DTMediaRead Programmers Interface

Table of Contents

Methods and Properties

Open

long Open(BSTR bstrFileName, long dwFlags);

Open a new file, stream or network source for reading. 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 I can build a sequence out of stills, it will. Here are the basic types that may be automatically added to you 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	- Meta data information (dt: space)
xxx.RDF	- Uniform description XML
xxx.TCI	- Time code information file
xxx.NDX	- Frame index file
xxx.TC	- Time code stream

Close

long Close();

Close the currently open stream or file

SourceFileName

void SourceFileName(BSTR *FileName);

The final file name used for the source file

SourceHeight

long SourceHeight();

Source video media's height

SourceWidth

long SourceWidth();

Source video media's width

SourceBitDepth

long SourceBitDepth();
Source video media's bit depth

SourceFourCC

long SourceFourCC(l);
Source video media's fourcc compression code

SourceBitRate

long SourceBitRate();
Source video media's bit rate

SourceFrameSize

long SourceFrameSize(long dwFrame);
Source video media's frame size for the requested or current frame

SourceVideoChannels

long SourceVideoChannels();
Source video total channels

SourceAudioChannels

long SourceAudioChannels();
Source audio total channels

SourceAudioFrequency

long SourceAudioFrequency();
Source audio media frequency

SourceAudioBitsPerSample

long SourceAudioBitsPerSample();
Source audio media bits per sample

Duration

long Duration();
Return the duration (total number of frames) of the media

SourceRate

long SourceRate();
Source video rate value (FPS = SourceRate / SourceScale)

SourceScale

long SourceScale();
Source video scale value (FPS = SourceRate / SourceScale)

SourceMetaDataDWORD

long SourceMetaDataDWORD(long dwMetaDataElement);
Return source metadata information that are numeric (DWORDs or longs)

SourceMetaDataSTR

BSTR * SourceMetaDataSTR(long dwMetaDataElement);
Return source metadata information that are string data

GetReadTypes

long GetReadTypes(long dwIndex);
Returns recommended and supported read types. Please see the section **Output Video Formats** 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. GetReadTypes() should always be checked unless you are using SetReadType(DTMR_READTYPE_RGBA).

Type	dwIndex	Return
DPX:	0	DTMR_READTYPE_RGBA
	1	DTMR_READTYPE_RGB10Bit
	2	DTMR_READTYPE_INVALID
v210Mov:	0	DTMR_READTYPE_RGBA
	1	DTMR_READTYPE_V210
	2	DTMR_READTYPE_INVALID
AbacusYUV:	0	DTMR_READTYPE_RGBA
	1	DTMR_READTYPE_UYVY
	2	DTMR_READTYPE_INVALID

MPEG:	0	DTMR_READTYPE_RGBA
	1	DTMR_READTYPE_UYVY
	2	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)

SetReadType

long SetReadType(long lReadType);

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

Frame

long Frame();
void Frame(long newVal);

Get or set the current absolute (zero based) Frame

GetVideoFrame

long GetVideoFrame(VARIANT * psaFrame, long * plSize);

GetVideoFrame returns a safe array containing one video frame. 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 section **Output Video Formats** for more information on the available types.

GetAudioFrame

long GetAudioFrame(VARIANT * psaFrame, long * plSize);

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 materials types returned by SourceXXX methods. Please see the section **Output Audio Formats** for more information.

LastVitcFrame

long LastVitcFrame();

Return the last GetVideoFrame VITC (vertical blank) time code

LastVitcUb

long LastVitcUb();
Return the last GetVideoFrame VITC (vertical blank time code) user bits

LastLtcFrame

long LastLtcFrame();
Return the last GetVideoFrame LTC (SMPTE) time code

LastLtcUb

long LastLtcUb();
Return the last GetVideoFrame LTC (SMPTE time code) user bits

Defines And Constants

These formats are used by GetReadTypes() and SetReadType() to setup the frame return type for GetVideoFrame(). See Video Output Formats for more information on these frame layouts.

//! Windows RGBA (like bitmap, tga, etc)
const long **DTMR_READTYPE_RGBA** = 0;
//! 8 Bit YCbCr (yuv2, D1/HDSDI raw 4:2:2 video
const long **DTMR_READTYPE_UVYV** = 1;
//! 10 Bit v210 (quicktime packing) 4:2:2 video
const long **DTMR_READTYPE_V210** = 2;
//! 10 Bit RGB 4:4;4 (dpx packing)
const long **DTMR_READTYPE_RGB10Bit** = 3;
//! Returned if there are no more suggested types
const 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 files 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 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 normally output as two channels of sixteen bits per sample PCM audio. This is written in the same format as windows wave files.

Left Channel (2 Bytes)
Right Channel (2 Bytes)
[repeats with no padding]

The frequency is dependent on the SourceAudioFrequency return.

Example

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 setup 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 acces to the frame data, it can be maipulated however we like. It can be loaded into a device context, or save 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.

Visual C++ MFC Wrapper Header

```
#if !defined(AFX_DTREADX_H__0FC032D3_B7D0_4D98_9112_EB50A7445297__INCLUDED_)
#define AFX_DTREADX_H__0FC032D3_B7D0_4D98_9112_EB50A7445297__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Machine generated IDispatch wrapper class(es) created by Microsoft Visual C++

// NOTE: Do not modify the contents of this file.  If this class is regenerated
by
//  Microsoft Visual C++, your modifications will be overwritten.

/////////////////////////////////////////////////////////////////////////////
// CDTReadX wrapper class

class CDTReadX : public CWnd
{
protected:
	DECLARE_DYNCREATE(CDTReadX)
public:
	CLSID const& GetClsid()
	{
		static CLSID const clsid
			= { 0x429137c0, 0x164e, 0x4cdd, { 0x89, 0x52, 0xa4, 0x63,
0x55, 0x9a, 0x3d, 0x53 } };
		return clsid;
	}
	virtual BOOL Create(LPCTSTR lpszClassName,
		LPCTSTR lpszWindowName, DWORD dwStyle,
		const RECT& rect,
		CWnd* pParentWnd, UINT nID,
		CCreateContext* pContext = NULL)
	{ return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
pParentWnd, nID); }

    BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle,
		const RECT& rect, CWnd* pParentWnd, UINT nID,
		CFile* pPersist = NULL, BOOL bStorage = FALSE,
		BSTR bstrLicKey = NULL)
	{ return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect,
pParentWnd, nID,
		pPersist, bStorage, bstrLicKey); }

// Attributes
public:

// Operations
public:
	long GetReadTypes(long dwIndex);
	long Open(LPCTSTR bstrFileName, long dwFlags);
	long Close();
	void GetSourceFileName(BSTR* FileName);
	long GetSourceHeight();
	long GetSourceWidth();
	long GetSourceBitDepth();
	long GetSourceFourCC();
	long GetSourceBitRate();
	long GetSourceFrameSize(long dwFrame);
	long GetSourceVideoChannels();
	long GetSourceAudioChannels();
	long GetSourceAudioFrequency();
	long GetSourceAudioBitsPerSample();
	long GetDuration();
	long GetSourceRate();
	long GetSourceScale();
```

```
        long GetSourceMetaDataDWORD(long dwMetaDataElement);
        CString GetSourceMetaDataSTR(long dwMetaDataElement);
        long SetReadType(long lReadType);
        long GetFrame();
        void SetFrame(long nNewValue);
        long GetLastVitcFrame();
        long GetLastVitcUb();
        long GetLastLtcFrame();
        long GetLastLtcUb();
        long GetVideoFrame(VARIANT* psaFrame, long* plSize);
        long GetAudioFrame(VARIANT* psaFrame, long* plSize);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif // !
defined(AFX_DTREADX_H__0FC032D3_B7D0_4D98_9112_EB50A7445297__INCLUDED_)
```

Visual C++ MFC Wrapper Implementation

```
// Machine generated IDispatch wrapper class(es) created by Microsoft Visual C++

// NOTE: Do not modify the contents of this file.  If this class is regenerated
by
//  Microsoft Visual C++, your modifications will be overwritten.


#include "stdafx.h"
#include "dtreadx.h"

/////////////////////////////////////////////////////////////////////////////
// CDTReadX

IMPLEMENT_DYNCREATE(CDTReadX, CWnd)

/////////////////////////////////////////////////////////////////////////////
// CDTReadX properties

/////////////////////////////////////////////////////////////////////////////
// CDTReadX operations

long CDTReadX::GetReadTypes(long dwIndex)
{
	long result;
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0x1, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
		dwIndex);
	return result;
}

long CDTReadX::Open(LPCTSTR bstrFileName, long dwFlags)
{
	long result;
	static BYTE parms[] =
		VTS_BSTR VTS_I4;
	InvokeHelper(0x2, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
		bstrFileName, dwFlags);
	return result;
}

long CDTReadX::Close()
{
	long result;
	InvokeHelper(0x3, DISPATCH_METHOD, VT_I4, (void*)&result, NULL);
	return result;
}

void CDTReadX::GetSourceFileName(BSTR* FileName)
{
	static BYTE parms[] =
		VTS_PBSTR;
	InvokeHelper(0x4, DISPATCH_PROPERTYGET, VT_EMPTY, NULL, parms,
		 FileName);
}

long CDTReadX::GetSourceHeight()
{
	long result;
	InvokeHelper(0x5, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceWidth()
{
```

```
	long result;
	InvokeHelper(0x6, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceBitDepth()
{
	long result;
	InvokeHelper(0x7, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceFourCC()
{
	long result;
	InvokeHelper(0x8, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceBitRate()
{
	long result;
	InvokeHelper(0x9, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceFrameSize(long dwFrame)
{
	long result;
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0xa, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, parms,
		dwFrame);
	return result;
}

long CDTReadX::GetSourceVideoChannels()
{
	long result;
	InvokeHelper(0xb, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceAudioChannels()
{
	long result;
	InvokeHelper(0xc, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceAudioFrequency()
{
	long result;
	InvokeHelper(0xd, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceAudioBitsPerSample()
{
	long result;
	InvokeHelper(0xe, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetDuration()
{
	long result;
```

```
	InvokeHelper(0xf, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceRate()
{
	long result;
	InvokeHelper(0x10, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceScale()
{
	long result;
	InvokeHelper(0x11, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetSourceMetaDataDWORD(long dwMetaDataElement)
{
	long result;
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0x12, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, parms,
		dwMetaDataElement);
	return result;
}

CString CDTReadX::GetSourceMetaDataSTR(long dwMetaDataElement)
{
	CString result;
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0x13, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, parms,
		dwMetaDataElement);
	return result;
}

long CDTReadX::SetReadType(long lReadType)
{
	long result;
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0x14, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
		lReadType);
	return result;
}

long CDTReadX::GetFrame()
{
	long result;
	InvokeHelper(0x15, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

void CDTReadX::SetFrame(long nNewValue)
{
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0x15, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
		 nNewValue);
}

long CDTReadX::GetLastVitcFrame()
{
	long result;
	InvokeHelper(0x16, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
```

```
	return result;
}

long CDTReadX::GetLastVitcUb()
{
	long result;
	InvokeHelper(0x17, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetLastLtcFrame()
{
	long result;
	InvokeHelper(0x18, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetLastLtcUb()
{
	long result;
	InvokeHelper(0x19, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long CDTReadX::GetVideoFrame(VARIANT* psaFrame, long* plSize)
{
	long result;
	static BYTE parms[] =
		VTS_PVARIANT VTS_PI4;
	InvokeHelper(0x1a, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
		psaFrame, plSize);
	return result;
}

long CDTReadX::GetAudioFrame(VARIANT* psaFrame, long* plSize)
{
	long result;
	static BYTE parms[] =
		VTS_PVARIANT VTS_PI4;
	InvokeHelper(0x1b, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
		psaFrame, plSize);
	return result;
}
```