
0x1741304852544 StructureSize: 8 Allocated: True
```

If a class object is returned from any method, to be sure if such object is valid, we can check the pointer field and compare it to non-zero value. But there is also *_IsValid* property in the ***AVBase*** class which performs the same:

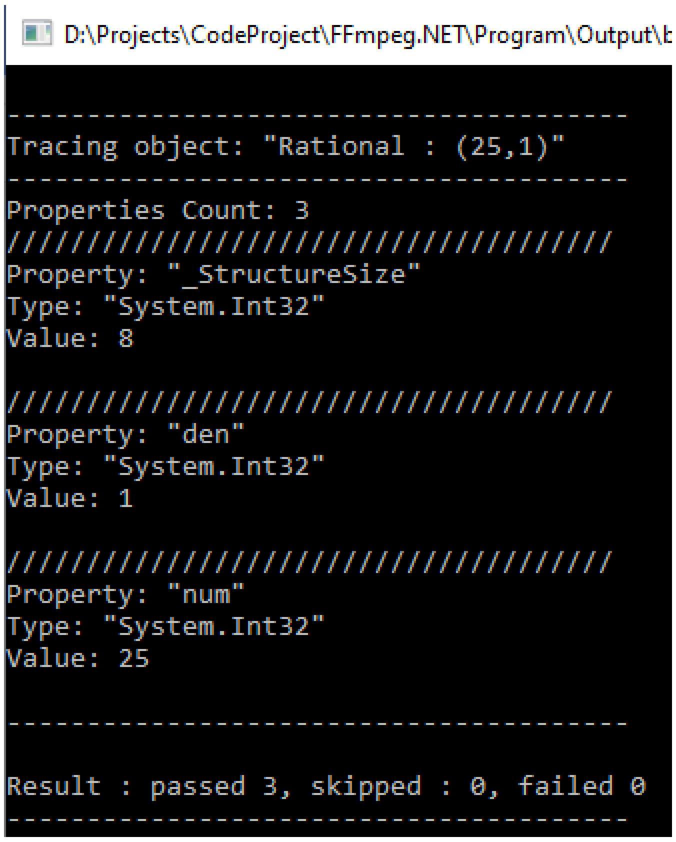
```
/* allocate and init a re-usable frame */
ost.frame = alloc_picture(c.pix_fmt, c.width, c.height);
if (!ost.frame._IsValid) {
    Console.WriteLine("Could not allocate video frame\n");
    Environment.Exit(1);
}
```

Decided to add the “_” prefix into such internal and not related to FFmpeg structure properties, so the user knows that it is not a structure field.

Methods

One method available for testing objects and its fields is the *TraceValues()*. This method uses .NET Reflection and enumerates available properties of the object with the values. If your code is critical to use of .NET Reflection then that code should be excluded from assembly. Altrought *TraceValues* code worked only in **Debug** build configuration. Example of execution that method is:

```
var r = new AVRational(25, 1);
r.TraceValues();
r.Dispose();
```


D:\Projects\CodeProject\FFmpeg.NET\Program\Output\b

```
---------------------------------------
Tracing object: "Rational : (25,1)"
---------------------------------------
Properties Count: 3
//////////////////////////////////////
Property: "_StructureSize"
Type: "System.Int32"
Value: 8

//////////////////////////////////////
Property: "den"
Type: "System.Int32"
Value: 1

//////////////////////////////////////
Property: "num"
Type: "System.Int32"
Value: 25


---------------------------------------

Result : passed 3, skipped : 0, failed 0
---------------------------------------
```

There are also a couple protected methods of the class which can help to build inherited classes with your own needs. I just point few of them which may be interested:

```
// Validate object not disposed
void ValidateObject();
```

That method checks whether an object is disposed or not, and if disposed then it throws an ***ObjectDisposedException*** exception.

```
// Check that object is availeble to be accessed
// throw exception if not
bool _EnsurePointer();
bool _EnsurePointer(bool bThrow);
```

Those two methods check if the structure pointer is zero or not, including the case if the object is disposed. This method is better for internal use than the *_IsValid* property described earlier.

```
// Object Memory Allocation
void AllocPointer(int _size);
```

This is the hepler method of allocation structure pointer. Usually an argument for this method is used in the *_StructureSize* field, but this is not a requirement. What method performs internal memory allocation for the structure pointer field: *_Pointer*, setting up the destructor and set allocation flag of an object: *_IsAllocated*.

```
if (!base._EnsurePointer(false))
{
    base.AllocPointer(_StructureSize);
}
```

Other protected methods of the class are described in objects management or memory management topics.

Destructor

As already described, that structure pointer in ***AVBase*** class is a pointer to a real FFmpeg structure. But any structure can be allocated and freed with different FFmpeg APIs. More of that after creation and call any initialization methods of any structure there is possible to call additional methods for destruction or specific uninitialization before performing actual object free. For example ***AVCodecContext*** allocation and free APIs

```
AVCodecContext *avcodec_alloc_context3(const AVCodec *codec);

/**
 * Free the codec context and everything associated with it and write NULL to
 * the provided pointer.
 */
void avcodec_free_context(AVCodecContext **avctx);
```

And at same time ***AVPacket*** object, along with allocation and free related API's, have the underlying buffer dereferencing API, which should be called before freeing the structure.

```
void av_packet_unref(AVPacket *pkt);
```

Depending on that ***AVBase*** class has a two pointers to API's which are performs structure destruction and free actual structure object memory:

```
typedef void TFreeFN(void *);
typedef void TFreeFNP(void **);
// Function to free object
TFreeFN * m_pFree;
// Object Destructor Function (may be set along with any free function)
TFreeFN * m_pDescructor;
// Free Pointer Function
TFreeFNP * m_pFreep;
```

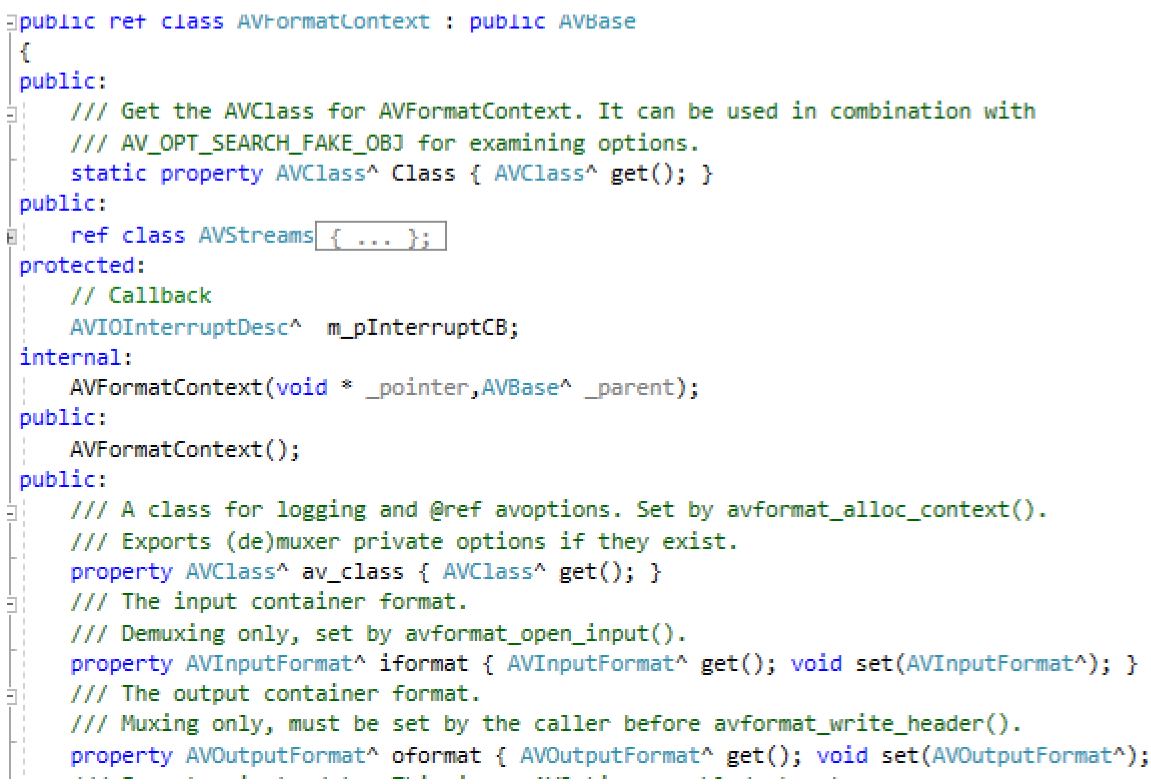
There are two types of free API pointers, as different FFmpeg structures can have different API arguments to free underlying objects, and properly destroy it. If one API is set then the other one is not used - those two pointers are mutual exclusion. Destructor API if set then called before any API to free structure. Example how that pointers are initialized internally for ***AVPacket*** class:

```
FFmpeg::AVPacket::AVPacket()
    : AVBase(nullptr,nullptr)
    , m_pOpaque(nullptr)
    , m_pFreeCB(nullptr)
{
    m_pPointer = av_packet_alloc();
    m_pFreep = (TFreeFNP*)av_packet_free;
    m_pDescructor = (TFreeFN*)av_packet_unref;

    av_init_packet((::AVPacket*)m_pPointer);
    ((::AVPacket*)m_pPointer)->data = nullptr;
    ((::AVPacket*)m_pPointer)->size = 0;
}
```

Those pointers can be changed internally in code depending on which object methods are called. So, in some cases, for an object can be set a different free function pointer or different destructor depending on the object state. This allows the object to properly destroy allocated data and clean up used memory.

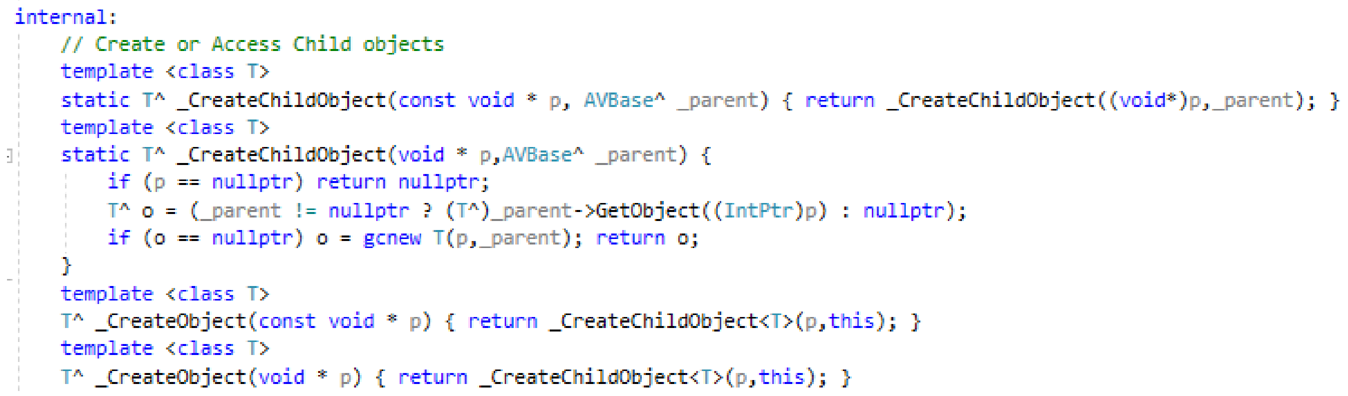
Objects management

Each wrapper library object can belong to another object as well as the underlying structure field containing a pointer to another FFmpeg structure. As an example: structure ***AVCodecContext*** contains *codec* field which points to ***AVCodec*** structure; *av_class* field - ***AVClass*** structure and others; ***AVFormatContext*** have similar fields: *iformat*, *oformat*, *pb* and others. More of that, the structure fields may not be read only and can be changed by the user, so in the code we need to handle that. In related properties it designed next way:



```
public ref class AVFormatContext : public AVBase
{
public:
    /// Get the AVClass for AVFormatContext. It can be used in combination with
    /// AV_OPT_SEARCH_FAKE_OBJ for examining options.
    static property AVClass^ Class { AVClass^ get(); }
public:
    ref class AVStreams { ... };
protected:
    // Callback
    AVIOInterruptDesc^  m_pInterruptCB;
internal:
    AVFormatContext(void * _pointer,AVBase^ _parent);
public:
    AVFormatContext();
public:
    /// A class for logging and @ref avoptions. Set by avformat_alloc_context().
    /// Exports (de)muxer private options if they exist.
    property AVClass^ av_class { AVClass^ get(); }
    /// The input container format.
    /// Demuxing only, set by avformat_open_input().
    property AVInputFormat^ iformat { AVInputFormat^ get(); void set(AVInputFormat^); }
    /// The output container format.
    /// Muxing only, must be set by the caller before avformat_write_header().
    property AVOutputFormat^ oformat { AVOutputFormat^ get(); void set(AVOutputFormat^); }
```

Internally, the child object of the such structure is created with no referencing allocated data - as initially the object is part of the parent. So disposing of that object makes no sense. But, if such an object is set by the user, which means that it was allocated as an independent object initially, then the parent holds reference to the child until it will be replaced via property or the main object will be released. In that case the call of child object disposing does not free the actual object - it just decrements the internal reference counter. In that manner: one object can be created and can be set via property to different parents, and in all cases that single object will be used without coping and it will be disposed automatically, once all parents will be freed. In other words, disposing of the child object will be performed then it does not belong to any other parents. So, an object can be created manually, an object can be created from a pointer as part of the parent object, or an object can be set as a pointer property of another object, and from all of that object can be properly released. This is handled by having two reference counters. One used for in-object access another from outer-object calls and once all references become zero - object freed.
Most child objects created once the program accessed the property for the first time. Then such an object is put into the children collection, and if the program makes a new call of the same property then a previously created object from that collection will be used, so no new instance created.
There are helper templates in the ***AVBase*** class for child instance creation:



```
internal:
    // Create or Access Child objects
    template <class T>
    static T^ _CreateChildObject(const void * p, AVBase^ _parent) { return _CreateChildObject((void*)p,_parent); }
    template <class T>
    static T^ _CreateChildObject(void * p,AVBase^ _parent) {
        if (p == nullptr) return nullptr;
        T^ o = (_parent != nullptr ? (T^)_parent->GetObject((IntPtr)p) : nullptr);
        if (o == nullptr) o = gcnew T(p,_parent); return o;
    }
    template <class T>
    T^ _CreateObject(const void * p) { return _CreateChildObject<T>(p,this); }
    template <class T>
    T^ _CreateObject(void * p) { return _CreateChildObject<T>(p,this); }
```

Children collection is freed once the main object is released. Class contains helper methods for accessing children objects in a collection:

```
protected:
    // Accessing children
    AVBase^ GetObject(IntPtr p);
    bool AddObject(IntPtr p,AVBase^ _object);
    bool AddObject(AVBase^ _object);
    void RemoveObject(IntPtr p);
```

Most interesting method here is the *AddObject* which allows associating a child ***AVBase*** object with a specified pointer. If that pointer is already associated with another object then it will be replaced, and the previous object disposed.

Memory management

Each ***AVBase*** object can contain allocated memory for certain properties, internal data or other needs. That memory is stored in the named collection which is related to that object. If a memory pointer is recreated then it is replaced in the collection. Once an object is freed - then all allocated memory is also free.

```
Generic::SortedList<String^,IntPtr>^ m_ObjectMemory;
```

There are couple methods for memory manipulation in a class:

```
IntPtr GetMemory(String^ _key);
void SetMemory(String^ _key,IntPtr _pointer);
IntPtr AllocMemory(String^ _key,int _size);
IntPtr AllocString(String^ _key,String^ _value);
IntPtr AllocString(String^ _key,String^ _value,bool bUnicode);
void FreeMemory(String^ _key);
bool IsAllocated(String^ _key);
```

Those methods help allocate and free memory and check whether it is allocated or not. Naming access done for controls allocation for specified properties.

```
void FFmpeg::AVCodecContext::subtitle_header::set(array<byte>^ value)
{
    if (value != nullptr && value->Length > 0)
    {
        ((::AVCodecContext*)m_pPointer)->subtitle_header_size = value->Length;
        ((::AVCodecContext*)m_pPointer)->subtitle_header =
            (uint8_t *)AllocMemory("subtitle_header",value->Length).ToPointer();
        Marshal::Copy(value,0,
            (IntPtr)((::AVCodecContext*)m_pPointer)->subtitle_header,value->Length);
    }
    else
    {
        FreeMemory("subtitle_header");
        ((::AVCodecContext*)m_pPointer)->subtitle_header_size = 0;
        ((::AVCodecContext*)m_pPointer)->subtitle_header = nullptr;
    }
}
```

Allocation of the memory is done by using *av_maloc* API of the FFmpeg library. There is also a static collection of allocated memory. That collection is used to call the static methods for memory allocation provided by the ***AVBase*** class.

```
static IntPtr AllocMemory(int _size);
static IntPtr AllocString(String^ _value);
static IntPtr AllocString(String^ _value,bool bUnicode);
static void FreeMemory(IntPtr _memory);
```

Static memory collection located in a special ***AVMemory*** class. As that class is the base for ***AVMemPtr*** - which represents the special object of memory allocated pointer, and for ***AVBase*** - as that class is base for all imported structures and, as were mentioned, there may require memory allocation. Static memory is freed automatically once all objects which may use it are disposed.

Arrays

In FFmpeg structures there are a lot of fields which represent arrays of data with different types. Those arrays can be fixed size, arrays ending with specified data value, arrays of arrays and arrays of other structures. Each type can be designed personally, but there is also some common implementation. For example array of streams in format context implemented as its own class with enumerator and indexed property.

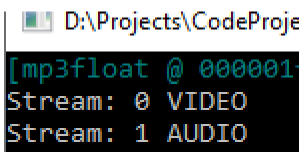
```
ref class AVStreams
    : public System::Collections::IEnumerable
    , public System::Collections::Generic::IEnumerable<AVStream^>
{
private:
    ref class AVStreamsEnumerator { ... };
protected:
    AVFormatContext^ m_pParent;
internal:
    AVStreams(AVFormatContext^ _parent) : m_pParent(_parent) {}
public:
    property AVStream^ default[int] { AVStream^ get(int index) { return m_pParent->GetStream(index); } }
    property int Count { int get() { return m_pParent->nb_streams; } }
public:
    // IEnumerable
    virtual System::Collections::IEnumerator^ GetEnumerator() sealed = System::Collections::IEnumerable::
public:
    // IEnumerable<AVStream^>
    virtual System::Collections::Generic::IEnumerator<AVStream^>^ GetEnumeratorGeneric() sealed = System:
    { ... }
};
```

In that case we have an array with pointers to other structure classes - ***AVStream***, and each indexed property call - performing creation of the child object of the parent ***AVFormatContext*** object, as described in objects management topic.

```
FFmpeg::AVStream^ FFmpeg::AVFormatContext::GetStream(int idx)
{
    if (((::AVFormatContext*)m_pPointer)->nb_streams <= (unsigned int)idx || idx < 0) return nullptr;
    auto p = ((::AVFormatContext*)m_pPointer)->streams[idx];
    return _CreateObject<AVStream>((void*)p);
}
```

In .NET that is looks just as regular property and array access:

```
var _input = AVFormatContext.OpenInputFile(@"test.avi");
if (_input.FindStreamInfo() == 0)
{
    for (int i = 0; i < _input.streams.Count; i++)
    {
        Console.WriteLine("Stream: {0} {1}",i,_input.streams[i].codecpar.codec_type);
    }
}
```



```
D:\Projects\CodeProje
[mp3float @ 000001
Stream:  0 VIDEO
Stream:  1 AUDIO
```

There are also possible cases where you need to set an array of the ***AVBase*** objects as property - in that situation each object is put into the parent objects collection, so disposing of that object can be done safely. Along with the main structure array, the property which is related to the counter of objects in that array can be modified internally. Also the previously allocated objects and array memory itself are freed. The next code displays how is that done:

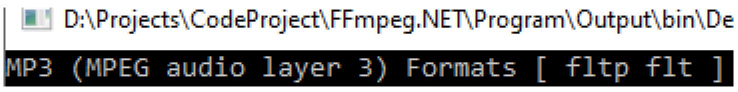
```
void FFmpeg::AVFormatContext::chapters::set(array<AVChapter^>^ value)
{
    {
        int nCount = ((::AVFormatContext*)m_pPointer)->nb_chapters;
        ::AVChapter ** p = (::AVChapter **)((::AVFormatContext*)m_pPointer)->chapters;
        if (p)
        {
            while (nCount-- > 0)
            {
                RemoveObject((IntPtr)*p++);
            }
        }
        ((::AVFormatContext*)m_pPointer)->nb_chapters = 0;
        ((::AVFormatContext*)m_pPointer)->chapters = nullptr;
        FreeMemory("chapters");
    }
    if (value != nullptr && value->Length > 0)
    {
        ::AVChapter ** p = (::AVChapter **)AllocMemory("chapters",value->Length * (sizeof(::AVChapter*))).ToPointer();
        if (p)
        {
            ((::AVFormatContext*)m_pPointer)->chapters = p;
            ((::AVFormatContext*)m_pPointer)->nb_chapters = value->Length;
            for (int i = 0; i < value->Length; i++)
            {
                AddObject((IntPtr)*p,value[i]);
                *p++ = (::AVChapter*)value[i]->_Pointer.ToPointer();
            }
        }
    }
}
```

In most cases with read only arrays, where array values are not required to be changed, the properties returns a simple .NET array:

```
array<FFmpeg::AVSampleFormat>^ FFmpeg::AVCodec::sample_fmts::get()
{
    List<AVSampleFormat>^ _array = nullptr;
    const ::AVSampleFormat * _pointer = ((::AVCodec*)m_pPointer)->sample_fmts;
    if (_pointer)
    {
        _array =  gcnew List<AVSampleFormat>();
        while (*_pointer != -1)
        {
            _array->Add((AVSampleFormat)*_pointer++);
        }
    }
    return _array != nullptr ? _array->ToArray() : nullptr;
}
```

The usage of such arrays done in regular way in .NET:

```
var codec = AVCodec.FindDecoder(AVCodecID.MP3);
Console.Write("{0} Formats [ ",codec.long_name);
foreach (var fmt in codec.sample_fmts)
{
    Console.Write("{0} ",fmt);
}
Console.WriteLine("]");
```



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\De
MP3 (MPEG audio layer 3) Formats [ fltp flt ]
```

For other cases for arrays accessing in the wrapper library implemented base class ***AVArrayBase***. It is inherited from the ***AVBase*** class and accesses memory chunks with specified data size. Class also contains a number of elements in that array.

```
public ref class AVArrayBase : public AVBase
{
protected:
    bool    m_bValidate;
    int     m_nItemSize;
    int     m_nCount;
protected:
    AVArrayBase(void * _pointer,AVBase^ _parent,int nItemSize,int nCount)
        : AVBase(_pointer,_parent) , m_nCount(nCount), m_nItemSize(nItemSize), m_bValidate(true) { }
    AVArrayBase(void * _pointer,AVBase^ _parent,int nItemSize,int nCount,bool bValidate)
        : AVBase(_pointer,_parent) , m_nCount(nCount), m_nItemSize(nItemSize), m_bValidate(bValidate) { }
protected:
    void ValidateIndex(int index) { if (index < 0 || index >= m_nCount) throw gcnew ArgumentOutOfRangeException(); }
    void * GetValue(int index)
    {
        if (m_bValidate) ValidateIndex(index);
        return (((LPBYTE)m_pPointer) + m_nItemSize * index);
    }
    void SetValue(int index,void * value)
    {
        if (m_bValidate) ValidateIndex(index);
        memcpy(((LPBYTE)m_pPointer) + m_nItemSize * index,value,m_nItemSize);
    }
public:
    property int Count { int get() { return m_nCount; } }
};
```

That class is base without types specifying. And the main template for the typed arrays is the ***AVArray*** class. It is inherited from ***AVArrayBase*** and already has access to the indexed property of the array elements. Along with it, the class supports enumerators. This class has special modifications for some types like ***AVMemPtr*** and ***IntPtr*** to make proper access to array elements.The template classes are used in most cases in the library, as they give direct access to underlying array memory - without any .NET marshaling and copy. For example: the class ***AVFrame***/***AVPicture*** have properties:

```
///< pointers to the image data planes
property AVArray<AVMemPtr^>^ data { AVArray<AVMemPtr^>^ get(); }
///< number of bytes per line
property AVArray<int>^ linesize { AVArray<int>^ get(); }
```

And in the same place the ***AVMemPtr*** class has the ability to directly access the memory with its properties, as different typed arrays which helps modifying or reading data. Next example shows how to access the frame data:

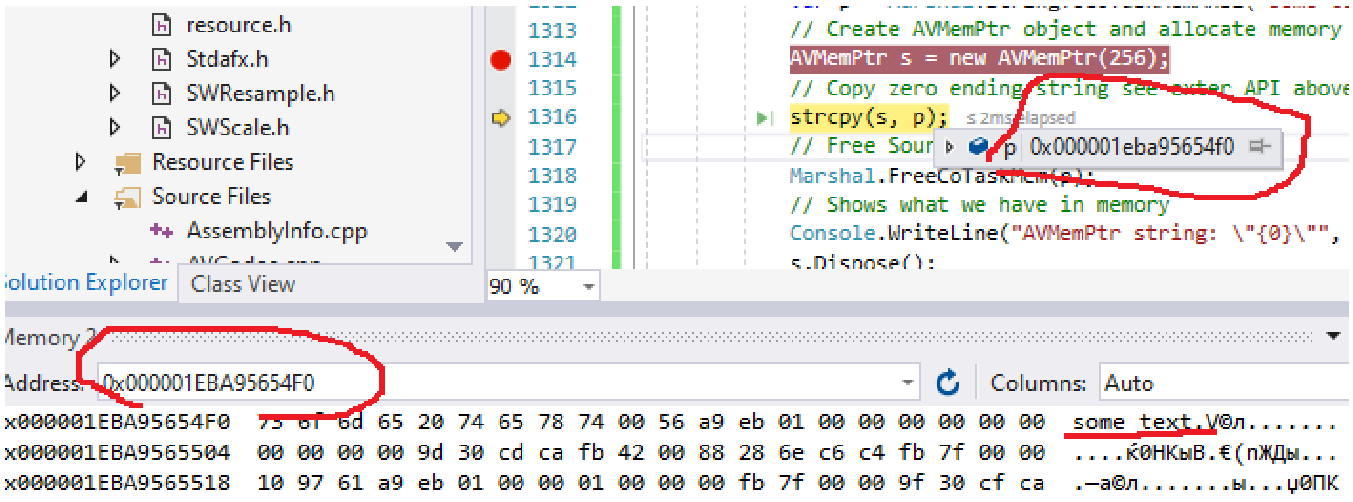
```
var frame = AVFrame.FromImage((Bitmap)Bitmap.FromFile(@"d:\image.jpg"),AVPixelFormat.YUV420P);
for (int j = 0; j < frame.height; j++)
{
    for (int i = 0; i < frame.linesize[0]; i++)
    {
        Console.Write("{0:X2}",frame.data[0][i + j * frame.linesize[0]]);
    }
    Console.WriteLine();
}
frame.Dispose();
```

Class AVMemPtr

Initially in the wrapper library all properties, which operate with pointers, were designed as ***IntPtr*** type. But it is not an easy way to work with memory in .NET directly with ***IntPtr***. As in application may require to change picture data, generate image, perform some processing of audio and video. To make that easier the ***AVMemPtr*** memory helper class was involved. However it contains an implicit casting operator which allows it to be used in the same methods which have the ***IntPtr*** type as argument in .NET. The basic usage of this class as regular ***IntPtr*** value demonstrated in next C# example:

```
[DllImport("msvcrt.dll", EntryPoint = "strcpy")]
public static extern IntPtr strcpy(IntPtr dest, IntPtr src);

static void Main(string[] args)
{
    // Allocate pointer from string
    var p = Marshal.StringToCoTaskMemAnsi("some text");
    // Create AVMemPtr object and allocate memory buffer of 256 bytes
    AVMemPtr s = new AVMemPtr(256);
    // Copy zero ending string see exter API above
    strcpy(s, p);
    // Free Source memory
    Marshal.FreeCoTaskMem(p);
    // Shows what we have in memory
    Console.WriteLine("AVMemPtr string: \"{0}\"", Marshal.PtrToStringAnsi(s));
    s.Dispose();
}
```

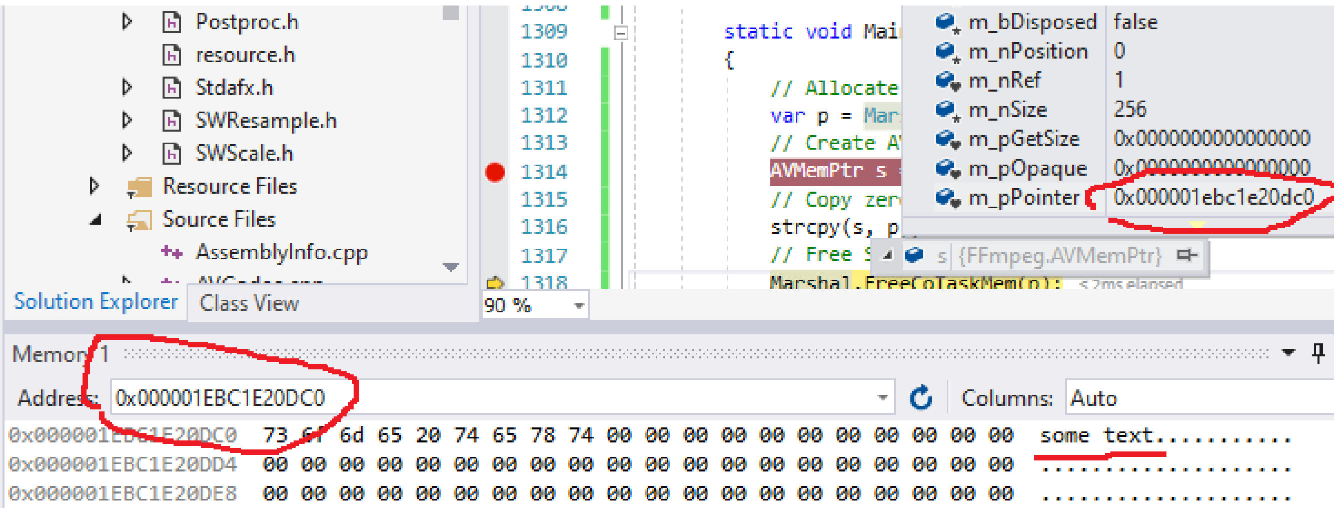
In this example we allocate memory from given text. The result of that memory pointer and the data can be showed in next picture:



```
// Create AVMemPtr object and allocate memory
AVMemPtr s = new AVMemPtr(256);
// Copy zero ending string see exter API above
strcpy(s, p);  ≤ 2ms elapsed
// Free Sour   p 0x000001eba95654f0
Marshal.FreeCoTaskMem(p);
// Shows what we have in memory
Console.WriteLine("AVMemPtr string: \"{0}\"",
s.Dispose();
```

```
Address: 0x000001EBA95654F0                      Columns: Auto
x000001EBA95654F0  73 6f 6d 65 20 74 65 78 74 00 56 a9 eb 01 00 00 00 00 00 00  some text.V©л.......
x000001EBA9565504  00 00 00 00 9d 30 cd ca fb 42 00 88 28 6e c6 c4 fb 7f 00 00  ....ќ0НКыB.€(nЖДы...
x000001EBA9565518  10 97 61 a9 eb 01 00 00 01 00 00 00 fb 7f 00 00 9f 30 cf ca  .—a©л.......ы...џ0ПК
```

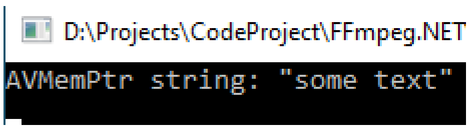
After execution of C++ strcpy API .NET wrapper, which performs zero ending string copy, you can see that the text copied into allocated pointer of ***AVMemPtr*** object:



```
m_bDisposed   false
m_nPosition   0
m_nRef        1
m_nSize       256
m_pGetSize    0x0000000000000000
m_pOpaque     0x0000000000000000
m_pPointer    0x000001ebc1e20dc0
s {FFmpeg.AVMemPtr}
```

```
Address: 0x000001EBC1E20DC0                      Columns: Auto
0x000001EBC1E20DC0  73 6f 6d 65 20 74 65 78 74 00 00 00 00 00 00 00 00 00 00 00  some text...........
0x000001EBC1E20DD4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0x000001EBC1E20DE8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
```

And after we displaying the string value which we have in there:

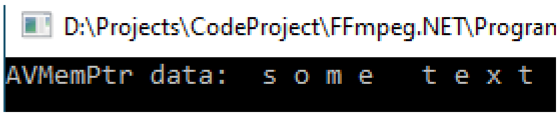


```
D:\Projects\CodeProject\FFmpeg.NET
AVMemPtr string: "some text"
```

Memory allocation and free in class done with FFmpeg API *av_alloc* and *av_free*. Class have comparison operators with different types. ***AVMemPtr*** also can be casted directly from ***IntPtr*** structure. Along with it, the class has the ability to access data as a regular bytes array:

```
var p = Marshal.StringToCoTaskMemAnsi("some text");
AVMemPtr s = p;
int idx = 0;
Console.Write("AVMemPtr data: ");
while (s[idx] != 0) Console.Write(" {0}", (char)s[idx++]);
Console.Write("\n");
Marshal.FreeCoTaskMem(p);
```

Result of execution code above:



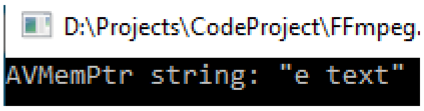
```
D:\Projects\CodeProject\FFmpeg.NET\Progran
AVMemPtr data:  s o m e   t e x t
```

So we have allocated pointer data as ***IntPtr*** type and easily accessed it directly. ***AVMemPtr*** also can handle addition and subtraction operators they gave another ***AVMemPtr*** object which points to another address with resulted offset:

```
var p = Marshal.StringToCoTaskMemAnsi("some text");
AVMemPtr s = p;
Console.Write("AVMemPtr data: ");
while ((byte)s != 0) { Console.Write(" {0}", (char)((byte)s)); s += 1; }
Console.Write("\n");
Marshal.FreeCoTaskMem(p);
```

Each addition or subtraction creates a new instance of ***AVMemPtr*** but internally it is implemented that the base data pointer stays the same even if data is allocated and the main instance is *DIsposed*. That is done also with counting references of main instance, for example this code will work correctly:

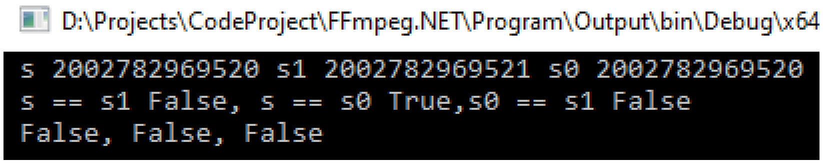
```
var p = Marshal.StringToCoTaskMemAnsi("some text");
AVMemPtr s = new AVMemPtr(256);
strcpy(s,p);
Marshal.FreeCoTaskMem(p);
AVMemPtr s1 = s + 3;
s.Dispose();
Console.WriteLine("AVMemPtr string: \"{0}\"", Marshal.PtrToStringAnsi(s1));
s1.Dispose();
```



```
D:\Projects\CodeProject\FFmpeg.
AVMemPtr string: "e text"
```

In next code we can see that instances of *s*, *s0* and *s1* objects are different but comparison operators determine equals of the data:

```
var p = Marshal.StringToCoTaskMemAnsi("some text");
AVMemPtr s = p;
AVMemPtr s1 = s + 1;
AVMemPtr s0 = s1 - 1;
Console.WriteLine(" s {0} s1 {1} s0 {2}", s,s1,s0);
Console.WriteLine(" s == s1 {0}, s == s0 {1},s0 == s1 {2}",
    (s == s1),(s == s0),(s0 == s1));
Console.WriteLine(" {0}, {1}, {2}",
    object.ReferenceEquals(s,s1), object.ReferenceEquals(s,s0),
    object.ReferenceEquals(s0,s1));
Marshal.FreeCoTaskMem(p);
```



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64
 s 2002782969520 s1 2002782969521 s0 2002782969520
 s == s1 False, s == s0 True,s0 == s1 False
 False, False, False
```

Class ***AVMemPtr*** also contains helper methods and properties which may be useful, one of them to determine if data allocated and size of buffer in bytes and debug helper method to perform dump pointer data to a file or ***Stream***.

The main feature of this class is the ability to represent data as arrays of different types. This is done as different properties of ***AVArray*** templates:

```
property AVArray<byte>^   bytes              { AVArray<byte>^ get();   }
property AVArray<short>^  shorts             { AVArray<short>^ get();  }
property AVArray<int>^    integers           { AVArray<int>^ get();    }
property AVArray<float>^  floats             { AVArray<float>^ get();  }
property AVArray<double>^ doubles            { AVArray<double>^ get();  }
property AVArray<IntPtr>^ pointers           { AVArray<IntPtr>^ get(); }
property AVArray<unsigned int>^ uints        { AVArray<unsigned int>^ get(); }
property AVArray<unsigned short>^ ushorts    { AVArray<unsigned short>^ get();  }
property AVArray<RGB^>^   rgb                { AVArray<RGB^>^ get();    }
property AVArray<RGBA^>^  rgba               { AVArray<RGBA^>^ get();   }
property AVArray<AYUV^>^  ayuv               { AVArray<AYUV^>^ get();   }
property AVArray<YUY2^>^  yuy2               { AVArray<YUY2^>^ get();   }
property AVArray<UYVY^>^  uyvy               { AVArray<UYVY^>^ get();   }
```

So, it is easy fill *data* of ***AVFrame*** object, for example, of the audio with IEEE float type or signed short:

```
var samples = frame.data[0].shorts;
for (int j = 0; j < c.frame_size; j++)
{
    samples[2 * j] = (short)(int)(Math.Sin(t) * 10000);

    for (int k = 1; k < c.channels; k++)
        samples[2 * j + k] = samples[2 * j];
    t += tincr;
}
```

More of it, as you can see, there are some arrays with specified pixel format structures: ***RGB***, ***RGBA***, ***AYUV***, ***YUY2*** and ***UYVY***. Using these properties is a designed way to address pixels with those formats which is described later.

Enumerators

FFmpeg APIs for accessing lists of resources for example enumerate existing input or output formats have special types of APIs - iterators. The new API have iterate with opaque argument and old ones uses previous element as reference:

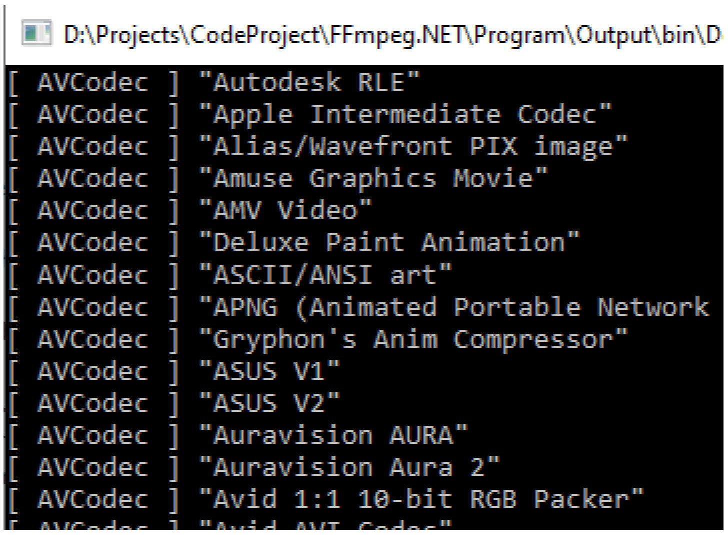
```
const AVCodec *av_codec_iterate(void **opaque);
```

And the old one:

```
AVCodec *av_codec_next(const AVCodec *c);
```

Initially in the wrapper library were designed a way to use those APi in the same manner as they are present in FFmpeg, but later implementations were done with usage of ***IEnumerable*** and ***IEnumerator*** interfaces. That gave us the way of performing enumeration with *foreach* C# operator instead of calling API directly:

```
foreach (AVCodec codec in AVCodec.Codecs)
{
    if (codec.IsDecoder()) Console.WriteLine(codec);
}
```


```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\D
[ AVCodec ] "Autodesk RLE"
[ AVCodec ] "Apple Intermediate Codec"
[ AVCodec ] "Alias/Wavefront PIX image"
[ AVCodec ] "Amuse Graphics Movie"
[ AVCodec ] "AMV Video"
[ AVCodec ] "Deluxe Paint Animation"
[ AVCodec ] "ASCII/ANSI art"
[ AVCodec ] "APNG (Animated Portable Network
[ AVCodec ] "Gryphon's Anim Compressor"
[ AVCodec ] "ASUS V1"
[ AVCodec ] "ASUS V2"
[ AVCodec ] "Auravision AURA"
[ AVCodec ] "Auravision Aura 2"
[ AVCodec ] "Avid 1:1 10-bit RGB Packer"
```
Code above can be done in a couple other modifications. Most of enumerator objects expose *Count* property and indexing operator:

```
var codecs = AVCodec.Codecs;
for (int i = 0; i < codecs.Count; i++)
{
    if (codecs[i].IsDecoder()) Console.WriteLine(codecs[i]);
}
```

Either also possible to use old way with calling API wrapper method:

```
AVCodec codec = null;
while (null != (codec = AVCodec.Next(codec)))
{
    if (codec.IsDecoder()) Console.WriteLine(codec);
}
```

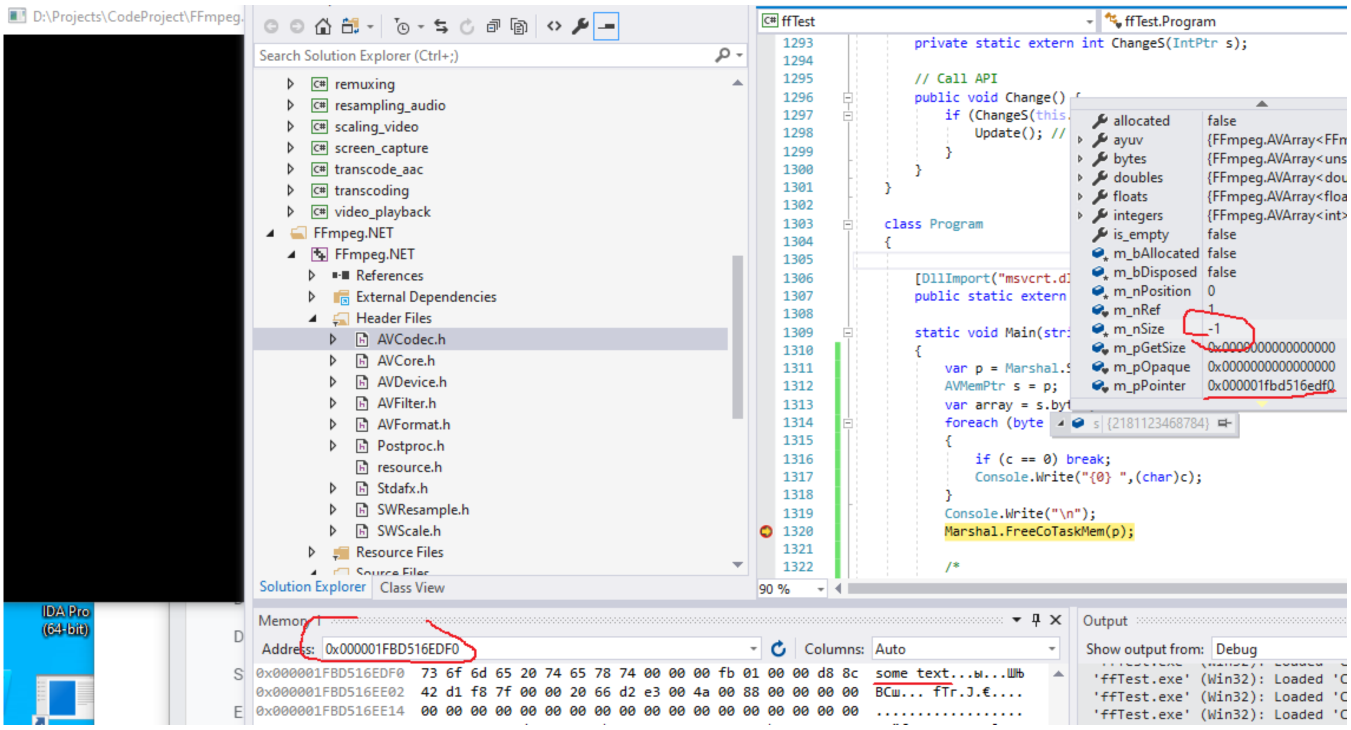
There is no need to worry if FFmpeg libraries expose only one API: *av_codec_iterate* or *av_codec_next*. In the wrapper library all FFmpeg APIs which are marked as deprecated are linked dynamically, so it checks runtime for what API to use. How that is implemented is described later in a separate topic.
AVArray classes which are done in the library also expose an enumerator interface:

```
var p = Marshal.StringToCoTaskMemAnsi("some text");
AVMemPtr s = new AVMemPtr(256);
strcpy(s, p);
Marshal.FreeCoTaskMem(p);
var array = s.bytes;
foreach (byte c in array)
{
    if (c == 0) break;
    Console.Write("{0} ",(char)c);
}
Console.Write("\n");
```

In the example above, we have allocated data length of memory while creating ***AVMemPtr*** object. But it is possible that enumeration may not work as arrays can be created dynamically or without size initialization - so the enumerators are not able to determine the total number of elements. I limit that functionality to avoid out of boundary crashes and other exceptions. For example, the next code will not work as the data size of ***AVMemPtr*** object is 0 due conversion from ***IntPtr*** pointer into ***AVMemPtr***, so you can find the difference with the previous example:

```
var p = Marshal.StringToCoTaskMemAnsi("some text");
AVMemPtr s = p;
var array = s.bytes;
foreach (byte c in array)
{
    if (c == 0) break;
    Console.Write("{0} ",(char)c);
}
Console.Write("\n");
Marshal.FreeCoTaskMem(p);
```

On the screenshot below it is possible to see that data of ***AVMemPtr*** object pointed correctly to the text line, but as size is unknown and the enumeration is not available.



```
allocated       false
ayuv            {FFmpeg.AVArray<FFm
bytes           {FFmpeg.AVArray<uns
doubles         {FFmpeg.AVArray<dou
floats          {FFmpeg.AVArray<floa
integers        {FFmpeg.AVArray<int>
is_empty        false
m_bAllocated    false
m_bDisposed     false
m_nPosition     0
m_nRef          1
m_nSize         -1
m_pGetSize      0x0000000000000000
m_pOpaque       0x0000000000000000
m_pPointer      0x000001fbd516edf0
```

```
Address: 0x000001FBD516EDF0
0x000001FBD516EDF0  73 6f 6d 65 20 74 65 78 74 00 00 00 fb 01 00 00 d8 8c  some text...ы...Шь
0x000001FBD516EE02  42 d1 f8 7f 00 00 20 66 d2 e3 00 4a 00 88 00 00 00 00  BCш... fTr.J.€....
0x000001FBD516EE14  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
```

The wrapper library properly handles ***AVPicture***/***AVFrame*** classes *data* field enumeration. That is done with an internal function which handles detecting object field size. So next code example work correctly:

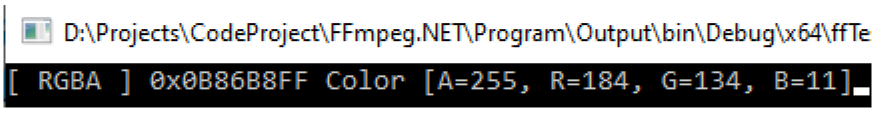
```
AVFrame frame = new AVFrame();
frame.Alloc(AVPixelFormat.BGR24, 320, 240);
frame.MakeWritable();
var rgb = frame.data[0].rgb;
foreach (RGB c in rgb)
{
    c.Color = Color.AliceBlue;
}
var bmp = frame.ToBitmap();
bmp.Save(@"test.bmp");
bmp.Dispose();
frame.Dispose();
```

Lots of classes in a library support enumerators. Along with enumerating codecs in the examples above enumerator types are used in enumeration input and output formats, parsers, filters, devices and other library resources.

Class AVColor

As already mentioned, the ***AVMemPtr*** class can produce data access as arrays of some colorspace types like ***RGB24***, ***RGBA***, ***AYUV***, ***YUY2*** or ***UYVY***. Those colorspaces are designed as structures with the ability to access color components. Base class of those structures is the ***AVColor***. It contains basic operators, helper properties and methods. For example it support assigning regular color value and override string representation:

```
AVFrame frame = new AVFrame();
frame.Alloc(AVPixelFormat.BGRA, 320, 240);
frame.MakeWritable();
var rgb = frame.data[0].rgba;
rgb[0].Color = Color.DarkGoldenrod;
Console.Write("{0} {1}",rgb[0].ToString(),rgb[0].Color);
frame.Dispose();
```

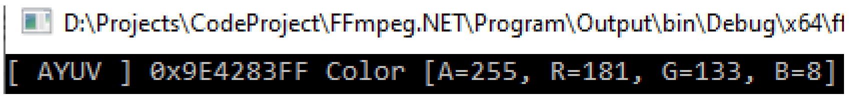


```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTe
[ RGBA ] 0x0B86B8FF Color [A=255, R=184, G=134, B=11]
```

Along with it, the class contains the ability for color component access and implicit conversion into direct ***IntPtr*** type. Also, internally, the class supports colorspace conversions **RGB** into **YUV** and **YUV** to **RGB**.

```
var ayuv = frame.data[0].ayuv;
ayuv[0].Color = Color.DarkGoldenrod;
Console.Write("{0} {1}",ayuv[0].ToString(),ayuv[0].Color);
```

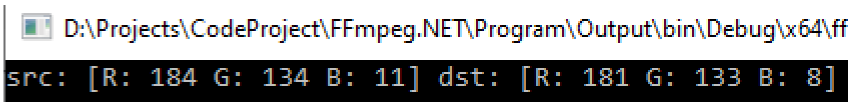
In code above the same color as in the previous example set into ***AYUV*** colorspace. So internally it converted from ***ARGB*** into ***AYUV***, which gave output:



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ff
[ AYUV ] 0x9E4283FF Color [A=255, R=181, G=133, B=8]
```

Conversion results have little difference from original value, but that is acceptable. By default used the BT.601 conversion matrix. It is possible to change matrix coefficients by calling *SetColorspaceMatrices* static method of ***AVColor*** class. Additionally class contains static methods for colorspace components conversion:

```
Color c = Color.DarkGoldenrod;
int y = 0, u = 0, v = 0;
int r = 0, g = 0, b = 0;
AVColor.RGBToYUV(c.R,c.G,c.B, ref y, ref u, ref v);
AVColor.YUVToRGB(y,u,v, ref r, ref g, ref b);
Console.Write("src: [R: {0} G: {1} B: {2}] dst: [R: {3} G: {4} B: {5}]",c.R,c.G,c.B,r,g,b);
```



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ff
src: [R: 184 G: 134 B: 11] dst: [R: 181 G: 133 B: 8]
```

Necessary to keep in mind that ***YUY2*** and ***UYVY*** control 2 pixels. So setting the *Color* property affects both those pixels. To control each pixel you should use “*Y0*” and “*Y1*” class properties to change luma values.

Core Classes

The wrapper library contains lots of classes which, as were mentioned, represents each one of FFmpeg structures with exposing data fields as properties along with related usage methods. This topic describes most of those classes but not all of them. Description also contains basic usage examples of those classes, without describing class fields, as they are the same as in original C++ FFmpeg libraries. In code parts skipped calling the *Dispose*() method of some global object destruction. Also, that is required to check error results of the methods calls, and you should do it in your code but in the samples that is also skipped to decrease code size.

AVPacket

Class is the wrapper around the ***AVPacket*** structure of the FFmpeg *libavcodec* library. It has different constructors with the ability to pass its own buffer in there, and with a callback to detect when the buffer will be freed. After every successive return from any decoding or encoding operation it is required to call the *Free* method to un-reference the underlying buffer. Class contains static methods to create ***AVPacket*** objects from specified .NET arrays with different types. *Dump* method allows you to save packet data into a file or stream.

```
var f = AVFrame.FromImage((Bitmap)Bitmap.FromFile(@"image.jpg"),AVPixelFormat.YUV420P);
var c = new AVCodecContext(AVCodec.FindEncoder(AVCodecID.MPEG2VIDEO));
c.bit_rate = 400000;
c.width = f.width;
c.height = f.height;
c.time_base = new AVRational(1, 25);
c.framerate = new AVRational(25, 1);
c.pix_fmt = f.format;
c.Open(null);
AVPacket pkt = new AVPacket();
bool got_packet = false;
while (!got_packet)
{
    c.EncodeVideo(pkt,f,ref got_packet);
}
pkt.Dump(@"pkt.bin");
pkt.Free();
pkt.Dispose();
```

AVPicture

Class which is wrapped around related structure in the old version of the FFmpeg *libavcodec* library.In wrapper library It is supported for all versions of FFmpeg in current implementation. Class contains methods for the *av_image_** API's of FFmpeg library and pointers to data and plane sizes. The ***AVFrame*** inherits from this class so it is recommended to use ***AVFrame*** directly instead.

AVFrame

Class for the ***AVFrame*** structure of the FFmpeg *libavutil* library. Class used as input for encoding audio or video data, or as output from the decoding process. It can handle audio and video data. After every successive return from any decoding or encoding operation it is required to call the *Free* method to un-reference the underlying buffer.

```
var _input = AVFormatContext.OpenInputFile(@"test.mp4");
if (_input.FindStreamInfo() == 0) {
    int idx = _input.FindBestStream(AVMediaType.VIDEO, -1, -1);
    AVCodecContext decoder = _input.streams[idx].codec;
    var codec = AVCodec.FindDecoder(decoder.codec_id);
    if (decoder.Open(codec) == 0) {
        AVPacket pkt = new AVPacket();
        AVFrame frame = new AVFrame();
        int index = 0;
        while ((bool)_input.ReadFrame(pkt)) {
            if (pkt.stream_index == idx) {
                bool got_frame = false;
                int ret = decoder.DecodeVideo(frame, ref got_frame, pkt);
                if (got_frame) {
                    var bmp = AVFrame.ToBitmap(frame);
                    bmp.Save(string.Format(@"image{0}.png",++index));
                    bmp.Dispose();
                    frame.Free();
                }
            }
            pkt.Free();
        }
    }
}
```

Frame objects can be initialized from existing .NET ***Bitmap*** object. In the code below the frame is created from the image and converted into **YUV 420** planar colorspace format.

```
var frame = AVFrame.FromImage((Bitmap)Bitmap.FromFile(@"image.jpg"),AVPixelFormat.YUV420P);
```

It is also possible to have wrap around existing bitmap data in that case no need to specify target pixel format as the second argument. The pixel format of such frame data will be **BGRA**, and the image data will not be copied into the newly allocated buffer. But, internally, the child class object of the wrapped image data will be created which will be freed once the actual frame will be disposed.
There are also some helper methods which can be used for converting video frames:

```
var src = AVFrame.FromImage((Bitmap)Bitmap.FromFile(@"image.jpg"));
var dst = AVFrame.ConvertVideoFrame(src,AVPixelFormat.YUV420P);
```

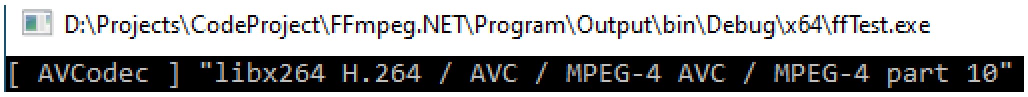
And conversion back into .NET Bitmap object:

```
var bmp = AVFrame.ToBitmap(frame);
bmp.Save(@"image.png");
```

AVCodec

Class which is implemented managed access to ***AVCodec*** structure of the FFmpeg *libavcodec* library. As recommended, it contains only public fields.of the underlying structure. Class describes registered codec in the library.

```
var codec = AVCodec.FindEncoder("libx264");
Console.WriteLine("{0}",codec);
```



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.exe
[ AVCodec ] "libx264 H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"
```

Class able to enumerate existing codecs:

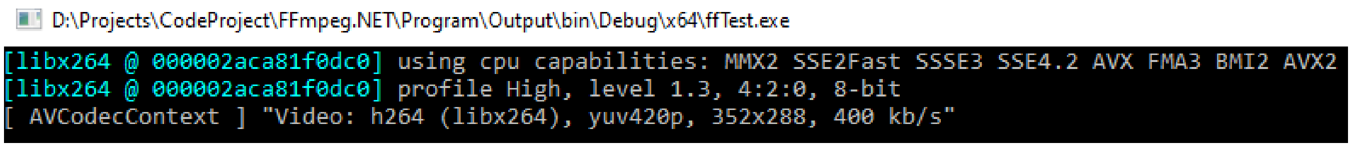
```
foreach (AVCodec codec in AVCodec.Codecs)
{
    Console.WriteLine(codec);
}
```

AVCodecContext

Class which is implemented managed access to ***AVCodecContext*** structure of the FFmpeg *libavcodec* library. Object is used for encoding or decoding video and audio data.

```
var c = new AVCodecContext(AVCodec.FindEncoder("libx264"));
c.bit_rate = 400000;
c.width = 352;
c.height = 288;
c.time_base = new AVRational(1, 25);
c.framerate = new AVRational(25, 1);
c.gop_size = 10;
c.max_b_frames = 1;
c.pix_fmt =  AVPixelFormat.YUV420P;
c.Open(null);
Console.WriteLine(c);
```

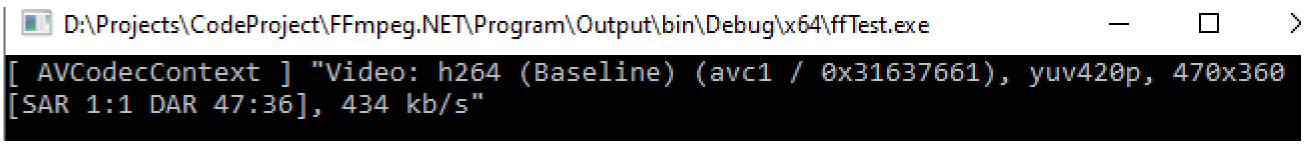
In the example above we create a context object and initialize the **H264** encoder of the video with resolution 352x288 with 25 frames per second and 400 kbps bit rate.



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.exe
[libx264 @ 000002aca81f0dc0] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 000002aca81f0dc0] profile High, level 1.3, 4:2:0, 8-bit
[ AVCodecContext ] "Video: h264 (libx264), yuv420p, 352x288, 400 kb/s"
```

Next sample demonstrates opening the file and initializing the decoder for the video stream:

```
AVFormatContext fmt_ctx;
AVFormatContext.OpenInput(out fmt_ctx, @"test.mp4");
fmt_ctx.FindStreamInfo();
var s = fmt_ctx.streams[fmt_ctx.FindBestStream(AVMediaType.VIDEO)];
var c = new AVCodecContext(AVCodec.FindDecoder(s.codecpar.codec_id));
s.codecpar.CopyTo(c);
c.Open(null);
Console.WriteLine(c);
```


D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.exe

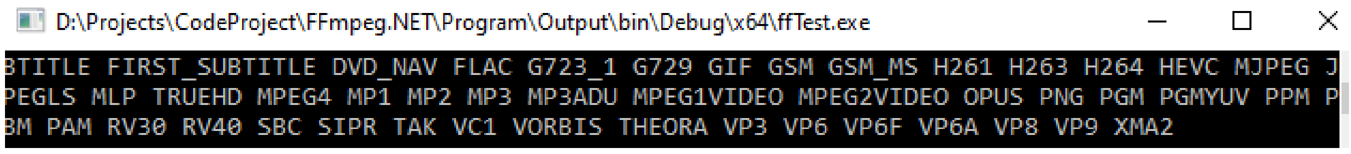
```
[ AVCodecContext ] "Video: h264 (Baseline) (avc1 / 0x31637661), yuv420p, 470x360
[SAR 1:1 DAR 47:36], 434 kb/s"
```

Class contains methods for initializing structure and decoding or encoding audio and video.

AVCodecParser

Class for accessing the ***AVCodecParser*** structure of the FFmpeg *libavcodec* library from managed code. Structure describes registered codec parsers. Class mostly used for enumerating registered parsers in the library:

```
foreach (AVCodecParser parser in AVCodecParser.Parsers)
{
    for (int i = 0; i < parser.codec_ids.Length; i++)
    {
        Console.Write("{0} ", parser.codec_ids[i]);
    }
}
```


D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.exe

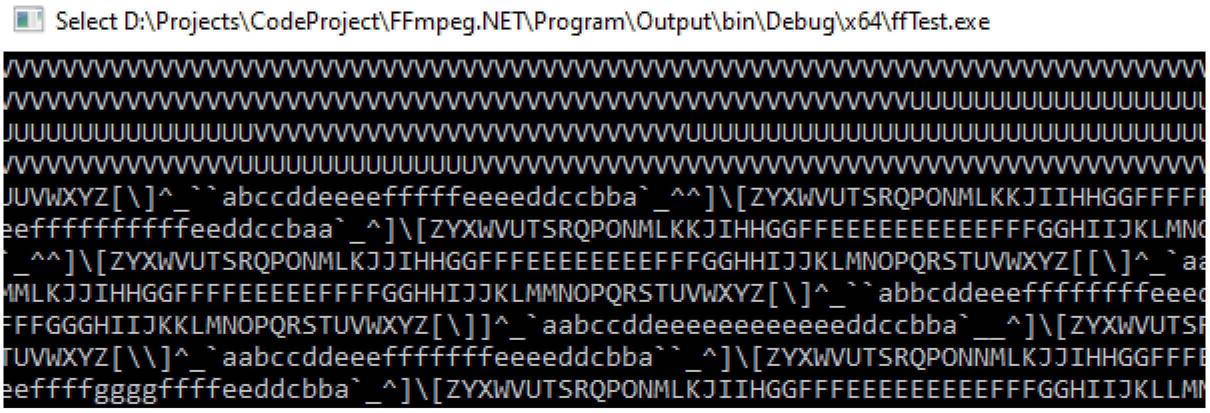
```
BTITLE FIRST_SUBTITLE DVD_NAV FLAC G723_1 G729 GIF GSM GSM_MS H261 H263 H264 HEVC MJPEG J
PEGLS MLP TRUEHD MPEG4 MP1 MP2 MP3 MP3ADU MPEG1VIDEO MPEG2VIDEO OPUS PNG PGM PGMYUV PPM P
BM PAM RV30 RV40 SBC SIPR TAK VC1 VORBIS THEORA VP3 VP6 VP6F VP6A VP8 VP9 XMA2
```

AVCodecParserContext

Class for performing bitstream parse operation. It is wrapped around the ***AVCodecParserContext*** structure of the FFmpeg *libavcodec* library. It is created with the specified codec and contains different variations of the parse method.

```
var decoder = new AVCodecContext(AVCodec.FindDecoder(AVCodecID.MP3));
decoder.sample_fmt = AVSampleFormat.FLTP;
if (decoder.Open(null) == 0) {
    var parser = new AVCodecParserContext(decoder.codec_id);
    AVPacket pkt = new AVPacket();
    AVFrame frame = new AVFrame();
    var f = fopen(@"test.mp3", "rb");
    const int buffer_size = 1024;
    IntPtr data = Marshal.AllocCoTaskMem(buffer_size);
    int data_size = fread(data, 1, buffer_size, f);
    while (data_size > 0) {
        int ret = parser.Parse(decoder, pkt, data, data_size);
        if (ret < 0) break;
        data_size -= ret;
        if (pkt.size > 0) {
             bool got_frame = false;
            decoder.DecodeAudio(frame,ref got_frame,pkt);
            if (got_frame) {
                var p = frame.data[0].floats;
                for (int i = 0; i < frame.nb_samples; i++) {
                    Console.Write((char)(((p[i] + 1.0) * 54) + 32));
                }
                frame.Free();
            }
            pkt.Free();
        }
        memmove(data, data + ret, data_size);
        data_size += fread(data + data_size, 1, buffer_size - data_size, f);
    }
    Marshal.FreeCoTaskMem(data);
    fclose(f);
}
```

Example demonstrates reading data from raw **mp3** file, parsing it by chunks and decode with simple print clamped data into output. Example uses some wrappers of the unmanaged API's: fopen, fread and fclose. Those function declarations can be found in other samples in this document.



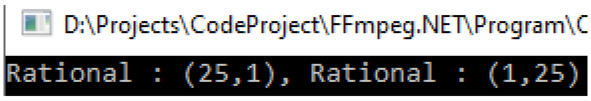
Select D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.exe

```
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVUUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUVVVVVVVVVVVVVVVVVVVVVVVVVVVVUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
VVVVVVVVVVVVVVVUUUUUUUUUUUUUVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
UUVWXYZ[\]^_``abccddeeeefffffeeeeddccbba`_^^]\[ZYXWVUTSRQPONMLKKJIIHHGGFFFFF
eeffffffffffeeddccbaa`_^]\[ZYXWVUTSRQPONMLKKJIHHGGFFEEEEEEEEEEEEFFFGGHIIJKLMNO
`_^^]\[ZYXWVUTSRQPONMLKJJIHHGGFFFEEEEEEEEEFFFGGHHIJJKLMNOPQRSTUVWXYZ[[\]^_`aa
MMLKJJIHHGGFFFFEEEEEEFFFFGGHHIJJKLMMNOPQRSTUVWXYZ[\]^_``abbcddeeefffffffffeeed
FFFGGGHIIJKKLMNOPQRSTUVWXYZ[\]]^_`aabccddeeeeeeeeeeeeeeddccbba`__^]\[ZYXWVUTSR
TUVWXYZ[\\]^_`aabccddeeeffffffffeeeeddcbba``_^]\[ZYXWVUTSRQPONNMLKJJIHHGGFFFE
eeffffggggffffeeddcbba`_^]\[ZYXWVUTSRQPONMLKJIIHGGFFFEEEEEEEEEEEFFFGGHIIJKLLMN
```

AVRational

Class which is implemented managed access to ***AVRational*** structure of the FFmpeg *libavutil* library. It contains all methods which are related to that structure.

```
AVRational r = new AVRational(25,1);
Console.WriteLine("{0}, {1}",r.ToString(),r.inv_q().ToString());
```



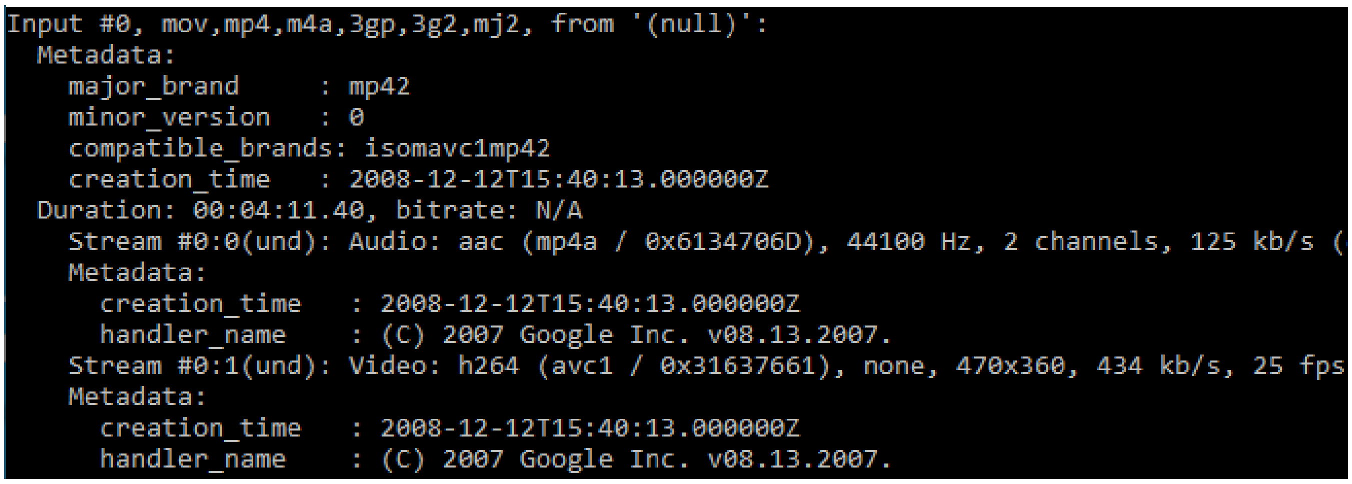
D:\Projects\CodeProject\FFmpeg.NET\Program\C

```
Rational : (25,1), Rational : (1,25)
```

AVFormatContext

Class which is implemented managed access to ***AVFormatContext*** structure of the FFmpeg *libavformat* library. Class contains methods for operating with output or input media data.

```
var ic = AVFormatContext.OpenInputFile(@"test.mp4");
ic.DumpFormat(0,null,false);
```



```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '(null)':
  Metadata:
    major_brand     : mp42
    minor_version   : 0
    compatible_brands: isomavc1mp42
    creation_time   : 2008-12-12T15:40:13.000000Z
  Duration: 00:04:11.40, bitrate: N/A
    Stream #0:0(und): Audio: aac (mp4a / 0x6134706D), 44100 Hz, 2 channels, 125 kb/s (
    Metadata:
      creation_time   : 2008-12-12T15:40:13.000000Z
      handler_name    : (C) 2007 Google Inc. v08.13.2007.
    Stream #0:1(und): Video: h264 (avc1 / 0x31637661), none, 470x360, 434 kb/s, 25 fps
    Metadata:
      creation_time   : 2008-12-12T15:40:13.000000Z
      handler_name    : (C) 2007 Google Inc. v08.13.2007.
```

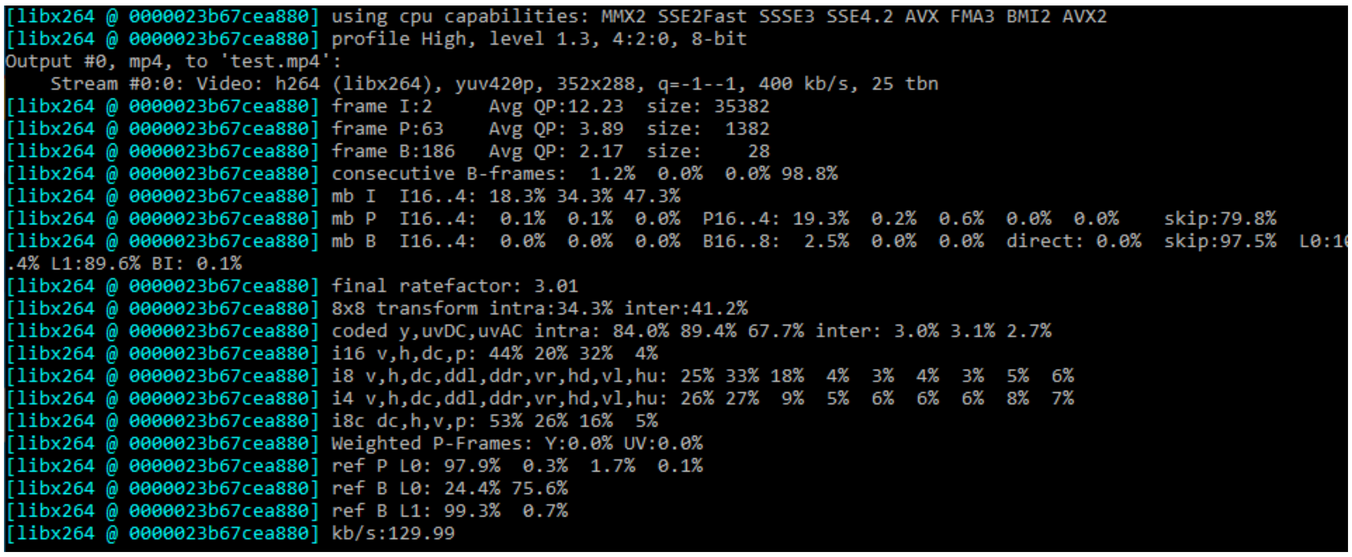
Most use case of the class is getting audio or video packets from input or writing such packets into output. Next sample demonstrates reading **mp3** audio packets from an **AVI** file and saves them into a separate file.

```
var ic = AVFormatContext.OpenInputFile(@"1.avi");
if (0 == ic.FindStreamInfo()) {
    int idx = ic.FindBestStream(AVMediaType.AUDIO);
    if (idx >= 0
        && ic.streams[idx].codecpar.codec_id == AVCodecID.MP3) {
        AVPacket pkt = new AVPacket();
        bool append = false;
        while (ic.ReadFrame(pkt) == 0) {
            if (pkt.stream_index == idx) {
                pkt.Dump(@"out.mp3",append);
                append = true;
            }
            pkt.Free();
        }
        pkt.Dispose();
    }
}
ic.Dispose();
```

And another main task, as mentioned, is the output packets. Next example demonstrates outputting a static picture to a **mp4** file with **H264** encoding for a 10 seconds long:

```
string filename = @"test.mp4";
var oc = AVFormatContext.OpenOutput(filename);
var frame = AVFrame.FromImage((Bitmap)Bitmap.FromFile(@"image.jpg"),AVPixelFormat.YUV420P);
var codec = AVCodec.FindEncoder(AVCodecID.H264);
var ctx = new AVCodecContext(codec);
var st = oc.AddStream(codec);
st.id = oc.nb_streams-1;
st.time_base = new AVRational( 1, 25 );
ctx.bit_rate = 400000;
ctx.width = frame.width;
ctx.height = frame.height;
ctx.time_base = st.time_base;
ctx.framerate = ctx.time_base.inv_q();
ctx.pix_fmt = frame.format;
ctx.Open(null);
st.codecpar.FromContext(ctx);
if ((int)(oc.oformat.flags & AVfmt.NOFILE) == 0) {
    oc.pb = new AVIOContext(filename,AvioFlag.WRITE);
}
oc.DumpFormat(0, filename, true);
oc.WriteHeader();

AVPacket pkt = new AVPacket();
int idx = 0;
bool flush = false;
while (true) {
    flush = (idx > ctx.framerate.num * 10 / ctx.framerate.den);
    frame.pts = idx++;
    bool got_packet = false;
    if (0 < ctx.EncodeVideo(pkt, flush ? null : frame, ref got_packet)) break;
    if (got_packet) {
        pkt.RescaleTS(ctx.time_base, st.time_base);
        oc.WriteFrame(pkt);
        pkt.Free();
        continue;
    }
    if (flush) break;
}
oc.WriteTrailer();
pkt.Dispose();
frame.Dispose();
ctx.Dispose();
oc.Dispose();
```

And the application output results:



```
[libx264 @ 0000023b67cea880] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 0000023b67cea880] profile High, level 1.3, 4:2:0, 8-bit
Output #0, mp4, to 'test.mp4':
    Stream #0:0: Video: h264 (libx264), yuv420p, 352x288, q=-1--1, 400 kb/s, 25 tbn
[libx264 @ 0000023b67cea880] frame I:2     Avg QP:12.23  size: 35382
[libx264 @ 0000023b67cea880] frame P:63    Avg QP: 3.89  size:  1382
[libx264 @ 0000023b67cea880] frame B:186   Avg QP: 2.17  size:    28
[libx264 @ 0000023b67cea880] consecutive B-frames:  1.2%  0.0%  0.0% 98.8%
[libx264 @ 0000023b67cea880] mb I  I16..4: 18.3% 34.3% 47.3%
[libx264 @ 0000023b67cea880] mb P  I16..4:  0.1%  0.1%  0.0%  P16..4: 19.3%  0.2%  0.6%  0.0%  0.0%    skip:79.8%
[libx264 @ 0000023b67cea880] mb B  I16..4:  0.0%  0.0%  0.0%  B16..8:  2.5%  0.0%  0.0%  direct: 0.0%  skip:97.5%  L0:1
.4% L1:89.6% BI: 0.1%
[libx264 @ 0000023b67cea880] final ratefactor: 3.01
[libx264 @ 0000023b67cea880] 8x8 transform intra:34.3% inter:41.2%
[libx264 @ 0000023b67cea880] coded y,uvDC,uvAC intra: 84.0% 89.4% 67.7% inter: 3.0% 3.1% 2.7%
[libx264 @ 0000023b67cea880] i16 v,h,dc,p: 44% 20% 32%  4%
[libx264 @ 0000023b67cea880] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 25% 33% 18%  4%  3%  4%  3%  5%  6%
[libx264 @ 0000023b67cea880] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 26% 27%  9%  5%  6%  6%  6%  8%  7%
[libx264 @ 0000023b67cea880] i8c dc,h,v,p: 53% 26% 16%  5%
[libx264 @ 0000023b67cea880] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0000023b67cea880] ref P L0: 97.9%  0.3%  1.7%  0.1%
[libx264 @ 0000023b67cea880] ref B L0: 24.4% 75.6%
[libx264 @ 0000023b67cea880] ref B L1: 99.3%  0.7%
[libx264 @ 0000023b67cea880] kb/s:129.99
```

AVIOContext

Class which is implemented managed access to ***AVIOContext*** structure of the FFmpeg *libavformat* library. Class contains helper methods for writing/reading data from byte streams. The most interesting usage of this class is the ability to set up custom reading or writing callbacks. To demonstrates how to use those features let's modify previous code sample with replacement ***AVIOContext*** creation code:

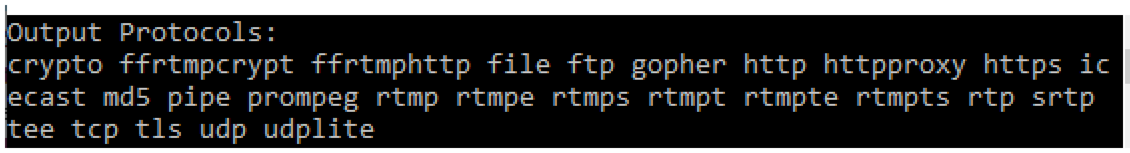
```
AVMemPtr ptr = new AVMemPtr(20 * 1024);
OutputFile file = new OutputFile(filename);
if ((int)(oc.oformat.flags & AVfmt.NOFILE) == 0) {
    oc.pb = new AVIOContext(ptr,1,file,null,OutputFile.WritePacket,OutputFile.Seek);
}
```

So right now we create a context with defining output and seek callback. Those callbacks are defined in ***OutputFile*** class of the example:

```
class OutputFile
{
    IntPtr stream = IntPtr.Zero;

    public OutputFile(string filename) {
        stream = fopen(filename, "w+b");
    }
    ~OutputFile() {
        IntPtr s = Interlocked.Exchange(ref stream, IntPtr.Zero);
        if (s != IntPtr.Zero) fclose(s);
    }

    public static int WritePacket(object opaque, IntPtr buf, int buf_size) {
        return fwrite(buf,1,buf_size,(opaque as OutputFile).stream);
    }

    public static int Seek(object opaque, long offset, AVSeek whence) {
        return fseek((opaque as OutputFile).stream,(int)offset, (int)whence);
    }

    [DllImport("msvcrt.dll", EntryPoint = "fopen")]
    public static extern IntPtr fopen(
        [MarshalAs(UnmanagedType.LPStr)] string filename,
        [MarshalAs(UnmanagedType.LPStr)] string mode);

    [DllImport("msvcrt.dll")]
    public static extern int fwrite(IntPtr buffer, int size, int count, IntPtr stream);

    [DllImport("msvcrt.dll")]
    public static extern int fclose(IntPtr stream);

    [DllImport("msvcrt.dll")]
    public static extern int fseek(IntPtr stream,long offset,int origin);
}
```

By running the sample you got the same output file and same results as in the previous topic, but now you can control each writing and seeking of the data.
Class also contains some helper static methods. It can check for **URL**s and enumerate supported protocols:

```
Console.Write("Output Protocols: \n");
var protocols = AVIOContext.EnumProtocols(true);
foreach (var s in protocols) {
    Console.Write("{0} ",s);
}
```

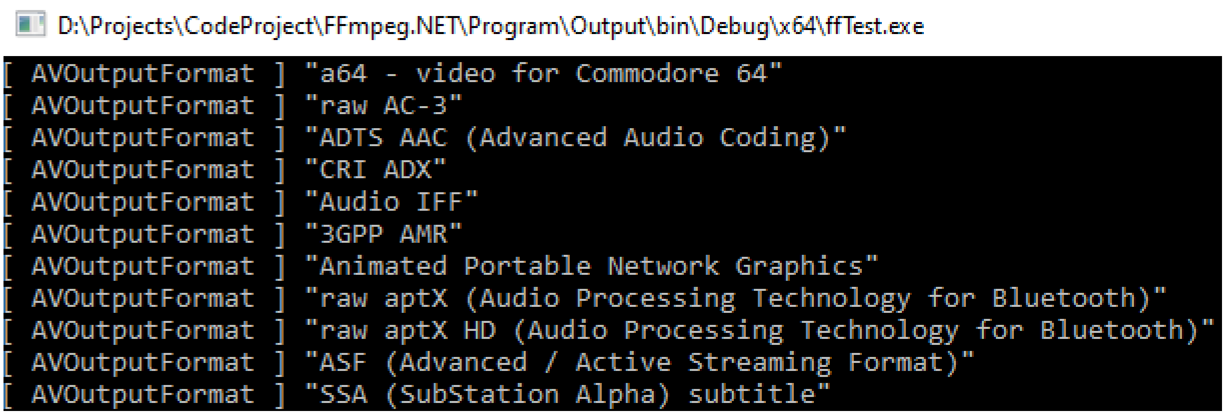


```
Output Protocols:
crypto ffrtmpcrypt ffrtmphttp file ftp gopher http httpproxy https ic
ecast md5 pipe prompeg rtmp rtmpe rtmps rtmpt rtmpte rtmpts rtp srtp
tee tcp tls udp udplite
```

AVOutputFormat

Class which is implemented managed access to ***AVOutputFormat*** structure of the ffmpeg libavformat library. Structure describes parameters of an output format supported by the libavformat library. Class contains enumerator to see all available formats:

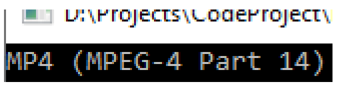
```
foreach (AVOutputFormat format in AVOutputFormat.Formats)
{
    Console.WriteLine(format.ToString());
}
```



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.exe
[ AVOutputFormat ] "a64 - video for Commodore 64"
[ AVOutputFormat ] "raw AC-3"
[ AVOutputFormat ] "ADTS AAC (Advanced Audio Coding)"
[ AVOutputFormat ] "CRI ADX"
[ AVOutputFormat ] "Audio IFF"
[ AVOutputFormat ] "3GPP AMR"
[ AVOutputFormat ] "Animated Portable Network Graphics"
[ AVOutputFormat ] "raw aptX (Audio Processing Technology for Bluetooth)"
[ AVOutputFormat ] "raw aptX HD (Audio Processing Technology for Bluetooth)"
[ AVOutputFormat ] "ASF (Advanced / Active Streaming Format)"
[ AVOutputFormat ] "SSA (SubStation Alpha) subtitle"
```

Class have no public constructor, but it can be accessed with enumerator or static methods:

```
var fmt = AVOutputFormat.GuessFormat(null,@"test.mp4",null);
Console.WriteLine(fmt.long_name);
```

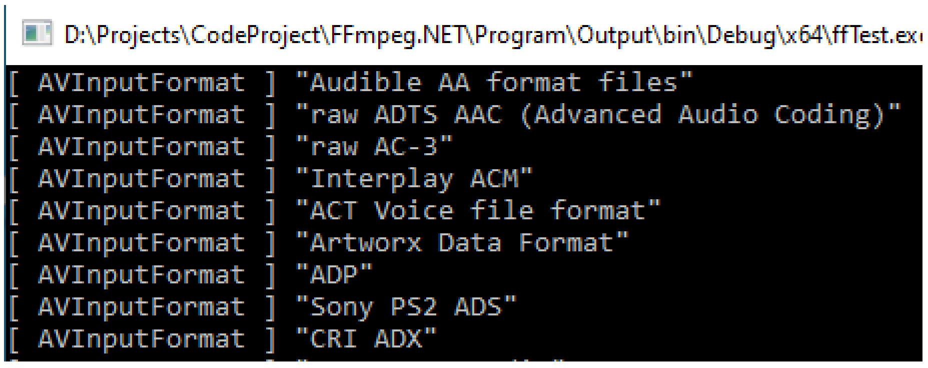


```
D:\Projects\CodeProject\
MP4 (MPEG-4 Part 14)
```

AVInputFormat

Class which is implemented managed access to ***AVInputFormat*** structure of the FFmpeg *libavformat* library. The structure describes parameters of an input format supported by the *libavformat* library. Class also contains enumerator to see all available formats:

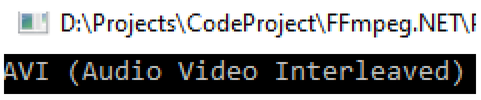
```
foreach (AVInputFormat format in AVInputFormat.Formats)
{
    Console.WriteLine(format.ToString());
}
```



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.ex
[ AVInputFormat ] "Audible AA format files"
[ AVInputFormat ] "raw ADTS AAC (Advanced Audio Coding)"
[ AVInputFormat ] "raw AC-3"
[ AVInputFormat ] "Interplay ACM"
[ AVInputFormat ] "ACT Voice file format"
[ AVInputFormat ] "Artworx Data Format"
[ AVInputFormat ] "ADP"
[ AVInputFormat ] "Sony PS2 ADS"
[ AVInputFormat ] "CRI ADX"
```

Class has no public constructor. Can be accessed from static methods:

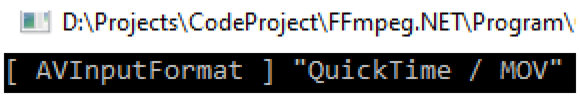
```
var fmt = AVInputFormat.FindInputFormat("avi");
Console.WriteLine(fmt.long_name);
```



```
D:\Projects\CodeProject\FFmpeg.NET\
AVI (Audio Video Interleaved)
```

There are also possible to detect format of giving buffer, also using static methods:

```
var f = fopen(@"test.mp4", "rb");
AVProbeData data = new AVProbeData();
data.buf_size = 1024;
data.buf = Marshal.AllocCoTaskMem(data.buf_size);
data.buf_size = fread(data.buf,1,data.buf_size,f);
var fmt = AVInputFormat.ProbeInputFormat(data,true);
Console.WriteLine(fmt.ToString());
Marshal.FreeCoTaskMem(data.buf);
data.Dispose();
fclose(f);
```

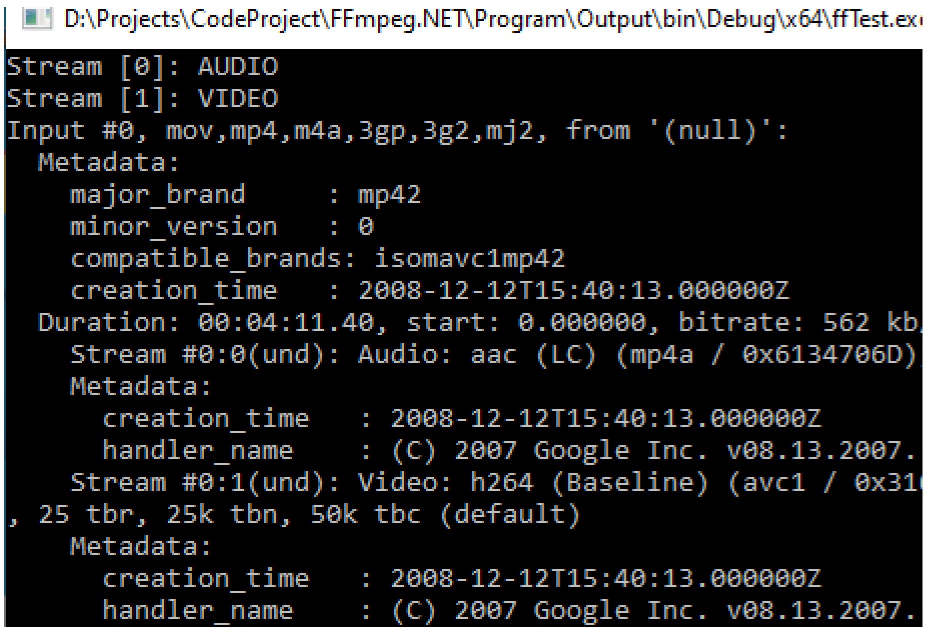


```
D:\Projects\CodeProject\FFmpeg.NET\Program\
[ AVInputFormat ] "QuickTime / MOV"
```

AVStream

Class which is implemented managed access to ***AVStream*** structure of the FFmpeg *libavformat* library. Class has no public constructor and can be accessed from the ***AVFormatContext*** streams property.

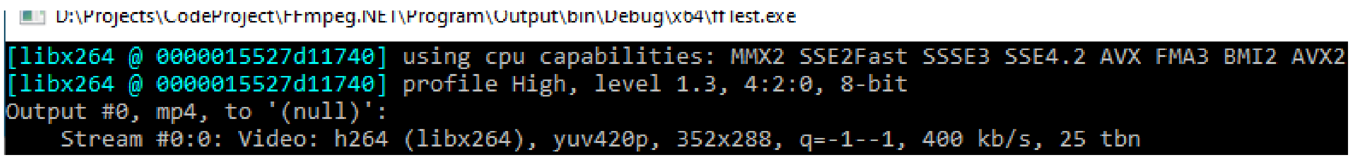
```
var input = AVFormatContext.OpenInputFile(@"test.mp4");
if (input.FindStreamInfo() == 0){
    for (int idx = 0; idx < input.streams.Count; idx++)
    {
        Console.WriteLine("Stream [{0}]: {1}",idx,
            input.streams[idx].codecpar.codec_type);
    }
}
input.DumpFormat(0,null,false);
```



```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.ex
Stream [0]: AUDIO
Stream [1]: VIDEO
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from '(null)':
  Metadata:
    major_brand     : mp42
    minor_version   : 0
    compatible_brands: isomavc1mp42
    creation_time   : 2008-12-12T15:40:13.000000Z
  Duration: 00:04:11.40, start: 0.000000, bitrate: 562 kb
    Stream #0:0(und): Audio: aac (LC) (mp4a / 0x6134706D)
    Metadata:
      creation_time   : 2008-12-12T15:40:13.000000Z
      handler_name    : (C) 2007 Google Inc. v08.13.2007.
    Stream #0:1(und): Video: h264 (Baseline) (avc1 / 0x31
, 25 tbr, 25k tbn, 50k tbc (default)
    Metadata:
      creation_time   : 2008-12-12T15:40:13.000000Z
      handler_name    : (C) 2007 Google Inc. v08.13.2007.
```

Streams can be created with the ***AVFormatContext*** *AddStream* method.

```
var oc = AVFormatContext.OpenOutput(@"test.mp4");
var codec = AVCodec.FindEncoder(AVCodecID.H264);
var c = new AVCodecContext(codec);
var st = oc.AddStream(codec);
st.id = oc.nb_streams-1;
st.time_base = new AVRational( 1, 25 );
c.codec_id = codec.id;
c.bit_rate = 400000;
c.width     = 352;
c.height    = 288;
c.time_base  = st.time_base;
c.gop_size   = 12;
c.pix_fmt    = AVPixelFormat.YUV420P;
c.Open(codec);
st.codecpar.FromContext(c);
oc.DumpFormat(0, null, true);
```



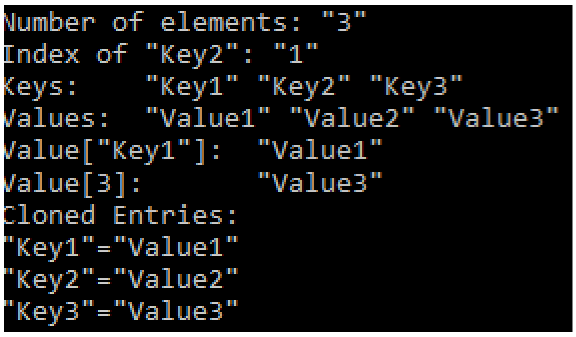
```
D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x64\ffTest.exe
[libx264 @ 0000015527d11740] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 0000015527d11740] profile High, level 1.3, 4:2:0, 8-bit
Output #0, mp4, to '(null)':
    Stream #0:0: Video: h264 (libx264), yuv420p, 352x288, q=-1--1, 400 kb/s, 25 tbn
```

AVDictionary

Class which is implemented managed access to ***AVDictionary*** collection type of the FFmpeg *libavutil* library. This class is the name-value collection of the string values and contains methods, enumerators and properties for accessing them. There is no ability in FFmpeg API to remove special entries from the collection, so we have the same functionality in the wrapper library. ***AVDictionary*** class supports enumeration of the key-values entries. Class also supports copy data with usage of the ***ICloneable*** interface.

```
AVDictionary dict = new AVDictionary();
dict.SetValue("Key1","Value1");
dict.SetValue("Key2","Value2");
dict.SetValue("Key3","Value3");
Console.Write("Number of elements: \"{0}\"\nIndex of \"Key2\": \"{1}\"\n",
    dict.Count,dict.IndexOf("Key2"));
Console.Write("Keys:\t");
foreach (string key in dict.Keys) {
    Console.Write(" \"" + key + "\"");
}
Console.Write("\nValues:\t");
for (int i = 0; i < dict.Values.Count; i++) {
    Console.Write(" \"" + dict.Values[i] + "\"");
}
Console.Write("\nValue[\"Key1\"]:\t\"{0}\"\nValue[3]:\t\"{1}\"\n",
    dict["Key1"],dict[2]);
AVDictionary cloned = (AVDictionary)dict.Clone();
Console.Write("Cloned Entries: \n");
foreach (AVDictionaryEntry ent in cloned) {
    Console.Write("\"{0}\"=\"{1}\"\n", ent.key, ent.value);
}
cloned.Dispose();
dict.Dispose();
```

Result of execution code above is:

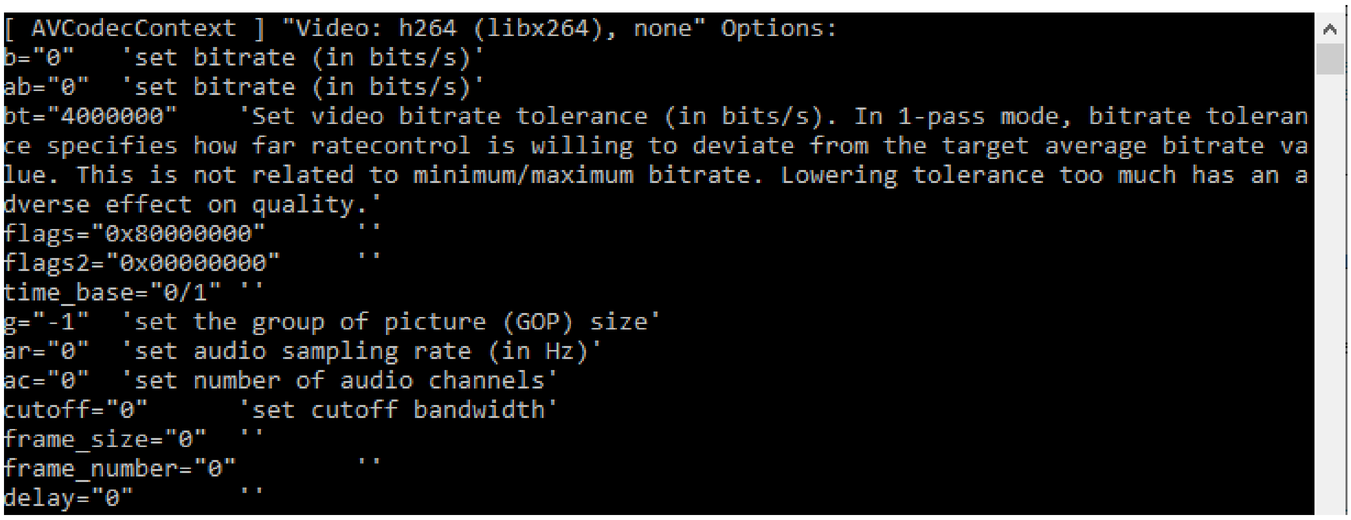


```
Number of elements: "3"
Index of "Key2": "1"
Keys:     "Key1" "Key2" "Key3"
Values:   "Value1" "Value2" "Value3"
Value["Key1"]:  "Value1"
Value[3]:       "Value3"
Cloned Entries:
"Key1"="Value1"
"Key2"="Value2"
"Key3"="Value3"
```

AVOptions

Helper class which exposes functionality accessing options of the library objects. Objects can be created from pointers or from ***AVBase*** class as were already mentioned. Example of enumerating existing object options:

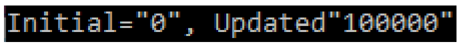
```
var ctx = new AVCodecContext(AVCodec.FindEncoder(AVCodecID.H264));
Console.WriteLine("{0} Options:",ctx);
var options = new AVOptions(ctx);
for (int i = 0; i < options.Count; i++)
{
    string value = "";
    if (0 == options.get(options[i].name, AVOptSearch.None, out value))
    {
        Console.WriteLine("{0}=\"{1}\"\t'{2}'",options[i].name,value,options[i].help);
    }
}
```



```
[ AVCodecContext ] "Video: h264 (libx264), none" Options:
b="0"   'set bitrate (in bits/s)'
ab="0"  'set bitrate (in bits/s)'
bt="4000000"    'Set video bitrate tolerance (in bits/s). In 1-pass mode, bitrate toleran
ce specifies how far ratecontrol is willing to deviate from the target average bitrate va
lue. This is not related to minimum/maximum bitrate. Lowering tolerance too much has an a
dverse effect on quality.'
flags="0x80000000"      ''
flags2="0x00000000"     ''
time_base="0/1" ''
g="-1"  'set the group of picture (GOP) size'
ar="0"  'set audio sampling rate (in Hz)'
ac="0"  'set number of audio channels'
cutoff="0"      'set cutoff bandwidth'
frame_size="0"  ''
frame_number="0"        ''
delay="0"       ''
```

Class gives ability to enumerate option names and option parameters and also gets or sets options values of the object.

```
var ctx = new AVCodecContext(AVCodec.FindEncoder(AVCodecID.H264));
var options = new AVOptions(ctx);
string initial = "";
options.get("b", AVOptSearch.None, out initial);
options.set_int("b", 100000, AVOptSearch.None);
string updated = "";
options.get("b", AVOptSearch.None, out updated);
Console.WriteLine("Initial=\"{0}\", Updated\"{1}\"",initial,updated);
```



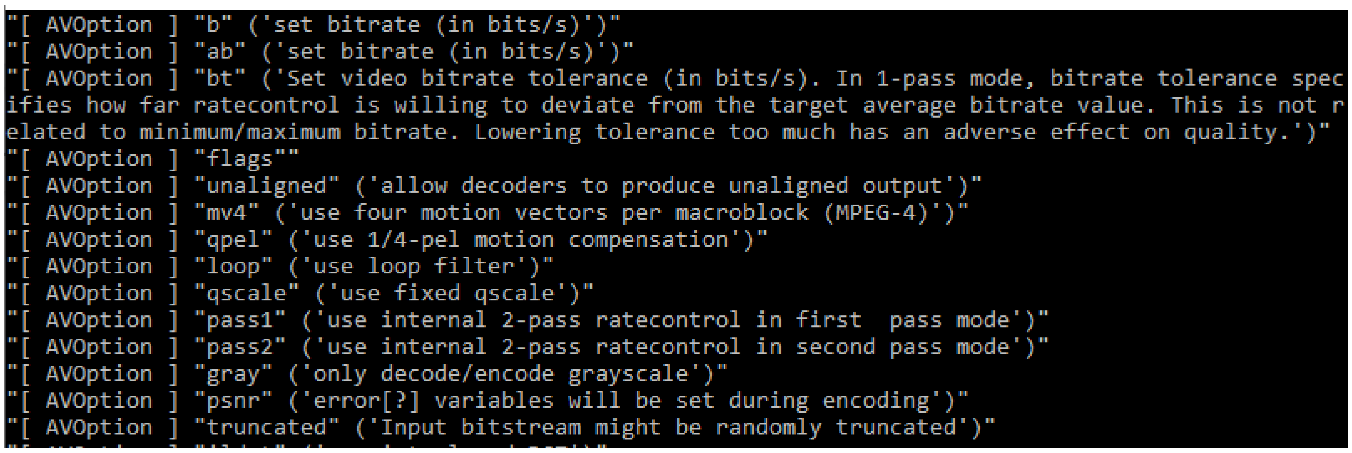
```
Initial="0", Updated"100000"
```

Due compatibility with old wrapper library versions some properties of library classes internally set object options instead of usage direct fields.

```
void FFmpeg::AVFilterGraph::scale_sws_opts::set(String^ value)
{
    auto p = _CreateObject<AVOptions>(m_pPointer);
    p->set("scale_sws_opts",value,AVOptSearch::None);
}
```

Available options also can be enumerated via ***AVClass*** object of the library:

```
var options = AVClass.GetCodecClass().option;
for (int i = 0; i < options.Length; i++)
{
    Console.WriteLine("\"{0}\"",options[i]);
}
```



```
"[ AVOption ] "b" ('set bitrate (in bits/s)')"
"[ AVOption ] "ab" ('set bitrate (in bits/s)')"
"[ AVOption ] "bt" ('Set video bitrate tolerance (in bits/s). In 1-pass mode, bitrate tolerance spec
ifies how far ratecontrol is willing to deviate from the target average bitrate value. This is not r
elated to minimum/maximum bitrate. Lowering tolerance too much has an adverse effect on quality.')"
"[ AVOption ] "flags""
"[ AVOption ] "unaligned" ('allow decoders to produce unaligned output')"
"[ AVOption ] "mv4" ('use four motion vectors per macroblock (MPEG-4)')"
"[ AVOption ] "qpel" ('use 1/4-pel motion compensation')"
"[ AVOption ] "loop" ('use loop filter')"
"[ AVOption ] "qscale" ('use fixed qscale')"
"[ AVOption ] "pass1" ('use internal 2-pass ratecontrol in first  pass mode')"
"[ AVOption ] "pass2" ('use internal 2-pass ratecontrol in second pass mode')"
"[ AVOption ] "gray" ('only decode/encode grayscale')"
"[ AVOption ] "psnr" ('error[?] variables will be set during encoding')"
"[ AVOption ] "truncated" ('Input bitstream might be randomly truncated')"
```

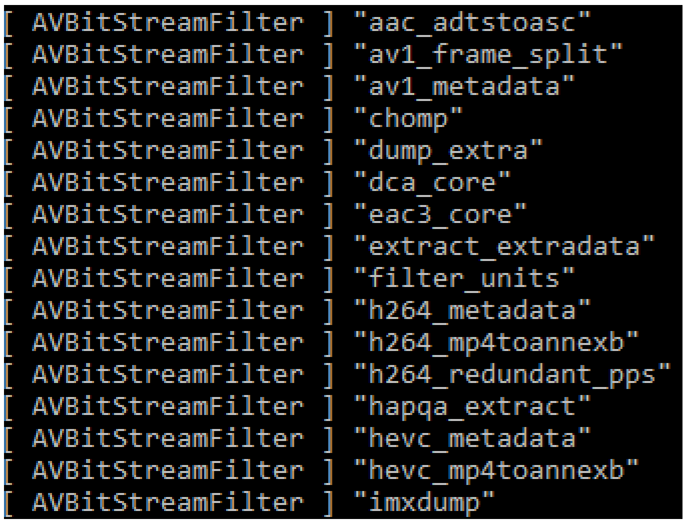
As in FFmpeg each context structure have ***AVClass*** object access, so next code gave same results:

```
var ctx = new AVCodecContext(AVCodec.FindEncoder(AVCodecID.H264));
var options = ctx.av_class.option;
for (int i = 0; i < options.Length; i++)
{
    Console.WriteLine("\"{0}\"",options[i]);
}
```

AVBitStreamFilter

Class which is implemented managed access to ***AVBitStreamFilter*** type of the FFmpeg *libavcodec* library. It contains the name and array of codec ids to which filter can be applied. All filters can be enumerated:

```
foreach(AVBitStreamFilter f in AVBitStreamFilter.Filters)
{
    Console.WriteLine("{0}",f);
}
```



```
[ AVBitStreamFilter ] "aac_adtstoasc"
[ AVBitStreamFilter ] "av1_frame_split"
[ AVBitStreamFilter ] "av1_metadata"
[ AVBitStreamFilter ] "chomp"
[ AVBitStreamFilter ] "dump_extra"
[ AVBitStreamFilter ] "dca_core"
[ AVBitStreamFilter ] "eac3_core"
[ AVBitStreamFilter ] "extract_extradata"
[ AVBitStreamFilter ] "filter_units"
[ AVBitStreamFilter ] "h264_metadata"
[ AVBitStreamFilter ] "h264_mp4toannexb"
[ AVBitStreamFilter ] "h264_redundant_pps"
[ AVBitStreamFilter ] "hapqa_extract"
[ AVBitStreamFilter ] "hevc_metadata"
[ AVBitStreamFilter ] "hevc_mp4toannexb"
[ AVBitStreamFilter ] "imxdump"
```

Filter can be searched by name:

```
var f = AVBitStreamFilter.GetByName("h264_mp4toannexb");
Console.Write("{0} Codecs: ",f);
foreach(AVCodecID id in f.codec_ids)
{
    Console.WriteLine("{0} ",id);
}
```

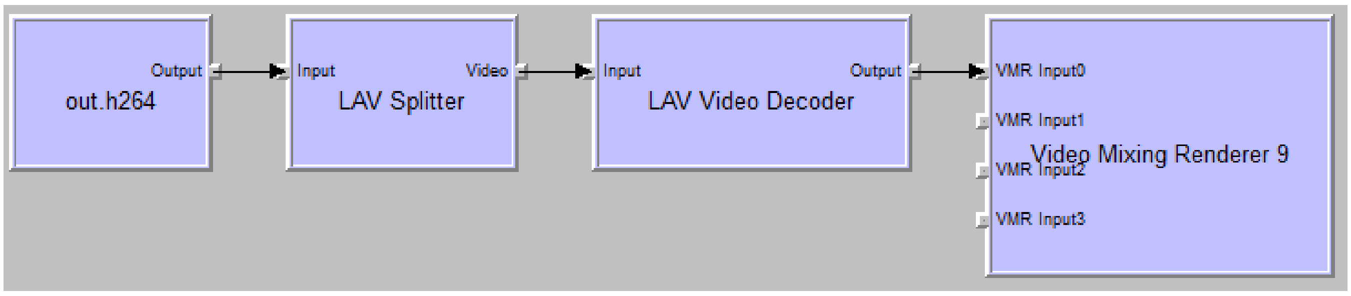
```
[ AVBitStreamFilter ] "h264_mp4toannexb" Codecs: H264
```

AVBSFContext

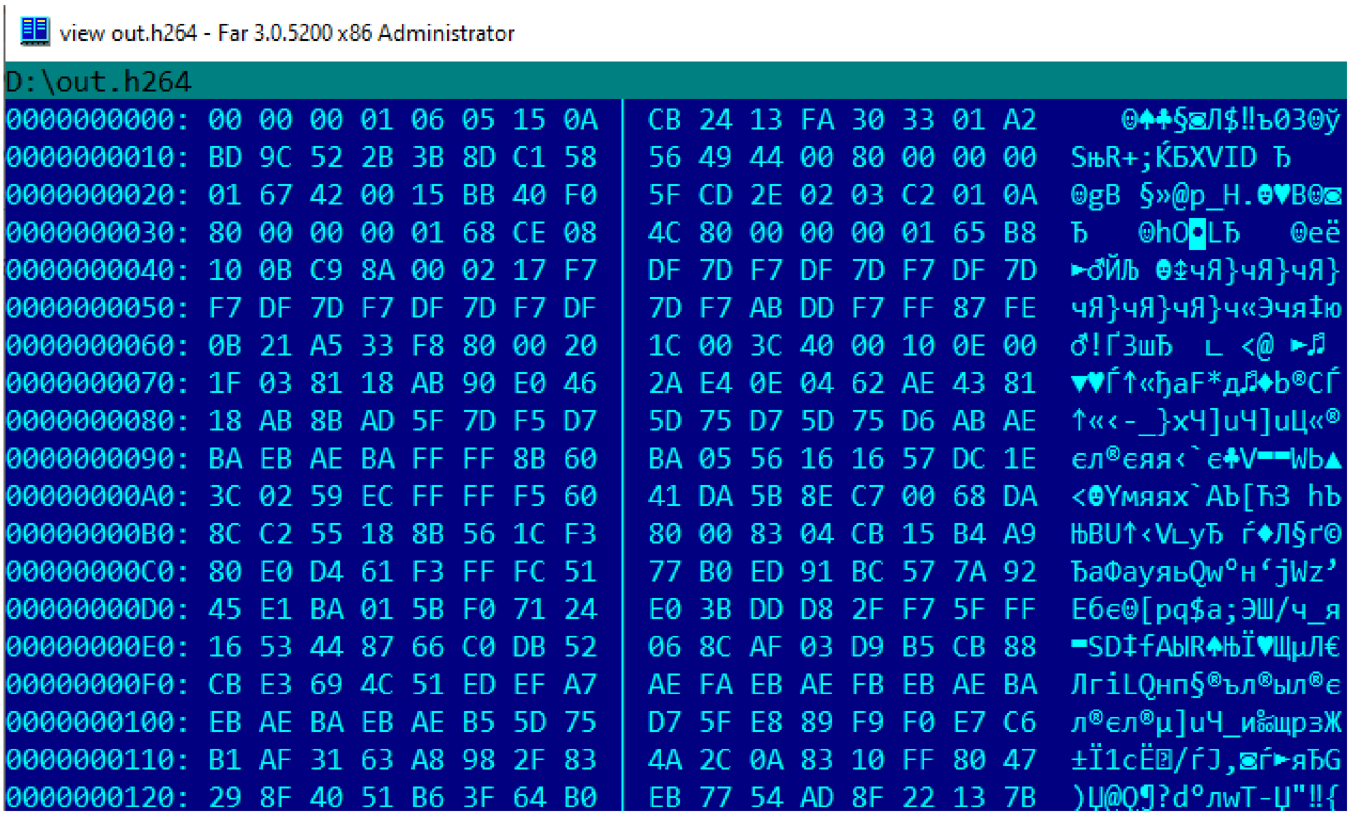
Class which is implemented managed access to ***AVBSFContext*** structure type of the FFmpeg *libavcodec* library. It is an instance of Bit Stream Filter and contains methods for filtering operations. Example of class usage:

```
var input = AVFormatContext.OpenInputFile(@"test.mp4");
if (input.FindStreamInfo() == 0) {
    int idx = input.FindBestStream(AVMediaType.VIDEO);
    var ctx = new AVBSFContext(
        AVBitStreamFilter.GetByName("h264_mp4toannexb"));
    input.streams[idx].codecpar.CopyTo(ctx.par_in);
    ctx.time_base_in = input.streams[idx].time_base;
    if (0 == ctx.Init()) {
        bool append = false;
        AVPacket pkt = new AVPacket();
        while (0 == input.ReadFrame(pkt)) {
            if (pkt.stream_index == idx) {
                ctx.SendPacket(pkt);
            }
            pkt.Free();
            if (0 == ctx.ReceivePacket(pkt)) {
                pkt.Dump(@"out.h264",append);
                pkt.Free();
                append = true;
            }
        }
        pkt.Dispose();
    }
    ctx.Dispose();
}
```

The code above performs conversion of **H264** video stream from opened **mp4** media file into annexb format - format with start codes, and saves the resulting bitstream into a file. The file, which is produced, can be played with **VLC** media player, with **graphedit** tool or with **ffplay**.



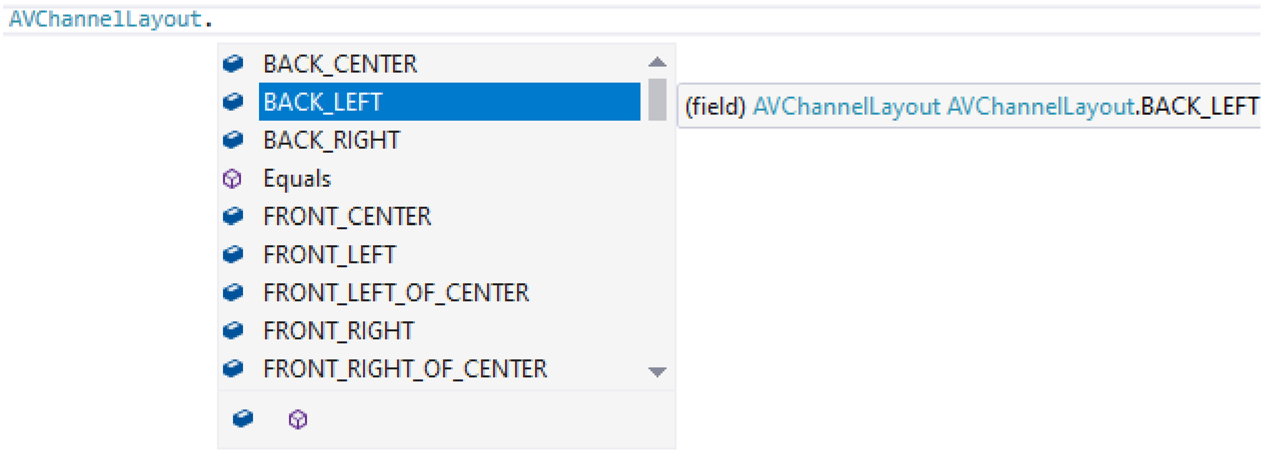
In the hex dump of a file it is possible to see that data is in annexb format:



```
view out.h264 - Far 3.0.5200 x86 Administrator
D:\out.h264
0000000000: 00 00 00 01 06 05 15 0A │ CB 24 13 FA 30 33 01 A2     ☺♠♣§◙Л$‼ъ03☺ў
0000000010: BD 9C 52 2B 3B 8D C1 58 │ 56 49 44 00 80 00 00 00  SњR+;ЌБXVID Ђ
0000000020: 01 67 42 00 15 BB 40 F0 │ 5F CD 2E 02 03 C2 01 0A  ☺gB §»@p_Н.☻♥В☺◙
0000000030: 80 00 00 00 01 68 CE 08 │ 4C 80 00 00 00 01 65 B8  Ђ   ☺hО◘LЂ   ☺eё
0000000040: 10 0B C9 8A 00 02 17 F7 │ DF 7D F7 DF 7D F7 DF 7D  ►♂ЙЉ ☻↨чЯ}чЯ}чЯ}
0000000050: F7 DF 7D F7 DF 7D F7 DF │ 7D F7 AB DD F7 FF 87 FE  чЯ}чЯ}чЯ}ч«Эчя‡ю
0000000060: 0B 21 A5 33 F8 80 00 20 │ 1C 00 3C 40 00 10 0E 00  ♂!Ґ3шЂ  ∟ <@ ►♫
0000000070: 1F 03 81 18 AB 90 E0 46 │ 2A E4 0E 04 62 AE 43 81  ▼♥Ѓ↑«ђaF*д♫♦b®CЃ
0000000080: 18 AB 8B AD 5F 7D F5 D7 │ 5D 75 D7 5D 75 D6 AB AE  ↑«‹-_}хЧ]uЧ]uЦ«®
0000000090: BA EB AE BA FF FF 8B 60 │ BA 05 56 16 16 57 DC 1E  єл®єяя‹`є♣V▬▬WЬ▲
00000000A0: 3C 02 59 EC FF FF F5 60 │ 41 DA 5B 8E C7 00 68 DA  <☻Yмяях`AЪ[Ћ3 hЪ
00000000B0: 8C C2 55 18 8B 56 1C F3 │ 80 00 83 04 CB 15 B4 A9  ЊBU↑‹V∟уЂ ѓ♦Л§ґ©
00000000C0: 80 E0 D4 61 F3 FF FC 51 │ 77 B0 ED 91 BC 57 7A 92  ЂаФауяьQw°н‘jWz’
00000000D0: 45 E1 BA 01 5B F0 71 24 │ E0 3B DD D8 2F F7 5F FF  Ебє☺[pq$а;ЭШ/ч_я
00000000E0: 16 53 44 87 66 C0 DB 52 │ 06 8C AF 03 D9 B5 CB 88  ▬SD‡fАЫR♠ЊЇ♥ЩµЛ€
00000000F0: CB E3 69 4C 51 ED EF A7 │ AE FA EB AE FB EB AE BA  ЛгiLQнп§®ъл®ыл®є
0000000100: EB AE BA EB AE B5 5D 75 │ D7 5F E8 89 F9 F0 E7 C6  л®єл®µ]uЧ_и‰щрзЖ
0000000110: B1 AF 31 63 A8 98 2F 83 │ 4A 2C 0A 83 10 FF 80 47  ±Ї1cЁ/ѓJ,◙ѓ►яЂG
0000000120: 29 8F 40 51 B6 3F 64 B0 │ EB 77 54 AD 8F 22 13 7B  )Џ@Q¶?d°лwT-Џ"‼{
```

AVChannelLayout

Helper value class of definitions *AV_CH_** for audio channels masks from *libavutil* library. Class contains all available definitions as public static class fields:



```
AVChannelLayout.
```

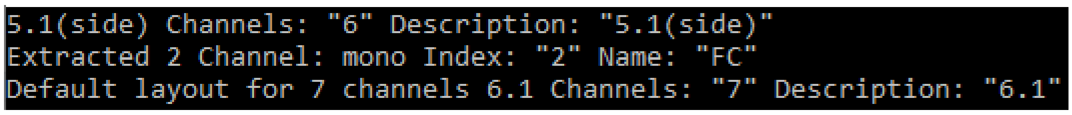
- BACK_CENTER
- BACK_LEFT
- BACK_RIGHT
- Equals
- FRONT_CENTER
- FRONT_LEFT
- FRONT_LEFT_OF_CENTER
- FRONT_RIGHT
- FRONT_RIGHT_OF_CENTER

(field) AVChannelLayout AVChannelLayout.BACK_LEFT

Class represent the 64 bit integer value and cover implicit conversions operators. It also contains helper methods as wrap of FFmpeg library APIs and has overridden string representation of the value.

```
AVChannelLayout ch = AVChannelLayout.LAYOUT_5POINT1;
Console.WriteLine("{0} Channels: \"{1}\" Description: \"{2}\"",
    ch,ch.channels,ch.description);
AVChannelLayout ex = ch.extract_channel(2);
Console.WriteLine("Extracted 2 Channel: {0} Index: \"{1}\" Name: \"{2}\"",
    ex,ch.get_channel_index(ex),ex.name);
ch = AVChannelLayout.get_default_channel_layout(7);
Console.WriteLine("Default layout for 7 channels {0} "+
    "Channels: \"{1}\" Description: \"{2}\"",
    ch,ch.channels,ch.description);
```

The code above displays basic operations with the class. All class methods are based on FFmpeg library APIs, so it is easy to understand how to use them. The execution result of the code above:



```
5.1(side) Channels: "6" Description: "5.1(side)"
Extracted 2 Channel: mono Index: "2" Name: "FC"
Default layout for 7 channels 6.1 Channels: "7" Description: "6.1"
```

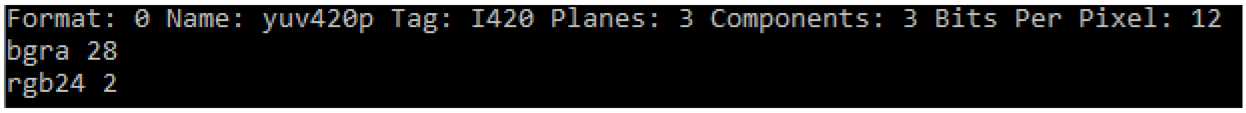
There is also an ***AVChannels*** class in a wrapper library which is the separate static class with exported APIs for ***AVChannelLayout***.

AVPixelFormat

Another helper value class for enumeration with the same name from *libavutil* library. It describes the pixel format of video data. Class contains all formats which are exposed by original enumeration. Along with it, the class extended with methods and properties based on FFmpeg APIs. It handles implicit conversion from to integer types.

```
AVPixelFormat fmt = AVPixelFormat.YUV420P;
byte[] bytes = BitConverter.GetBytes(fmt.codec_tag);
string tag = "";
for (int i = 0; i < bytes.Length; i++) { tag += (char)bytes[i]; }
Console.WriteLine("Format: {0} Name: {1} Tag: {2} Planes: {3} " +
    "Components: {4} Bits Per Pixel: {5}",(int)fmt,
    fmt.name,tag,fmt.planes, fmt.format.nb_components,
    fmt.format.bits_per_pixel);
fmt = AVPixelFormat.get_pix_fmt("rgb32");
Console.WriteLine("{0} {1}",fmt.name, (int)fmt);
FFLoss loss = FFLoss.NONE;
fmt = AVPixelFormat.find_best_pix_fmt_of_2(AVPixelFormat.RGB24,
    AVPixelFormat.RGB32,AVPixelFormat.RGB444BE,false,ref loss);
Console.WriteLine("{0} {1}",fmt.name, (int)fmt);
```

Execution of the code above gives a result:

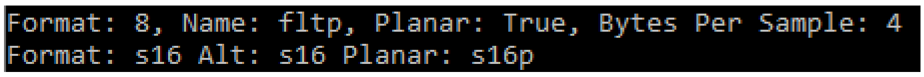


```
Format: 0 Name: yuv420p Tag: I420 Planes: 3 Components: 3 Bits Per Pixel: 12
bgra 28
rgb24 2
```

AVSampleFormat

Also a helper class for *libavutil* enumeration with the same name. It manages the format description of the audio data. As other classes, it handles basic operations and exposes properties and methods which are done as FFmpeg APIs.

```
AVSampleFormat fmt = AVSampleFormat.FLTP;
Console.WriteLine("Format: {0}, Name: {1}, Planar: {2}, Bytes Per Sample: {3}",
    (int)fmt,fmt.name,fmt.is_planar,fmt.bytes_per_sample);
fmt = AVSampleFormat.get_sample_fmt("s16");
var alt = fmt.get_alt_sample_fmt(false);
var pln = fmt.get_planar_sample_fmt();
Console.WriteLine("Format: {0} Alt: {1} Planar: {2}",fmt, alt, pln);
```



```
Format: 8, Name: fltp, Planar: True, Bytes Per Sample: 4
Format: s16 Alt: s16 Planar: s16p
```

FFmpeg APIs which operate with ***AVSampleFormat*** are also exported in a separate static class ***AVSampleFmt***.

AVSamples

Helper static class which exposes methods for *libavutil* APIs and operates with audio data samples. It is able to allocate buffers for storing audio data with different formats and copy data from extended buffers into the ***AVFrame***/***AVPicture*** class.
Next code displays how to allocate a buffer for audio data, filling that buffer with silence and copying the decoded audio frame to that buffer.

```
var input = AVFormatContext.OpenInputFile(@"test.mp4");
if (input.FindStreamInfo() == 0) {
    int idx = input.FindBestStream(AVMediaType.AUDIO);
    AVCodecContext decoder = input.streams[idx].codec;
    var codec = AVCodec.FindDecoder(decoder.codec_id);
    if (decoder.Open(codec) == 0) {
        AVPacket pkt = new AVPacket();
        AVFrame frame = new AVFrame();
        bool got_frame = false;
        while (input.ReadFrame(pkt) == 0 && !got_frame) {
            if (pkt.stream_index == idx) {
                decoder.DecodeAudio(frame,ref got_frame,pkt);
                if (got_frame) {
                    int size = AVSamples.get_buffer_size(frame.channels,
                        frame.nb_samples, frame.format);
                    AVMemPtr ptr = new AVMemPtr(size);
                    IntPtr[] data = new IntPtr[frame.channels];
                    AVSamples.fill_arrays(ref data, ptr,
                        frame.channels, frame.nb_samples, frame.format);
                    AVSamples.set_silence(data, 0,
                        frame.nb_samples, frame.channels, frame.format);
                    ptr.Dump(@"silence.bin");
                    AVSamples.copy(data, frame, 0, 0, frame.nb_samples);
                    ptr.Dump(@"data.bin");
                    ptr.Dispose();
                    frame.Free();
                }
            }
            pkt.Free();
        }
        frame.Dispose();
        pkt.Dispose();
    }
}
```

AVMath

Helper static class which exposes some useful mathematics APIs from *libavutil* library. Of course, it is not mandatory to use them as we have math from .NET, but in some cases it can be helpful. For example timestamps conversion from one base into another and timestamps comparing.

```
AVCodecContext c = new AVCodecContext(null);
//... Initalize encoder context
AVFrame frame = new AVFrame();
//... Prepare frame data
frame.pts = AVMath.rescale_q(frame.nb_samples, new AVRational(1, c.sample_rate), c.time_base);
//... Pass frame to encoder
```

In code snippet above performed conversion from sample rate basis into context time base timestamp which later will be passed to encoder.

SwrContext

Managed wrapper of resampling context structure of *libswresample* library. Class contains methods and properties to work with underlying library structure. It allows to set properties, initialize context, and perform resampling of the audio data.
Next code sample displays how to initialize resampling context and perform conversion. It opens the file with an audio stream. Decode audio data. Resampling resulted data into **S16** stereo format with the same sample rate as were on input and saved the result into a binary file. Definitions of *fopen*, *fwrite* and *fclose* functions can be found in ***AVIOContext*** code samples.

```
var input = AVFormatContext.OpenInputFile(@"test.mp4");
if (input.FindStreamInfo() == 0) {
    int idx = input.FindBestStream(AVMediaType.AUDIO);
    AVCodecContext decoder = input.streams[idx].codec;
    var codec = AVCodec.FindDecoder(decoder.codec_id);
    decoder.sample_fmt = codec.sample_fmts != null ? codec.sample_fmts[0] : AVSampleFormat.FLTP;
    if (decoder.Open(codec) == 0) {
        AVPacket pkt = new AVPacket();
        AVFrame frame = new AVFrame();
        AVMemPtr ptr = null;
        SwrContext swr = null;
        IntPtr file = fopen(@"out.bin","w+b");
        while (input.ReadFrame(pkt) == 0) {
            if (pkt.stream_index == idx) {
                bool got_frame = false;
                decoder.DecodeAudio(frame,ref got_frame,pkt);
                if (got_frame) {
                    int size = AVSamples.get_buffer_size(frame.channels,
                        frame.nb_samples, frame.format);
                    if (swr == null) {
                        swr = new SwrContext(AVChannelLayout.LAYOUT_STEREO,
                            AVSampleFormat.S16, frame.sample_rate,
                            frame.channel_layout, frame.format, frame.sample_rate);
                        swr.Init();
                    }
                    if (ptr != null && ptr.size < size) {
                        ptr.Dispose();
                        ptr = null;
                    }
                    if (ptr == null) {
                        ptr = new AVMemPtr(size);
                    }
                    int count = swr.Convert(new IntPtr[] { ptr },frame.nb_samples,frame.data,frame.nb_samples);
                    if (count > 0) {
                        int bps = AVSampleFmt.get_bytes_per_sample(AVSampleFormat.S16) * frame.channels;
                        fwrite(ptr,bps,count,file);
                    }
                    frame.Free();
                }
            }
            pkt.Free();
        }
        fclose(file);
        if (swr != null) swr.Dispose();
        if (ptr != null) ptr.Dispose();
        frame.Dispose();
        pkt.Dispose();
    }
}
```

The resulting file can be played with **ffplay** with format parameters. In case your audio data has 44100 Hz sample rate, otherwise replace rate in next command line arguments.

```
ffplay -f s16le -channel_layout 3 -channels 2 -ar 44100 out.bin
```
.

SwsContext

Managed wrapper of scaling context structure of *libswscale* library. Class supports scaling and colorspace conversion of the video data.
Next sample code displays how to initialize scaling context and perform conversion of the video frames into image files. It opens the file with a video stream. Decode video data. Initialize context and temporal data pointer, create a .NET bitmap object associated with an allocated data pointer, and save each frame to a file.

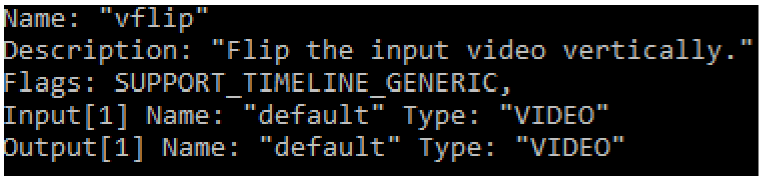
```
var input = AVFormatContext.OpenInputFile(@"test.mp4");
if (input.FindStreamInfo() == 0) {
    int idx = input.FindBestStream(AVMediaType.VIDEO);
    AVCodecContext decoder = input.streams[idx].codec;
    var codec = AVCodec.FindDecoder(decoder.codec_id);
    if (decoder.Open(codec) == 0) {
        AVPacket pkt = new AVPacket();
        AVFrame frame = new AVFrame();
        AVMemPtr ptr = null;
        SwsContext sws = null;
        int image = 0;
        Bitmap bmp = null;
        while (input.ReadFrame(pkt) == 0) {
            if (pkt.stream_index == idx) {
                bool got_frame = false;
                decoder.DecodeVideo(frame,ref got_frame,pkt);
                if (got_frame) {
                    if (sws == null) {
                        sws = new SwsContext(frame.width, frame.height, frame.format,
                            frame.width, frame.height, AVPixelFormat.BGRA, SwsFlags.FastBilinear);
                    }
                    if (ptr == null) {
                        int size = AVPicture.GetSize(AVPixelFormat.BGRA,frame.width, frame.height);
                        ptr = new AVMemPtr(size);
                    }
                    sws.Scale(frame,0,frame.height,new IntPtr[] { ptr },new int[] { frame.width << 2 });
                    if (bmp == null)
                    {
                        bmp = new Bitmap(frame.width, frame.height,
                            frame.width << 2, PixelFormat.Format32bppRgb, ptr);
                    }
                    bmp.Save(string.Format(@"image{0}.png",image++));
                    frame.Free();
                }
            }
            pkt.Free();
        }
        if (sws != null) sws.Dispose();
        if (ptr != null) ptr.Dispose();
        if (bmp != null) bmp.Dispose();
        frame.Dispose();
        pkt.Dispose();
    }
}
```

Such frame conversion is implemented by the library with the ***AVFrame***.*ToBitmap()* method. You can see how that is done in ***AVFrame*** sample code. Difference that in the current example is that the data buffer and bitmap object are allocated only once, which gives better performance.

AVFilter

Managed wrapper of ***AVFilter*** structure of *libavfilter* library. Class describes every filter item in the filtering platform. It defines object fields and methods. Class has no public constructor, and can be accessed from the static methods.

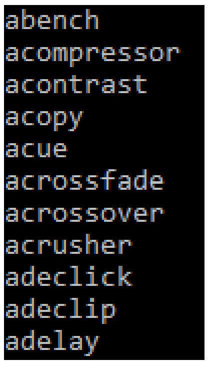
```
var f = AVFilter.GetByName("vflip");
Console.WriteLine("Name: \"{0}\"\nDescription: \"{1}\"\nFlags: {2}, ",
    f.name, f.description,f.flags);
for (int i = 0; i < f.inputs.Count; i++)
    Console.WriteLine("Input[{0}] Name: \"{1}\" Type: \"{2}\"",
        i + 1, f.inputs[i].name, f.inputs[i].type);
for (int i = 0; i < f.outputs.Count; i++)
    Console.WriteLine("Output[{0}] Name: \"{1}\" Type: \"{2}\"",
        i + 1, f.outputs[i].name, f.outputs[i].type);
```



```
Name: "vflip"
Description: "Flip the input video vertically."
Flags: SUPPORT_TIMELINE_GENERIC,
Input[1] Name: "default" Type: "VIDEO"
Output[1] Name: "default" Type: "VIDEO"
```

Class also has the ability to enumerate existing filters available in the library.

```
foreach (var f in AVFilter.Filters)
{
    Console.WriteLine(f);
}
```

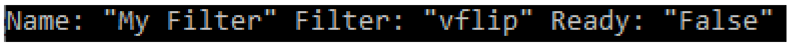


```
abench
acompressor
acontrast
acopy
acue
acrossfade
acrossover
acrusher
adeclick
adeclip
adelay
```

AVFilterContext

Managed class represents ***AVFilterContext*** structure of the *libavfilter* library. It is used to describe filter instances in a filter graph. Have no public constructor, only able to be created by calling ***AVFilterGraph*** class methods.

```
AVFilterGraph graph = new AVFilterGraph();
AVFilterContext ctx = graph.CreateFilter(AVFilter.GetByName("vflip"),"My Filter");
Console.WriteLine("Name: \"{0}\" Filter: \"{1}\" Ready: \"{2}\"",
    ctx.name, ctx.filter,ctx.ready);
```



```
Name: "My Filter" Filter: "vflip" Ready: "False"
```

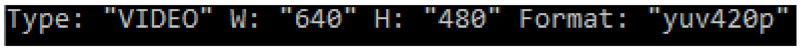
Each filter context output can be linked with the context input of the other filter. There are few special filters: sink and source. Source able to receive data and should be inserted first into the graph chain. And the sink is the endpoint of the chain. Sink provides output from the filter graph.
Source filter have special name: "*buffer*" for video and "*abuffer*" for the audio, creation such filter always must have the initialization parameters:

```
AVFilterGraph graph = new AVFilterGraph();
AVFilterContext src = graph.CreateFilter(AVFilter.GetByName("buffer"),"in",
    "video_size=640x480:pix_fmt=0:time_base=1/25:pixel_aspect=1/1",IntPtr.Zero);
```

Destination filter names are "*buffersink*" and "*abuffersink*" for video and audio respectively. Extend the previous code by adding sink filter creation and connecting it with previously created video source.

```
var sink = graph.CreateFilter(AVFilter.GetByName("buffersink"),"out");
src.Link(sink);
graph.Config();
var input = sink.inputs[0];
Console.WriteLine("Type: \"{0}\" W: \"{1}\" H: \"{2}\" Format: \"{3}\"",
    input.type, input.w, input.h,((AVPixelFormat)input.format).name);
```

Once we call the configuring method of the graph, it sets up the chain parameters, so we have format settings on the sink input.



```
Type: "VIDEO" W: "640" H: "480" Format: "yuv420p"
```

There are special classes which help to operate with sink and source filters context. ***AVBufferSrc*** - have methods for filter initialization and providing input frames into underlying source filter context.

AVBufferSink - contains abilities for getting properties of the graph endpoint and receiving resulting frames.
The next complete example code performs a vertical flip filtering effect of the picture from one image file and saves the result into another file:

```
var frame = AVFrame.FromImage((Bitmap)Bitmap.FromFile(@"image.jpg"),
    AVPixelFormat.YUV420P);
string fmt = string.Format(
        "video_size={0}x{1}:pix_fmt={2}:time_base=1/1:pixel_aspect1/1",
        frame.width, frame.height, (int)frame.format);

AVFilterGraph graph = new AVFilterGraph();
AVFilterContext src = graph.CreateFilter(AVFilter.GetByName("buffer"),"in",fmt,
    IntPtr.Zero);
var flip = graph.CreateFilter(AVFilter.GetByName("vflip"),"My Filter");
var sink = graph.CreateFilter(AVFilter.GetByName("buffersink"),"out");
src.Link(flip);
flip.Link(sink);
graph.Config();

AVBufferSink _sink = new AVBufferSink(sink);
AVBufferSrc _source = new AVBufferSrc(src);
_source.add_frame(frame);
frame.Free();
_sink.get_frame(frame);
frame.ToBitmap().Save(@"out_image.jpg");
```

AVFilterGraph

Managed class for ***AVFilterGraph*** structure of the *libavfilter* library. Class manage filters chain and connections between them. It contains graph configuring and filter context creation methods which were described previously. In addition, graphs have the ability to enumerate all filters in a chain:

```
AVFilterGraph graph = new AVFilterGraph();
var src = graph.CreateFilter(AVFilter.GetByName("buffer"),"in",
    "video_size=640x480:pix_fmt=0:time_base=1/25:pixel_aspect=1/1",IntPtr.Zero);
var flip = graph.CreateFilter(AVFilter.GetByName("vflip"),"My Filter");
var sink = graph.CreateFilter(AVFilter.GetByName("buffersink"),"out");

foreach (AVFilterContext f in graph.filters)
{
    Console.Write("\"" + f + "\" ");
}
```

```
"in" "My Filter" "out"
```

In an additional way of graph initialization, class allows it to be generated from the parameters string, which includes filter name and its parameters. Source and sink filters in that case should be created in a regular way with setting up the Inputs and output parameters for intermediate filter chain initialization.

```
AVFilterGraph graph = new AVFilterGraph();
var src = graph.CreateFilter(AVFilter.GetByName("buffer"),"in",
    "video_size=640x480:pix_fmt=0:time_base=1/25:pixel_aspect=1/1",IntPtr.Zero);
var sink = graph.CreateFilter(AVFilter.GetByName("buffersink"),"out");

AVFilterInOut outputs = new AVFilterInOut();
outputs.name       = "in";
outputs.filter_ctx = src;

AVFilterInOut inputs  = new AVFilterInOut();
inputs.name       = "out";
inputs.filter_ctx = sink;

graph.ParsePtr("vflip,scale=2:3", inputs, outputs);
graph.Config();
foreach (AVFilterContext f in graph.filters)
{
    Console.Write("\"" + f + "\" ");
}
```

The code above demonstrates filters chain creation and configuring graph from initialization parameters. The output from the sample:

```
"in" "out" "Parsed_vflip_0" "Parsed_scale_1"
```

We add "*in*" and "*out*" filters to the graph, setup inputs and outputs structures and call method to build a chain with two intermediate filters: *vertical flip* and *scaling*.

AVDevices

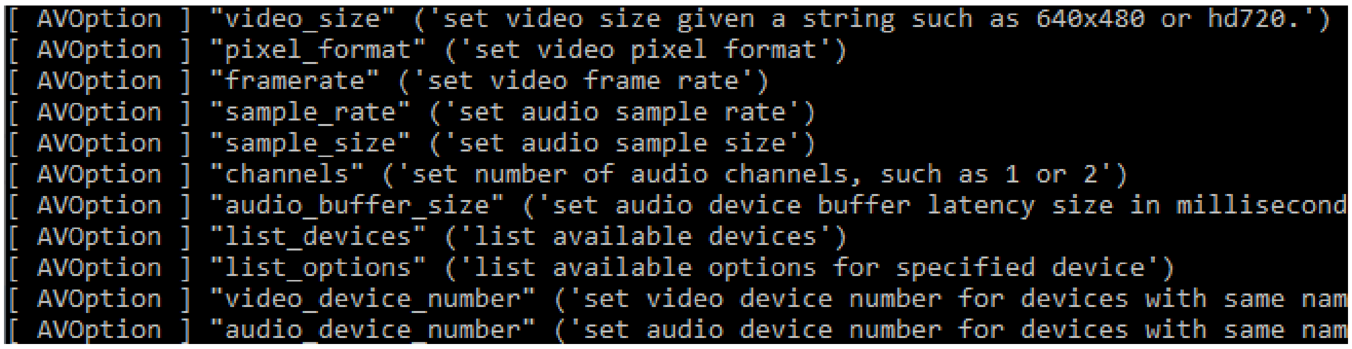
Managed static class for accessing *libavdevice* library APIs. All methods are static and allow access to collection of devices.

```
AVInputFormat fmt = null;
do {
    fmt = AVDevices.input_video_device_next(fmt);
    if (fmt != null) {
        Console.WriteLine(fmt);
    }
} while (fmt != null);
```

```
[ AVInputFormat ] "DirectShow capture"
[ AVInputFormat ] "GDI API Windows frame grabber"
[ AVInputFormat ] "Libavfilter virtual input device"
[ AVInputFormat ] "VfW video capture"
```

The devices registered for ability accessing with the *libavformat* API. Additional registration API call is not required as all necessary API calls are performed once you access either ***AVDevices*** class or any *libavformat* wrapper classes APIs. Each format class contains available options to access the target device from the input device subsystem. Options can be listed from the input format:

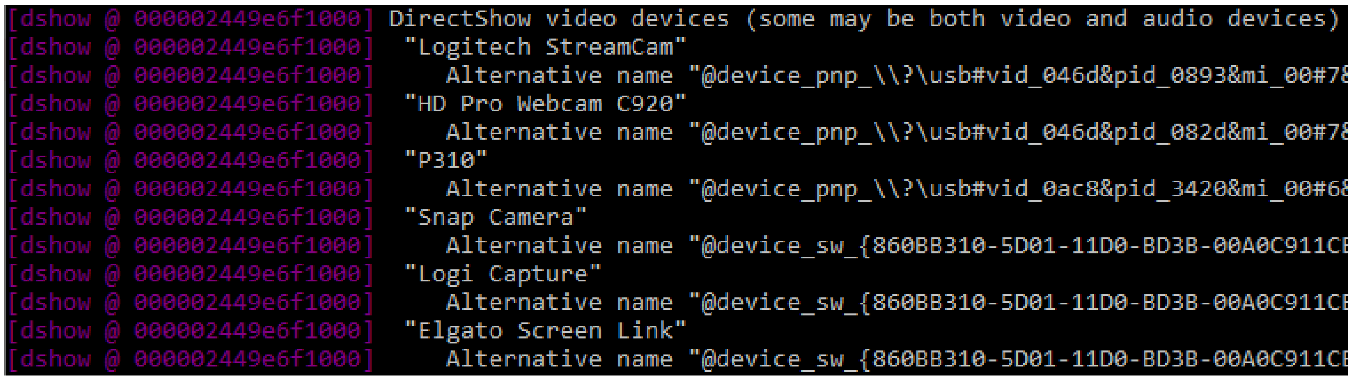
```
var fmt = AVInputFormat.FindInputFormat("dshow");
foreach (var opt in fmt.priv_class.option) {
    Console.WriteLine(opt);
}
```



```
[ AVOption ] "video_size" ('set video size given a string such as 640x480 or hd720.')
[ AVOption ] "pixel_format" ('set video pixel format')
[ AVOption ] "framerate" ('set video frame rate')
[ AVOption ] "sample_rate" ('set audio sample rate')
[ AVOption ] "sample_size" ('set audio sample size')
[ AVOption ] "channels" ('set number of audio channels, such as 1 or 2')
[ AVOption ] "audio_buffer_size" ('set audio device buffer latency size in millisecond
[ AVOption ] "list_devices" ('list available devices')
[ AVOption ] "list_options" ('list available options for specified device')
[ AVOption ] "video_device_number" ('set video device number for devices with same nam
[ AVOption ] "audio_device_number" ('set audio device number for devices with same nam
```

Options should be set on ***AVFormatContext*** creation. You can specify resolution or device number. For example let's see how can be displayed list of available **DirectShow** devices:

```
AVDictionary opt = new AVDictionary();
opt.SetValue("list_devices", "true");
var fmt = AVInputFormat.FindInputFormat("dshow");
AVFormatContext ctx = null;
AVFormatContext.OpenInput(out ctx, fmt, "video=dummy", opt);
```



```
[dshow @ 000002449e6f1000] DirectShow video devices (some may be both video and audio devices)
[dshow @ 000002449e6f1000]  "Logitech StreamCam"
[dshow @ 000002449e6f1000]     Alternative name "@device_pnp_\\?\usb#vid_046d&pid_0893&mi_00#78
[dshow @ 000002449e6f1000]  "HD Pro Webcam C920"
[dshow @ 000002449e6f1000]     Alternative name "@device_pnp_\\?\usb#vid_046d&pid_082d&mi_00#78
[dshow @ 000002449e6f1000]  "P310"
[dshow @ 000002449e6f1000]     Alternative name "@device_pnp_\\?\usb#vid_0ac8&pid_3420&mi_00#68
[dshow @ 000002449e6f1000]  "Snap Camera"
[dshow @ 000002449e6f1000]     Alternative name "@device_sw_{860BB310-5D01-11D0-BD3B-00A0C911CE
[dshow @ 000002449e6f1000]  "Logi Capture"
[dshow @ 000002449e6f1000]     Alternative name "@device_sw_{860BB310-5D01-11D0-BD3B-00A0C911CE
[dshow @ 000002449e6f1000]  "Elgato Screen Link"
[dshow @ 000002449e6f1000]     Alternative name "@device_sw_{860BB310-5D01-11D0-BD3B-00A0C911CE
```

AVDeviceInfoList

Managed class describing list of ***AVDeviceInfo*** structures of *libavdevice* library. Class handling devices collection, and can be accessed from methods of ***AVDevices*** class, without a public constructor.

```
AVDeviceInfoList devices = AVDevices.list_input_sources(
        AVInputFormat.FindInputFormat("dshow"),null,null);
if (devices != null) {
    foreach (AVDeviceInfo dev in devices) {
        Console.WriteLine(dev);
    }
}
```

AVDeviceInfo

Managed class for ***AVDeviceInfo*** structure of *libavdevice* library. Class contains strings of device description and device name. It can be accessed from the ***AVDeviceInfoList*** class.

Advanced

Based on the documentation above we can arrange some advanced topics. In most cases of wrapper library usage it may not be needed, but it is good to know any other features of implementation.

Delegates and callbacks

In some cases of API usage it is required to have the ability to set up callbacks for an API or a structure method. Such callback functions are designed to be implemented as static methods and they have *opaque* object as parameter. Each callback method has a delegate:

```
public delegate void AVBufferFreeCB(Object^ opaque, IntPtr buf);
```

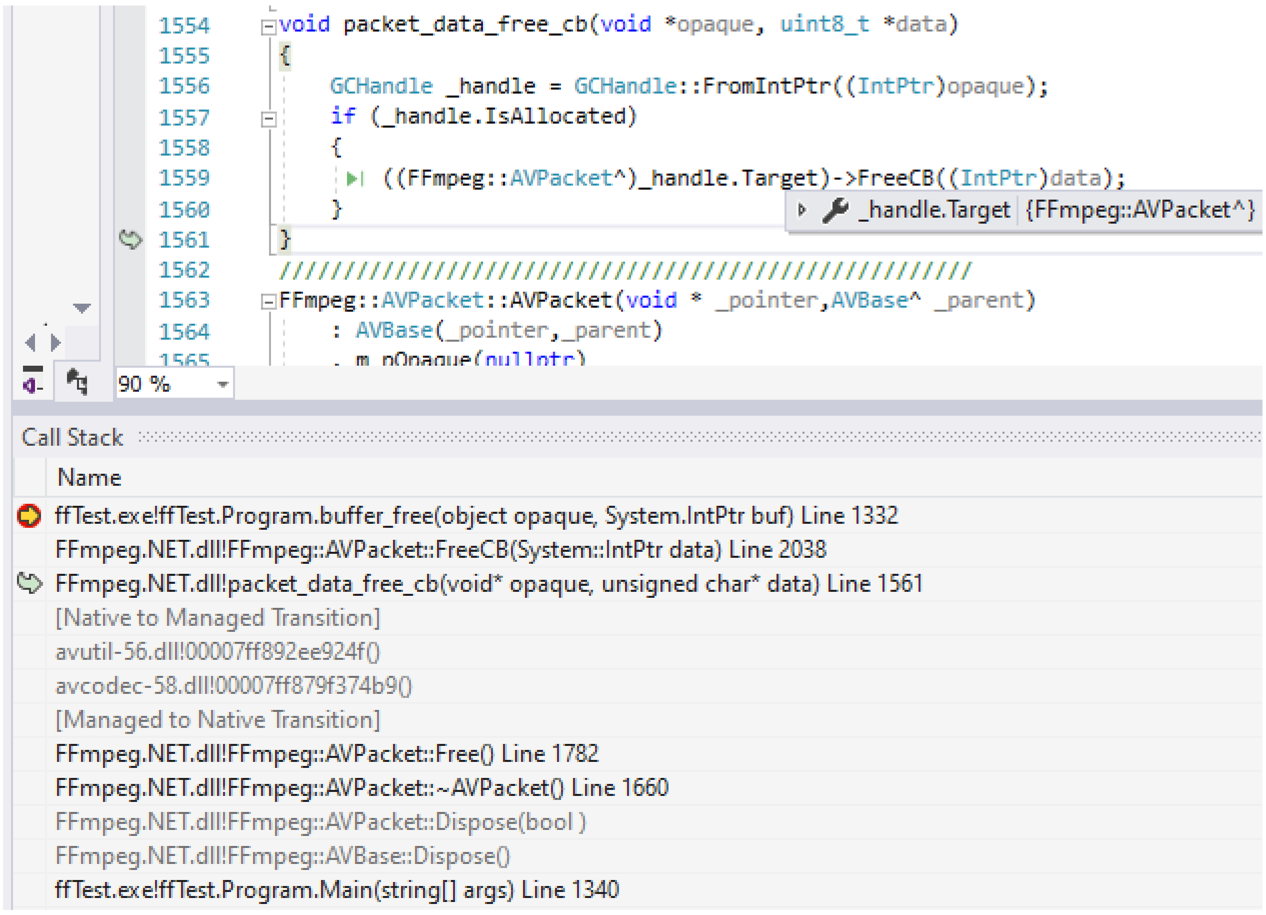
In the library it is only needed to know how the delegate of the method looks, and make a callback method with the same arguments and return type. Callbacks can be set as an argument of function or object constructor or even designed as object property.

```
static void buffer_free(object opaque, IntPtr buf)
{
    Marshal.FreeCoTaskMem(buf);
}

static void Main(string[] args)
{
    int cb = 1024 * 1024;
    var pkt = new AVPacket(Marshal.AllocCoTaskMem(cb), cb, buffer_free, null);
    //...
    pkt.Dispose();
}
```

Example above shows how to create ***AVPacket*** with custom allocated data. Once the packet is disposed then the passed free callback method is called. As were mentioned, it is possible to use *opaque* and cast it to another object which is used as another argument for callback creation:

```
static int read_packet(object opaque, IntPtr buf, int buf_size)
{
    var s = (opaque as Stream);
    long available = (s.Length - s.Position);
    if (available == 0) return AVRESULT.EOF;
    if ((long)buf_size > available) buf_size = (int)available;
    var buffer = new byte[buf_size];
    buf_size = s.Read(buffer,0,buf_size);
    Marshal.Copy(buffer, 0, buf,buf_size);
    return buf_size;
}

static void Main(string[] args)
{
    var stream = File.OpenRead(@"Test.avi");
    var avio_ctx = new AVIOContext(new AVMemPtr(4096), 0, stream, read_packet, null, null);
    //...
}
```

Example above demonstrates callback implementation which passed in the constructor of an ***AVIOContext*** object.
All callbacks in a library have their own background - the native callback function which stays as a layer between managed and unmanaged code and can provide arguments in a proper way into managed callback methods. This can be easily displayed then we set breakpoint in the callback method of the ***AVPacket*** sample code above:

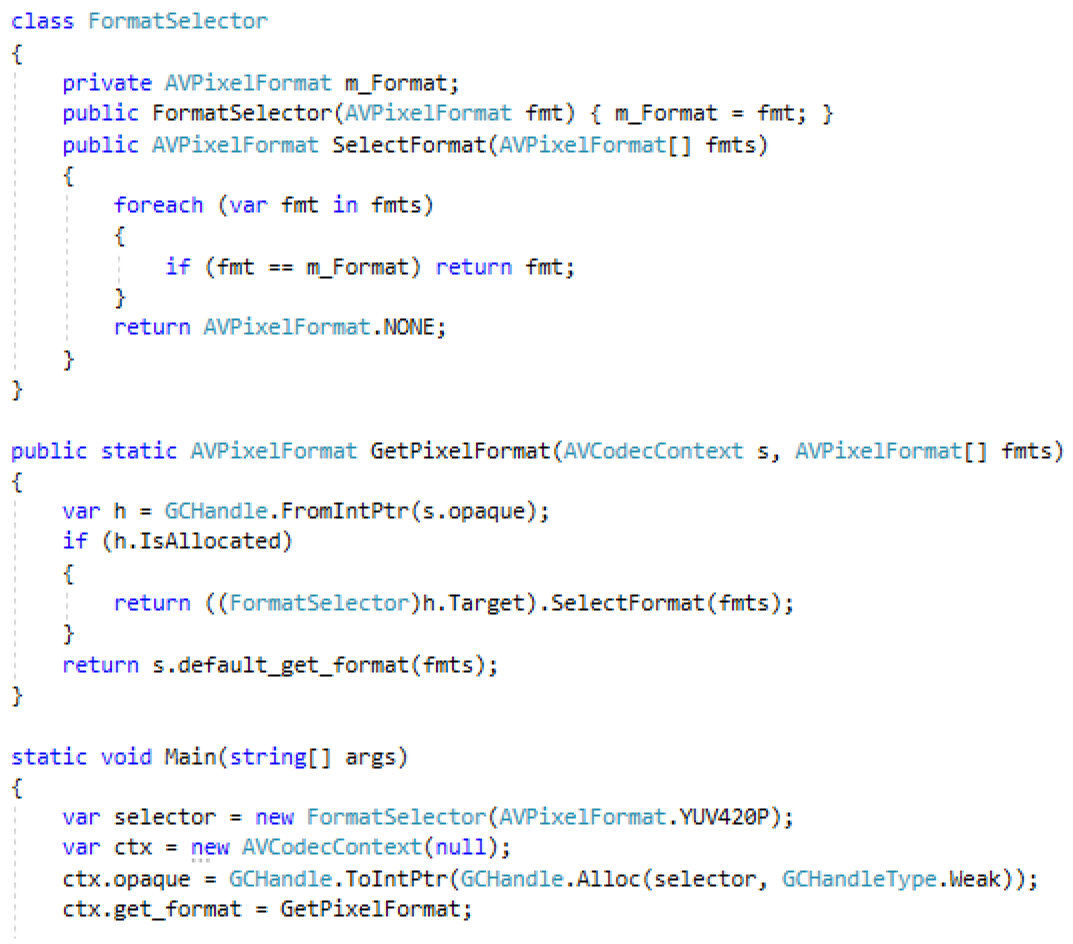


```
void packet_data_free_cb(void *opaque, uint8_t *data)
{
    GCHandle _handle = GCHandle::FromIntPtr((IntPtr)opaque);
    if (_handle.IsAllocated)
    {
        ((FFmpeg::AVPacket^)_handle.Target)->FreeCB((IntPtr)data);
    }                                       _handle.Target {FFmpeg::AVPacket^}
}
////////////////////////////////////////////////////////////
FFmpeg::AVPacket::AVPacket(void * _pointer,AVBase^ _parent)
    : AVBase(_pointer,_parent)
    , m_pOpaque(nullptr)
```

```
Call Stack
Name
ffTest.exe!ffTest.Program.buffer_free(object opaque, System.IntPtr buf) Line 1332
FFmpeg.NET.dll!FFmpeg::AVPacket::FreeCB(System::IntPtr data) Line 2038
FFmpeg.NET.dll!packet_data_free_cb(void* opaque, unsigned char* data) Line 1561
[Native to Managed Transition]
avutil-56.dll!00007ff892ee924f()
avcodec-58.dll!00007ff879f374b9()
[Managed to Native Transition]
FFmpeg.NET.dll!FFmpeg::AVPacket::Free() Line 1782
FFmpeg.NET.dll!FFmpeg::AVPacket::~AVPacket() Line 1660
FFmpeg.NET.dll!FFmpeg::AVPacket::Dispose(bool )
FFmpeg.NET.dll!FFmpeg::AVBase::Dispose()
ffTest.exe!ffTest.Program.Main(string[] args) Line 1340
```

In the call stack it is possible to see that *buffer_free* callback method is called from an internal library function which converts opaque pointer into ***GCHandle*** of the related ***AVPacket*** object. And that object calls the common free handler of that class which as the result made call the passed callback method which was saved as a variable along with the user's *opaque* value.

The callbacks, which are set as property, do not have the same implementation. For example: ***AVCodecContext*** structure class uses an *opaque* object which is set to context property - so same way it is done in native FFmpeg library:



```
class FormatSelector
{
    private AVPixelFormat m_Format;
    public FormatSelector(AVPixelFormat fmt) { m_Format = fmt; }
    public AVPixelFormat SelectFormat(AVPixelFormat[] fmts)
    {
        foreach (var fmt in fmts)
        {
            if (fmt == m_Format) return fmt;
        }
        return AVPixelFormat.NONE;
    }
}

public static AVPixelFormat GetPixelFormat(AVCodecContext s, AVPixelFormat[] fmts)
{
    var h = GCHandle.FromIntPtr(s.opaque);
    if (h.IsAllocated)
    {
        return ((FormatSelector)h.Target).SelectFormat(fmts);
    }
    return s.default_get_format(fmts);
}

static void Main(string[] args)
{
    var selector = new FormatSelector(AVPixelFormat.YUV420P);
    var ctx = new AVCodecContext(null);
    ctx.opaque = GCHandle.ToIntPtr(GCHandle.Alloc(selector, GCHandleType.Weak));
    ctx.get_format = GetPixelFormat;
```

In the code above we demonstrate how to set up a callback for the format selection property of the ***AVCodecContext*** structure. We create a selector class object where we try to select the **YUV420P** pixel format. This class we provide as an *opaque* object for the context as a ***GCHandle*** pointer. The selector object must stay alive until the callback method can be used as we set a **Weak** handle type. We have a *GetPixelFormat* callback method where we use an ***AVCodecContext*** *opaque* object as our format selector class and use its method for performing selection. As we have a **Weak** handle type then the underlying object can be freed and in such case we call the default format selector method of the ***AVCodecContext*** class.

Dynamic API

The wrapper library is designed for linking to the specified .lib files from the FFmpeg libraries. Current implementation links to the FFmpeg API function entries which are exposed with those .lib files. But, as were mentioned, the wrapper library is designed to support different versions of FFmpeg libraries, and on newer versions APIs can be deprecated or added optionally depending on FFmpeg build options. To properly handle building such cases some imported API's are checked dynamically in code. And internal object method implementation can differ depending on what API present in FFmpeg libraries. That implementation is hidden internally, so users just see the regular one method. For example on some version of FFmpeg *av_codec_next* not present and replacement of the functionality done with *av_codec_iterate* API:

```
const AVCodec *av_codec_iterate(void **opaque);

#if FF_API_NEXT
/**
 * If c is NULL, returns the first registered codec,
 * if c is non-NULL, returns the next registered codec after c,
 * or NULL if c is the last one.
 */
attribute_deprecated
AVCodec *av_codec_next(const AVCodec *c);
#endif
```

Also on new FFmpeg versions an *avcodec_register_all* API is also deprecated and not present as an exported API.
To handle such version differences and avoid build errors, access to those API is made dynamically. Here is how the implementation of assessing *avcodec_register_all* API looks:

```
void FFmpeg::LibAVCodec::RegisterAll()
{
    if (!s_bRegistered)
    {
        s_bRegistered = true;
        VOID_API(AVCodec,avcodec_register_all)
        avcodec_register_all();
    }
}
```

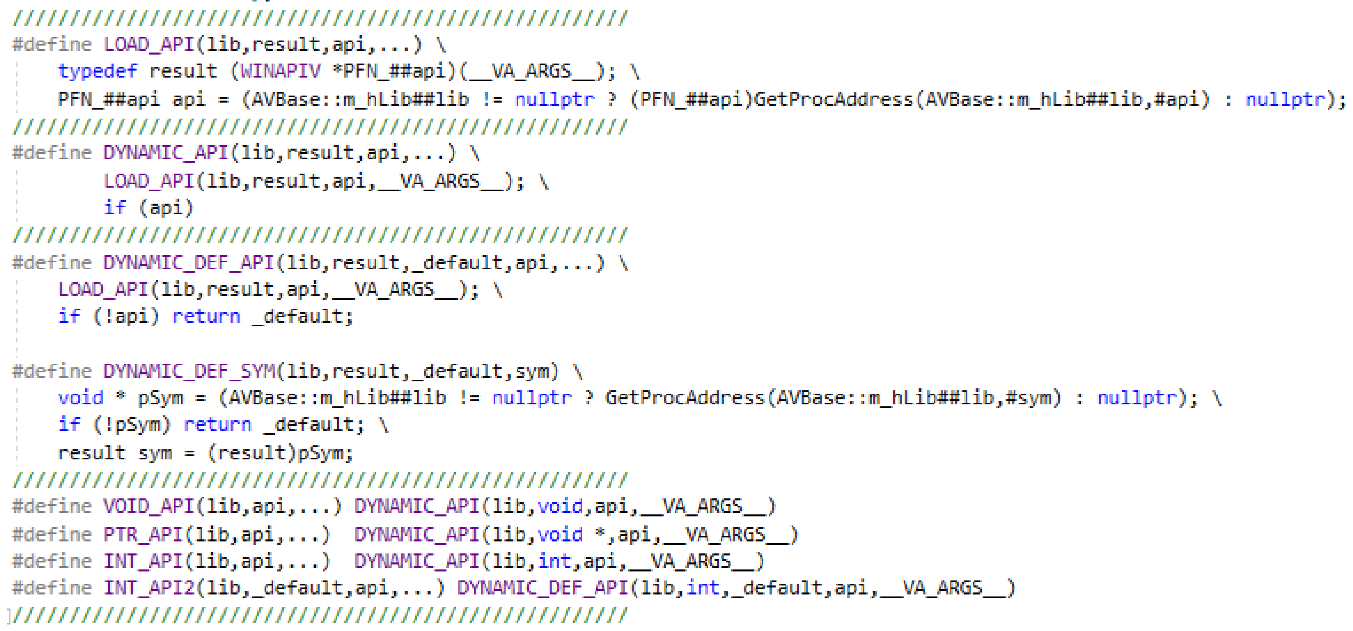
And here is how done the codec iteration:

```
bool FFmpeg::AVCodec::AVCodecs::AVCodecEnumerator::MoveNext()
{
    const ::AVCodec * p = nullptr;
    void * opaque = m_pOpaque.ToPointer();
    LOAD_API(AVCodec,::AVCodec *,av_codec_next,const ::AVCodec*);
    LOAD_API(AVCodec,::AVCodec *,av_codec_iterate,void **);
    if (av_codec_iterate)
    {
        p = av_codec_iterate(&opaque);
    }
    else
    {
        if (av_codec_next)
        {
            p = av_codec_next((const ::AVCodec*)opaque);
            opaque = (void*)p;
        }
    }
    m_pOpaque = IntPtr(opaque);
    m_pCurrent = (p != nullptr) ? gcnew AVCodec((void*)p, nullptr) : nullptr;
    return (m_pCurrent != nullptr);
}
```

In the code above there are some helper macros: **VOID_API** and **LOAD_API** which load API from specified DLL modules and if one or another API present - uses it, or in other case skip API call. The dynamic DLLs are static parts of ***AVBase*** class and loaded in its constructor call for the first time.

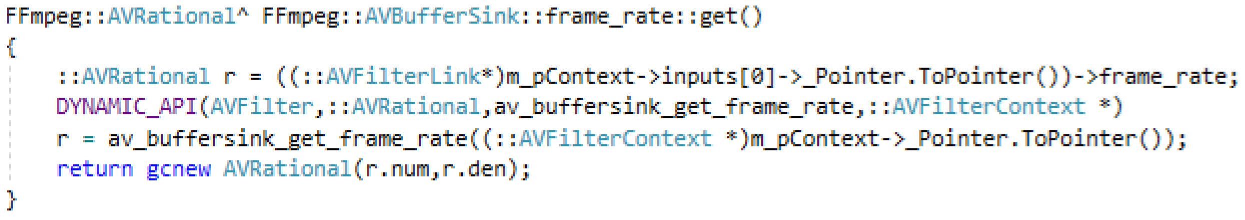
```
internal:
    static bool s_bDllLoaded = false;
    static HMODULE m_hLibAVUtil = nullptr;
    static HMODULE m_hLibAVCodec = nullptr;
    static HMODULE m_hLibAVFormat = nullptr;
    static HMODULE m_hLibAVFilter = nullptr;
    static HMODULE m_hLibAVDevice = nullptr;
    static HMODULE m_hLibPostproc = nullptr;
    static HMODULE m_hLibSwscale = nullptr;
    static HMODULE m_hLibSwresample = nullptr;
```

The helper macro defined in AVCore.h file:



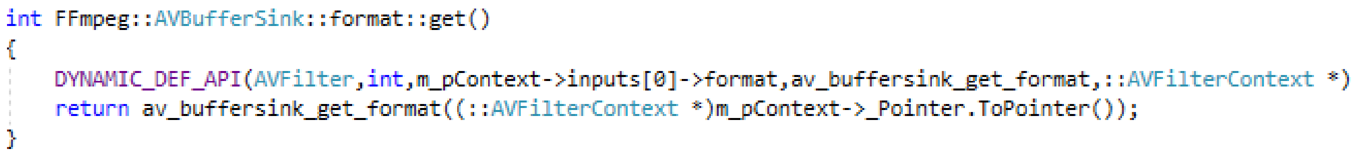
```
//////////////////////////////////////////////////////
#define LOAD_API(lib,result,api,...) \
    typedef result (WINAPIV *PFN_##api)(__VA_ARGS__); \
    PFN_##api api = (AVBase::m_hLib##lib != nullptr ? (PFN_##api)GetProcAddress(AVBase::m_hLib##lib,#api) : nullptr);
//////////////////////////////////////////////////////
#define DYNAMIC_API(lib,result,api,...) \
        LOAD_API(lib,result,api,__VA_ARGS__); \
        if (api)
//////////////////////////////////////////////////////
#define DYNAMIC_DEF_API(lib,result,_default,api,...) \
    LOAD_API(lib,result,api,__VA_ARGS__); \
    if (!api) return _default;

#define DYNAMIC_DEF_SYM(lib,result,_default,sym) \
    void * pSym = (AVBase::m_hLib##lib != nullptr ? GetProcAddress(AVBase::m_hLib##lib,#sym) : nullptr); \
    if (!pSym) return _default; \
    result sym = (result)pSym;
//////////////////////////////////////////////////////
#define VOID_API(lib,api,...) DYNAMIC_API(lib,void,api,__VA_ARGS__)
#define PTR_API(lib,api,...)  DYNAMIC_API(lib,void *,api,__VA_ARGS__)
#define INT_API(lib,api,...)  DYNAMIC_API(lib,int,api,__VA_ARGS__)
#define INT_API2(lib,_default,api,...) DYNAMIC_DEF_API(lib,int,_default,api,__VA_ARGS__)
//////////////////////////////////////////////////////
```

LOAD_API macro defines the API variable and loads it from the specified FFmpeg library. **DYNAMIC_API** - loads an API and calls the next code line if the API persists. This is good if required to switch between dynamic or static API linkage, or use different way access.



```
FFmpeg::AVRational^ FFmpeg::AVBufferSink::frame_rate::get()
{
    ::AVRational r = ((::AVFilterLink*)m_pContext->inputs[0]->_Pointer.ToPointer())->frame_rate;
    DYNAMIC_API(AVFilter,::AVRational,av_buffersink_get_frame_rate,::AVFilterContext *)
    r = av_buffersink_get_frame_rate((::AVFilterContext *)m_pContext->_Pointer.ToPointer());
    return gcnew AVRational(r.num,r.den);
}
```

DYNAMIC_DEF_API - loads API and if it is not able to be loaded return specified default value.

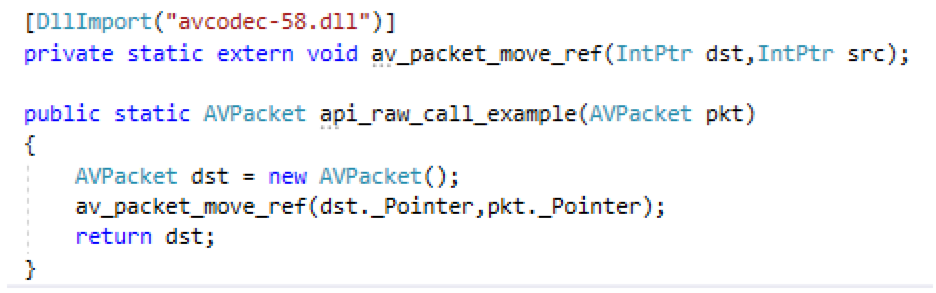


```
int FFmpeg::AVBufferSink::format::get()
{
    DYNAMIC_DEF_API(AVFilter,int,m_pContext->inputs[0]->format,av_buffersink_get_format,::AVFilterContext *)
    return av_buffersink_get_format((::AVFilterContext *)m_pContext->_Pointer.ToPointer());
}
```

Other macros are just variations of return types.
In case if your exported API is used as the static method, or in class which is not subclass of ***AVBase*** then make sure that before using those macro ***AVBase***::*EnsureLibraryLoaded()*; method is called, or just check ***AVBase***::*s_bDllLoaded* variable.

Structure pointers

As mentioned earlier: each ***AVBase*** class exposes a pointer to underlying FFmpeg structure. This is done for ability to extend library functionality or manually managing existing API. As an example ***AVBase***._*Pointer* field can be used for the direct object cast or for manual usage in exported API:



```
[DllImport("avcodec-58.dll")]
private static extern void av_packet_move_ref(IntPtr dst,IntPtr src);

public static AVPacket api_raw_call_example(AVPacket pkt)
{
    AVPacket dst = new AVPacket();
    av_packet_move_ref(dst._Pointer,pkt._Pointer);
    return dst;
}
```

Note: the object methods can internally manage destructor and free structure API's, so such usage of raw API has risk of memory leaks.

Raw pointers also can be used to access any structure fields directly.

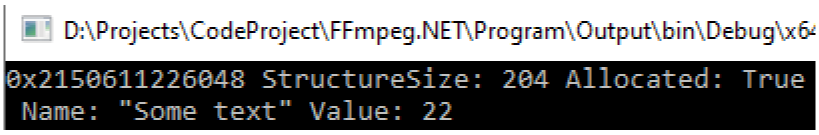
Extending the library

Just to show the benefits of implementing an entire wrapper library in C++/CLI this topic was added. It is possible to implement different parts of the code in C# or other .NET languages directly by accessing the existing library objects properties. It is also possible to extend existing functionality by adding support for any API, which may be missed, or for any other needs. As already specified, those API can be used with the pointers to structures, and to handle such structures is used the ***AVBase*** class. To wrap your own structure it is just necessary to inherit such a structure from the ***AVBase*** class and manage its properties. Here is a simple example of such implementation:

```
public class MyAVStruct : AVBase
{
    [StructLayout(LayoutKind.Sequential,CharSet=CharSet.Ansi)]
    private struct S
    {
        [MarshalAs(UnmanagedType.I4)]
        public int Value;
        [MarshalAs(UnmanagedType.ByValTStr,SizeConst=200)]
        public string Name;
    }

    public MyAVStruct()
    {
        if (!base._EnsurePointer(false))
        {
            base.AllocPointer(_StructureSize);
        }
    }
    public override int _StructureSize { get { return Marshal.SizeOf(typeof(S)); } }

    public int Value
    {
        get { return Marshal.PtrToStructure<S>(base._Pointer).Value; }
        set {
            if (base._EnsurePointer())
            {
                var s = Marshal.PtrToStructure<S>(base._Pointer);
                s.Value = value;
                Marshal.StructureToPtr<S>(s, base._Pointer, false);
            }
        }
    }

    public string Name
    {
        get { return Marshal.PtrToStructure<S>(base._Pointer).Name; }
        set { var s = Marshal.PtrToStructure<S>(base._Pointer);
            s.Name = value;
            Marshal.StructureToPtr<S>(s, base._Pointer, false);
        }
    }
}
```

We have a structure named “*S*” for which we make a wrap object “*MyAVStruct*” for the ability to access it from .NET. We allocate a data pointer by calling the *AllocPointer* method of the ***AVBase*** class. Also we have access to each field of that structure by marshaling the whole structure each time we get or set property value. An example of the call that structure:

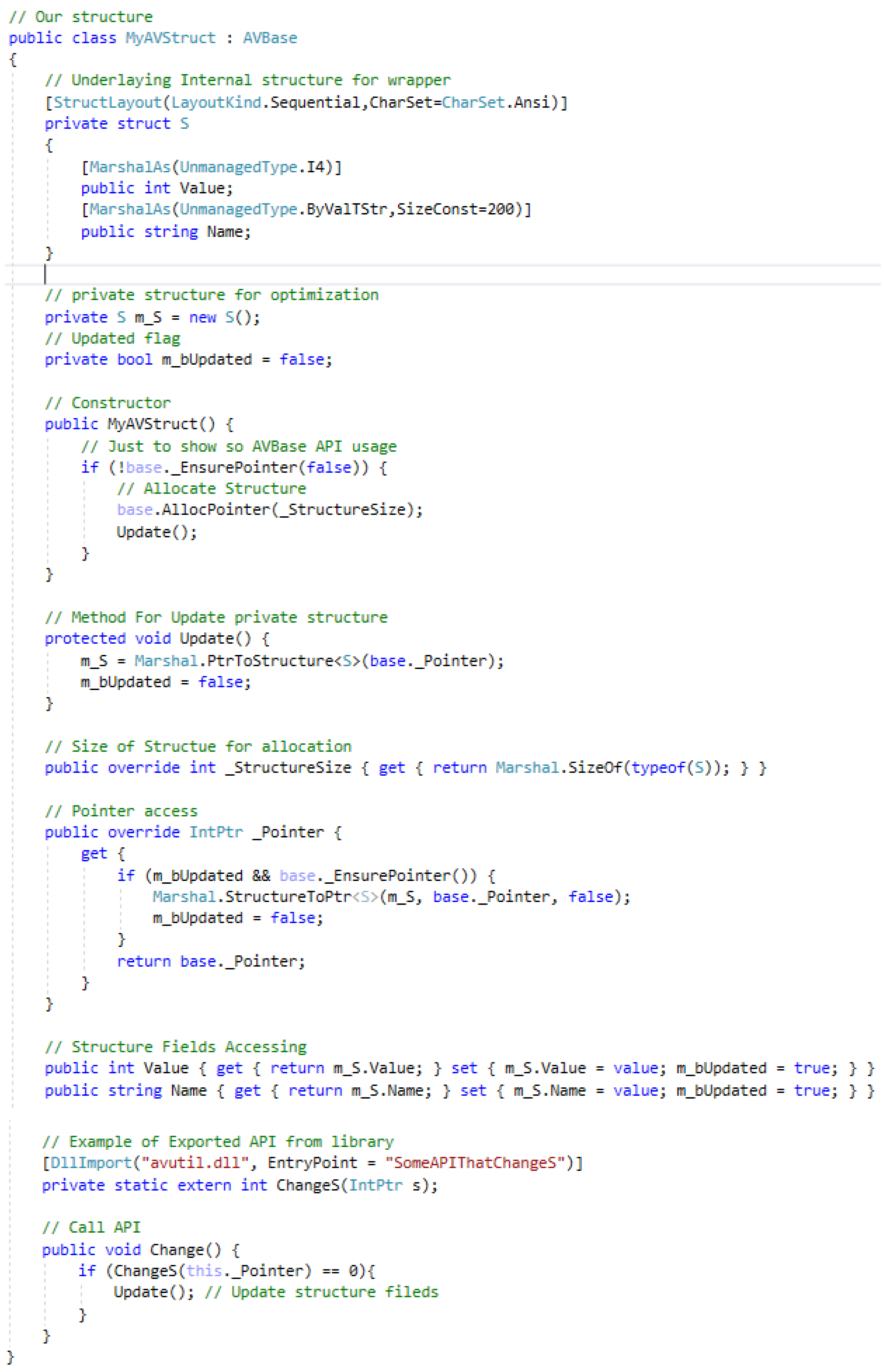
```
// Create Structure instance
var s = new MyAVStruct();
if (s._IsValid) // Check if the pointer allocated
{
    // Set field values
    s.Value = 22;
    s.Name = "Some text";
    // Get values
    Console.WriteLine("0x{0:x} StructureSize: {1} Allocated: {2}\n Name: \"{3}\" Value: {4}",
    s._Pointer, s._StructureSize, s._IsAllocated,
    s.Name,s.Value);
}
// Destroy structure and free data
s.Dispose();
```



D:\Projects\CodeProject\FFmpeg.NET\Program\Output\bin\Debug\x6

```
0x2150611226048 StructureSize: 204 Allocated: True
 Name: "Some text" Value: 22
```

Of course the structure implementation in code above is not optimized. It's just for displaying functionality. Keep in mind that those operations for marshaling structure each time are not performing either in case of C++/CLI implementation as those fields are accessed directly internally and this is a big plus. In .NET case we can optimize the above structure:



```csharp
// Our structure
public class MyAVStruct : AVBase
{
    // Underlaying Internal structure for wrapper
    [StructLayout(LayoutKind.Sequential,CharSet=CharSet.Ansi)]
    private struct S
    {
        [MarshalAs(UnmanagedType.I4)]
        public int Value;
        [MarshalAs(UnmanagedType.ByValTStr,SizeConst=200)]
        public string Name;
    }

    // private structure for optimization
    private S m_S = new S();
    // Updated flag
    private bool m_bUpdated = false;

    // Constructor
    public MyAVStruct() {
        // Just to show so AVBase API usage
        if (!base._EnsurePointer(false)) {
            // Allocate Structure
            base.AllocPointer(_StructureSize);
            Update();
        }
    }

    // Method For Update private structure
    protected void Update() {
        m_S = Marshal.PtrToStructure<S>(base._Pointer);
        m_bUpdated = false;
    }

    // Size of Structue for allocation
    public override int _StructureSize { get { return Marshal.SizeOf(typeof(S)); } }

    // Pointer access
    public override IntPtr _Pointer {
        get {
            if (m_bUpdated && base._EnsurePointer()) {
                Marshal.StructureToPtr<S>(m_S, base._Pointer, false);
                m_bUpdated = false;
            }
            return base._Pointer;
        }
    }

    // Structure Fields Accessing
    public int Value { get { return m_S.Value; } set { m_S.Value = value; m_bUpdated = true; } }
    public string Name { get { return m_S.Name; } set { m_S.Name = value; m_bUpdated = true; } }

    // Example of Exported API from library
    [DllImport("avutil.dll", EntryPoint = "SomeAPIThatChangeS")]
    private static extern int ChangeS(IntPtr s);

    // Call API
    public void Change() {
        if (ChangeS(this._Pointer) == 0){
            Update(); // Update structure fileds
        }
    }
}
```

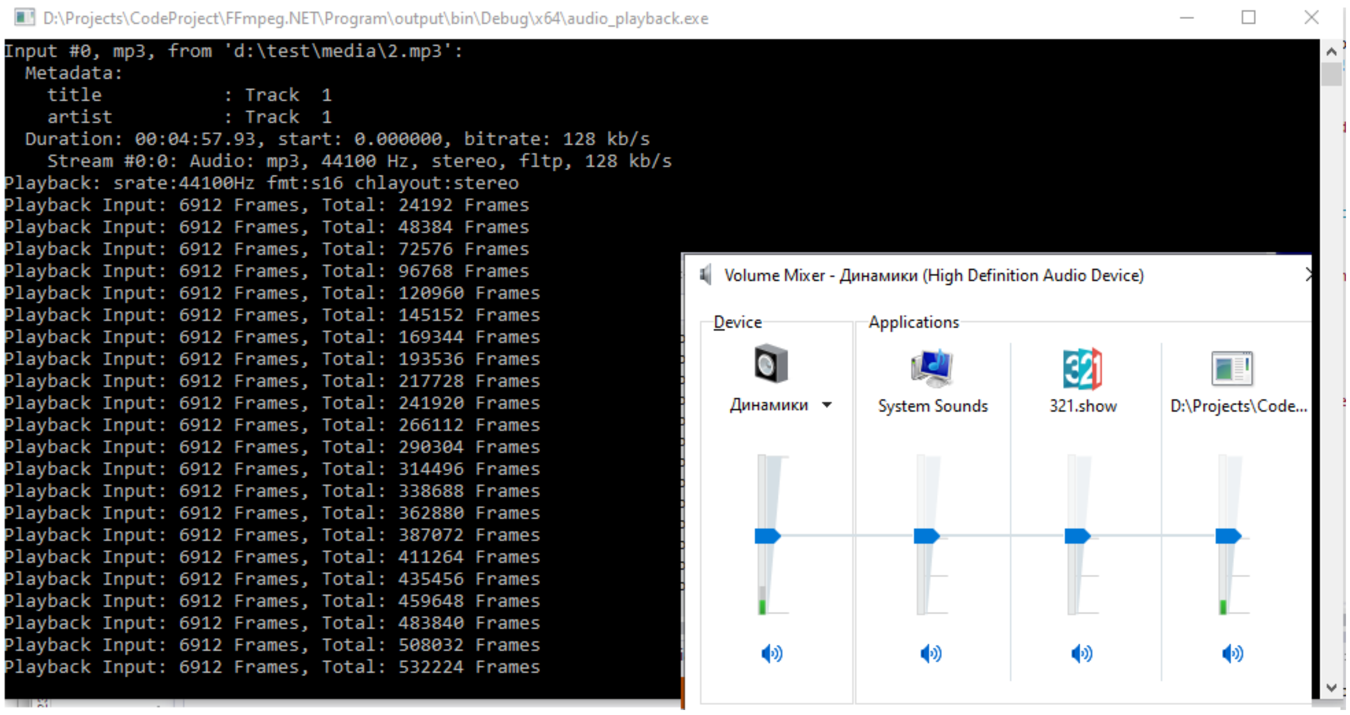
In the modified structure we have the flag: "*m_bUpdate*" which controls the field updates. If any value of the field is updated then the flag is set, and when we need to access raw pointer - structure mashaling is performed. Along with it there is an example of some exported API calls with the structure. After calling the temporal internal structure updated with values from the actual pointer.
This is a simple example of implementation as you can see in the FFmpeg libraries structures are not simple and not possible to manage all them this way, or at least not all fields, it is also hard in the case of marshaling each field directly from a pointer and casting then into one or another objects. But that was not hard to handle with C++/CLI, that's why it was selected for implementation.

Examples

I made a couple of C# examples to show how to use the wrapper library. Some of them are wrappers of examples which come with native FFmpeg documentation, so you can easily compare the implementation. I also add a few more my own examples, which I think will be interesting. All samples code is located in the "*Sources\Examples*" folder. The project files can be loaded from the "*Sources\Examples\proj*" folder, or just the library solution file can be opened to access all examples. All C# samples which are wrappers of standard FFmpeg examples are working the same way as native. Some sample descriptions contain screenshots of the execution.

Audio_playback

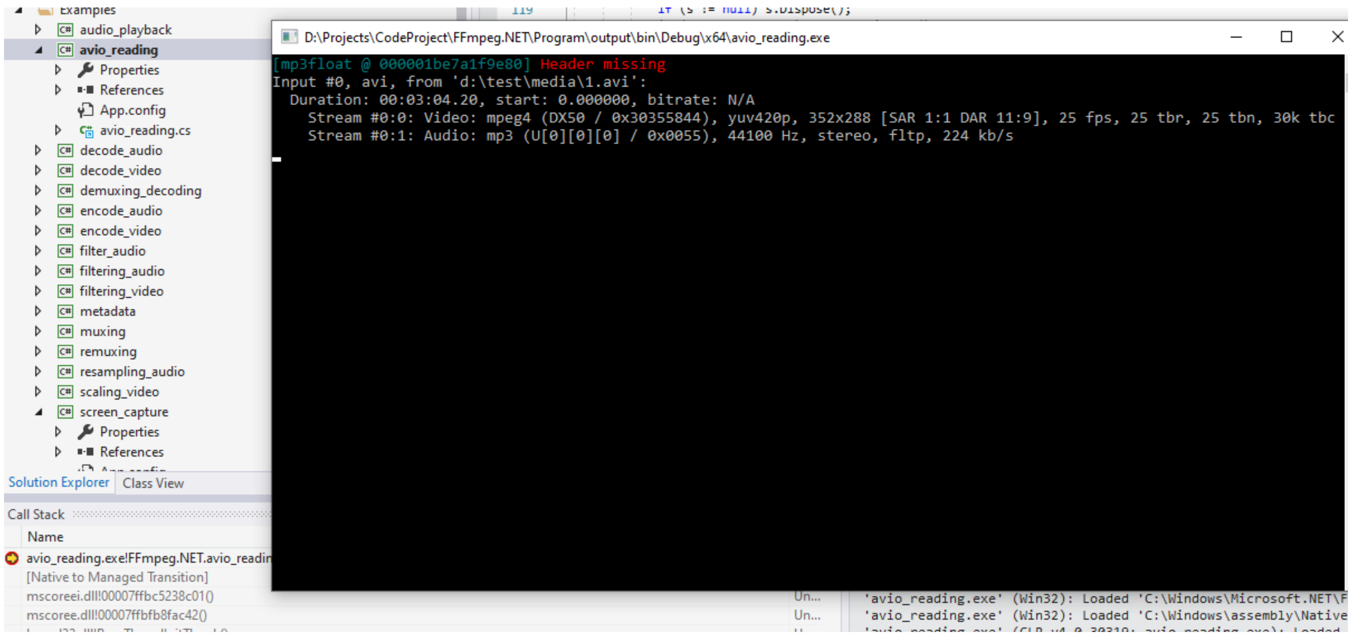
Example of loading media file with *libavformat* decoding and resampling audio data and output it to playback with WinMM Windows API.


D:\Projects\CodeProject\FFmpeg.NET\Program\output\bin\Debug\x64\audio_playback.exe
Volume Mixer - Динамики (High Definition Audio Device)
Device
Applications
Динамики
System Sounds
321.show
D:\Projects\Code...

```
Input #0, mp3, from 'd:\test\media\2.mp3':
  Metadata:
    title           : Track  1
    artist          : Track  1
  Duration: 00:04:57.93, start: 0.000000, bitrate: 128 kb/s
    Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 128 kb/s
Playback: srate:44100Hz fmt:s16 chlayout:stereo
Playback Input: 6912 Frames, Total: 24192 Frames
Playback Input: 6912 Frames, Total: 48384 Frames
Playback Input: 6912 Frames, Total: 72576 Frames
Playback Input: 6912 Frames, Total: 96768 Frames
Playback Input: 6912 Frames, Total: 120960 Frames
Playback Input: 6912 Frames, Total: 145152 Frames
Playback Input: 6912 Frames, Total: 169344 Frames
Playback Input: 6912 Frames, Total: 193536 Frames
Playback Input: 6912 Frames, Total: 217728 Frames
Playback Input: 6912 Frames, Total: 241920 Frames
Playback Input: 6912 Frames, Total: 266112 Frames
Playback Input: 6912 Frames, Total: 290304 Frames
Playback Input: 6912 Frames, Total: 314496 Frames
Playback Input: 6912 Frames, Total: 338688 Frames
Playback Input: 6912 Frames, Total: 362880 Frames
Playback Input: 6912 Frames, Total: 387072 Frames
Playback Input: 6912 Frames, Total: 411264 Frames
Playback Input: 6912 Frames, Total: 435456 Frames
Playback Input: 6912 Frames, Total: 459648 Frames
Playback Input: 6912 Frames, Total: 483840 Frames
Playback Input: 6912 Frames, Total: 508032 Frames
Playback Input: 6912 Frames, Total: 532224 Frames
```

Avio_reading

Standard FFmpeg *libavformat* ***AVIOContext*** API example. Make *libavformat* demuxer access media content through a custom ***AVIOContext*** read callback.



```
[mp3float @ 000001be7a1f9e80] Header missing
Input #0, avi, from 'd:\test\media\1.avi':
  Duration: 00:03:04.20, start: 0.000000, bitrate: N/A
    Stream #0:0: Video: mpeg4 (DX50 / 0x30355844), yuv420p, 352x288 [SAR 1:1 DAR 11:9], 25 fps, 25 tbr, 25 tbn, 30k tbc
    Stream #0:1: Audio: mp3 (U[0][0][0] / 0x0055), 44100 Hz, stereo, fltp, 224 kb/s
```

```
avio_reading.exe!FFmpeg.NET.avio_readir
[Native to Managed Transition]
mscoreei.dll!00007ffbc5238c01()
mscoree.dll!00007ffbfb8fac42()
```

```
'avio_reading.exe' (Win32): Loaded 'C:\Windows\Microsoft.NET\F
'avio_reading.exe' (Win32): Loaded 'C:\Windows\assembly\Native
```

Decode_audio

A C# wrapper of standard FFmpeg example of audio decoding with *libavcodec* API.

Decode_video

A C# wrapper of standard FFmpeg example of video decoding with *libavcodec* API.

Demuxing_decoding

A C# wrapper of standard FFmpeg demuxing and decoding example. Show how to use the *libavformat* and *libavcodec* API to demux and decode audio and video data.

Encode_audio

A C# wrapper of standard FFmpeg audio encoding with *libavcodec* API example.

Encode_video

A C# wrapper of standard FFmpeg video encoding with *libavcodec* API example.

Filter_audio

A C# wrapper of standard FFmpeg *libavfilter* API usage example. This example will generate a sine wave audio, pass it through a simple filter chain, and then compute the MD5 checksum of the output data.

Filtering_audio

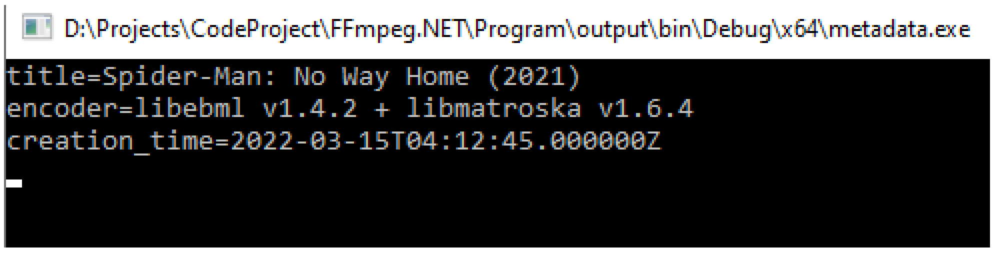
A C# wrapper of standard FFmpeg API example for audio decoding and filtering.

Filtering_video

A C# wrapper of standard FFmpeg API example for decoding and filtering video.

Metadata

A C# wrapper of standard FFmpeg example. Shows how the metadata API can be used in application programs.

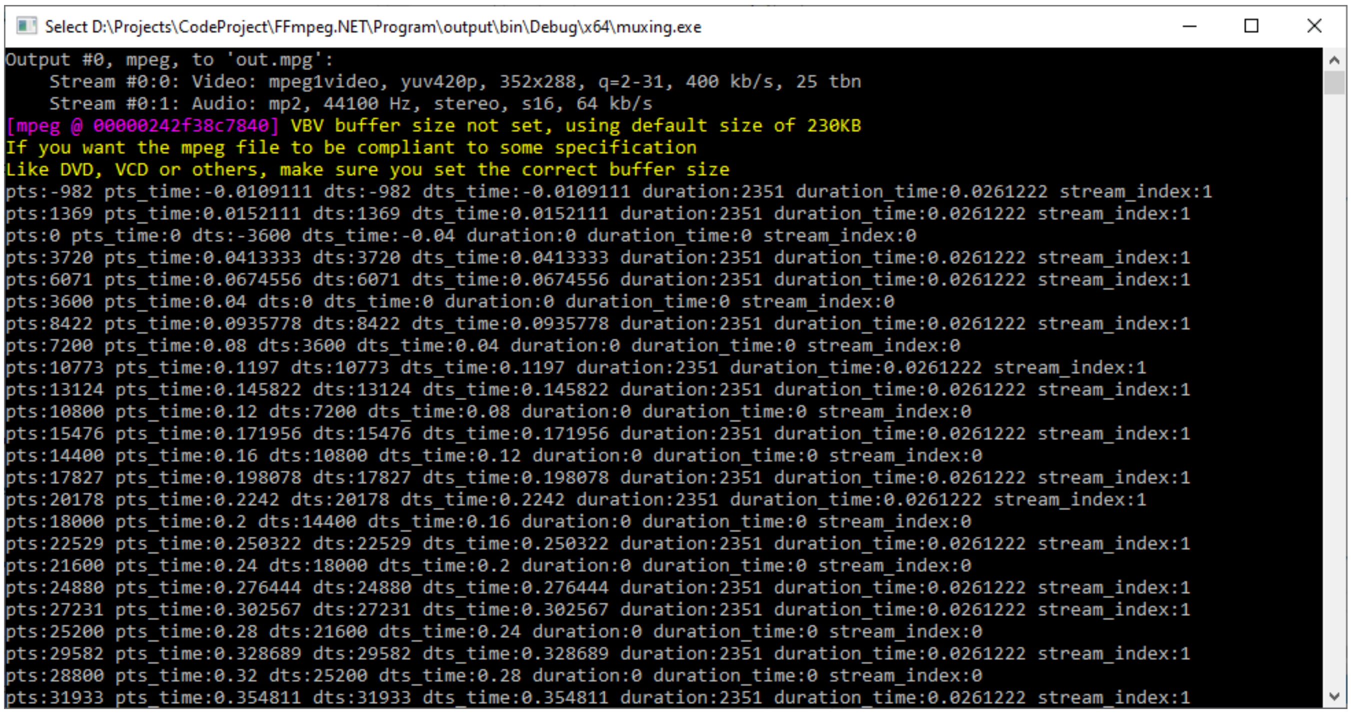


D:\Projects\CodeProject\FFmpeg.NET\Program\output\bin\Debug\x64\metadata.exe

```
title=Spider-Man: No Way Home (2021)
encoder=libebml v1.4.2 + libmatroska v1.6.4
creation_time=2022-03-15T04:12:45.000000Z
```

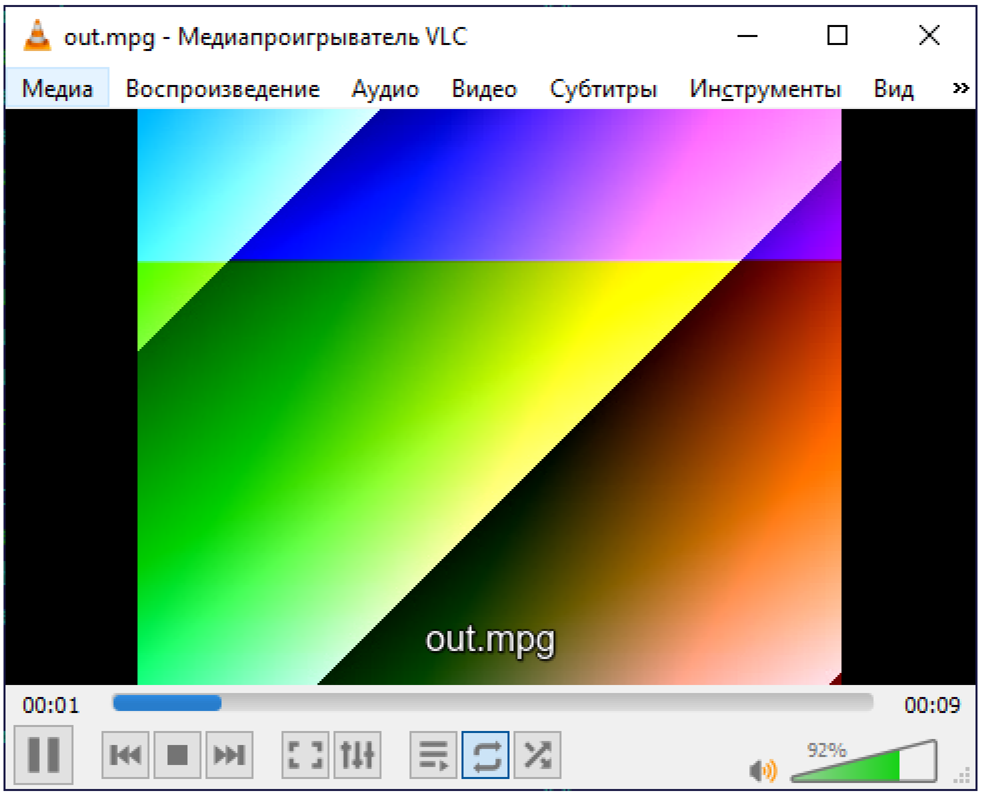
Muxing

A C# wrapper of standard FFmpeg *libavformat* API example. Output a media file in any supported *libavformat* format.



Select D:\Projects\CodeProject\FFmpeg.NET\Program\output\bin\Debug\x64\muxing.exe

```
Output #0, mpeg, to 'out.mpg':
    Stream #0:0: Video: mpeg1video, yuv420p, 352x288, q=2-31, 400 kb/s, 25 tbn
    Stream #0:1: Audio: mp2, 44100 Hz, stereo, s16, 64 kb/s
[mpeg @ 00000242f38c7840] VBV buffer size not set, using default size of 230KB
If you want the mpeg file to be compliant to some specification
Like DVD, VCD or others, make sure you set the correct buffer size
pts:-982 pts_time:-0.0109111 dts:-982 dts_time:-0.0109111 duration:2351 duration_time:0.0261222 stream_index:1
pts:1369 pts_time:0.0152111 dts:1369 dts_time:0.0152111 duration:2351 duration_time:0.0261222 stream_index:1
pts:0 pts_time:0 dts:-3600 dts_time:-0.04 duration:0 duration_time:0 stream_index:0
pts:3720 pts_time:0.0413333 dts:3720 dts_time:0.0413333 duration:2351 duration_time:0.0261222 stream_index:1
pts:6071 pts_time:0.0674556 dts:6071 dts_time:0.0674556 duration:2351 duration_time:0.0261222 stream_index:1
pts:3600 pts_time:0.04 dts:0 dts_time:0 duration:0 duration_time:0 stream_index:0
pts:8422 pts_time:0.0935778 dts:8422 dts_time:0.0935778 duration:2351 duration_time:0.0261222 stream_index:1
pts:7200 pts_time:0.08 dts:3600 dts_time:0.04 duration:0 duration_time:0 stream_index:0
pts:10773 pts_time:0.1197 dts:10773 dts_time:0.1197 duration:2351 duration_time:0.0261222 stream_index:1
pts:13124 pts_time:0.145822 dts:13124 dts_time:0.145822 duration:2351 duration_time:0.0261222 stream_index:1
pts:10800 pts_time:0.12 dts:7200 dts_time:0.08 duration:0 duration_time:0 stream_index:0
pts:15476 pts_time:0.171956 dts:15476 dts_time:0.171956 duration:2351 duration_time:0.0261222 stream_index:1
pts:14400 pts_time:0.16 dts:10800 dts_time:0.12 duration:0 duration_time:0 stream_index:0
pts:17827 pts_time:0.198078 dts:17827 dts_time:0.198078 duration:2351 duration_time:0.0261222 stream_index:1
pts:20178 pts_time:0.2242 dts:20178 dts_time:0.2242 duration:2351 duration_time:0.0261222 stream_index:1
pts:18000 pts_time:0.2 dts:14400 dts_time:0.16 duration:0 duration_time:0 stream_index:0
pts:22529 pts_time:0.250322 dts:22529 dts_time:0.250322 duration:2351 duration_time:0.0261222 stream_index:1
pts:21600 pts_time:0.24 dts:18000 dts_time:0.2 duration:0 duration_time:0 stream_index:0
pts:24880 pts_time:0.276444 dts:24880 dts_time:0.276444 duration:2351 duration_time:0.0261222 stream_index:1
pts:27231 pts_time:0.302567 dts:27231 dts_time:0.302567 duration:2351 duration_time:0.0261222 stream_index:1
pts:25200 pts_time:0.28 dts:21600 dts_time:0.24 duration:0 duration_time:0 stream_index:0
pts:29582 pts_time:0.328689 dts:29582 dts_time:0.328689 duration:2351 duration_time:0.0261222 stream_index:1
pts:28800 pts_time:0.32 dts:25200 dts_time:0.28 duration:0 duration_time:0 stream_index:0
pts:31933 pts_time:0.354811 dts:31933 dts_time:0.354811 duration:2351 duration_time:0.0261222 stream_index:1
```



Remuxing

A C# wrapper of standard FFmpeg *libavformat/libavcodec* demuxing and muxing API example. Remux streams from one container format to another.

Resampling_audio

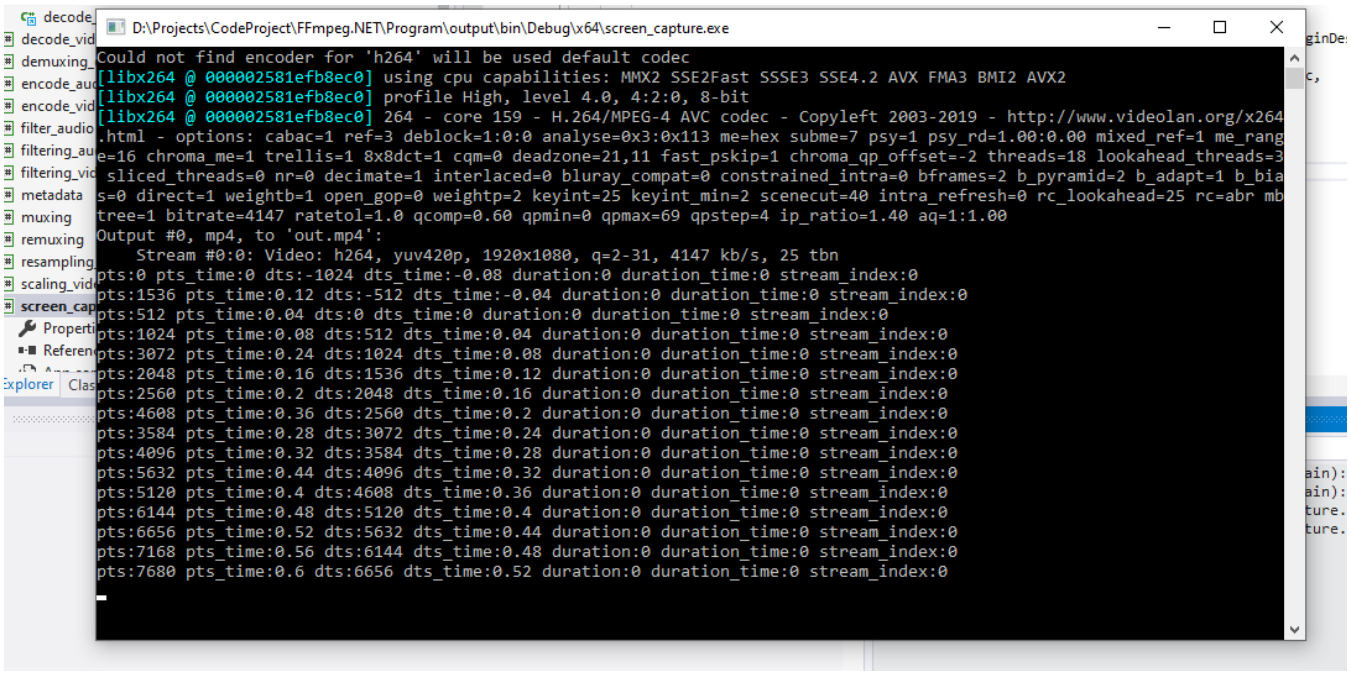
A C# wrapper of standard FFmpeg *libswresample* API use example. Program shows how to resample an audio stream with *libswresample*. It generates a series of audio frames, resamples them to a specified output format and rate and saves them to an output file.

Scaling_video

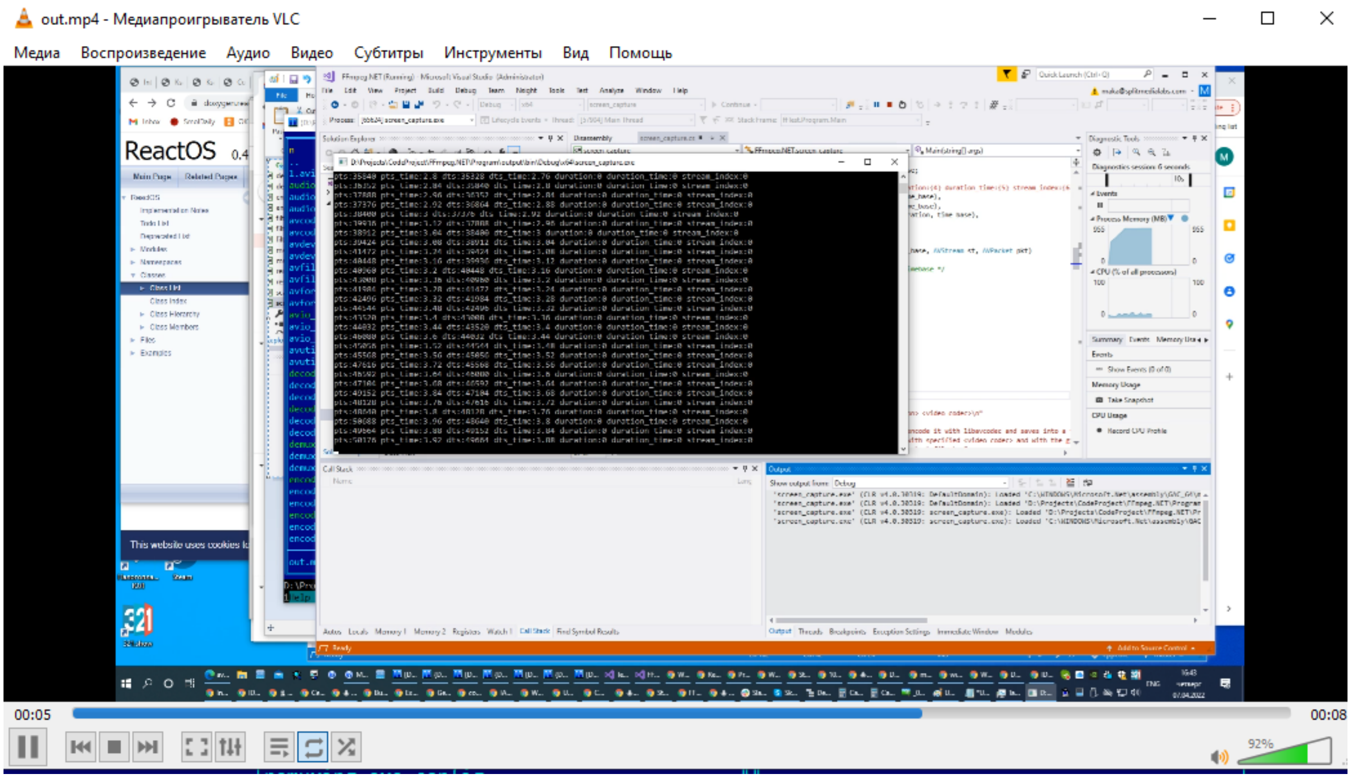
A C# wrapper of standard FFmpeg *libswscale* API use example. Program shows how to scale an image with *libswscale*.It generates a series of pictures, rescales them to the given size and saves them to an output file.

Screen_capture

Capture Screen with Windows GDI scale with *libswscale*, encode with *libavcodec* and save into a file with *libavformat*.


D:\Projects\CodeProject\FFmpeg.NET\Program\output\bin\Debug\x64\screen_capture.exe

```
Could not find encoder for 'h264' will be used default codec
[libx264 @ 000002581efb8ec0] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 000002581efb8ec0] profile High, level 4.0, 4:2:0, 8-bit
[libx264 @ 000002581efb8ec0] 264 - core 159 - H.264/MPEG-4 AVC codec - Copyleft 2003-2019 - http://www.videolan.org/x264
.html - options: cabac=1 ref=3 deblock=1:0:0 analyse=0x3:0x113 me=hex subme=7 psy=1 psy_rd=1.00:0.00 mixed_ref=1 me_rang
e=16 chroma_me=1 trellis=1 8x8dct=1 cqm=0 deadzone=21,11 fast_pskip=1 chroma_qp_offset=-2 threads=18 lookahead_threads=3
 sliced_threads=0 nr=0 decimate=1 interlaced=0 bluray_compat=0 constrained_intra=0 bframes=2 b_pyramid=2 b_adapt=1 b_bia
s=0 direct=1 weightb=1 open_gop=0 weightp=2 keyint=25 keyint_min=2 scenecut=40 intra_refresh=0 rc_lookahead=25 rc=abr mb
tree=1 bitrate=4147 ratetol=1.0 qcomp=0.60 qpmin=0 qpmax=69 qpstep=4 ip_ratio=1.40 aq=1:1.00
Output #0, mp4, to 'out.mp4':
    Stream #0:0: Video: h264, yuv420p, 1920x1080, q=2-31, 4147 kb/s, 25 tbn
pts:0 pts_time:0 dts:-1024 dts_time:-0.08 duration:0 duration_time:0 stream_index:0
pts:1536 pts_time:0.12 dts:-512 dts_time:-0.04 duration:0 duration_time:0 stream_index:0
pts:512 pts_time:0.04 dts:0 dts_time:0 duration:0 duration_time:0 stream_index:0
pts:1024 pts_time:0.08 dts:512 dts_time:0.04 duration:0 duration_time:0 stream_index:0
pts:3072 pts_time:0.24 dts:1024 dts_time:0.08 duration:0 duration_time:0 stream_index:0
pts:2048 pts_time:0.16 dts:1536 dts_time:0.12 duration:0 duration_time:0 stream_index:0
pts:2560 pts_time:0.2 dts:2048 dts_time:0.16 duration:0 duration_time:0 stream_index:0
pts:4608 pts_time:0.36 dts:2560 dts_time:0.2 duration:0 duration_time:0 stream_index:0
pts:3584 pts_time:0.28 dts:3072 dts_time:0.24 duration:0 duration_time:0 stream_index:0
pts:4096 pts_time:0.32 dts:3584 dts_time:0.28 duration:0 duration_time:0 stream_index:0
pts:5632 pts_time:0.44 dts:4096 dts_time:0.32 duration:0 duration_time:0 stream_index:0
pts:5120 pts_time:0.4 dts:4608 dts_time:0.36 duration:0 duration_time:0 stream_index:0
pts:6144 pts_time:0.48 dts:5120 dts_time:0.4 duration:0 duration_time:0 stream_index:0
pts:6656 pts_time:0.52 dts:5632 dts_time:0.44 duration:0 duration_time:0 stream_index:0
pts:7168 pts_time:0.56 dts:6144 dts_time:0.48 duration:0 duration_time:0 stream_index:0
pts:7680 pts_time:0.6 dts:6656 dts_time:0.52 duration:0 duration_time:0 stream_index:0
```

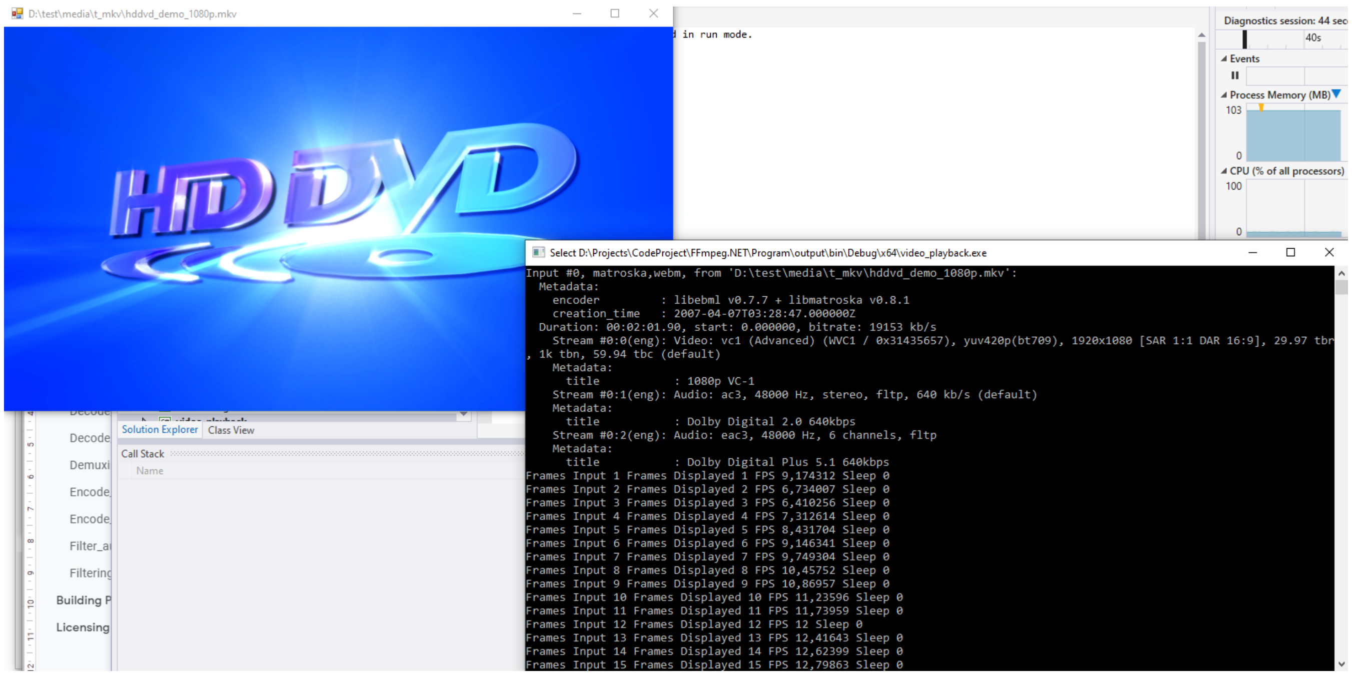


Transcoding

A C# wrapper of standard FFmpeg API example for demuxing, decoding, filtering, encoding and muxing.

Video_playback

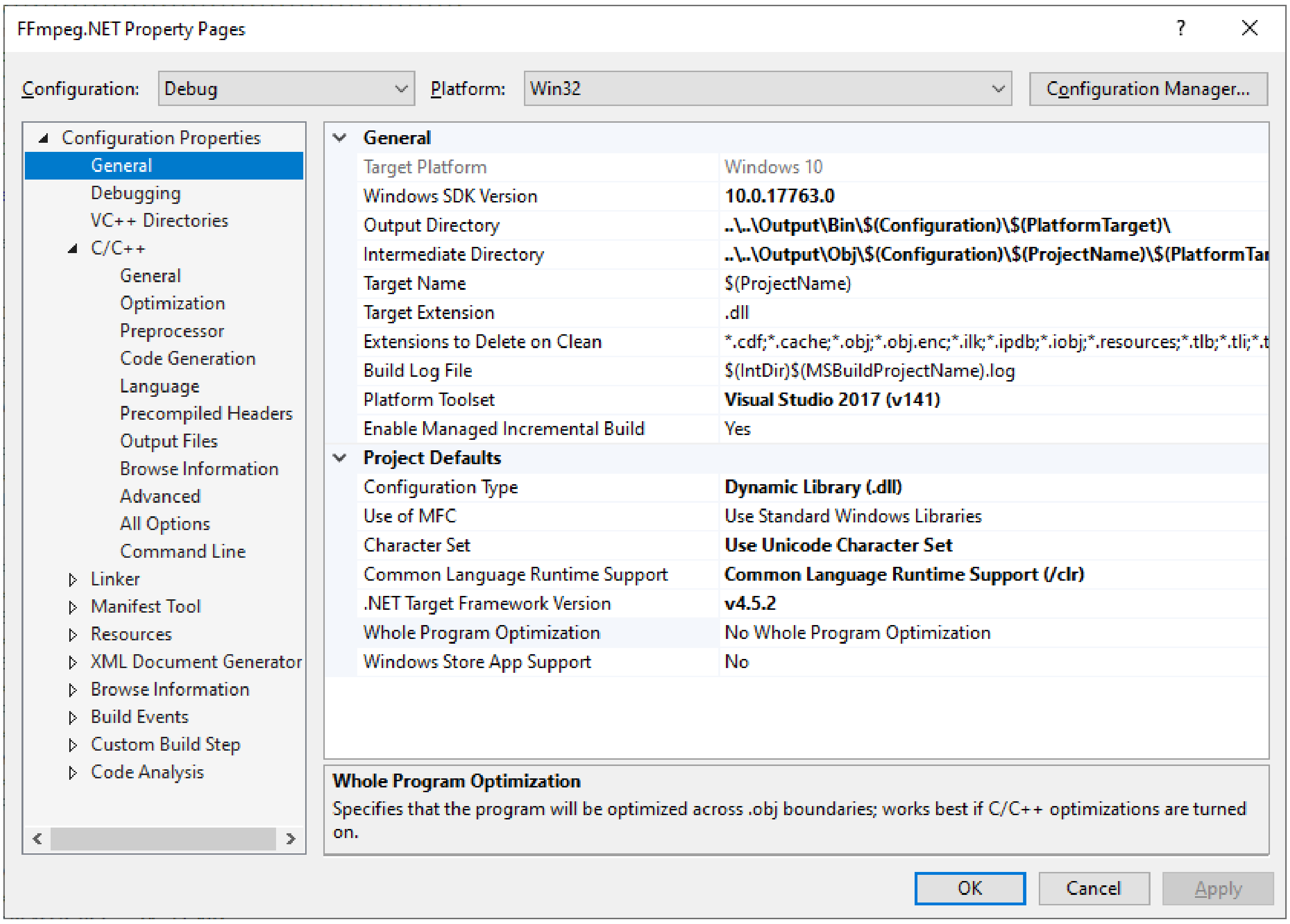
Example of loading a media file with libavformat decoding and scaling video frames and output it to playback with Winforms and GDI+ API. This is just an example, I do not suggest using the same way for video displaying. For video output it is better to use playback with GPU output not GDI+, as there is lots of CPU overhead.


Select D:\Projects\CodeProject\FFmpeg.NET\Program\output\bin\Debug\x64\video_playback.exe

```
Input #0, matroska,webm, from 'D:\test\media\t_mkv\hddvd_demo_1080p.mkv':
  Metadata:
    encoder         : libebml v0.7.7 + libmatroska v0.8.1
    creation_time   : 2007-04-07T03:28:47.000000Z
  Duration: 00:02:01.90, start: 0.000000, bitrate: 19153 kb/s
    Stream #0:0(eng): Video: vc1 (Advanced) (WVC1 / 0x31435657), yuv420p(bt709), 1920x1080 [SAR 1:1 DAR 16:9], 29.97 tbr
, 1k tbn, 59.94 tbc (default)
    Metadata:
      title           : 1080p VC-1
    Stream #0:1(eng): Audio: ac3, 48000 Hz, stereo, fltp, 640 kb/s (default)
    Metadata:
      title           : Dolby Digital 2.0 640kbps
    Stream #0:2(eng): Audio: eac3, 48000 Hz, 6 channels, fltp
    Metadata:
      title           : Dolby Digital Plus 5.1 640kbps
Frames Input 1 Frames Displayed 1 FPS 9,174312 Sleep 0
Frames Input 2 Frames Displayed 2 FPS 6,734007 Sleep 0
Frames Input 3 Frames Displayed 3 FPS 6,410256 Sleep 0
Frames Input 4 Frames Displayed 4 FPS 7,312614 Sleep 0
Frames Input 5 Frames Displayed 5 FPS 8,431704 Sleep 0
Frames Input 6 Frames Displayed 6 FPS 9,146341 Sleep 0
Frames Input 7 Frames Displayed 7 FPS 9,749304 Sleep 0
Frames Input 8 Frames Displayed 8 FPS 10,45752 Sleep 0
Frames Input 9 Frames Displayed 9 FPS 10,86957 Sleep 0
Frames Input 10 Frames Displayed 10 FPS 11,23596 Sleep 0
Frames Input 11 Frames Displayed 11 FPS 11,73959 Sleep 0
Frames Input 12 Frames Displayed 12 FPS 12 Sleep 0
Frames Input 13 Frames Displayed 13 FPS 12,41643 Sleep 0
Frames Input 14 Frames Displayed 14 FPS 12,62399 Sleep 0
Frames Input 15 Frames Displayed 15 FPS 12,79863 Sleep 0
```

Building project

Build solution of the library located in the "*Solutions*" folder. On the target PC should be installed WIndows SDK (version 10 on my system) and .NET Framework (v 4.5.2), and the project can be built with Visual Studio 2017 (v141) platform toolset. It is possible to use different versions of SDK, .NET Framework and build toolsets. They can be configured in a project directly.


FFmpeg.NET Property Pages

Configuration: Debug   Platform: Win32   Configuration Manager...

General	
Target Platform	Windows 10
Windows SDK Version	**10.0.17763.0**
Output Directory	**..\..\Output\Bin\$(Configuration)\$(PlatformTarget)**
Intermediate Directory	**..\..\Output\Obj\$(Configuration)\$(ProjectName)\$(PlatformTar**
Target Name	$(ProjectName)
Target Extension	.dll
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.t
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	**Visual Studio 2017 (v141)**
Enable Managed Incremental Build	Yes
Project Defaults	
Configuration Type	**Dynamic Library (.dll)**
Use of MFC	Use Standard Windows Libraries
Character Set	**Use Unicode Character Set**
Common Language Runtime Support	**Common Language Runtime Support (/clr)**
.NET Target Framework Version	**v4.5.2**
Whole Program Optimization	No Whole Program Optimization
Windows Store App Support	No

Whole Program Optimization
Specifies that the program will be optimized across .obj boundaries; works best if C/C++ optimizations are turned on.

FFmpeg libraries and headers located in "*ThirdParty\ffmpeg\x.x.x*" where "*x.x.x*" is the FFmpeg version. There are up to four subfolders inside: "*bin*" with dll libraries located in there, "*lib*" - with linker libraries,

“*include*” - with exported api headers, also possible to put FFmpeg source into “*sources*” folder. Sources, if present, must be the same version. Depending on the build configuration, “*lib*” and “*bin*” contain “*x64*” and “*x86*” divisions.
The FFmpeg version with the path must be specified in project settings for compiler include path configuration (*Properties\C/C++\General\Additional Include directories*), linker library search path configuration (*Properties\Linker\General\Additional Library directories*) and post build library copy events (*Properties\Build Events\Post-Build Event*).
If sources present then the “**HAVE_FFMPEG_SOURCES**” definition may be set in project settings or in a precompiled header file.
If everything is set up correctly then the building project succeeded.

Licensing and distribution

The FFmpeg.NET Wrapper of FFmpeg libraries later "Wrapper Library" is provided "as is" without warranty of any kind; without even the implied warranty of merchantability or fitness for a particular purpose.

Wrapper Library is the source available software. It is Free to use in non-commercial software and open source projects.

Usage terms described in attached license text file of the project.