

DTMediaRead Programmers Interface



Copyright 2004-2022 Drastic Technologies Ltd.

All Rights Reserved



www.drastic.tv

Table of Contents

DTMediaRead Programmers Interface.....	1
Introduction.....	4
ActiveX Usage.....	4
Direct Link Usage.....	4
Methods and Properties.....	6
dtmrOpen.....	6
dtmrOpenMulti.....	6
dtmrClose.....	6
dtmrSourceFileName.....	7
dtmrSourceHeight.....	7
dtmrSourceWidth.....	7
dtmrSourceBitDepth.....	7
dtmrSourceFourCC.....	7
dtmrSourceBitRate.....	7
dtmrSourceFrameSize.....	7
dtmrSourceVideoChannels.....	7
dtmrSourceAudioChannels.....	8
dtmrSourceAudioFrequency.....	8
dtmrSourceAudioBitsPerSample.....	8
dtmrSourceAudioFourCC.....	8
dtmrDuration.....	8
dtmrAudioDuration.....	8
dtmrSourceRate.....	8
dtmrSourceScale.....	9
dtmrSourceMetaDataDWORD.....	9
dtmrSourceMetaDataSTR.....	9
dtmrGetReadTypes.....	9
dtmrSetReadType.....	10
dtmrGetFrame.....	11
dtmrSetFrame.....	11
dtmrSetVideoChannel.....	11
dtmrGetVideoFrame.....	11
dtmrAudioChannelPair.....	11
dtmrGetAudioFrame.....	12
dtmrGetCurExtendedData.....	12
dtmrGetCurCloseCaptions.....	12
dtmrSourceAudioFourCC.....	13
dtmrSetVideoChannel.....	13
dtmrAudioChannelPair.....	13
dtmrSetMode.....	13
dtmrLastVitcFrame.....	13
dtmrLastVitcUb.....	14

dtmrLastLtcFrame.....	14
dtmrLastLtcUb.....	14
dtmrGetFileFrameInfo.....	14
dtmrGetCaptureDiscontinuities.....	16
dtmrFreeCaptureDiscontinuities.....	17
Clip Detection API.....	18
Defines And Constants.....	23
Output Video Formats.....	24
ARGB 32 (8 bits per component, vertical invert).....	24
DTMR_READTYPE_RGBA.....	24
RGB 24 (8 bits per component, vertical invert).....	24
DTMR_READTYPE_RGB.....	24
AAA 24 (8 bits per component, vertical invert).....	24
DTMR_READTYPE_AAA.....	24
RGB 30 (10 bits per component).....	24
DTMR_READTYPE_RGB10Bit.....	24
YCrCb 8 (8 bits per component 4:2:2).....	25
DTMR_READTYPE_UVYV.....	25
YCrCb 10 (10 bits per component 4:2:2).....	25
DTMR_READTYPE_V210.....	25
RGBA 4:4:4:4 16 Per (64 Total) Integer.....	25
RGB 4:4:4 16 Per (48 Total) Integer.....	26
Alpha 16 Bit Integer (48 Total Repeated 3 Times).....	26
RGBA 4:4:4:4 16 Per (64 Total) Half Float.....	26
RGB 4:4:4 16 Per (48 Total) Half Float.....	26
Image Invert.....	26
Output Audio Formats.....	27
Example - Direct.....	28
Example - ActiveX.....	29
Example: Reading Size/Position Via dtmrGetFileFrameInfo.....	30
Metadata Elements.....	31
Direct Link Header.....	40

Introduction

The DTMediaRead interface is designed to give programmers a simple yet powerful access to all of the still, video, streaming and audio formats Drastic's MediaReactor and Drastic DDR software is able to read. This document describes the various methods and properties exported by DTMediaRead.

There are two ways to access the DTMediaRead API:

- ActiveX via direct 32 bit, or 32/64 bit RPC
- Direct link to Windows 32, Windows 64, Mac 32/64 and Linux 64

Both methods share the same set of functions and properties, but in the case of direct link, all the names are preceded by 'dtmr' to avoid namespace collisions. For instance, the ActiveX function LastLtcFrame(), would be dtmrLastLtcFrame() in the direct link version.

ActiveX Usage

Deprecated The ActiveX will install with the required DLLs into "C:\Program Files\MediaReactor" or "C:\Program Files(x86)\MediaReactor". The ActiveX component can be added to your project by inserting the MediaReactor ActiveX control via your IDE. For the functions below, remove the dtmrXXX start.

Direct Link Usage

All the functions in the direct link model have 'dtmr' prepended to the function name. This means the 'GetVideoFrame' becomes 'dtmrGetVideoFrame' to avoid naming conflicts. The direct link setup depends on the platform being used:

Windows 32

"C:\Program Files\MediaReactor"

Windows 64 – using 32 bit

"C:\Program Files(x86)\MediaReactor"

Windows 64 – using 64 bit

"C:\Program Files\MediaReactor"

Mac OS-X

/Libraries/Frameworks/DrasticDDR.framework

Linux 64
/usr/bin
/usr/lib

To use the direct link, you will need to include "dtmediaread.h" in your source file, and link to libdtmediaread.lib/.a/framework, depending on your platform.

Soft link is also an option for the direct link API. Each function prototype includes a function pointer typedef. It is the same as the prototype with a 'p_' added to the front. The SDK also ships with a C file dtmr_loader.cpp that has all the functions as point, and a load/unloader function for your convenience.

Methods and Properties

dtmrOpen

```
long dtmrOpen(BSTR bstrFileName, long dwFlags);
```

Open a new file, stream or network source for reading. The BSTR is a UTF-8 string on all platforms (Windows will be automatically converted to Unicode). This will attempt to open as much media as is available automatically. Along with the specified file, any 'side bar' audio file will be added, as well as information files (XML, RDF, TCI, NDX) and time code tracks. In the case of a series of stills, only one still needs to be specified. DTMediaRead attempts to treat everything as a video stream, so if it can build a sequence out of stills, it will. Here are the basic types that may be automatically added to your main video stream:

xxx.avi	- Main stream (could be mov, gen, omf, etc)
xxx.wav	- Will replace audio 1 & 2
xxx.A12.wav/aiff	- Also, will replace audio 1 & 2
xxx.AA.wav/aiff	- Also, will replace audio 1 & 2
xxx.A1.wav/aiff	- Mono wave pairs
xxx.A34.wav	- Add more audio channels
xxx.XML	- Metadata information (dt: space)
xxx.RDF	- Uniform description XML
xxx.TCI	- Time code information file
xxx.NDX	- Frame index file
xxx.TC	- Time code stream

dtmrOpenMulti

```
DTMRHANDLE dtmrOpenMulti(char * szFileNameVAA[17],  
unsigned long dwFlags);
```

Similar to dtmrOpen, but used to open a group of one (or no) video file, and up to 16 audio files. Pointers are const, and must be filled in order except for video, which may be null.

dtmrClose

```
long dtmrClose(DTMRHANDLE dtmr);
```

Close the currently open opaque handle to the stream or file.

dtmrSourceFileName

```
void dtmrSourceFileName(DTMRHANDLE dtmr, char *  
    tszMAX_PATHString);
```

The final file name used for the source file returned as UTF-8.

dtmrSourceHeight

```
long dtmrSourceHeight(DTMRHANDLE dtmr, long *pVal);
```

Source video media's height.

dtmrSourceWidth

```
long dtmrSourceWidth(DTMRHANDLE dtmr, long *pVal);
```

Source video media's width.

dtmrSourceBitDepth

```
long dtmrSourceBitDepth(DTMRHANDLE dtmr, long *pVal);
```

Source video media's bit depth.

dtmrSourceFourCC

```
long dtmrSourceFourCC(DTMRHANDLE dtmr, long *pVal);
```

Source video media's four character code compression type.

dtmrSourceBitRate

```
long dtmrSourceBitRate(DTMRHANDLE dtmr, long *pVal);
```

Source video media's bit rate.

dtmrSourceFrameSize

```
long dtmrSourceFrameSize(DTMRHANDLE dtmr, long dwFrame,  
    long *pVal);
```

Source video media's frame size for the requested or current frame. This will return the minimum size required for the frame, including padding, and should be used to pass into the read, unless a different padding is required.

dtmrSourceVideoChannels

```
long dtmrSourceVideoChannels(DTMRHANDLE dtmr, long  
    *pVal);
```

Source video total channels as a bitwise array.

dtmrSourceAudioChannels

```
long dtmrSourceAudioChannels(DTMRHANDLE dtmr, long *pVal);
```

Source audio total channels as a bitwise array. Here are some examples of the dtmrSourceAudioChannels return:

```
Mono = 0x0001 (1)  
Stereo = 0x0003 (2)  
Quad = 0x000F (15)  
Eight Channel = 0x00FF (255)
```

dtmrSourceAudioFrequency

```
long dtmrSourceAudioFrequency(DTMRHANDLE dtmr, long *pVal);
```

Source audio media frequency.

dtmrSourceAudioBitsPerSample

```
long dtmrSourceAudioBitsPerSample(DTMRHANDLE dtmr, long *pVal);
```

Source audio media bits per sample.

dtmrSourceAudioFourCC

```
long dtmrSourceAudioFourCC(DTMRHANDLE dtmr, long *pVal);
```

Source audio four character code. The most common would be 0/1 – little endian PCM and 'sowt' – big endian PCM.

dtmrDuration

```
long dtmrDuration(DTMRHANDLE dtmr, long *pVal);
```

Return the duration (total number of frames) of the media.

dtmrAudioDuration

```
long dtmrAudioDuration(DTMRHANDLE dtmr, long *pVal);
```

Return the audio duration (total number of samples) of the media.

dtmrSourceRate

```
long dtmrSourceRate(DTMRHANDLE dtmr, long *pVal);
```


Source video rate value (FPS = SourceRate / SourceScale)

dtmrSourceScale

```
long dtmrSourceScale(DTMRHANDLE dtmr, long *pVal);
```

Source video scale value (FPS = SourceRate / SourceScale)

NOTE: It is best to use standard Rate/Scale descriptors when setting up files. Here are the most common: 24/1, 24000/1001, 25/1, 30000/1001, 30/1, 50/1, 60000/1001, 60/1

dtmrSourceMetaDataDWORD

```
long dtmrSourceMetaDataDWORD(DTMRHANDLE dtmr, long dwMetaDataElement, long *pVal);
```

Return source metadata information that are numeric (DWORDs or longs). See the **Metadata Elements** section towards the end of the manual. Works for vvwTimeCode to vvwWhiteBalance inclusive, and vvwVideoWidth to vvwAudioBits inclusive.

dtmrSourceMetaDataSTR

```
BSTR * dtmrSourceMetaDataSTR(DTMRHANDLE dtmr, long dwMetaDataElement, char * szMAX_PATHString);
```

Return source metadata information that are string data. See the **Metadata Elements** section towards the end of the manual. Works for vvwFileName to vvwUMID inclusive.

dtmrGetReadTypes

```
long dtmrGetReadTypes(DTMRHANDLE dtmr, unsigned long dwIndex, unsigned long * pdwTypes);
```

Returns recommended and supported read types. Please see the **Output Video Formats** section for more information on the available types. One type is returned for each index specified until DTMR_READTYPE_INVALID is returned. The first return (where dwIndex = 0) will always be DTMR_READTYPE_RGBA as all video types can decode to our native RGBA. The next return depends on the source type. For RGB(A) sources, dwIndex = 1 will return DTMR_READTYPE_INVALID indicating that on RGBA decoding is supported. Below are a few sample returns for different file types. dtmrGetReadTypes() should always be checked unless you are using dtmrSetReadType(DTMR_READTYPE_RGBA).

Type	dwIndex	Return
DPX:	0	DTMR_READTYPE_RGB10Bit
	1	DTMR_READTYPE_RGBA
	2	DTMR_READTYPE_INVALID
v210Mov:	0	DTMR_READTYPE_V210
	1	DTMR_READTYPE_UYVY
	2	DTMR_READTYPE_RGBA
	3	DTMR_READTYPE_INVALID
AbacusYUV:	0	DTMR_READTYPE_UYVY
	1	DTMR_READTYPE_RGBA
	2	DTMR_READTYPE_INVALID
MPEG:	0	DTMR_READTYPE_UYVY
	1	DTMR_READTYPE_RGB
	2	DTMR_READTYPE_INVALID
DNG:	0	DTMR_READTYPE_RGBHALFFLOAT
	1	DTMR_READTYPE_RGB48
	2	DTMR_READTYPE_RGBA
	3	DTMR_READTYPE_INVALID

(Note: Formats such as MPEG, MJPEG and MPEG-4 will return DTMR_READTYPE_UYVY as a possible type because they are YCbCr based. The DTMR_READTYPE_UYVY frame will always return 4:2:2 YCbCr interleaved samples even if the source format is a lower sampling rate such as 4:2:0 or 4:1:1)

dtmrSetReadType

```
long dtmrSetReadType(DTMRHANDLE dtmr, long lReadType);
```

Set the read type for the video frames. Please see the **Output Video Formats** section for more information on the available types. This function should only be set to one of the types available as specified in the GetReadTypes() return.

Set read type also allows the caller to set the audio bit size returned to 16 bit or 32 bit. All audio will be decoded and converted to the requested format. If this is not set, it will return 16 for 16 and 32 for any other bit depth.

dtmrGetFrame

dtmrSetFrame

```
long dtmrGetFrame(DTMRHANDLE dtmr, long *pVal);  
long dtmrSetFrame(DTMRHANDLE dtmr, long newVal);
```

Get or set the current absolute (zero based) frame.

dtmrSetVideoChannel

```
long dtmrSetVideoChannel(DTMRHANDLE dtmr, long  
IVideoChannel);
```

Set the video channel to be read by get video frame.

dtmrGetVideoFrame

```
long dtmrGetVideoFrame(DTMRHANDLE dtmr, unsigned char *  
psvFrame, long * pISize);
```

GetVideoFrame returns a safe array containing one video frame. Passing NULL psvFrame will return size of the allocation for the frame, including any temporary space. Once allocated, the pISize should include the actual frame size. This can be retrieved from dtmrSourceFrameSize() for the smallest representation. Larger values can be passed in, and this will cause the read to pad out to those sizes, if possible. This is only available for RGB frame types. Please note, this function will only return frames in a few specified formats. These formats do not change regardless of the parameters returned by the SourceXXX methods. Please see the **Output Video Formats** section for more information on the available types.

dtmrAudioChannelPair

```
long dtmrSetAudioChannelPair(DTMRHANDLE dtmr, long  
IAudioChannelPair);
```

Set the audio channel pair to be read by get audio frame. Since the audio frame returns a stereo set, you can then select the two channels you want to read from the available bits. Basically:

- 0 - 0x0003
- 1 - 0x000C
- 2 - 0x0030
- 3 - 0x00C0

dtmrGetAudioFrame

```
long dtmrGetAudioFrame(DTMRHANDLE dtmr, unsigned char *  
psaFrame, long * pISize);
```

GetAudioFrame returns a safe array containing one video frame worth of audio data. Please note that the audio data is always returned as uncompressed, stereo PCM regardless of the values describing the source material's type returned by SourceXXX methods. Please see the **Output Audio Formats** section for more information.

dtmrGetCurExtendedData

dtmrGetCurCloseCaptions

```
long dtmrGetCurExtendedData(DTMRHANDLE dtmr, unsigned  
char *pvData, unsigned long * pIFlags, long *pISize);  
long dtmrGetCurClosedCaptions(DTMRHANDLE dtmr, unsigned  
char *pvCC, long *pIccSize, long * pIccFlags);
```

Get current extended data or close captions (*NOTE: to set this for a particular frame, dtmrSetFrame and dtmrGetVideoFrame() or dtmrGetFileFrameInfo() must be called first*). Normally both these calls return some combination of closed captions. With certain files, like Navy or NASA embedded data, the extended data may be something other than closed captions. The first two bytes are always CC1/CC3. If the FRAMEINFO_DATA_F1_EIA608 flag is not set, their value is undefined, but will likely be 0x80 0x80. The second two bytes are always CC2/CC4 if the FRAMEINFO_DATA_F2_EIA608 flag is set, otherwise they are undefined but will likely be 0x80 0x80. Everything from byte 4 on are 708 or OP-47 SMPTE 436 packets of closed captions, active format description and v-chip IDs. Each ANC packet will start with its DID SDID and size (for example for 708 captions 0x61 0x01 0x49). That size can be used to run through multiple ANC packets for a given frame. The CC, if it exists, will always be first, followed by any AFD, V-Chip or other custom packets.

```
//! Data is EIA-608B SD closed caption data field one (uses 2 bytes)  
#define FRAMEINFO_DATA_F1_EIA608          0x00000001  
//! Data is EIA-608B SD closed caption data field two (uses 2 bytes)  
#define FRAMEINFO_DATA_F2_EIA608          0x00000002  
//! Data is EIA-708 HD closed caption data (uses remaining bytes =  
minus the above)  
#define FRAMEINFO_DATA_EIA708             0x00001000  
//! Data is OP-47 closed caption data
```

```
#define FRAMEINFO_DATA_OP47 0x00002000
```

dtmrSourceAudioFourCC

```
long dtmrSourceAudioFourCC(DTMRHANDLE dtmr, long *pVal)
```

Get the audio four character code

dtmrSetVideoChannel

```
long dtmrSetVideoChannel(DTMRHANDLE dtmr, long  
IVideoChannel);
```

Select the video channel to read. This is a bitwise array.

dtmrAudioChannelPair

```
long dtmrSetAudioChannelPair(DTMRHANDLE dtmr, long  
IAudioChannelPair);
```

Select the audio pair to read. This is NOT bitwise. Each pair is a number starting at 0:

Pair	Bitwise Channel Set
0	0x0003
1	0x000F
2	0x0030
3	0x00F0

dtmrSetMode

```
long dtmrSetMode(DTMRHANDLE dtmr, void * pMediCmd);
```

Set mediacmd (advanced). This can be used to set advanced settings like GPU enable, number of threads in codec, number of threads in file, white balance, matrix enable and other settings. Please contact Drastic for more information.

dtmrLastVtcFrame

```
long dtmrLastVtcFrame(DTMRHANDLE dtmr, long *pVal);
```

Return the last dtmrGetVideoFrame VITC (vertical blank) time code as a frame value. To set this for a particular frame, dtmrSetFrame and dtmrGetVideoFrame() or dtmrGetFileFrameInfo() must be called first.

dtmrLastVitcUb

long dtmrLastVitcUb(DTMRHANDLE dtmr, long *pVal);

Return the last GetVideoFrame VITC (vertical blank time code) user bits. To set this for a particular frame, dtmrSetFrame and dtmrGetVideoFrame() or dtmrGetFileFrameInfo() must be called first.

dtmrLastLtcFrame

long dtmrLastLtcFrame(DTMRHANDLE dtmr, long *pVal);

Return the last dtmrGetVideoFrame LTC (SMPTE) time code. To set this for a particular frame, dtmrSetFrame and dtmrGetVideoFrame() or dtmrGetFileFrameInfo() must be called first.

dtmrLastLtcUb

long dtmrLastLtcUb(DTMRHANDLE dtmr, long *pVal);

Return the last dtmrGetVideoFrame LTC (SMPTE time code) user bits. To set this for a particular frame, dtmrSetFrame and dtmrGetVideoFrame() or dtmrGetFileFrameInfo() must be called first.

dtmrGetFileFrameInfo

long dtmrGetFileFrameInfo(DTMRHANDLE dtmr, unsigned long dwFrame, unsigned long dwChannels, unsigned long dwFlags, size_t * pnPosition, size_t * pnSize, unsigned long * pdwFrameFlags, char * szFilePathAndName);

This call returns information about a frame (or group of samples) of audio or video. It will return the position, size, frame flags and file name for a video sample or audio sample groups.

1.

//! Send this in if you just need the filename (faster than getting all the info)

```
#define DPOSSIZENAME_FILENAME_ONLY 0x40000000
```

```
    // Same as DFRAME_SKIP_FRAME
```

```
    //! Flag for mediafile/avhal to get audio dframe
```

```
#define GetAudio 0x00000000
```

```
    //! Flag for mediafile/avhal to get video dframe
```

```
#define GetVideo 0x00000001
```

```
// dwFrameFlags
```

```
#define DPOSSIZENAME_VIDEO_FRAME 0x00000001
```

```

    //! Is this file type currently recording
#define DPOSSIZENAME_RECORDING          0x00000004
    //! This frame needs to be made black (default frame) in
MediaFile
#define DPOSSIZENAME_PLEASE_BLACK
_PDFRAMEFLAGS_PLEASE_BLACK //    0x00000080
    //! This is a mono audio chunk
#define DPOSSIZENAME_MONO_AUDIO_FRAME   0x00000100
    //! This is a stereo audio chunk
#define DPOSSIZENAME_STEREO_AUDIO_FRAME 0x00000200
#define DPOSSIZENAME_QUAD_AUDIO_FRAME   0x00000400
#define DPOSSIZENAME_4_1_AUDIO_FRAME    0x00000800
#define DPOSSIZENAME_5_1_AUDIO_FRAME    0x00001000
#define DPOSSIZENAME_7_1_AUDIO_FRAME    0x00002000
#define DPOSSIZENAME_9_1_AUDIO_FRAME    0x00004000
#define DPOSSIZENAME_AUDIO_MASK
(DPOSSIZENAME_MONO_AUDIO_FRAME|
DPOSSIZENAME_STEREO_AUDIO_FRAME|
DPOSSIZENAME_STEREO_AUDIO_FRAME|
DPOSSIZENAME_QUAD_AUDIO_FRAME|
DPOSSIZENAME_4_1_AUDIO_FRAME|
DPOSSIZENAME_5_1_AUDIO_FRAME|
DPOSSIZENAME_7_1_AUDIO_FRAME|
DPOSSIZENAME_9_1_AUDIO_FRAME)
#define DPOSSIZENAME_FRAME_MASK          0x0000FFFF
    //! This frame contains audio data see DFRAME::dwType
#define DFRAME_TYPE_AUDIO                0x00010000
    //! 16 bit audio
#define DPOSSIZENAME_AUD_16_16_BIT       0x00100000
    //! 20 bit audio in 24
#define DPOSSIZENAME_AUD_20_24_BIT       0x00200000
    //! 24 bit audio in 24
#define DPOSSIZENAME_AUD_24_24_BIT       0x00400000
    //! 24/32 bit audio in 32
#define DPOSSIZENAME_AUD_24_32_BIT       0x00800000
    //! 32/32 bit audio in 32
#define DPOSSIZENAME_AUD_32_32_BIT       0x01000000
    //! Audio is compressed
#define DPOSSIZENAME_AUD_COMPRESSED      0x02000000
    //! Audio is big endian, else little endian
#define DPOSSIZENAME_AUD_BIGENDIAN_BIT   0x00080000
    //! Just for completeness
#define DPOSSIZENAME_AUD_LITTLEENDIAN_BIT 0x00000000
    //! This frame is independent of other frames for decode see

```

```

DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME 0x10000000
    //! This frame is independent of other frames for decode (an
MPEG I Frame) see DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME_I    0x10000000
    //! This frame requires previous keyframe(s) (for MPEG a P
Frame) see DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME_P    0x80000000
    //! This frame requires more than one frame to decode (for
MPEG a B Frame) see DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME_B    0x20000000
    //! This frame should be skipped (decoded, but not displayed) -
Used to reach seek frame on a non key frame from key frame see
DFRAME::dwType
#define DFRAME_SKIP_FRAME          0x40000000

```

dtmrGetCaptureDiscontinuities

```

long dtmrGetCaptureDiscontinuities(DTMRHANDLE dtmr,
unsigned char ** pszDiscontinuities);

```

Return the list, or part of the list, of discontinuities that have occurred while capturing the file. The function may need to be called more than once, for long lists of discontinuities. The format is XML, with each individual <Discontinuity ...> is with a single <DISCONTINUITIES> </DISCONTINUITIES> key pair. The first call will return nothing, if there are no discontinuities. If there are, the first call will start with <DISCONTINUITIES> followed by lines of <Discontinuity .. >. For small numbers the entire list will be returned on the first call with the ending </DISCONTINUITIES>. For larger lists, subsequent calls will return the next set of discontinuity lines, until all are returned along with the closing </DISCONTINUITIES> tag. In the case that the file you're checking is still writing you can treat it the same as larger lists, just keep calling dtmrGetCaptureDiscontinuities and you'll get the next piece if available.

If you want force a reread just free the pointer with dtmrFreeCaptureDiscontinuities and the next call to dtmrGetCaptureDiscontinuities will start from the beginning.

Below is sample list:

```

<DISCONTINUITIES>
    <Discontinuity type="video" time="10:25:24 Wednesday, July
27, 2016" frame="286" timecode="2046723"

```



```
timecodetype="D">18:58:12;28</Discontinuity>
  <Discontinuity type="audio" time="10:25:24 Wednesday, July
27, 2016" sample="13728000" timecode="2046723"
timecodetype="D">18:58:12;29</Discontinuity>
  <Discontinuity type="meta tc" time="10:25:24 Wednesday, July
27, 2016" frame="286" timecode="2046723"
timecodetype="D">18:58:12;29</Discontinuity>
  <Discontinuity type="timecode" time="10:25:25 Wednesday,
July 27, 2016" frame="305" timecode="2046723" timecodetype="D"
comment="Last I TC: 2046723 Current:
2046423">18:58:13;17</Discontinuity>
</DISCONTINUITIES>
```

dtmrFreeCaptureDiscontinuities

```
long dtmrFreeCaptureDiscontinuities(DTMRHANDLE dtmr,
unsigned char ** pszDiscontinuities);
```

Free the buffer returned by dtmrGetCaptureDiscontinuities. This only needs to be called after all the calls to the 'get' function and the entire list has been returned.

Clip Detection API

DTMediaRead includes a clip detection API. This API finds clips within a directory structure, like P2 or BDMV, as well as sequences of files and other standard formats. It also includes stitching together of files that were recorded as segments and combining audio, video, closed caption and metadata files as a single entity. Once found, the clips may be opened with the read API above, and the individual elements can be retrieved, as well as reading each clip with all its components automatically.

Header: detectdir.h

ddtFileDir

Normal file directory

ddtAVCHD

Sony, JVC, Panasonic, Canon camera format

AVCHD

AVF_INFO

private

AVCHD

BDMV

CLIPINF

PLAYLIST

STREAM

SONY

ddtBDMV

BDMV - BluRay, subset of AVCHD

BDMV

CLIPINF

PLAYLIST

STREAM

ddtXAVCS

Sony XAVC-S cameras

XAVC-S

M4ROOT

CLIP

GENERAL

SUB

THMBNL

ddtP2Atom

Panasonic P2 Atom (DV25, 50, 100, AVCi 100)

*CONTENTS**AUDIO**CLIP**ICON**PROXY**VIDEO**VOICE***ddtP2OP1b**

Panasonic P2 OP1b (AVCi 100/200, Long G)

*CONTENTS**AUDIO**AVCLIP**CLIP**ICON**PROXY**VIDEO**VOICE***ddtXDCam**

Sony XDCam

Clip

Edit

General

Sub

ddtHDV

Sony HDV Camcorder

ddtC500

Canon C Series 500/700

ddtCanonXF

Canon XF Cameras

CONTENTS

CLIPS001

<NAME>

JOURNAL

ddtGVJpeg2000

Grass Valley

<Name>.bmp

<Name>.xml

<Name>_DM.xml

<Name>-<num>_H.mxf

<Name>-<num>_H.mxf

ddtCodex

Codex RAW formats

<FileNameAsDir>

<FileName><NUM>.rmf

<FileNameBase>.xml

ddtGemini

Gemini Recorder

<FileNameAsDir>

<FileNameRoot>.xml

<FileName><NUM>.rmf

ddtGF

GF (Similar to P2)

BIN001

ANC

AUDIO

CLIPINF

PIC

PROXY

VBI

VIDEO

PLAYLIST

ddtArriRaw

Arri Raw

ddtRed

RED camera

ddtHasSequences

Generic sequence(s)

ddtK2Server

Grass Valley server (K2)

audio.A#

data.N#

timecode.T#

video.F#

<clipname>.xml

detectDirType __stdcall ddtCheckDir(char * szDirectory);

Quickly determine the type of the directory specified

**DDTHANDLE __stdcall ddtOpenDir(char * szDirectory,
detectDirType ddType);**

Open a directory as a particular type

DWORD __stdcall ddtCloseDir(DDTHANDLE ddtH);

Close a previously open directory

DWORD __stdcall ddtGetTotalFiles(DDTHANDLE ddtH);

Get total files/clips

**DWORD __stdcall ddtGetFilePath(DDTHANDLE ddtH, int nIndex,
char * szFullPath);**

Get a clip file name with full path

**DWORD __stdcall ddtGetBaseName(char * szFullPath, char *
szBaseName);**

Get base file name for sequences

**DWORD __stdcall ddtGetFileInfo(DDTHANDLE ddtH, int nIndex,
WIN32_FIND_DATA * pFileInfo);**

Get a clip file info

DWORD __stdcall ddtCheckSequence(char * szFileName);

Detect if file is part of a sequence

Defines And Constants

These formats are used by dtmrGetReadTypes() and dtmrSetReadType() to set up the frame return type for dtmrGetVideoFrame(). See the **Video Output Formats** section for more information on these frame layouts.

```
//! Windows RGBA (like bitmap, tga, etc)
const unsigned long DTMR_READTYPE_ARGB = 0;
//! RGB 8 bits per component, 24 total
const unsigned long DTMR_READTYPE_RGB = 0x10000000;
//! Alpha only 8 bits per component, repeated to 24
const unsigned long DTMR_READTYPE_AAA = 0x20000000;
//! 8 Bit YCbCr (yuv2, D1/HDSI raw 4:2:2 video
const unsigned long DTMR_READTYPE_UYVY = 1;
//! 10 Bit v210 (quicktime packing) 4:2:2 video
const unsigned long DTMR_READTYPE_V210 = 2;
//! 10 Bit RGB 4:4:4 (dpx packing)
const unsigned long DTMR_READTYPE_RGB10Bit = 3;
//! 16 bit per component (64 bit) RGBA 4:4:4:4
const unsigned long DTMR_READTYPE_RGBA64 = 4;
//! RGB 16 bits per component, 48 total
const unsigned long DTMR_READTYPE_RGB48 = 0x10000004;
//! Alpha only 16 bits per component, repeated to 48
const unsigned long DTMR_READTYPE_AAA16 = 0x20000004;
//! 16 bit half float per component RGBA (GPU)
const unsigned long DTMR_READTYPE_RGBAHALFFLOAT = 5;
//! 16 bit half float per component RGB (GPU)
const unsigned long DTMR_READTYPE_RGBHALFFLOAT = 6;
//! Set to invert the picture vertically
const unsigned long DTMR_READFLAG_FLIP = 0x80000000;
//! Invalid file
const unsigned long DTMR_READTYPE_INVALID = -1;
```

Output Video Formats

These are the formats supported by GetVideoFrame(). Each of these formats only appears as specified here for this return. The SourceXXX series of methods (including SourceBitDepth and SourceFourCC) refer to the video media as it is saved on disk. The DTMediaRead library will decompress, and where necessary convert, from the file's native format to the requested format set by SetReadType(). For each file opened, the GetReadTypes() should be called to determine the available read types.

ARGB 32 (8 bits per component, vertical invert)

DTMR_READTYPE_RGBA

ARGB Decreasing Address Order																															
Byte 3				Byte 2				Byte 1				Byte 0																			
Alpha				Red				Green				Blue																			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

RGB 24 (8 bits per component, vertical invert)

DTMR_READTYPE_RGB

RGB Decreasing Address Order																											
				Byte 2				Byte 1				Byte 0															
				Red				Green				Blue															
				7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

AAA 24 (8 bits per component, vertical invert)

DTMR_READTYPE_AAA

RGB Decreasing Address Order																											
				Byte 2				Byte 1				Byte 0															
				Alpha				Alpha				Alpha															
				7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

RGB 30 (10 bits per component)

DTMR_READTYPE_RGB10Bit

RGB 10 Bit Decreasing Address Order											
Byte 3			Byte 2			Byte 1			Byte 0		
Blue			Green	Blue		Re d	Green		Red		

RGB 10 Bit Decreasing Address Order																															
5	4	3	2	1	0			3	2	1	0	9	8	7	6	1	0	9	8	7	6	5	4	9	8	7	6	5	4	3	2

Please note: This is the standard DPX file layout, which was originally big endian, but is viewed here as little endian.

YCrCb 8 (8 bits per component 4:2:2)

DTMR_READTYPE_UVYV

YCbCr8 2 Pixels, Decreasing Address Order																															
Byte 3				Byte 2				Byte 1				Byte 0																			
Cr				Y1				Cb				Y0																			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

YCrCb 10 (10 bits per component 4:2:2)

DTMR_READTYPE_V210

YCbCr10 Pixels, Decreasing Address Order																																					
Byte 3				Byte 2				Byte 1				Byte 0																									
Cr 0				Y 0				Cb 0																													
								9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
Byte 7				Byte 6				Byte 5				Byte 4																									
Y 2				Cb 1				Y 1																													
								9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
Byte 11				Byte 10				Byte 9				Byte 8																									
Cb 2				Y 3				Cr 1																													
								9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
Byte 15				Byte 14				Byte 13				Byte 12																									
Y 5				Cr 2				Y 4																													
								9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0

RGBA 4:4:4:4 16 Per (64 Total) Integer

```

//! 16 bit per component (64 bit) RGBA 4:4:4:4
const unsigned long DTMR_READTYPE_RGBA64 = 4;

```

RGB 4:4:4 16 Per (48 Total) Integer

```
//! RGB 16 bits per component, 48 total  
const unsigned long DTMR_READTYPE_RGB48 = 0x10000004;
```

Alpha 16 Bit Integer (48 Total Repeated 3 Times)

```
//! Alpha only 16 bits per component, repeated to 48  
const unsigned long DTMR_READTYPE_AAA16 = 0x20000004;
```

RGBA 4:4:4:4 16 Per (64 Total) Half Float

```
//! 16 bit half float per component RGBA (GPU)  
const unsigned long DTMR_READTYPE_RGBAHALFFLOAT = 5;
```

RGB 4:4:4 16 Per (48 Total) Half Float

```
//! 16 bit half float per component RGB (GPU)  
const unsigned long DTMR_READTYPE_RGBHALFFLOAT = 6;
```

Image Invert

```
//! Set to invert the picture vertically  
const unsigned long DTMR_READFLAG_FLIP = 0x80000000;
```

```
//! Invalid file  
const unsigned long DTMR_READTYPE_INVALID = -1;
```

Output Audio Formats

These are the formats supported by dtmrGetAudioFrame(). By default, it will return video frame sized chunks of audio in 16 bit container for 16 bit files, and 32 bit containers for other bit sizes. Using dtmrSetReadType, this can be modified to return a specific bit depth for the file, and switch from video frame audio sized groups to native/audio sample based reads:

```
//! Set readtype to video frame size AUDIO to 16 bits LE
const unsigned long DTMR_READTYPE_FRAME_AUDIO_16LE = (0x00010000 | 16);
//! Set readtype to video frame size AUDIO to 32 bits (note, 16, 20, 24 will be shifted
to most significant, LE)
const unsigned long DTMR_READTYPE_FRAME_AUDIO_32LE = (0x00010000 | 32);
//! Set readtype to arbitrary sample AUDIO to 16 bits LE
const unsigned long DTMR_READTYPE_SAMPLE_AUDIO_16LE = (0x00110000 | 16);
//! Set readtype to arbitrary sample AUDIO to 32 bits (note, 16, 20, 24 will be
shifted to most significant, LE)
const unsigned long DTMR_READTYPE_SAMPLE_AUDIO_32LE = (0x00110000 | 32);
```

Audio is always output as two channels of either 32 bit or 16 per sample PCM audio. This is written in the same format as Windows wave files.

Left Channel (2 or 4 bytes little endian)
Right Channel (2 or 5 bytes little endian)
[repeats with no padding]

The frequency is dependent on the dtmrSourceAudioFrequency return. The bit size is dependent on dtmrSourceAudioBitsPerSample. If the dtmrSourceAudioBitsPerSample is 16 or less, then it will return 16 bit samples. If it is greater than 16 bits (normally 20, 24 or 32), then it will return 32 bits, where the 20 or 24 have been shifted up to become 32 bits.

The size of the return is dependent on the frame rate of the file. This can vary from 23.98 fps, or 2000/2001 samples per frame, down to 60 fps, or 800 samples per frame. The size will also vary, depending on how the frame rate divides into the sample rate. For example:

48,000 hz audio at 29.97 video = 1601.6 samples
Because we can only return an even number of samples, the audio is returned in a 5 frame cadence of 1601 or 1602 samples. Because these are stereo, this means the application will receive 6404/6408 bytes in 16 bit, and 12808/12816 bytes in 20/24/32 bit.

Example - Direct

Follow these steps to read using the direct interface:

1. Open the file with `dtmrOpen(szUTF8FileName, 0)`. Keep the opaque handle returned for all further calls.
2. Get the file information via the `dtmrSourceHeight/Width/FourCC/etc` and the `dtmrSourceAudioChannels/Frequency/etc`
3. Optionally, get the best read type from `dtmrGetReadTypes`
4. Set the read type you want for audio and video. For simplicity, start with `dtmrSetReadType(h, DTMR_READTYPE_ARGB)` and `dtmrSetReadType(h, DTMR_READTYPE_FRAME_AUDIO_16LE)`.
5. Set the first frame you want to read `dtmrSetFrame(h, 0)`
6. Get the video suggested buffer size, and allocate memory for the video frame. Sending `dtmrGetVideoFrame(h, NULL, &lSize)` will return the suggested buffer size in `lSize`.
7. Read the video frame, passing in the actual video size of the frame. The actual size, without padding, can be retrieved from `dtmrSourceFrameSize()`. The second call to read would be `dtmrGetVideoFrame(h, pAllocation, &lSourceFrameSize)`
8. Read any audio channels you need with `dtmrGetAudioFrame(h, pAudAllocation, &lAudioSize)`, which works the same way as video size.
9. Get any per frame metadata with `dtmrLastVtcType/Frame/Ub`, `dtmrLastLtcType/Frame/Ub` and `dtmrGetCurExtendedData/dtmrGetCurClosedCaptions`
10. Repeat steps 7 through 9 for any other frames you need
11. Close the opaque handle with `dtmrClose(h)`

Example - ActiveX

Follow these steps to load a media file into the DTMediaRead ActiveX SDK and then get the desired data.

1. Open a file. Somewhere in your application, you must open the desired file using the `CDTReadX::Open(LPCTSTR bstrFileName, long dwFlags)`; function call. If the function returns -1, then the file has not been opened. Otherwise, proceed to step two.
2. Set the frame. After the file has been opened, set the current frame in the ActiveX by calling the `CDTReadX::SetFrame(long nNewValue)`; where the `nNewValue` is the desired frame. This procedure can be attached to a slider control or an editbox, used for setting the frame dynamically. (NOTE: the total number of frames should be known before setting the current frame.)
3. Obtaining data. Now that everything has been set up correctly, we can now access the ActiveX and retrieve the frame data for both audio and video. Variables you will need include a `VARIANT` to get the data, a `DWORD` to determine the size of the data, and a `LPVOID` to store the data.

```
DWORD dwSize;  
VARIANT vid_frame;  
void* lpVidData;  
m_dtRead.GetVideoFrame(&vid_frame, (long*)&dwSize);
```

If at this point, the size of the data (`dwSize`) is 0, then the ActiveX was unable to find any data for this frame. Now access the data using the `lpVidData` variable. Continue if successful.

```
if(SafeArrayAccessData(vid_frame.parray, &lpVidData) == S_OK)  
{  
    Now that we have access to the frame data, it can be manipulated however we like. It can be loaded into a device context, or saved to a bitmap, etc...
```
4. Freeing the data. Now that we are done with the data, it should be cleared.

```
SafeArrayUnaccessData(vid_frame.parray);  
SafeArrayDestroy(vid_frame.parray);
```
5. To get the audio frame data from the ActiveX, perform step 3 using the `CDTReadX::GetAudioFrame(VARIANT* psaFrame, long* plSize)`; function in place of `GetVideoFrame`.

Example: Reading Size/Position Via dtmrGetFileFrameInfo

Most file formats, and especially RTIndex files, can return the frame position and size, rather than the actual decoded audio/video data. This allows reading of files with your own decoders, and rewrapping of audio/video with direct reads to the data using normal OS reads. It is similar to a direct read, other than the read itself.

1. Open the file with `dtmrOpen(szUTF8FileName, 0)`. Keep the opaque handle returned for all further calls.
2. Get the file information via the `dtmrSourceHeight/Width/FourCC/etc` and the `dtmrSourceAudioChannels/Frequency/etc`
3. Set the first frame you want to read `dtmrSetFrame(h, 0)`
4. Call `dtmrGetFileFrameInfo` with the frame, channel you want and Audio/Video flags for the media you want. It will return the position, size, flags and file that media exists in. The flags will have audio info or video IPB flags.
5. Call `dtmrGetFileFrameInfo` for any other audio channels you want the media file.
6. Get any per frame metadata with `dtmrLastVitcType/Frame/Ub`, `dtmrLastLtcType/Frame/Ub` and `dtmrGetCurExtendedData/dtmrGetCurClosedCaptions`
7. Repeat steps 3 through 6 for any other frames you need
8. Close the opaque handle with `dtmrClose(h)`

Metadata Elements

The functions dtmrSourceMetaDataDWORD() and dtmrSourceMetaDataSTR() use the defines below to return specific metadata from the file. The first enums are string values for dtmrSourceMetaDataSTR() (from vvwFileName to vvwUMID). The second set of enums are the DWORD values (from vvwTimeCode to vvwAudioBits).

```
/** Numeric values for all the metadata information types available in MR and VVW
*/
enum vvwInfoMetaTypes {
    /** see VVWINFO::szFileName
    vvwFileName,
    /** see VVWINFO::szNativeLocator
    vvwNativeLocator,
    /** see VVWINFO::szUniversalName
    vvwUniversalName,
    /** see VVWINFO::szIP
    vvwIP,
    /** see VVWINFO::szSourceLocator
    vvwSourceLocator,

    /** see VVWINFO::szChannel
    vvwChannel,
    /** see VVWINFO::szChannelName
    vvwChannelName,
    /** see VVWINFO::szChannelDescription
    vvwChannelDescription,
    /** see VVWINFO::szTitle
    vvwTitle,
    /** see VVWINFO::szSubject
    vvwSubject,
    /** see VVWINFO::szCategory
    vvwCategory, // <-- 10
    /** see VVWINFO::szKeywords
    vvwKeywords,
    /** see VVWINFO::szRatings
    vvwRatings,
    /** see VVWINFO::szComments
    vvwComments,
    /** see VVWINFO::szOwner
    vvwOwner,
    /** see VVWINFO::szEditor
    vvwEditor,
    /** see VVWINFO::szSupplier
    vvwSupplier,
    /** see VVWINFO::szSource
    vvwSource,
    /** see VVWINFO::szProject
    vvwProject,
```

```
    //! see VVWINFO::szStatus
vzwiStatus,
    //! see VVWINFO::szAuthor
vzwiAuthor, // <-- 20
    //! see VVWINFO::szRevisionNumber
vzwiRevisionNumber,
    //! see VVWINFO::szProduced
vzwiProduced,
    //! see VVWINFO::szAlbum
vzwiAlbum,
    //! see VVWINFO::szArtist
vzwiArtist,
    //! see VVWINFO::szComposer
vzwiComposer,
    //! see VVWINFO::szCopyright
vzwiCopyright,
    //! see VVWINFO::szCreationData
vzwiCreationData,
    //! see VVWINFO::szDescription
vzwiDescription,
    //! see VVWINFO::szDirector
vzwiDirector,
    //! see VVWINFO::szDisclaimer
vzwiDisclaimer, // <-- 30
    //! see VVWINFO::szEncodedBy
vzwiEncodedBy,
    //! see VVWINFO::szFullName
vzwiFullName,
    //! see VVWINFO::szGenre
vzwiGenre,
    //! see VVWINFO::szHostComputer
vzwiHostComputer,
    //! see VVWINFO::szInformation
vzwiInformation,
    //! see VVWINFO::szMake
vzwiMake,
    //! see VVWINFO::szModel
vzwiModel,
    //! see VVWINFO::szOriginalArtist
vzwiOriginalArtist,
    //! see VVWINFO::szOriginalFormat
vzwiOriginalFormat,
    //! see VVWINFO::szPerformers
vzwiPerformers, // <-- 40
    //! see VVWINFO::szProducer
vzwiProducer,
    //! see VVWINFO::szProduct
vzwiProduct,
    //! see VVWINFO::szSoftware
vzwiSoftware,
    //! see VVWINFO::szSpecialPlaybackRequirements
vzwiSpecialPlaybackRequirements,
    //! see VVWINFO::szTrack
vzwiTrack,
```



```
    //! see VVWINFO::szWarning
    vwwiWarning,
    //! see VVWINFO::szURLLink
    vwwiURLLink,
    //! see VVWINFO::szEditData1
    vwwiEditData1,
    //! see VVWINFO::szEditData2
    vwwiEditData2,
    //! see VVWINFO::szEditData3
    vwwiEditData3, // <-- 50
    //! see VVWINFO::szEditData4
    vwwiEditData4,
    //! see VVWINFO::szEditData5
    vwwiEditData5,
    //! see VVWINFO::szEditData6
    vwwiEditData6,
    //! see VVWINFO::szEditData7
    vwwiEditData7,
    //! see VVWINFO::szEditData8
    vwwiEditData8,
    //! see VVWINFO::szEditData9
    vwwiEditData9,
    //! see VVWINFO::szVersionString
    vwwiVersionString,
    //! see VVWINFO::szManufacturer
    vwwiManufacturer,
    //! see VVWINFO::szLanguage
    vwwiLanguage,
    //! see VVWINFO::szFormat
    vwwiFormat, // <-- 60
    //! see VVWINFO::szInputDevice
    vwwiInputDevice,
    //! see VVWINFO::szDeviceModelNum
    vwwiDeviceModelNum,
    //! see VVWINFO::szDeviceSerialNum
    vwwiDeviceSerialNum,
    //! see VVWINFO::szReel
    vwwiReel,
    //! see VVWINFO::szShot
    vwwiShot,
    //! see VVWINFO::szTake
    vwwiTake,
    //! see VVWINFO::szSlateInfo
    vwwiSlateInfo,
    //! see VVWINFO::szFrameAttribute
    vwwiFrameAttribute,
    //! see VVWINFO::szEpisode
    vwwiEpisode,
    //! see VVWINFO::szScene
    vwwiScene, // <-- 70
    //! see VVWINFO::szDailyRoll
    vwwiDailyRoll,
    //! see VVWINFO::szCamRoll
    vwwiCamRoll,
```

```
    //! see VVWINFO::szSoundRoll
    vwiSoundRoll,
    //! see VVWINFO::szLabRoll
    vwiLabRoll,
    //! see VVWINFO::szKeyNumberPrefix
    vwiKeyNumberPrefix,
    //! see VVWINFO::szInkNumberPrefix
    vwiInkNumberPrefix,
    //! see VVWINFO::szPictureIcon
    vwiPictureIcon,
    //! see VVWINFO::szProxyFile
    vwiProxyFile,
    //!
    vwiCustomMetadataBlockPointer,
    //!
    vwiImageInfo,
    //!
    vwiUMID,
    //
    vwiEND_OF_STRINGS,

    vwiNumericStart = 0x1000,
    //! see VVWINFO::dwTimeCode
    vwiTimeCode,
    //! see VVWINFO::dwUserBits
    vwiUserBits,
    //! see VVWINFO::dwVITCTimeCode
    vwiVITCTimeCode,
    //! see VVWINFO::dwVITCUserBits
    vwiVITCUserBits,
    //! see VVWINFO::dwVITCLine3
    vwiVITCLine3,
    //! see VVWINFO::dwPosterFrame
    vwiPosterFrame,
    //! see VVWINFO::dwAFrame
    vwiAFrame,
    //! see VVWINFO::dwAspectRatio
    vwiAspectRatio,
    //! see VVWINFO::dwOriginalRate
    vwiOriginalRate,
    //! see VVWINFO::dwOriginalScale
    vwiOriginalScale,
    //! see VVWINFO::dwConversions
    vwiConversions,
    //! see VVWINFO::dwVersionNumber
    vwiVersionNumber,
    //! see VVWINFO::dwFileSize
    vwiFileSize,
    //! see VVWINFO::dwFileDate
    vwiFileDate,
    //! see VVWINFO::dwFileTime
    vwiFileTime,
    //! see VVWINFO::dwSequenceNumber
    vwiSequenceNumber,
```

```

    //! see VVWINFO::dwTotalStreams
    vwwiTotalStreams,
    //! see VVWINFO::dwTotalLength
    vwwiTotalLength,
    //! see VVWINFO::dwFilmManufacturerCode
    vwwiFilmManufacturerCode,
    //! see VVWINFO::dwFilmTypeCode
    vwwiFilmTypeCode,
    //! see VVWINFO::dwWhitePoint
    vwwiWhitePoint,
    //! see VVWINFO::dwBlackPoint
    vwwiBlackPoint,
    //! see VVWINFO::dwBlackGain
    vwwiBlackGain,
    //! see VVWINFO::dwBreakPoint
    vwwiBreakPoint,
    //! see VVWINFO::dwGamma1000
    vwwiGamma1000,
    //! see VVWINFO::dwTagNumber
    vwwiTagNumber,
    //! see VVWINFO::dwFlags
    vwwiFlags,
    //! see VVWINFO::dwTimeCodeType
    vwwiTimeCodeType,
    //! see VVWINFO::dwLTCTimeCodeType
    vwwiLTCTimeCodeType,
    //! see VVWINFO::dwVITCTimeCodeType
    vwwiVITCTimeCodeType,
    //! see VVWINFO::dwProdDate
    vwwiProdDate,
    //End: v3.0
    //! see VVWINFO::dwUniqueID
    vwwiUniqueID,
    //!
    vwwiCustomMetadataBlockType,
    vwwiCustomMetadataBlockSize,
    vwwiNorthSouthEastWest,
    vwwiLatitude,
    vwwiLongitude,
    vwwiExposure,
    vwwiRedGain,
    vwwiBlueGain,
    vwwiWhiteBalance,

    vwwiEND_OF_DWORD_V2,
    // Add elements here
    //VVVID STRUCT
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwwiVideoWidth = 0x10000,
    //! XML tag name for width
#define VVWINFOTAG_woVideoWidth "Width"
#define VVWINFODESC_woVideoWidth "Width"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwwiVideoHeight,

```

```

    //! XML tag name for height
#define VVWINFOTAG_woVideoHeight                "Height"
#define VVWINFODESC_woVideoHeight              "Height"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoPlanes,
    //! XML tag name for planes
#define VVWINFOTAG_woVideoPlanes                "Planes"
#define VVWINFODESC_woVideoPlanes              "Planes"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoBitCount,
    //! XML tag name for bit count
#define VVWINFOTAG_woVideoBitCount              "BitCount"
#define VVWINFODESC_woVideoBitCount            "BitCount"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoCompression,
    //! XML tag name for compression (fourcc)
#define VVWINFOTAG_woVideoCompression          "Compression"
#define VVWINFODESC_woVideoCompression        "Compression"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoSizeImage,
    //! XML tag name for size of the image in unsigned chars
#define VVWINFOTAG_woVideoSizeImage            "SizeImage"
#define VVWINFODESC_woVideoSizeImage          "SizeImage"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoXPelsPerMeter,
    //! XML tag name for X pels per meter
#define VVWINFOTAG_woVideoXPelsPerMeter        "XPelsPerMeter"
#define VVWINFODESC_woVideoXPelsPerMeter      "XPelsPerMeter"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoYPelsPerMeter,
    //! XML tag name for Y pels per meter
#define VVWINFOTAG_woVideoYPelsPerMeter        "YPelsPerMeter"
#define VVWINFODESC_woVideoYPelsPerMeter      "YPelsPerMeter"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoClrUsed,
    //! XML tag name for color elements used
#define VVWINFOTAG_woVideoClrUsed              "ClrUsed"
#define VVWINFODESC_woVideoClrUsed            "ClrUsed"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoClrImportant,
    //! XML tag name for
#define VVWINFOTAG_woVideoClrImportant          "ClrImportant"
#define VVWINFODESC_woVideoClrImportant        "ClrImportant"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoReserved,
    //! XML tag name for reserved array
#define VVWINFOTAG_woVideoReserved              "Reserved"
#define VVWINFODESC_woVideoReserved            "Reserved"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO
    vwiVideoFccType,
    //! XML tag name for four cc type (video/audio)
#define VVWINFOTAG_woVideoFccType              "FccType"
#define VVWINFODESC_woVideoFccType            "FccType"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VWVAUDIO

```

```

    vwiVideoFccHandler,
    //! XML tag name for four cc handler
#define VWINFOTAG_woVideoFccHandler          "FccHandler"
#define VWINFODESC_woVideoFccHandler        "FccHandler"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoFlags,
    //! XML tag name for flags
#define VWINFOTAG_woVideoFlags              "Flags"
#define VWINFODESC_woVideoFlags            "Flags"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoCaps,
    //! XML tag name for capabilities
#define VWINFOTAG_woVideoCaps              "Caps"
#define VWINFODESC_woVideoCaps            "Caps"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoPriority,
    //! XML tag name for priority
#define VWINFOTAG_woVideoPriority           "Priority"
#define VWINFODESC_woVideoPriority         "Priority"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoLanguage,
    //! XML tag name for language
#define VWINFOTAG_woVideoLanguage          "Language"
#define VWINFODESC_woVideoLanguage        "Language"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoScale,
    //! XML tag name for scale (fps = rate / scale)
#define VWINFOTAG_woVideoScale             "Scale"
#define VWINFODESC_woVideoScale           "Scale"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoRate,
    //! XML tag name for rate (fps = rate / scale)
#define VWINFOTAG_woVideoRate              "Rate"
#define VWINFODESC_woVideoRate            "Rate"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoStart,
    //! XML tag name for start frame
#define VWINFOTAG_woVideoStart             "Start"
#define VWINFODESC_woVideoStart           "Start"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoLength,
    //! XML tag name for the length in frames
#define VWINFOTAG_woVideoLength            "Length"
#define VWINFODESC_woVideoLength          "Length"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoInitialFrames,
    //! XML tag name for number of initial frames to load
#define VWINFOTAG_woVideoInitialFrames     "InitialFrames"
#define VWINFODESC_woVideoInitialFrames   "InitialFrames"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoSuggestedBufferSize,
    //! XML tag name for suggested maximum buffer size
#define VWINFOTAG_woVideoSuggestedBufferSize "SuggestedBufferSize"
#define VWINFODESC_woVideoSuggestedBufferSize "SuggestedBufferSize"

```

```

    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoQuality,
    //! XML tag name for quality
#define VWINFOTAG_woVideoQuality "Quality"
#define VWINFODESC_woVideoQuality "Quality"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoSampleSize,
    //! XML tag name for recommended sample size
#define VWINFOTAG_woVideoSampleSize "SampleSize"
#define VWINFODESC_woVideoSampleSize "SampleSize"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoEditCount,
    //! XML tag name for number of edits done on this file
#define VWINFOTAG_woVideoEditCount "EditCount"
#define VWINFODESC_woVideoEditCount "EditCount"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoFormatChangeCount,
    //! XML tag name for number of format changes
#define VWINFOTAG_woVideoFormatChangeCount "FormatChangeCount"
#define VWINFODESC_woVideoFormatChangeCount "FormatChangeCount"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoPitch,
    //! XML tag name for video line pitch
#define VWINFOTAG_woVideoPitch "Pitch"
#define VWINFODESC_woVideoPitch "Pitch"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoDrFlags,
    //! XML tag name for internal drastic flags
#define VWINFOTAG_woVideoDrFlags "DrFlags"
#define VWINFODESC_woVideoDrFlags "DrFlags"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoFileType,
    //! XML tag name for drastic 'mft' file type
#define VWINFOTAG_woVideoFileType "FileType"
#define VWINFODESC_woVideoFileType "FileType"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoResDrastic,
    //! XML tag name for reserved drastic array of DWORDs
#define VWINFOTAG_woVideoResDrastic "ResDrastic"
#define VWINFODESC_woVideoResDrastic "ResDrastic"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiAudioType,
    //! XML tag
#define VWINFOTAG_woAudioType "AudioType"
#define VWINFODESC_woAudioType "AudioType"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiAudioChannels,
    //! XML tag
#define VWINFOTAG_woAudioChannels "AudioChannels"
#define VWINFODESC_woAudioChannels "AudioChannels"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiAudioFrequency,
    //! XML tag

```

```
#define VWINFOTAG_woAudioFrequency "AudioFrequency"
#define VWINFODESC_woAudioFrequency "AudioFrequency"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiAudioBits,
    //! XML tag
#define VWINFOTAG_woAudioBits "AudioBits"
#define VWINFODESC_woAudioBits "AudioBits"
    //char szName[_VWXXX_NAME_SIZE]; // Stream identifier
    //RECT/*16*/ rcFrame; // Frame dimensions
    vwiLastElementPlus1
    // DO NOT ADD ANYTHING BELOW vwiLastElementPlus1
};
```

Direct Link Header

dtmediaread.h

```
/
*****
*****
*
*
* Copyright (c) 1998-2022 Drastic Technologies Ltd. All Rights Reserved.
* 523 The Queensway, Suite 201 Toronto ON M8Y 1J7
* phone (416) 255 5636 fax (416) 255 8780
* engineering@drastictech.com http://www.drastic.tv

*****
****/
// drmediaread.h : Declaration of the dtmediaread api

// Hacking class from activex control

#ifndef __DTMEDIAREAD_DRASTIC_API_9204jrewf348j4_H_
#define __DTMEDIAREAD_DRASTIC_API_9204jrewf348j4_H_

////////////////////////////////////

#define DTMRHANDLE void*

#ifdef _WIN32
#define DTMRCALLTYPE __stdcall
#include <windows.h>
#else
#define DTMRCALLTYPE
#include <stddef.h>
#endif

#ifdef __cplusplus
extern "C" { // PREVENT C++ NAME-MANGLING
#endif

/** The read types
 */
//! Windows RGBA 8 bits per component, 32 total (like bitmap, tga, etc)
const unsigned long DTMR_READTYPE_ARGB = 0;
//! RGB 8 bits per component, 24 total
const unsigned long DTMR_READTYPE_RGB = 0x10000000;
//! Alpha only 8 bits per component, repeated to 24
const unsigned long DTMR_READTYPE_AAA = 0x20000000;
//! 8 Bit YCbCr (yuv2, D1/HDSI raw 4:2:2 video
const unsigned long DTMR_READTYPE_UYVY = 1;
//! 10 Bit v210 (quicktime packing) 4:2:2 video
const unsigned long DTMR_READTYPE_V210 = 2;
//! 10 Bit RGB 4:4:4 (dpx packing)
const unsigned long DTMR_READTYPE_RGB10Bit = 3;
//! 16 bit per component (64 bit) RGBA 4:4:4:4
```



```

const unsigned long DTMR_READTYPE_RGBA64 = 4;
//! RGB 16 bits per component, 48 total
const unsigned long DTMR_READTYPE_RGB48 = 0x10000004;
//! Alpha only 16 bits per component, repeated to 48
const unsigned long DTMR_READTYPE_AAA16 = 0x20000004;
//! 16 bit half float per component RGBA (GPU)
const unsigned long DTMR_READTYPE_RGBAHALFFLOAT = 5;
//! 16 bit half float per component RGB (GPU)
const unsigned long DTMR_READTYPE_RGBHALFFLOAT = 6;
//! Set to invert the picture vertically
const unsigned long DTMR_READFLAG_FLIP = 0x80000000;
//! Invalid file
const unsigned long DTMR_READTYPE_INVALID = -1;
//! Set readtype to video frame size AUDIO to 16 bits LE
const unsigned long DTMR_READTYPE_FRAME_AUDIO_16LE = (0x00010000 | 16);
//! Set readtype to video frame size AUDIO to 32 bits (note, 16, 20, 24 will be shifted
to most significant, LE)
const unsigned long DTMR_READTYPE_FRAME_AUDIO_32LE = (0x00010000 | 32);
//! Set readtype to arbitrary sample AUDIO to 16 bits LE
const unsigned long DTMR_READTYPE_SAMPLE_AUDIO_16LE = (0x00110000 | 16);
//! Set readtype to arbitrary sample AUDIO to 32 bits (note, 16, 20, 24 will be shifted
to most significant, LE)
const unsigned long DTMR_READTYPE_SAMPLE_AUDIO_32LE = (0x00110000 | 32);

/** Open a new file, stream or network source for preview
 */
DTMRHANDLE DTMRCALLTYPE dtmrOpen(char * szFileName, unsigned long
dwFlags);
typedef DTMRHANDLE (DTMRCALLTYPE * p_dtmrOpen)(char * szFileName, unsigned
long dwFlags);

/** Special case: Open a video and array of audio files
 * szFileNameVAA[0] = video file name
 * szFileNameVAA[1] = first audio file
 * szFileNameVAA[n] = last audio file
 * szFileNameVAA[n+1] = NULL for rest of audio entries
 */
DTMRHANDLE DTMRCALLTYPE dtmrOpenMulti(char * szFileNameVAA[17], unsigned
long dwFlags);

/** Close the currently open stream or file
 */
long DTMRCALLTYPE dtmrClose(DTMRHANDLE dtmr);
typedef long (DTMRCALLTYPE * p_dtmrClose)(DTMRHANDLE dtmr);

/** Returns recommended and supported read types
 */
long DTMRCALLTYPE dtmrGetReadTypes(DTMRHANDLE dtmr, unsigned long
dwIndex, unsigned long * pdwTypes);
typedef long (DTMRCALLTYPE * p_dtmrGetReadTypes)(DTMRHANDLE dtmr, unsigned
long dwIndex, unsigned long * pdwTypes);

```

```

/** The final file name used for the source file
 */
long DTMRCALLTYPE dtmrSourceFileName(DTMRHANDLE dtmr, char *
tszMAX_PATHString);
typedef long (DTMRCALLTYPE * p_dtmrSourceFileName)(DTMRHANDLE dtmr, char *
tszMAX_PATHString);

/** Source video media's height
 */
long DTMRCALLTYPE dtmrSourceHeight(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceHeight)(DTMRHANDLE dtmr, long
*pVal);

/** Source video media's width
 */
long DTMRCALLTYPE dtmrSourceWidth(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceWidth)(DTMRHANDLE dtmr, long
*pVal);

/* Source video media's bit depth
 */
long DTMRCALLTYPE dtmrSourceBitDepth(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceBitDepth)(DTMRHANDLE dtmr, long
*pVal);

/* Source video media's fourcc compression code
 */
long DTMRCALLTYPE dtmrSourceFourCC(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceFourCC)(DTMRHANDLE dtmr, long
*pVal);

/* Source video media's bit rate
 */
long DTMRCALLTYPE dtmrSourceBitRate(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceBitRate)(DTMRHANDLE dtmr, long
*pVal);

/* Source video media's frame size for the requested or current frame
 */
long DTMRCALLTYPE dtmrSourceFrameSize(DTMRHANDLE dtmr, long dwFrame, long
*pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceFrameSize)(DTMRHANDLE dtmr, long
dwFrame, long *pVal);

/* Source video total channels
 */
long DTMRCALLTYPE dtmrSourceVideoChannels(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceVideoChannels)(DTMRHANDLE dtmr,
long *pVal);

/* Source audio total channels
 */
long DTMRCALLTYPE dtmrSourceAudioChannels(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceAudioChannels)(DTMRHANDLE dtmr,

```

```

long *pVal);

/** Source audio media frequency
 */
long DTMRCALLTYPE dtmrSourceAudioFrequency(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceAudioFrequency)(DTMRHANDLE dtmr,
long *pVal);

/** Source audio media bits per sample
 */
long DTMRCALLTYPE dtmrSourceAudioBitsPerSample(DTMRHANDLE dtmr, long
*pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceAudioBitsPerSample)(DTMRHANDLE
dtmr, long *pVal);

/* Source audio media's fourcc compression code
 */
long DTMRCALLTYPE dtmrSourceAudioFourCC(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceAudioFourCC)(DTMRHANDLE dtmr,
long *pVal);

/** Return the duration (total number of frames) of the media
 */
long DTMRCALLTYPE dtmrDuration(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrDuration)(DTMRHANDLE dtmr, long *pVal);

/** Return the audio duration (total number of audio samples) of the media
 */
long DTMRCALLTYPE dtmrAudioDuration(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrAudioDuration)(DTMRHANDLE dtmr, long
*pVal);

/** Source video rate value (FPS = SourceRate / SourceScale)
 */
long DTMRCALLTYPE dtmrSourceRate(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceRate)(DTMRHANDLE dtmr, long
*pVal);

/** Source video scale value (FPS = SourceRate / SourceScale)
 */
long DTMRCALLTYPE dtmrSourceScale(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceScale)(DTMRHANDLE dtmr, long
*pVal);

/** Return source metadata information that are numeric (DWORDs or longs)
 */
long DTMRCALLTYPE dtmrSourceMetaDataDWORD(DTMRHANDLE dtmr, long
dwMetaDataElement, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrSourceMetaDataDWORD)(DTMRHANDLE dtmr,
long dwMetaDataElement, long *pVal);

/** Return source metadata information that are string data
 */
long DTMRCALLTYPE dtmrSourceMetaDataSTR(DTMRHANDLE dtmr, long

```

```

dwMetaDataElement, char * szMAX_PATHString);
typedef long (DTMRCALLTYPE * p_dtmrSourceMetaDataSTR)(DTMRHANDLE dtmr,
long dwMetaDataElement, char * szMAX_PATHString);

/** Set the read type for the video frames
 */
long DTMRCALLTYPE dtmrSetReadType(DTMRHANDLE dtmr, long IReadType);
typedef long (DTMRCALLTYPE * p_dtmrSetReadType)(DTMRHANDLE dtmr, long
IReadType);

/** Get the current absolute (zero based) Frame
 */
long DTMRCALLTYPE dtmrGetFrame(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrGetFrame)(DTMRHANDLE dtmr, long *pVal);

/** Set the current absolute (zero based) Frame
 */
long DTMRCALLTYPE dtmrSetFrame(DTMRHANDLE dtmr, long newVal);
typedef long (DTMRCALLTYPE * p_dtmrSetFrame)(DTMRHANDLE dtmr, long newVal);

/** Set the channel for the video frames (0, 1, 2, 3, 4 etc) (0 = 0x03, 1 = 0x0C, 2 =
0x30, 3 = 0xC0 etc.)
 */
long DTMRCALLTYPE dtmrSetVideoChannel(DTMRHANDLE dtmr, long IVideoChannel);
typedef long (DTMRCALLTYPE * p_dtmrSetVideoChannel)(DTMRHANDLE dtmr, long
IVideoChannel);

/** Set the audio channel pair to monitor (0 = 1+2, 1 = 3+4, 2 = 5+6, 3 = 7+8
etc.)
 */
long DTMRCALLTYPE dtmrSetAudioChannelPair(DTMRHANDLE dtmr, long
IAudioChannelPair);
typedef long (DTMRCALLTYPE * p_dtmrSetAudioChannelPair)(DTMRHANDLE dtmr,
long IAudioChannelPair);

/** Return the last GetVideoFrame VITC (vertical blank) time code
 */
long DTMRCALLTYPE dtmrLastVitcFrame(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrLastVitcFrame)(DTMRHANDLE dtmr, long
*pVal);

/** Return the last GetVideoFrame VITC (vertical blank time code) user bits
 */
long DTMRCALLTYPE dtmrLastVitcUb(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrLastVitcUb)(DTMRHANDLE dtmr, long *pVal);

/** Return the last GetVideoFrame LTC (SMPTE) time code
 */
long DTMRCALLTYPE dtmrLastLtcFrame(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrLastLtcFrame)(DTMRHANDLE dtmr, long
*pVal);

/** Return the last GetVideoFrame LTC (SMPTE time code) user bits
 */

```

```

long DTMRCALLTYPE dtmrLastLtcUb(DTMRHANDLE dtmr, long *pVal);
typedef long (DTMRCALLTYPE * p_dtmrLastLtcUb)(DTMRHANDLE dtmr, long *pVal);

/** GetVideoFrame returns a safe array containing one video frame
 */
long DTMRCALLTYPE dtmrGetVideoFrame(DTMRHANDLE dtmr, unsigned char *
psvFrame, long * pISize);
typedef long (DTMRCALLTYPE * p_dtmrGetVideoFrame)(DTMRHANDLE dtmr,
unsigned char * psvFrame, long * pISize);

/** GetAudioFrame returns a safe array containing one video frame worth of audio
data
 */
long DTMRCALLTYPE dtmrGetAudioFrame(DTMRHANDLE dtmr, unsigned char *
psaFrame, long * pISize);
typedef long (DTMRCALLTYPE * p_dtmrGetAudioFrame)(DTMRHANDLE dtmr,
unsigned char * psaFrame, long * pISize);

/** Get current extended data
 */
long DTMRCALLTYPE dtmrGetCurExtendedData(DTMRHANDLE dtmr, unsigned char
*pvData, unsigned long * pIFlags, long *pISize);
typedef long (DTMRCALLTYPE * p_dtmrGetCurExtendedData)(DTMRHANDLE dtmr,
unsigned char *pvData, unsigned long * pIFlags, long *pISize);

//! Data is EIA-608B SD closed caption data field one (uses 2 bytes)
#define FRAMEINFO_DATA_F1_EIA608          0x00000001
//! Data is EIA-608B SD closed caption data field two (uses 2 bytes)
#define FRAMEINFO_DATA_F2_EIA608          0x00000002
//! Data is EIA-708 HD closed caption data (uses remaining bytes = minus the
above)
#define FRAMEINFO_DATA_EIA708              0x00000100

/** Get current closed captions (from last video frame loaded) including size and
flags
 */
long DTMRCALLTYPE dtmrGetCurClosedCaptions(DTMRHANDLE dtmr, unsigned char
*pvCC, long *pICCSize, long * pICCFlags);
typedef long (DTMRCALLTYPE * p_dtmrGetCurClosedCaptions)(DTMRHANDLE dtmr,
unsigned char *pvCC, long *pICCSize, long * pICCFlags);

/** Advanced - send/return a mediacmd structure
 */
long DTMRCALLTYPE dtmrSetMode(DTMRHANDLE dtmr, void * pMediCmd);
typedef long (DTMRCALLTYPE * p_dtmrSetMode)(DTMRHANDLE dtmr, void *
pMediCmd);

// dwFlags
    //! Send this in if you just need the filename (faster than getting all the info)
#define DPOSSIZENAME_FILENAME_ONLY          0x40000000    //
Same as DFRAME_SKIP_FRAME
    //! Flag for mediafile/avhal to get audio dframe
#define GetAudio    0x00000000
    //! Flag for mediafile/avhal to get video dframe

```

```

#define GetVideo    0x00000001

// dwFrameFlags
#define DPOSSIZENAME_VIDEO_FRAME    0x00000001
    //! Is this file type currently recording
#define DPOSSIZENAME_RECORDING    0x00000004
    //! This frame needs to be made black (default frame) in MediaFile
#define DPOSSIZENAME_PLEASE_BLACK    _PDFRAMEFLAGS_PLEASE_BLACK
    //    0x00000080
    //! This is a mono audio chunk
#define DPOSSIZENAME_MONO_AUDIO_FRAME    0x00000100
    //! This is a stereo audio chunk
#define DPOSSIZENAME_STEREO_AUDIO_FRAME    0x00000200
#define DPOSSIZENAME_QUAD_AUDIO_FRAME    0x00000400
#define DPOSSIZENAME_4_1_AUDIO_FRAME    0x00000800
#define DPOSSIZENAME_5_1_AUDIO_FRAME    0x00001000
#define DPOSSIZENAME_7_1_AUDIO_FRAME    0x00002000
#define DPOSSIZENAME_9_1_AUDIO_FRAME    0x00004000
#define DPOSSIZENAME_AUDIO_MASK
(DPOSSIZENAME_MONO_AUDIO_FRAME|DPOSSIZENAME_STEREO_AUDIO_FRAME|
DPOSSIZENAME_STEREO_AUDIO_FRAME|DPOSSIZENAME_QUAD_AUDIO_FRAME|
DPOSSIZENAME_4_1_AUDIO_FRAME|DPOSSIZENAME_5_1_AUDIO_FRAME|
DPOSSIZENAME_7_1_AUDIO_FRAME|DPOSSIZENAME_9_1_AUDIO_FRAME)
#define DPOSSIZENAME_FRAME_MASK    0x0000FFFF
    //! This frame contains audio data see DFRAME::dwType
#define DFRAME_TYPE_AUDIO    0x00010000
    //! 16 bit audio
#define DPOSSIZENAME_AUD_16_16_BIT    0x00100000
    //! 20 bit audio in 24
#define DPOSSIZENAME_AUD_20_24_BIT    0x00200000
    //! 24 bit audio in 24
#define DPOSSIZENAME_AUD_24_24_BIT    0x00400000
    //! 24/32 bit audio in 32
#define DPOSSIZENAME_AUD_24_32_BIT    0x00800000
    //! 32/32 bit audio in 32
#define DPOSSIZENAME_AUD_32_32_BIT    0x01000000
    //! Audio is compressed
#define DPOSSIZENAME_AUD_COMPRESSED    0x02000000
    //! Audio is big endian, else little endian
#define DPOSSIZENAME_AUD_BIGENDIAN_BIT    0x00080000
    //! Just for completeness
#define DPOSSIZENAME_AUD_LITTLEENDIAN_BIT    0x00000000
    //! This frame is independent of other frames for decode see
DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME    0x10000000
    //! This frame is independent of other frames for decode (an MPEG I Frame)
see DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME_I    0x10000000
    //! This frame requires previous keyframe(s) (for MPEG a P Frame) see
DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME_P    0x80000000
    //! This frame requires more than one frame to decode (for MPEG a B Frame)
see DFRAME::dwType
#define DFRAME_TYPE_KEYFRAME_B    0x20000000

```

```
        //!< This frame should be skipped (decoded, but not displayed) - Used to reach
        seek frame on a non key frame from key frame see DFRAME::dwType
#define DFRAME_SKIP_FRAME            0x40000000
```

```
/** Get info on a frame of audio or video
 */
long DTMRCALLTYPE dtmrGetFileFrameInfo(DTMRHANDLE dtmr, unsigned long
dwFrame, unsigned long dwChannels, unsigned long dwFlags,
                                         size_t *
pnPosition, size_t * pnSize, unsigned long * pdwFrameFlags,
                                         char *
szFilePathAndName);
typedef long (DTMRCALLTYPE * p_dtmrGetFileFrameInfo)(DTMRHANDLE dtmr,
unsigned long dwFrame, unsigned long dwChannels, unsigned long dwFlags,
                                         size_t *
pnPosition, size_t * pnSize, unsigned long * pdwFrameFlags,
                                         char *
szFilePathAndName);

#ifdef __cplusplus
}          // PREVENT C++ NAME-MANGLING
#endif

////////////////////////////////////

#endif // __DTMEDIAREAD_DRASTIC_API_9204jrewf348j4_H_
```