

DTMediaWrite Programmers Interface



Copyright 2010-2015 Drastic Technologies Ltd.
All Rights Reserved



www.drastic.tv

Table of Contents

DTMediaWrite Programmers Interface.....	1
Introduction.....	5
Direct Link Usage.....	5
Methods and Properties.....	6
dtmwOpen.....	6
dtmwClose.....	6
dtmwGetWriteTypes.....	6
dtmwTargetFileName.....	7
dtmwTargetHeight.....	7
dtmwTargetWidth.....	7
dtmwTargetPitch.....	7
dtmwTargetBitDepth.....	7
dtmwTargetFourCC.....	7
dtmwTargetBitRate.....	7
dtmwTargetQuality.....	8
dtmwTargetFrameSize.....	8
dtmwTargetVideoChannels.....	8
dtmwTargetAudioChannels.....	8
dtmwTargetAudioFrequency.....	8
dtmwTargetAudioBitsPerSample.....	8
dtmwTargetAudioFourCC.....	8
dtmwTargetRate.....	9
dtmwTargetScale.....	9
dtmwTargetMetaDataDWORD.....	9
dtmwTargetMetaDataSTR.....	9
dtmwSetWriteType.....	9
dtmwSetVideoChannel.....	9
dtmwSetAudioChannelPair.....	10
dtmwSetVitcType.....	10
dtmwNextVitcFrame.....	10
dtmwNextVitcUb.....	10
dtmwSetLtcType.....	10
dtmwNextLtcFrame.....	10
dtmwNextLtcUb.....	11
dtmwPutNextExtendedData.....	11
dtmwPutVideoFrame.....	11
dtmwPutAudioFrame.....	11
dtmwSetFileFrameInfo.....	11
Defines And Constants.....	14
Output File Formats.....	15
Windows Wave (audio only).....	15
QuickTime Movie.....	15
Windows AVI.....	15
Targa Files.....	15

TIFF Files.....	15
YUV Files.....	15
HDR YUV Raw Stream.....	15
WAV Extensible (audio only).....	15
AIFF (audio only).....	15
MXF Sony SD IMX.....	15
DPX Files.....	15
WAV Broadcast Wave Format (audio only).....	15
Sony XDCam 4:2:0 (Old XDCam).....	15
Panasonic P2 DV MXF.....	15
Avid OP-Atom MXF.....	15
Panasonic P2 AVCi MXF.....	15
DCP/DCI MXF.....	15
OP1a MXF.....	15
Sony HDCam MXF.....	16
Sony XDCam 4:2:2 50 Mbs.....	16
EasyDCP/DCI MXF.....	16
MPEG-4 h.264.....	16
AS-02 MXF.....	16
Compression Types.....	17
DTWAVE_FORMAT_PCM (Up to stereo little endian).....	17
DTWAVE_FORMAT_EXTENSIBLE (Multi channel audio little endian)	
.....	17
dtmwfck16BitBigEndianFormat (Mov/Aiff big endian audio)..	17
dtmwfccdv25.....	17
dtmwfccdv50.....	17
dtmwfccdvh.....	17
dtmwfckYCbCr8Bit.....	17
dtmwfckYCbCr10Bit.....	17
dtmwfccCineForm.....	17
dtmwBI_RGB.....	17
dtmwfckIMXD10_NTSC_50.....	17
dtmwfckIMXD10_NTSC_40.....	17
dtmwfckIMXD10_NTSC_30.....	17
dtmwfckIMXD10_PAL_50.....	17
dtmwfckIMXD10_PAL_40.....	17
dtmwfckIMXD10_PAL_30.....	17
dtmwfcc10LinDPX.....	17
dtmwfcc10LogDPX.....	17
dtmwfccDT_MPEGHD_VBR_I.....	17
dtmwfccDT_MPEGHD_VBR_P.....	17
dtmwfccDT_MPEGHD_VBR_I_17.....	18
dtmwfccDT_MPEGHD_VBR_P_17.....	18
dtmwfccDT_MPEGHD_VBR_I_25.....	18
dtmwfccDT_MPEGHD_VBR_P_25.....	18
dtmwfccDT_MPEGHD_VBR_I_35.....	18
dtmwfccDT_MPEGHD_VBR_P_35.....	18

dtmwfccDT_MPEGHD_CBR_I.....	18
dtmwfccDT_MPEGHD_CBR_P.....	18
drmwfcckH264CodecType.....	18
dtmwfcckDNxHD_220x_10.....	18
dtmwfcckDNxHD_145x.....	18
dtmwfcckDNxHD_220x.....	18
dtmwfcckDNxHD_220_10.....	18
dtmwfcckDNxHD_145.....	18
dtmwfcckDNxHD_220.....	18
dtmwfcckDNxHD_720_220x.....	18
dtmwfcckDNxHD_720_220.....	18
dtmwfcckDNxHD_720_145.....	18
dtmwfcckDNxHD_36.....	18
dtmwfccAVCi100.....	18
dtmwfccJ2_Cinema2K.....	18
dtmwfccJ2_Cinema4K.....	19
fccJPEG2000_YCbCr.....	19
dtmwfccHDCamSR.....	19
dtmwfccHDCamSR_444.....	19
dtmwfccDT_MPEG422.....	19
dtmwfcckXAVC.....	19
dtmwfcckXAVC4K.....	19
Output Video Formats.....	20
ARGB 32 (8 bits per component, vertical invert).....	21
RGB 30 (10 bits per component).....	21
YCrCb 8 (8 bits per component 4:2:2).....	21
YCrCb 10 (10 bits per component 4:2:2).....	21
Output Audio Formats.....	23
Example - Direct.....	24
Metadata Elements.....	25
Direct Link Header.....	33

Introduction

The DTMediaWrite interface is designed to give programmers a simple yet powerful access to Drastic's main write formats. This document describes the various methods and properties exported by DTMediaWrite.

The DTMediaWrite API is available as a direct link library under Windows 32, Windows 64, Mac 32/64 and Linux 64. With the properties and functions under direct link, all the names are preceded by 'dtmw' to avoid namespace collisions.

Direct Link Usage

All the functions in the direct link model have 'dtmw' prepended to the function name. This means the 'PutVideoFrame' becomes 'dtmrPutVideoFrame' 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 "dtmediawrite.h" in your source file, and link to "libdtmediawrite.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 dtmw_loader.cpp that has all the functions as point, and a load/unloader function for your convenience.

Methods and Properties

dtmwOpen

*DTMRHANDLE DTMRCALLTYPE dtmwOpen(char * szFileName, unsigned long dwFlags, unsigned long dwFileType, unsigned long dwFourCC, unsigned long dwWidth, unsigned long dwHeight, unsigned long dwRate, unsigned long dwScale, unsigned long dwAudioChannels, unsigned long dwAudioRate, unsigned long dwAudioBits);*

Open a new file, stream or network source for preview. The szFileName is a UTF-8 string (converted by DTMediaWrite to Unicode for Windows). All of the basic requirements for the file to be created are sent at this point

- szFileName – UTF-8 file path and name
- dwFlags – Read/Write flags
- dwFileType – Exact writer requested
- dwFourCC – Video compression four character code
- dwWidth – Width for new file
- dwHeight – Height for new file
- dwRate – Rate part of frames per second (24, 25, 30000, etc)
- dwScale – Scale part of the frames per second (1, 1, 1001, etc)
- dwAudioChannels – Number of audio channels to write as a bitwise array
- dwAudioRate – Audio sample rate
- dwAudioBits – Audio bits per sample size
- @return – an opaque handle to use with the rest of the API functions

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

NOTE: For audio, 16 and 24 bits are the most common. When writing, there are only two container sizes: 16 bits for 16, and 32 bits for 20, 24 and 32 bits. The samples are always shifted to the most significant bits.

dtmwClose

long DTMRCALLTYPE dtmwClose(DTMRHANDLE dtmw);

Close the currently open stream or file

dtmwGetWriteTypes

*long DTMRCALLTYPE dtmwGetWriteTypes(DTMRHANDLE dtmw, unsigned long dwIndex, unsigned long * pdwTypes);*

Returns recommended and supported write types.

dtmwTargetFileName

*long DTMRCALLTYPE dtmwTargetFileName(DTMRHANDLE dtmw, char * pszString);*

The final file name used for the target file.

dtmwTargetHeight

*long DTMRCALLTYPE dtmwTargetHeight(DTMRHANDLE dtmw, long *pVal);*

Target video media's height.

dtmwTargetWidth

*long DTMRCALLTYPE dtmwTargetWidth(DTMRHANDLE dtmw, long *pVal);*

Target video media's width.

dtmwTargetPitch

*long DTMRCALLTYPE dtmwTargetPitch(DTMRHANDLE dtmwPV, long lType, long *pVal);*

Target pitch depending on frame type

dtmwTargetBitDepth

*long DTMRCALLTYPE dtmwTargetBitDepth(DTMRHANDLE dtmw, long *pVal);*

Target video media's bit depth

dtmwTargetFourCC

*long DTMRCALLTYPE dtmwTargetFourCC(DTMRHANDLE dtmw, long *pVal);*

Target video media's fourcc compression code

dtmwTargetBitRate

long DTMRCALLTYPE dtmwTargetBitRate(DTMRHANDLE dtmw, long

**pVal);*

Target video media's bit rate in kilo bits per seconds (e.g. 4000 = 4 megabits). Setting this will disable any quality settings. This call must be made before the **dtmwSetWriteType()** function is called.

dtmwTargetQuality

*long DTMRCALLTYPE dtmwTargetQuality(DTMRHANDLE dtmw, long *pVal);*

Target video media's quality. This is a value between 0 and 10,000, with 0 being the lowest possible quality. Setting this will disable any data rate settings. This call must be made before the **dtmwSetWriteType()** function is called.

dtmwTargetFrameSize

*long DTMRCALLTYPE dtmwTargetFrameSize(DTMRHANDLE dtmw, long dwFrameType, long *pVal);*

Target video media's frame size for the requested or current frame.

dtmwTargetVideoChannels

*long DTMRCALLTYPE dtmwTargetVideoChannels(DTMRHANDLE dtmw, long *pVal);*

Target video total channels.

dtmwTargetAudioChannels

*long DTMRCALLTYPE dtmwTargetAudioChannels(DTMRHANDLE dtmw, long *pVal);*

Target audio total channels.

dtmwTargetAudioFrequency

*long DTMRCALLTYPE dtmwTargetAudioFrequency(DTMRHANDLE dtmw, long *pVal);*

Target audio media frequency.

dtmwTargetAudioBitsPerSample

*long DTMRCALLTYPE dtmwTargetAudioBitsPerSample(DTMRHANDLE dtmw, long *pVal);*

Target audio media bits per sample.

dtmwTargetAudioFourCC

*long DTMRCALLTYPE dtmwTargetAudioFourCC(DTMRHANDLE dtmw, long *pVal);*

Target audio media's fourcc compression code.

dtmwTargetRate

*long DTMRCALLTYPE dtmwTargetRate(DTMRHANDLE dtmw, long *pVal);*

Target video rate value (FPS = TargetRate / TargetScale).

dtmwTargetScale

*long DTMRCALLTYPE dtmwTargetScale(DTMRHANDLE dtmw, long *pVal);*

Target video scale value (FPS = TargetRate / TargetScale).

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

dtmwTargetMetaDataDWORD

long DTMRCALLTYPE dtmwTargetMetaDataDWORD(DTMRHANDLE dtmw, long dwMetaDataElement, long dwVal);

Return Target metadata information that are numeric (DWORDs or longs). Works for vvwTimeCode to vvwWhiteBalance inclusive, and vvwVideoWidth to vvwAudioBits inclusive.

dtmwTargetMetaDataSTR

*long DTMRCALLTYPE dtmwTargetMetaDataSTR(DTMRHANDLE dtmw, long dwMetaDataElement, char * szMAX_PATHString);*

Return Target metadata information that are string data. Works for vvwFileName to vvwUMID inclusive.

dtmwSetWriteType

long DTMRCALLTYPE dtmwSetWriteType(DTMRHANDLE dtmw, long IWriteType);

Set the write type for the video frames.

dtmwSetVideoChannel

long DTMRCALLTYPE dtmwSetVideoChannel(DTMRHANDLE dtmw, long IVideoChannel);

Set the channel for the video frames (0, 1, 2, 3, 4 etc) (0 = 0x03, 1 = 0x0C, 2 = 0x30, 3 = 0xC0 etc.).

dtmwSetAudioChannelPair

long DTMRCALLTYPE dtmwSetAudioChannelPair(DTMRHANDLE dtmw, long IAudioChannelPair);

Set the audio channel pair to monitor (0 = 1+2, 1 = 3+4, 2 = 5+6, 3 = 7+8 etc.).

dtmwSetVitcType

long DTMRCALLTYPE dtmwSetVitcType(DTMRHANDLE dtmw, long dwVal);

Set the VITC (vertical blank) time code's type. The types are TC2_TCTYPE_FILM, TC2_TCTYPE_NDF, TC2_TCTYPE_PAL, TC2_TCTYPE_50, TC2_TCTYPE_5994, TC2_TCTYPE_60, TC2_TCTYPE_NTSCFILM, and TC2_TCTYPE_IRIG.

dtmwNextVitcFrame

long DTMRCALLTYPE dtmwNextVitcFrame(DTMRHANDLE dtmw, long dwVal);

Sets the next frames VITC (vertical blank) time code.

dtmwNextVitcUb

long DTMRCALLTYPE dtmwNextVitcUb(DTMRHANDLE dtmw, long dwVal);

Sets the next VITC (vertical blank time code) user bits.

dtmwSetLtcType

long DTMRCALLTYPE dtmwSetLtcType(DTMRHANDLE dtmw, long dwVal);

Set the VITC (vertical blank) time code's type. The types are TC2_TCTYPE_FILM, TC2_TCTYPE_NDF, TC2_TCTYPE_PAL, TC2_TCTYPE_50, TC2_TCTYPE_5994, TC2_TCTYPE_60, TC2_TCTYPE_NTSCFILM, and TC2_TCTYPE_IRIG.

dtmwNextLtcFrame

long DTMRCALLTYPE dtmwNextLtcFrame(DTMRHANDLE dtmw, long dwVal);

Sets the next LTC (SMPTE) time code.

dtmwNextLtcUb

long DTMRCALLTYPE dtmwNextLtcUb(DTMRHANDLE dtmw, long dwVal);

Sets the next LTC (SMPTE time code) user bits.

dtmwPutNextExtendedData

*long DTMRCALLTYPE dtmwPutNextExtendedData(DTMRHANDLE dtmw, unsigned char *pvData, long ISize, long IFlags);*

Set the next extended data. Normally both these calls set some combination of close captions. The first two bytes are CC1/CC3 if the flag is set, otherwise they are undefined. The second two bytes are CC2/CC4 if the flag is set, otherwise they are undefined. Everything from byte 4 on are 708/SMPTE 436 packets of closed captions, active format description and v-chip IDs, including the startup and line numbers.

```
//! 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
```

dtmwPutVideoFrame

*long DTMRCALLTYPE dtmwPutVideoFrame(DTMRHANDLE dtmw, unsigned char *psvFrame, long dwSize);*

Sends one video frame. The format must match the format set by the write type. Please note, the video buffer is not guaranteed to be the same on function return. It is used directly by the writer, and will likely be changed during the write, so it must be a new redrawn/captured video frame on each call.

dtmwPutAudioFrame

*long DTMRCALLTYPE dtmwPutAudioFrame(DTMRHANDLE dtmw, unsigned char * psaFrame, long dwSize);*

Returns a safe array containing one video frame worth of audio data (if in video size mode) or an arbitrary amount of audio samples of size bytes (if in audio mode).

dtmwSetMode

*long DTMWCALLTYPE dtmwSetMode(DTMWHANDLE dtmwPV, void * pMediaCmd);*

Send custom MEDIACMD commands to the file writer.

dtmwVersion

*long DTMWCALLTYPE dtmwVersion(long *pVerMajor, long *pVerMinor, long *pVerMod, long *pVerBuild);*

Returns the version information for the writer build.

dtmwAddVideoChannel

*long DTMWCALLTYPE dtmwAddVideoChannel(DTMWHANDLE dtmwPV, char * szVideoFile, unsigned long dwFileType, unsigned long dwFourCC, unsigned long dwWidth, unsigned long dwHeight, unsigned long dwRate, unsigned long dwScale, unsigned long * pdwVideoChannelHandle);*

Add a video channel to the RTIN file.

dtmwCodecData

*long DTMWCALLTYPE dtmwCodecData(DTMWHANDLE dtmw, unsigned char * pData, unsigned long dwSize);*

Set extended codec data for an video channel. Should be called before dtmwAddVideoChannel

dtmwAddAudioChannel

*long DTMWCALLTYPE dtmwAddAudioChannel(DTMWHANDLE dtmwPV, char * szAudioFile, unsigned long dwFileType, unsigned long dwAudioChannels, unsigned long dwAudioRate, unsigned long dwAudioBits, unsigned long * pdwAudioChannelHandle);*

Add an audio channel to the RTIN file.

dtmwAudioCodecData

long DTMWCALLTYPE dtmwAudioCodecData(DTMWHANDLE dtmw, unsigned char * pData, unsigned long dwSize);
Set extended codec data for an audio channel. Should be called before dtmwAddAudioChannel

dtmwSetFileFrameInfo

long DTMWCALLTYPE dtmwPutFileFrameInfo(DTMWHANDLE dtmwPV, unsigned long dwRTChannel, unsigned long dwFrame, unsigned long dwFlags, size_t nPosition, size_t nSize, unsigned long dwFrameFlags, unsigned long dwRepsSamples);

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.

//! 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

```


Defines And Constants

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

```
/** The write video frame types
 */
//! Windows RGBA (like bitmap, tga, etc)
const long DTMR_WRITETYPE_ARGB = 0;
//! 8 Bit YCbCr (yuv2, D1/HDSI raw 4:2:2 video)
const long DTMR_WRITETYPE_UYVY = 1;
//! 10 Bit v210 (quicktime packing) 4:2:2 video
const long DTMR_WRITETYPE_V210 = 2;
//! 10 Bit RGB 4:4:4 (dpx packing)
const long DTMR_WRITETYPE_RGB10Bit = 3;
//! 16 bit per component (64 bit) RGBA 4:4:4:4
const long DTMR_WRITETYPE_RGBA64 = 4;
//! 16 bit half float per component RGBA (GPU)
const long DTMR_WRITETYPE_RGBAHALFFLOAT = 5;
//! Returned if there are no more suggested types
const long DTMR_WRITETYPE_INVALID = -1;

//! Set readtype AUDIO to 16 bits LE
const unsigned long DTMR_WRITETYPE_FRAME_AUDIO_16LE =
(0x00010000 | 16);
//! Set readtype AUDIO to 32 bits (note, 16, 20, 24 will be shifted to
most significant, LE)
const unsigned long DTMR_WRITETYPE_FRAME_AUDIO_32LE =
(0x00010000 | 32);
//! Invalid file
const long DTMR_WRITETYPE_INVALID = -1;
```


Output File Formats

RtIndex (RTIN) – Special

dtmftrtIndex = 172 // Special case, write and RTIN directly

Windows Wave (audio only)

dtmwWave = 1, // Windows WAV audio files (audio)

QuickTime Movie

dtmwMov = 2, // Quicktime movie files (audio video info)

Windows AVI

dtmwAvi = 3, // Video for windows, Audio Video Interleave (audio video info)

Targa Files

dtmwLiveTga = 99, // 32 Bit Uncompressed only

TIFF Files

dtmwLiveTiff = 101, // 32 Bit Uncompressed only

YUV Files

dtmwLiveYuv = 104, // 8/10 Bit Uncompressed YCbCr only

HDR YUV Raw Stream

DtmwHdrYuv = 106, //

WAV Extensible (audio only)

dtmwAdvWave = 107, // Windows WAVE format extension (multi channel) audio plugin (no dual mono)

AIFF (audio only)

dtmwAdvAiff = 108, // Apple/SGI format multi channel audio plugin

MXF Sony SD IMX

dtmwMXFSonySD = 110, // Sony IMX MPEG SD

DPX Files

dtmwLiveDpx = 111, // RGB10 or YCBCR10

WAV Broadcast Wave Format (audio only)

dtmwBWaveF = 117, // Broadcast wave format

Sony XDCam 4:2:0 (Old XDCam)

dtmwMXFSonyHD = 127, // Sony 25/35mbit 4:2:0 XDCam (old XDCam)

Panasonic P2 DV MXF

dtmwMXFP2DV = 134, // Panasonic P2 DV25/50/HD

Avid OP-Atom MXF

dtmwMXFAvid = 135, // Avid OPAtom direct to mediafiles

Panasonic P2 AVCi MXF

dtmwMXFP2AVCi = 163, // Panasonic AVCi 100/50 writer

DCP/DCI MXF

dtmwMXFDCP = 167, // Unencrypted DCP

OP1a MXF

dtmwMXFOP1a = 172, // Op1a - yuv, j2k, avci, dvhd

Sony HDCam MXF

dtmwMXFSMDK = 186, // Sony HDCam MXF

Sony XDCam 4:2:2 50 Mbs

dtmwMXFSony422 = 192, // Sony XDCam 4:2:2 50

Mbit

EasyDCP/DCI MXF

dtmwMXFEasyDCP = 196, // Encrypted DCP (requires
EasyDCP license)

MPEG-4 h.264

dtmwMP4 = 197, // MP4 with 264 compression

AS-02 MXF

dtmwMXFAS02 = 201, // MXF AS-02

Compression Types

DTWAVE_FORMAT_PCM (Up to stereo little endian)

DTWAVE_FORMAT_EXTENSIBLE (Multi channel audio little endian)

dtmwfck16BitBigEndianFormat (Mov/Aiff big endian audio)

Sent as PCM little endian 16 or 32 bits per channel, stereo pairs

dtmwfccdv25

DV-25 4:2:0

dtmwfccdv50

DV-50 4:2:2

dtmwfccdvhd

DV-100/DVHD

dtmwfccYCbCr8Bit

yuv2/uyvy 8 bit YbCr

dtmwfccYCbCr10Bit

V210 10 bit YCbCr

dtmwfccCineForm

CineForm lossless/lossy codec

dtmwBI_RGB

ABGR 32 bit (8 bits per component)

dtmwfccIMXD10_NTSC_50

50 Mbit NTSC IMX MPEG

dtmwfccIMXD10_NTSC_40

40 Mbit NTSC IMX MPEG

dtmwfccIMXD10_NTSC_30

30 Mbit NTSC IMX MPEG

dtmwfccIMXD10_PAL_50

50 Mbit PAL IMX MPEG

dtmwfccIMXD10_PAL_40

40 Mbit PAL IMX MPEG

dtmwfccIMXD10_PAL_30

30 Mbit PAL IMX MPEG

dtmwfcc10LinDPX

Big endian

dtmwfcc10LogDPX

Big endian

dtmwfccDT_MPEGHD_VBR_I

4:2:0 XDCAM HD VBR Interlace

dtmwfccDT_MPEGHD_VBR_P

4:2:0 XDCAM HD VBR Progressive

dtmwfccDT_MPEGHD_VBR_I_17
4:2:0 XDCAM HD VBR Interlace 17.5 Mbps

dtmwfccDT_MPEGHD_VBR_P_17
4:2:0 XDCAM HD VBR Progressive 17.5 Mbps

dtmwfccDT_MPEGHD_VBR_I_25
4:2:0 XDCAM HD VBR Interlace 25 Mbps

dtmwfccDT_MPEGHD_VBR_P_25
4:2:0 XDCAM HD VBR Progressive 25 Mbps

dtmwfccDT_MPEGHD_VBR_I_35
4:2:0 XDCAM HD VBR Interlace 35 Mbps

dtmwfccDT_MPEGHD_VBR_P_35
4:2:0 XDCAM HD VBR Progressive 35 Mbps

dtmwfccDT_MPEGHD_CBR_I
4:2:0 XDCAM HD CBR Interlace 25 Mbps

dtmwfccDT_MPEGHD_CBR_P
4:2:0 XDCAM HD CBR Progressive 25 Mbps

drmwfcckH264CodecType
AVC1 H.264 bitstream without start codes.

dtmwfcckDNxHD_220x_10
1920x1080 10 Bit P (220x/185x/175x)

dtmwfcckDNxHD_145x
1920x1080 8 Bit P (145/120/115) ~equiv hdcam/dvcpro100

dtmwfcckDNxHD_220x
1920x1080 8 Bit P (220/185/175)

dtmwfcckDNxHD_220_10
1920x1080 10 Bit i (220/185/175)

dtmwfcckDNxHD_145
1920x1080 8 Bit i (145/120/115)

dtmwfcckDNxHD_220
1920x1080 8 Bit i (220/185/175)

dtmwfcckDNxHD_720_220x
1280x720 10 Bit P (220x/175x/90x)

dtmwfcckDNxHD_720_220
1280x720 8 Bit P (220x/175x/90x)

dtmwfcckDNxHD_720_145
1280x720 8 Bit P (145x/120x/115x)

dtmwfcckDNxHD_36
1920x1080 8 Bit P (36)

dtmwfccAVCi100
Panasonic AVCi-100

dtmwfccJ2_Cinema2K
Digital cinema 2K (alias)

dtmwfccJ2_Cinema4K

Digital cinema 4K (alias)

fccJPEG2000_YCbCr

SAMA/YCbCrJ2K/Grass Valley Infinity

dtmwfccHDCamSR

HDCam SR 4:2:2 10 bit

dtmwfccHDCamSR_444

HDCam SR 4:4:4

dtmwfccDT_MPEG422

4:2:2 MPEG-2 50 Mbit

dtmwfcckXAVC

Sony XAVC 100 HD

dtmwfcckXAVC4K

Sony XAVC 100 4K

Output Video Formats

These are the formats supported by dtmwSetVideoFrame(). Each of these formats only appears as specified here for this return. The dtmwSourceXXX series of methods (including dtmwSourceBitDepth 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 dtmeSetReadType(). For each file opened, the dtmwGetReadTypes() 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 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		Red		Green		Red																	
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

Output Audio Formats

This is the format supported by GetAudioFrame().

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 dtmwSourceAudioFrequency return. The bit size is dependent on dtmwSourceAudioBitsPerSample. If the dtmwSourceAudioBitsPerSample 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. Alternately, the incoming bit size may be forced by setting dtmrSetWriteType to either DTMR_WRITETYPE_FRAME_AUDIO_16LE or DTMR_WRITETYPE_FRAME_AUDIO_32LE.

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 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.

Examples

Please see the sample code in
samples/simplifiedtmediawrite
samples/simplifiedtmediawritemulti

And for RTIN generation, please see
samples/raw2rtin

The sample media require for raw2rtin can be downloaded from:

http://www.drastic.tv/downloads/test_media/rtin-test-seq.zip

Metadata Elements

The functions SourceMetaDataDWORD() and SourceMetaDataSTR() use the defines below to return specific metadata from the file. The first enums are string values for SourceMetaDataSTR() (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  
    vvwStatus,  
    /** see VVWINFO::szAuthor  
    vvwAuthor, // <-- 20
```

```
!! see VVWINFO::szRevisionNumber
vwwiRevisionNumber,
!! see VVWINFO::szProduced
vwwiProduced,
!! see VVWINFO::szAlbum
vwwiAlbum,
!! see VVWINFO::szArtist
vwwiArtist,
!! see VVWINFO::szComposer
vwwiComposer,
!! see VVWINFO::szCopyright
vwwiCopyright,
!! see VVWINFO::szCreationData
vwwiCreationData,
!! see VVWINFO::szDescription
vwwiDescription,
!! see VVWINFO::szDirector
vwwiDirector,
!! see VVWINFO::szDisclaimer
vwwiDisclaimer, // <-- 30
!! see VVWINFO::szEncodedBy
vwwiEncodedBy,
!! see VVWINFO::szFullName
vwwiFullName,
!! see VVWINFO::szGenre
vwwiGenre,
!! see VVWINFO::szHostComputer
vwwiHostComputer,
!! see VVWINFO::szInformation
vwwiInformation,
!! see VVWINFO::szMake
vwwiMake,
!! see VVWINFO::szModel
vwwiModel,
!! see VVWINFO::szOriginalArtist
vwwiOriginalArtist,
!! see VVWINFO::szOriginalFormat
vwwiOriginalFormat,
!! see VVWINFO::szPerformers
vwwiPerformers, // <-- 40
!! see VVWINFO::szProducer
vwwiProducer,
!! see VVWINFO::szProduct
vwwiProduct,
!! see VVWINFO::szSoftware
vwwiSoftware,
!! see VVWINFO::szSpecialPlaybackRequirements
vwwiSpecialPlaybackRequirements,
!! see VVWINFO::szTrack
vwwiTrack,
!! 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
vwwiSoundRoll,
//! see VVWINFO::szLabRoll
vwwiLabRoll,
//! see VVWINFO::szKeyNumberPrefix
vwwiKeyNumberPrefix,
//! see VVWINFO::szInkNumberPrefix
vwwiInkNumberPrefix,
//! see VVWINFO::szPictureIcon
vwwiPictureIcon,

```

```
    /// see VVWINFO::szProxyFile
    vwwiProxyFile,
    ///
    vwwiCustomMetadataBlockPointer,
    ///
    vwwiImageInfo,
    ///
    vwwiUMID,
    ///
    vwwiEND_OF_STRINGS,

    vwwiNumericStart = 0x1000,
    /// see VVWINFO::dwTimeCode
    vwwiTimeCode,
    /// see VVWINFO::dwUserBits
    vwwiUserBits,
    /// see VVWINFO::dwVITCTimeCode
    vwwiVITCTimeCode,
    /// see VVWINFO::dwVITCUserBits
    vwwiVITCUserBits,
    /// see VVWINFO::dwVITCLine3
    vwwiVITCLine3,
    /// see VVWINFO::dwPosterFrame
    vwwiPosterFrame,
    /// see VVWINFO::dwAFrame
    vwwiAFrame,
    /// see VVWINFO::dwAspectRatio
    vwwiAspectRatio,
    /// see VVWINFO::dwOriginalRate
    vwwiOriginalRate,
    /// see VVWINFO::dwOriginalScale
    vwwiOriginalScale,
    /// see VVWINFO::dwConversions
    vwwiConversions,
    /// see VVWINFO::dwVersionNumber
    vwwiVersionNumber,
    /// see VVWINFO::dwFileSize
    vwwiFileSize,
    /// see VVWINFO::dwFileDate
    vwwiFileDate,
    /// see VVWINFO::dwFileTime
    vwwiFileTime,
    /// see VVWINFO::dwSequenceNumber
    vwwiSequenceNumber,
    /// 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
```

```

vwiBlackGain,
//! see VVWINFO::dwBreakPoint
vwiBreakPoint,
//! see VVWINFO::dwGamma1000
vwiGamma1000,
//! see VVWINFO::dwTagNumber
vwiTagNumber,
//! see VVWINFO::dwFlags
vwiFlags,
//! see VVWINFO::dwTimeCodeType
vwiTimeCodeType,
//! see VVWINFO::dwLTCTimeCodeType
vwiLTCTimeCodeType,
//! see VVWINFO::dwVITCTimeCodeType
vwiVITCTimeCodeType,
//! see VVWINFO::dwProdDate
vwiProdDate,
//End: v3.0
//! see VVWINFO::dwUniqueID
vwiUniqueID,
//!
vwiCustomMetadataBlockType,
vwiCustomMetadataBlockSize,
vwiNorthSouthEastWest,
vwiLatitude,
vwiLongitude,
vwiExposure,
vwiRedGain,
vwiBlueGain,
vwiWhiteBalance,

vwiEND_OF_DWORD_V2,
// Add elements here
//VVVID STRUCT
//! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
vwiVideoWidth = 0x10000,
//! XML tag name for width
#define VVWINFO_TAG_woVideoWidth "Width"
#define VVWINFODESC_woVideoWidth "Width"
//! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
vwiVideoHeight,
//! XML tag name for height
#define VVWINFO_TAG_woVideoHeight "Height"
#define VVWINFODESC_woVideoHeight "Height"
//! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
vwiVideoPlanes,
//! XML tag name for planes
#define VVWINFO_TAG_woVideoPlanes "Planes"
#define VVWINFODESC_woVideoPlanes "Planes"
//! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
vwiVideoBitCount,
//! XML tag name for bit count
#define VVWINFO_TAG_woVideoBitCount "BitCount"
#define VVWINFODESC_woVideoBitCount "BitCount"
//! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
vwiVideoCompression,
//! XML tag name for compression (fourcc)

```

```

#define VVWINFOTAG_woVideoCompression          "Compression"
#define VVWINFODESC_woVideoCompression        "Compression"
    /*! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    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/#VVWAUDIO
    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/#VVWAUDIO
    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/#VVWAUDIO
    vwiVideoClrUsed,
    /*! XML tag name for color elements used
#define VVWINFOTAG_woVideoClrUsed             "ClrUsed"
#define VVWINFODESC_woVideoClrUsed           "ClrUsed"
    /*! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoClrImportant,
    /*! XML tag name for
#define VVWINFOTAG_woVideoClrImportant        "ClrImportant"
#define VVWINFODESC_woVideoClrImportant      "ClrImportant"
    /*! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoReserved,
    /*! XML tag name for reserved array
#define VVWINFOTAG_woVideoReserved           "Reserved"
#define VVWINFODESC_woVideoReserved          "Reserved"
    /*! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    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/#VVWAUDIO
    vwiVideoFccHandler,
    /*! XML tag name for four cc handler
#define VVWINFOTAG_woVideoFccHandler         "FccHandler"
#define VVWINFODESC_woVideoFccHandler       "FccHandler"
    /*! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoFlags,
    /*! XML tag name for flags
#define VVWINFOTAG_woVideoFlags              "Flags"
#define VVWINFODESC_woVideoFlags            "Flags"
    /*! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoCaps,
    /*! XML tag name for capabilities
#define VVWINFOTAG_woVideoCaps               "Caps"
#define VVWINFODESC_woVideoCaps             "Caps"
    /*! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoPriority,
    /*! XML tag name for priority
#define VVWINFOTAG_woVideoPriority           "Priority"
#define VVWINFODESC_woVideoPriority         "Priority"

```



```

    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoLanguage,
    //! XML tag name for language
#define VVWINFOFOTAG_woVideoLanguage          "Language"
#define VVWINFODESC_woVideoLanguage          "Language"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoScale,
    //! XML tag name for scale (fps = rate / scale)
#define VVWINFOFOTAG_woVideoScale            "Scale"
#define VVWINFODESC_woVideoScale            "Scale"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoRate,
    //! XML tag name for rate (fps = rate / scale)
#define VVWINFOFOTAG_woVideoRate            "Rate"
#define VVWINFODESC_woVideoRate            "Rate"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoStart,
    //! XML tag name for start frame
#define VVWINFOFOTAG_woVideoStart           "Start"
#define VVWINFODESC_woVideoStart           "Start"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoLength,
    //! XML tag name for the length in frames
#define VVWINFOFOTAG_woVideoLength          "Length"
#define VVWINFODESC_woVideoLength          "Length"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoInitialFrames,
    //! XML tag name for number of initial frames to load
#define VVWINFOFOTAG_woVideoInitialFrames   "InitialFrames"
#define VVWINFODESC_woVideoInitialFrames    "InitialFrames"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoSuggestedBufferSize,
    //! XML tag name for suggested maximum buffer size
#define VVWINFOFOTAG_woVideoSuggestedBufferSize "SuggestedBufferSize"
#define VVWINFODESC_woVideoSuggestedBufferSize "SuggestedBufferSize"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoQuality,
    //! XML tag name for quality
#define VVWINFOFOTAG_woVideoQuality         "Quality"
#define VVWINFODESC_woVideoQuality         "Quality"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoSampleSize,
    //! XML tag name for recommended sample size
#define VVWINFOFOTAG_woVideoSampleSize      "SampleSize"
#define VVWINFODESC_woVideoSampleSize      "SampleSize"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoEditCount,
    //! XML tag name for number of edits done on this file
#define VVWINFOFOTAG_woVideoEditCount       "EditCount"
#define VVWINFODESC_woVideoEditCount       "EditCount"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoFormatChangeCount,
    //! XML tag name for number of format changes
#define VVWINFOFOTAG_woVideoFormatChangeCount "FormatChangeCount"
#define VVWINFODESC_woVideoFormatChangeCount "FormatChangeCount"
    //! INTERNAL: Auto generated for XML output from #VWVIDEO/#VWAUDIO
    vwiVideoPitch,

```

```

    //! XML tag name for video line pitch
#define VVWINFOTAG_woVideoPitch                "Pitch"
#define VVWINFODESC_woVideoPitch              "Pitch"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoDrFlags,
    //! XML tag name for internal drastic flags
#define VVWINFOTAG_woVideoDrFlags            "DrFlags"
#define VVWINFODESC_woVideoDrFlags          "DrFlags"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoFileType,
    //! XML tag name for drastic 'mft' file type
#define VVWINFOTAG_woVideoFileType          "FileType"
#define VVWINFODESC_woVideoFileType        "FileType"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiVideoResDrastic,
    //! XML tag name for reserved drastic array of DWORDs
#define VVWINFOTAG_woVideoResDrastic        "ResDrastic"
#define VVWINFODESC_woVideoResDrastic      "ResDrastic"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiAudioType,
    //! XML tag
#define VVWINFOTAG_woAudioType              "AudioType"
#define VVWINFODESC_woAudioType            "AudioType"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiAudioChannels,
    //! XML tag
#define VVWINFOTAG_woAudioChannels          "AudioChannels"
#define VVWINFODESC_woAudioChannels        "AudioChannels"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiAudioFrequency,
    //! XML tag
#define VVWINFOTAG_woAudioFrequency        "AudioFrequency"
#define VVWINFODESC_woAudioFrequency      "AudioFrequency"
    //! INTERNAL: Auto generated for XML output from #VVWVIDEO/#VVWAUDIO
    vwiAudioBits,
    //! XML tag
#define VVWINFOTAG_woAudioBits              "AudioBits"
#define VVWINFODESC_woAudioBits            "AudioBits"
    //char szName[_VVWXXX_NAME_SIZE];        // Stream identifier
    //RECT/*16*/ rcFrame;                    // Frame dimensions
    vwiLastElementPlus1
    // DO NOT ADD ANYTHING BELOW vwiLastElementPlus1
};

```

Direct Link Header

dtmediawrite.h

```
/
*****
*****
*
*
* Copyright (c) 1998-2012 Drastic Technologies Ltd. All Rights Reserved.
* 523 The Queensway, Suite 102 Toronto ON M8V 1Y7
* phone (416) 255 5636 fax (416) 255 8780
* engineering@drastictech.com http://www.drastic.tv

*****
****/
// drmediawrite.h : Declaration of the dtmediawrite api

// Hacking class from activex control

#ifndef __DTMEDIAWRITE_DRASTIC_API_9204jrewf348j4_H_
#define __DTMEDIAWRITE_DRASTIC_API_9204jrewf348j4_H_

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

#define DTMWHANDLE void*

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

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

/** The write types
*/
//! Windows RGBA (like bitmap, tga, etc)
const unsigned long DTMW_WRITETYPE_ARGB = 0;
//! 8 Bit YCbCr (yuv2, D1/HDSI raw 4:2:2 video
const unsigned long DTMW_WRITETYPE_UYVY = 1;
//! 10 Bit v210 (quicktime packing) 4:2:2 video
const unsigned long DTMW_WRITETYPE_V210 = 2;
//! 10 Bit RGB 4:4:4 (dpx packing)
const unsigned long DTMW_WRITETYPE_RGB10Bit = 3;
//! 16 bit per component (64 bit) RGBA 4:4:4:4
const unsigned long DTMW_WRITETYPE_RGBA64 = 4;
//! 16 bit half float per component RGBA (GPU)
const unsigned long DTMW_WRITETYPE_RGBAHALFFLOAT = 5;
//! Set readtype AUDIO to 16 bits LE
const unsigned long DTMW_WRITETYPE_FRAME_AUDIO_16LE = (0x00010000 | 16);
```

```

//! Set readtype AUDIO to 32 bits (note, 16, 20, 24 will be shifted to most
significant, LE)
const unsigned long DTMW_WRITETYPE_FRAME_AUDIO_32LE = (0x00010000 | 32);
//! Invalid file
const long DTMW_WRITETYPE_INVALID = -1;

enum {
    dtmwWave = 1,          // Windows WAV audio files (audio)
    //dtmwMov2 = 183,
    //dtmwMovh264 = 190,
    dtmwMov = 164,        // Quicktime movie files (audio video info)
    dtmwAvi = 3,         // Video for windows, Audio Video Interleave
(audio video info)
    dtmwLiveTga = 99, // 32 Bit Uncompressed only
    dtmwLiveTiff = 101, // 32 Bit Uncompressed only
    dtmwLiveYuv = 104, // 8/10 Bit Uncompressed YCbCr only
    dtmwHdrYuv = 106, //
    dtmwAdvWave = 107, // Windows WAVE format extension (multi
channel) audio plugin (no dual mono)
    dtmwAdvAiff = 108, // Apple/SGI format multi channel audio plugin
    dtmwMXFSonySD = 110, // Sony IMX MPEG SD
    dtmwLiveDpx = 111, // RGB10 or YCBCR10
    dtmwBWaveF = 117, // Broadcast wave format
    dtmwMXFSonyHD = 127, // Sony 25/35mbit 4:2:2 XDCam (old XDCam)
    //dtmwMPEG4 , // MPEG-2 h264 essence
    dtmwMXFP2DV = 134, // Panasonic P2 DV25/50/HD
    dtmwMXFAvid = 135, // Avid OPAAtom direct to mediafiles
    dtmwMXFP2AVCi = 163, // Panasonic AVCi 100/50 writer
    dtmwMXFDPCP = 167, // Unencrypted DCP
    dtmwMXFOP1a = 172, // Op1a - yuv, j2k, dnxhd, avci, dvhd
// dtmwLiveDng = 178, // DNG bayer (direct write only)
    dtmwMXFSMDK = 186, // Sony HDCam MXF
    dtmwMXFSony422 = 192, // Sony XDCam 4:2:2 50 MBit
    dtmwMXFEasyDCP = 196, // Encrypted DCP (requires EasyDCP license)
    dtmwMP4 = 197, // MP4 with 264 compression
// dtmwMXFSonyXAVC = 198, // Sony XAVC Conatainer
    dtmwMXFAS02 = 201, // MXF AS-02

    //
    //
    //
    dtmftrtIndex = 172 // Special case, write and RTIN directly
};

#ifndef DTFOUR_CHAR_CODE
#define DTFOUR_CHAR_CODE(x) ((unsigned long)(x))
#endif
#ifndef DTRFOUR_CHAR_CODE
#define DTRFOUR_CHAR_CODE(x) (((unsigned long) ((x) & 0x000000FF))
<< 24) \
+ (((unsigned long)
((x) & 0x0000FF00)) << 8) \
+ (((unsigned long)

```

```

((x & 0x00FF0000)) >> 8) \
                                                                    + (((unsigned long)
((x & 0xFF000000)) >> 24))
#endif

enum {
/** dtmwWave = 1,          // Windows WAV audio files (audio)
 * Audio only
 */
#define DTWAVE_FORMAT_PCM 1
/** dtmwMov2 = 183,
 * dv25, dv50, dvhd, dnxhd, ycbcr, cineform, rgb10, avci100
 */
    dtmwfccdv25 =
DTRFOUR_CHAR_CODE('dv25'), // DV-25 4:2:0
    dtmwfccdv50 =
DTRFOUR_CHAR_CODE('dv50'), // DV-50 4:2:2
    dtmwfccdvhd =
DTRFOUR_CHAR_CODE('dvhd'), // DV-100/DVHD
    dtmwfcckYCbCr8Bit = DTRFOUR_CHAR_CODE('2vuy'),
        // yuv2/uyvy
    dtmwfcckYCbCr10Bit = DTRFOUR_CHAR_CODE('v210'),
        // v210
    dtmwfccCineForm =
DTRFOUR_CHAR_CODE('CFHD'), // CineForm lossless/lossy codec

// dtmwMovh264 = 190,
// dtmwMov = 2,          // Quicktime movie files (audio video info)
/** dtmwAvi = 3,          // Video for windows, Audio Video Interleave
(audio video info)
 * dv25, dv50, dvhd, ycbcr8, ycbcr10, cineform
 */
    //dtmwfccdv25
    //dtmwfccdv50
    //dtmwfccdvhd
    //dtmwfcckYCbCr8Bit
    //dtmwfcckYCbCr10Bit
    //dtmwfccCineForm

/** dtmwLiveTga = 99, // 32 Bit Uncompressed only
 * 32 RGB only
 */
    dtmwBI_RGB = 0,

/** dtmwLiveTiff = 101, // 32 Bit Uncompressed only
 * 32 RGB only
 */
    //dtmwBI_RGB

/** dtmwLiveYuv = 104, // 8/10 Bit Uncompressed YCbCr only
 * 8 and 10 bit ycbcr
 */
    //dtmwfcckYCbCr8Bit
    //dtmwfcckYCbCr10Bit

```

```

/** dtmwHdrYuv = 106, //
 * 8 and 10 bit ycbcr
 */
    //dtmwfcckYCbCr8Bit
    //dtmwfcckYCbCr10Bit

/** dtmwAdvWave = 107, // Windows WAVE format extension (multi
channel) audio plugin (no dual mono)
 * Audio Only
 */
    //DTWAVE_FORMAT_PCM
#define DTWAVE_FORMAT_EXTENSIBLE    0xFFFE

/** dtmwAdvAiff = 108, // Apple/SGI format multi channel audio plugin
 * Audio Only
 */
    dtmwfcck16BitBigEndianFormat    = DTFOUR_CHAR_CODE('twos'),
    /*16-bit big endian*/

/** dtmwMXFSonySD = 110, // Sony IMX MPEG SD
 * IMX PAL and NTSC
 */
    dtmwfcckIMXD10_NTSC_50          = DTFOUR_CHAR_CODE('mx5n'),
    // FinalCut Pro 5.0 Studio
    dtmwfcckIMXD10_NTSC_40          = DTFOUR_CHAR_CODE('mx4n'),
    // FinalCut Pro 5.0 Studio
    dtmwfcckIMXD10_NTSC_30          = DTFOUR_CHAR_CODE('mx3n'),
    // FinalCut Pro 5.0 Studio
    dtmwfcckIMXD10_PAL_50           = DTFOUR_CHAR_CODE('mx5p'),
    // FinalCut Pro 5.0 Studio
    dtmwfcckIMXD10_PAL_40           = DTFOUR_CHAR_CODE('mx4p'),
    // FinalCut Pro 5.0 Studio
    dtmwfcckIMXD10_PAL_30           = DTFOUR_CHAR_CODE('mx3p'),
    // FinalCut Pro 5.0 Studio

/** dtmwLiveDpx = 111, // RGB10 or YCBCR10
 * YCbCr 10 and RGB10
 */
    dtmwfcc10LinDPX                  =
DTFOUR_CHAR_CODE('R10k'),          // Big endian
    dtmwfcc10LogDPX                  =
DTFOUR_CHAR_CODE('R10g'),          // Big endian
    //dtmwfcckYCbCr8Bit
    //dtmwfcckYCbCr10Bit

/** dtmwBWaveF = 117, // Broadcast wave format
 * Audio only
 */
    //DTWAVE_FORMAT_PCM

/** dtmwMXFSonyHD = 127, // Sony 25/35mbit 4:2:2 XDCam (old XDCam)
 * MPEG 4:2:0 only
 */

```

```

    dtmwfccDT_MPEGHD_VBR_I                =
DTRFOUR_CHAR_CODE('mgv1'), // 4:2:0 XDCAM HD VBR Interlace
    dtmwfccDT_MPEGHD_VBR_P                =
DTRFOUR_CHAR_CODE('mgv2'), // 4:2:0 XDCAM HD VBR Progressive
    dtmwfccDT_MPEGHD_VBR_I_17            = DTRFOUR_CHAR_CODE('mc17'),
// 4:2:0 XDCAM HD VBR Interlace 17.5 Mbps
    dtmwfccDT_MPEGHD_VBR_P_17            = DTRFOUR_CHAR_CODE('mv17'),
// 4:2:0 XDCAM HD VBR Progressive 17.5 Mbps
    dtmwfccDT_MPEGHD_VBR_I_25            = DTRFOUR_CHAR_CODE('mc25'),
// 4:2:0 XDCAM HD VBR Interlace 25 Mbps
    dtmwfccDT_MPEGHD_VBR_P_25            = DTRFOUR_CHAR_CODE('mv25'),
// 4:2:0 XDCAM HD VBR Progressive 25 Mbps
    dtmwfccDT_MPEGHD_VBR_I_35            = DTRFOUR_CHAR_CODE('mc35'),
// 4:2:0 XDCAM HD VBR Interlace 35 Mbps
    dtmwfccDT_MPEGHD_VBR_P_35            = DTRFOUR_CHAR_CODE('mv35'),
// 4:2:0 XDCAM HD VBR Progressive 35 Mbps
    dtmwfccDT_MPEGHD_CBR_I                =
DTRFOUR_CHAR_CODE('mgc1'), // 4:2:0 XDCAM HD CBR Interlace 25 Mbps
    dtmwfccDT_MPEGHD_CBR_P                =
DTRFOUR_CHAR_CODE('mgc2'), // 4:2:0 XDCAM HD CBR Progressive 25 Mbps

/** dtmwMPEG4 , // MPEG-2 h264 essence
 * h264
 */
    drmwfcckH264CodecType                = DTFOUR_CHAR_CODE('avc1'), /*
MEDIASUBTYPE_AVC1 'AVC1' H.264 bitstream without start codes.*/

/** dtmwMXFP2DV = 134, // Panasonic P2 DV25/50/HD
 * DV25, DV50, DVHD
 */
    //dtmwfccdv25
    //dtmwfccdv50
    //dtmwfccdvhd

/** dtmwMXFAvid = 135, // Avid OPAAtom direct to mediafiles
 * DNxHD
 */
    dtmwfcckDNxHD_220x_10                = DTFOUR_CHAR_CODE('AV10'),
// 1920x1080 10 Bit P (220x/185x/175x)
    dtmwfcckDNxHD_145x                    =
DTFOUR_CHAR_CODE('AVd2'), // 1920x1080 8 Bit P (145/120/115) ~equiv
hdcam/dvcpro100
    dtmwfcckDNxHD_220x                    =
DTFOUR_CHAR_CODE('AVd3'), // 1920x1080 8 Bit P (220/185/175)
    dtmwfcckDNxHD_220_10                  = DTFOUR_CHAR_CODE('AVd4'),
// 1920x1080 10 Bit i (220/185/175)
    dtmwfcckDNxHD_145                    =
DTFOUR_CHAR_CODE('AVd5'), // 1920x1080 8 Bit i (145/120/115)
    dtmwfcckDNxHD_220                    =
DTFOUR_CHAR_CODE('AVd6'), // 1920x1080 8 Bit i (220/185/175)
    dtmwfcckDNxHD_720_220x                = DTFOUR_CHAR_CODE('AVd7'),
// 1280x720 10 Bit P (220x/175x/90x)
    dtmwfcckDNxHD_720_220                = DTFOUR_CHAR_CODE('AVd8'),
// 1280x720 8 Bit P (220x/175x/90x)

```

```

        dtmwfccDNxHD_720_145                = DTFOUR_CHAR_CODE('AVd9'),
        // 1280x720 8 Bit P (145x/120x/115x)
        dtmwfccDNxHD_36                    = DTFOUR_CHAR_CODE('AVd0'),
        // 1920x1080 8 Bit P (36)

/** dtmwMXFP2AVCi = 163, // Panasonic AVCi 100/50 writer
 * AVCi 100
 */
        dtmwfccAVCi100                      =
DTFOUR_CHAR_CODE('ai16'),

/** dtmwMXFDCP = 167, // Unencrypted DCP
 * JPEG-2000
 */
        dtmwfccJ2_Cinema2K                  =
DTRFOUR_CHAR_CODE('J22K'), // Digital cinema 2K (alias)
        dtmwfccJ2_Cinema4K                  =
DTRFOUR_CHAR_CODE('J24K'), // Digital cinema 4K (alias)

/** dtmwMXFOP1a = 172, // Op1a - yuv, j2k, dnxhd, avci, dvhd
 * YCbCr 8, DVHD, AVCi, DNxHD, JPEG-2000
 */
        //dtmwfcckYCbCr8Bit
        //dtmwfccdvhd
        //dtmwfccAVCi100
        fccJPEG2000_YCbCr                    = DTRFOUR_CHAR_CODE('J2GV'),
        // SAMA/YCbCrJ2K/Grass Valley Infinity

/** dtmwLiveDng = 178, // DNG bayer (direct write only)
 * Bayer (direct write only)
 */

/** dtmwMXFSMDK = 186, // Sony HDCam MXF
 * HDCam
 */
        dtmwfccHDCamSR                      =
DTFOUR_CHAR_CODE('HDSR'),
        dtmwfccHDCamSR_444                  =
DTFOUR_CHAR_CODE('HDS4'),

/** dtmwMXFSony422 = 192, // Sony XDCam 4:2:2 50 MBit
 * MPEG 4:2:2
 */
        dtmwfccDT_MPEG422                    =
DTRFOUR_CHAR_CODE('mg01'), // 4:2:2 MPEG-2

/** dtmwMFXEasyDCP = 196, // Encrypted DCP (requires EasyDCP license)
 * JPEG-2000
 */
        //dtmwfccJ2_Cinema2K
        //dtmwfccJ2_Cinema4K

/** dtmwMP4 = 197, // MP4 with 264 compression
 * h264

```



```

*/
    //dtmwfcckH264CodecType

/** dtmwMXFSonyXAVC = 198, // Sony XAVC Conatainer
* XAVC
*/

/** dtmwMXFAS02 = 201, // MXF AS-02
* JPEG-2000 (SAMA), YCbCr 8, XDCam
*/
    //dtmwfcckYCbCr8Bit
    //fccJPEG2000_YCbCr
    //fccJPEG2000CodecType
    dtmwfcckXAVC =
DTFOUR_CHAR_CODE('xavc'),
    dtmwfcckXAVC4K =
DTFOUR_CHAR_CODE('xav4'),
    //dtmwfccDT_MPEG422 (XDCam)

};

/** Open a new file, stream or network source for preview
*/
DTMWHANDLE DTMWCALLTYPE dtmwOpen(char * szFileName, unsigned long
dwFlags,
    unsigned long dwFileType, unsigned long dwFourCC, unsigned long dwWidth,
unsigned long dwHeight,
    unsigned long dwRate, unsigned long dwScale, unsigned long
dwAudioChannels,
    unsigned long dwAudioRate, unsigned long dwAudioBits);
typedef DTMWHANDLE (DTMWCALLTYPE * p_dtmwOpen)(char * szFileName,
unsigned long dwFlags,
    unsigned long dwFileType, unsigned long dwFourCC, unsigned long dwWidth,
unsigned long dwHeight,
    unsigned long dwRate, unsigned long dwScale, unsigned long
dwAudioChannels,
    unsigned long dwAudioRate, unsigned long dwAudioBits);

/** Close the currently open stream or file
*/
long DTMWCALLTYPE dtmwClose(DTMWHANDLE dtmw);
typedef long (DTMWCALLTYPE * p_dtmwClose)(DTMWHANDLE dtmw);

/** Returns recommended and supported write types
*/
long DTMWCALLTYPE dtmwGetWriteTypes(DTMWHANDLE dtmw, unsigned long
dwIndex, unsigned long * pdwTypes);
typedef long (DTMWCALLTYPE * p_dtmwGetWriteTypes)(DTMWHANDLE dtmw,
unsigned long dwIndex, unsigned long * pdwTypes);

/** The final file name used for the target file
*/
long DTMWCALLTYPE dtmwTargetFileName(DTMWHANDLE dtmw, char * tszString);

```

```

typedef long (DTMWCALLTYPE * p_dtmwTargetFileName)(DTMWHANDLE dtmw, char
* tszString);

/** Target video media's height
*/
long DTMWCALLTYPE dtmwTargetHeight(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetHeight)(DTMWHANDLE dtmw, long
*pVal);

/** Target video media's width
*/
long DTMWCALLTYPE dtmwTargetWidth(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetWidth)(DTMWHANDLE dtmw, long
*pVal);

/** Target pitch depending on frame type
*/
long DTMWCALLTYPE dtmwTargetPitch(DTMWHANDLE dtmwPV, long IType, long
*pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetPitch)(DTMWHANDLE dtmwPV, long
IType, long *pVal);

/* Target video media's bit depth
*/
long DTMWCALLTYPE dtmwTargetBitDepth(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetBitDepth)(DTMWHANDLE dtmw, long
*pVal);

/* Target video media's fourcc compression code
*/
long DTMWCALLTYPE dtmwTargetFourCC(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetFourCC)(DTMWHANDLE dtmw, long
*pVal);

/* Target video media's bit rate
*/
long DTMWCALLTYPE dtmwTargetBitRate(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetBitRate)(DTMWHANDLE dtmw, long
*pVal);

/* Target video media's quality
*/
long DTMWCALLTYPE dtmwTargetQuality(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetQuality)(DTMWHANDLE dtmw, long
*pVal);

/* Target video media's frame size for the requested or current frame
*/
long DTMWCALLTYPE dtmwTargetFrameSize(DTMWHANDLE dtmw, long
dwFrameType, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetFrameSize)(DTMWHANDLE dtmw,
long dwFrameType, long *pVal);

/* Target video total channels

```

```

*/
long DTMWCALLTYPE dtmwTargetVideoChannels(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetVideoChannels)(DTMWHANDLE dtmw,
long *pVal);

/* Target audio total channels
*/
long DTMWCALLTYPE dtmwTargetAudioChannels(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetAudioChannels)(DTMWHANDLE dtmw,
long *pVal);

/** Target audio media frequency
*/
long DTMWCALLTYPE dtmwTargetAudioFrequency(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetAudioFrequency)(DTMWHANDLE
dtmw, long *pVal);

/** Target audio media bits per sample
*/
long DTMWCALLTYPE dtmwTargetAudioBitsPerSample(DTMWHANDLE dtmw, long
*pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetAudioBitsPerSample)(DTMWHANDLE
dtmw, long *pVal);

/* Target audio media's fourcc compression code
*/
long DTMWCALLTYPE dtmwTargetAudioFourCC(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetAudioFourCC)(DTMWHANDLE dtmw,
long *pVal);

/** Target video rate value (FPS = TargetRate / TargetScale)
*/
long DTMWCALLTYPE dtmwTargetRate(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetRate)(DTMWHANDLE dtmw, long
*pVal);

/** Target video scale value (FPS = TargetRate / TargetScale)
*/
long DTMWCALLTYPE dtmwTargetScale(DTMWHANDLE dtmw, long *pVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetScale)(DTMWHANDLE dtmw, long
*pVal);

/** Return Target metadata information that are numeric (DWORDs or longs)
*/
long DTMWCALLTYPE dtmwTargetMetaDataDWORD(DTMWHANDLE dtmw, long
dwMetaElement, long dwVal);
typedef long (DTMWCALLTYPE * p_dtmwTargetMetaDataDWORD)(DTMWHANDLE
dtmw, long dwMetaElement, long dwVal);

/** Return Target metadata information that are string data
*/
long DTMWCALLTYPE dtmwTargetMetaDataSTR(DTMWHANDLE dtmw, long
dwMetaElement, char * szMAX_PATHString);
typedef long (DTMWCALLTYPE * p_dtmwTargetMetaDataSTR)(DTMWHANDLE dtmw,

```

```

long dwMetaDataElement, char * szMAX_PATHString);

/** Set the write type for the video frames
 */
long DTMWCALLTYPE dtmwSetWriteType(DTMWHANDLE dtmw, long IWriteType);
typedef long (DTMWCALLTYPE * p_dtmwSetWriteType)(DTMWHANDLE dtmw, long
IWriteType);

/** Set the channel for the video frames (0, 1, 2, 3, 4 etc) (0 = 0x03, 1 = 0x0C, 2 =
0x30, 3 = 0xC0 etc.)
 */
long DTMWCALLTYPE dtmwSetVideoChannel(DTMWHANDLE dtmw, long
IVideoChannel);
typedef long (DTMWCALLTYPE * p_dtmwSetVideoChannel)(DTMWHANDLE dtmw,
long IVideoChannel);

/** Set the audio channel pair to monitor (0 = 1+2, 1 = 3+4, 2 = 5+6, 3 = 7+8
etc.)
 */
long DTMWCALLTYPE dtmwSetAudioChannelPair(DTMWHANDLE dtmw, long
IAudioChannelPair);
typedef long (DTMWCALLTYPE * p_dtmwSetAudioChannelPair)(DTMWHANDLE dtmw,
long IAudioChannelPair);

//
long DTMWCALLTYPE dtmwSetVitcType(DTMWHANDLE dtmwPV, long dwVal);
typedef long (DTMWCALLTYPE * p_dtmwSetVitcType)(DTMWHANDLE dtmw, long
dwVal);

//
long DTMWCALLTYPE dtmwSetLtcType(DTMWHANDLE dtmwPV, long dwVal);
typedef long (DTMWCALLTYPE * p_dtmwSetLtcType)(DTMWHANDLE dtmw, long
dwVal);

/** Set the next VITC (vertical blank) time code
 */
long DTMWCALLTYPE dtmwNextVitcFrame(DTMWHANDLE dtmw, long dwVal);
typedef long (DTMWCALLTYPE * p_dtmwNextVitcFrame)(DTMWHANDLE dtmw, long
dwVal);

/** Set the next VITC (vertical blank time code) user bits
 */
long DTMWCALLTYPE dtmwNextVitcUb(DTMWHANDLE dtmw, long dwVal);
typedef long (DTMWCALLTYPE * p_dtmwNextVitcUb)(DTMWHANDLE dtmw, long
dwVal);

/** Set the next LTC (SMPTE) time code
 */
long DTMWCALLTYPE dtmwNextLtcFrame(DTMWHANDLE dtmw, long dwVal);
typedef long (DTMWCALLTYPE * p_dtmwNextLtcFrame)(DTMWHANDLE dtmw, long
dwVal);

/** Set the next LTC (SMPTE time code) user bits
 */

```

```

long DTMWCALLTYPE dtmwNextLtcUb(DTMWHANDLE dtmw, long dwVal);
typedef long (DTMWCALLTYPE * p_dtmwNextLtcUb)(DTMWHANDLE dtmw, long
dwVal);

/** PutVideoFrame sends one video frame
 */
long DTMWCALLTYPE dtmwPutVideoFrame(DTMWHANDLE dtmw, unsigned char *
psvFrame, long dwSize);
typedef long (DTMWCALLTYPE * p_dtmwPutVideoFrame)(DTMWHANDLE dtmw,
unsigned char * psvFrame, long dwSize);

/** PutAudioFrame returns a safe array containing one video frame worth of audio
data
 */
long DTMWCALLTYPE dtmwPutAudioFrame(DTMWHANDLE dtmw, unsigned char *
psaFrame, long dwSize);
typedef long (DTMWCALLTYPE * p_dtmwPutAudioFrame)(DTMWHANDLE dtmw,
unsigned char * psaFrame, long dwSize);

/** Get current extended data
 */
long DTMWCALLTYPE dtmwPutNextExtendedData(DTMWHANDLE dtmw, unsigned
char *pvData, long lSize, long lFlags);
typedef long (DTMWCALLTYPE * p_dtmwPutNextExtendedData)(DTMWHANDLE
dtmw, unsigned char *pvData, long lSize, long lFlags);

/** SetMode - send a mediacmd structure (advanced)
 */
long DTMWCALLTYPE dtmwSetMode(DTMWHANDLE dtmwPV, void * pMediaCmd);
typedef long (DTMWCALLTYPE * p_dtmwSetMode)(void * pMediaCmd);

/** Get the version
 */
long DTMWCALLTYPE dtmwVersion(long *pVerMajor, long *pVerMinor, long
*pVerMod, long *pVerBuild);
typedef long (DTMWCALLTYPE * p_dtmwVersion)(long *pVerMajor, long *pVerMinor,
long *pVerMod, long *pVerBuild);

// dwFlags
    ///! Send this in if you just need the filename (faster then 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
    ///! Flag for mediafile/avhal to put audio dframe
#define PutAudio     GetAudio
    ///! Flag for mediafile/avhal to put video dframe
#define PutVideo     GetVideo
    ///! Film 24 FPS time code
#define TC2_TCTYPE_FILM          0x00000001 // 24 fps
    ///! Non Drop Frame 30 FPS time code
#define TC2_TCTYPE_NDF          0x00000002 // NTSC Non Drop Frame

```

```

//! Drop Frame 29.97 FPS time code
#define TC2_TCTYPE_DF          0x00000004 // NTSC Drop Frame
//! PAL 25 FPS time code
#define TC2_TCTYPE_PAL        0x00000008 // PAL
//! Double PAL 50 FPS
#define TC2_TCTYPE_50         0x00000010 // PAL 720p (double rate)
//! 720p DROP 59.94 FPS
#define TC2_TCTYPE_5994       0x00000020 // NTSC 59.94fps 720p (NTSC DF
double)
//! 720p DROP 59.97 FPS
#define TC2_TCTYPE_5997       0x00000022 // NTSC 59.94fps 720p (NTSC DF
double)
//! 720p 60 FPS
#define TC2_TCTYPE_60         0x00000040 // NTSC 60fps 720p (NTSC NDF
double)
//! 23.98 FILM for NTSC 23.98 FPS (This is actually 24)
#define TC2_TCTYPE_NTSCFILM   0x00000080 // NTSC FILM 23.98
//! 23.98 TRUE (actual 23.98 drop per Avid)
#define TC2_TCTYPE_2398       0x00000084 // TRUE 23.98
//! Hundredths of a second HH:MM:SS:/100 100 FPS effective
#define TC2_TCTYPE_100        0x00000044 //
Hours:Minutes:Seconds:Hundreds
//! IRIG time code, uses both time code and user bits
#define TC2_TCTYPE_IRIG       0x00000045 // Hours:Minutes:Seconds:Xxx

// 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

/** Set info on a frame of audio or video for RTIN files
*/
long DTMWCALLTYPE dtmwPutFileFrameInfo(DTMWHANDLE dtmwPV, unsigned long
dwRTChannel, unsigned long dwFrame, unsigned long dwFlags,
    size_t nPosition, size_t nSize, unsigned long
dwFrameFlags, unsigned long dwRepsSamples);
typedef long (DTMWCALLTYPE * p_dtmwPutFileFrameInfo)(DTMWHANDLE dtmwPV,
unsigned long dwRTChannel, unsigned long dwFrame, unsigned long dwFlags,
    size_t nPosition, size_t nSize, unsigned long
dwFrameFlags, unsigned long dwRepsSamples);

/** AddVideoChannel - rtIndex add a video channel to the rtindex file
*/
long DTMWCALLTYPE dtmwAddVideoChannel(DTMWHANDLE dtmwPV, char *
szVideoFile, unsigned long dwFileType, unsigned long dwFourCC, unsigned long
dwWidth, unsigned long dwHeight,
    unsigned long dwRate, unsigned long dwScale, unsigned long *
pdwVideoChannelHandle);
typedef long (DTMWCALLTYPE * p_dtmwAddVideoChannel)(DTMWHANDLE dtmwPV,
char * szVideoFile, unsigned long dwFileType, unsigned long dwFourCC, unsigned
long dwWidth, unsigned long dwHeight, unsigned long dwRate, unsigned long
dwScale, unsigned long * pdwVideoChannelHandle);

/** AddAudioChannel - rtIndex add an audio channel to the rtindex file
*/
long DTMWCALLTYPE dtmwAddAudioChannel(DTMWHANDLE dtmwPV, char *
szAudioFile, unsigned long dwFileType, unsigned long dwAudioChannels,
    unsigned long dwAudioRate, unsigned long dwAudioBits, unsigned long *

```

```

pdwAudioChannelHandle);
typedef long (DTMWCALLTYPE * p_dtmwAddAudioChannel)(DTMWHANDLE dtmwPV,
char * szAudioFile, unsigned long dwFileType, unsigned long dwAudioChannels,
    unsigned long dwAudioRate, unsigned long dwAudioBits, unsigned long *
pdwAudioChannelHandle);

/** Get the video codec extra data (e.g. avc1 MP4 avcC box)
* Passing NULL pData will return size
*/
long DTMWCALLTYPE dtmwCodecData(DTMWHANDLE dtmw, unsigned char * pData,
unsigned long dwSize);
typedef long (DTMWCALLTYPE * p_dtmwCodecData)(DTMWHANDLE dtmw, unsigned
char * pData, unsigned long dwSize);

/** Get the audio codec extra data (e.g. acc config bytes)
* Passing NULL pData will return size
*/
long DTMWCALLTYPE dtmwAudioCodecData(DTMWHANDLE dtmw, unsigned char *
pData, unsigned long dwSize);
typedef long (DTMWCALLTYPE * p_dtmwAudioCodecData)(DTMWHANDLE dtmw,
unsigned char * pData, unsigned long dwSize);

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

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

#endif // __DTMEDIAWRITE_DRASTIC_API_9204jrewf348j4_H_

```