MediaCMD HTTP/Ajax Interface

www.drastic.tv

Overview

Drastic's MediaCMD RESTful HTTP API is a standard method of accessing and controlling Drastic's VVW series servers, Net-X-SDI I/O ports, and videoQC. The API is based directly on the binary MediaCMD SDK and can use any of the commands specified there. For this API, standard HTTP strings are sent, and XML responses are returned. Any language that supports HTTP can be used, but most interfaces are built on HTML 5 and Javascript. When you install any of the above products, a ***vvwlibrary.js*** will be installed that contains helper functions for dealing with this API. There will also be HTML files installed that use the API to control and retrieve status from the API. These are valuable examples and should be examined, along with this documentation.

There are three major types of commands available from the REST/HTTP interface:

1. Simple string commands and returns
2. Full MediaCmd AJAX/XML access
3. Special XML access commands

Full MediaCmd Ajax/XML Access

This access methods allows our javascript or php application to access all the same functions used by Drastic's GUIs from an html interface. The basic form of the commands is:

http://localhost:1080/VVWXMLMediaCmd?<mediacmd>

The <mediacmd> is a series of ampersand delimited (&) commands and modifiers. Normally this command will be sent via an HTTPObject, and will return synchronously or asynchronously a standard XML return that can be parsed. To send a command in AJAX/Javascript, you will first need to instantiate an HTTPObject to send it through. Here is an HTTPObject instantiation that will work in most browsers:

```
// Create an HttpObject
function getHTTPObject()
{
	var xmlhttp;

	/*@cc_on
		@if (@_jscript_version >= 5)
			try
			{
				xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
			}catch (e)
			{
				try
				{
					xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
				} catch (E)
				{
					xmlhttp = false;
				}
			}
		@else
			xmlhttp = false;
		@end @*/

	if (!xmlhttp && typeof XMLHttpRequest != 'undefined')
	{
		try
		{
			xmlhttp = new XMLHttpRequest();
		} catch (e)
		{
			xmlhttp = false;
		}
	}
	return xmlhttp;
```

```
}

// Instantiate the various HTTP Objects
var _xmlHttp = getHTTPObject(); // Create the HTTP Object
```

Once the HTTPObject is instantiated into a variable, the variable (_xmlHttp in this case) can be used to call the DDR and send and receive MediaCmds. These commands can be sent synchronously (the command will complete and return the XML immediately) or asynchronously (the command will process, but return immediately. Later a callback will be called with the return XML data). Either way the return will be the same.

To send a command synchronously (return after processing) without using the return:

```
function play()
{
	// Build the URL to connect to
	var url = "VVWXMLMediaCmd?Play”;
	// Open a connection to the server
	_xmlHttp.open("GET", url, false);  // indicates sync call
	// Send the request
	_xmlHttp.send(null);
}
```

For a command that is sent synchronously, but the return needs to be processed, the call is very similar:

```
function getClipMode()
{
	// Build the URL to connect to
	var url = "VVWXMLMediaCmd?GetValue&position=0&cmdalt=ClipMode&Flags=-1”;
	// Open a connection to the server
	_xmlHttp.open("GET", url, false);  // indicates sync call
	// Send the request
	_xmlHttp.send(null);
	// Get the clip mode out of the XML response
	var xmlobject = _xmlHttpMode.responseXML;
	if(xmlobject == null)	{
		return;
	}
	// Get MediaCmd return (in XML)
	var mCmd = xmlobject.getElementsByTagName("MediaCmd");
	if(mCmd[0])
	{
		// Return the current mode
		return mCmd[0].getElementsByTagName("Position")[0].getAttribute("Value")
    }
	return “errorValue”;
```

```
}
```

A typical XML return would look like this:

```
<MediaCmd>
  <!--  Drastic MEDIACMD xml structure version 1,0  -->
  <CmdID Value="-98237952"/>
  <StructSize Value="1116"/>
  <Channel Value="0"/>
  <Cmd Value="2">Play</Cmd>
  <Speed Value="65520">0</Speed>
  <Position Value="-1" TcType="drop-frame">99:59:59;29</Position>
</MediaCmd>
```

A status return (VVWXMLGetStatus) would look like this:

```
<MediaCmd>
  <!--  Drastic MEDIACMD xml structure version 1,0  -->
  <CmdID Value="-98237952"/>
  <StructSize Value="1116"/>
  <Channel Value="0"/>
  <Cmd Value="1" UseClipID="1">Pause</Cmd>
  <Speed Value="0">0</Speed>
  <VideoChannels Value="-1"/>
  <AudioChannels Value="0"/>
  <InfoChannels Value="-1"/>
  <CmdAlt Value="0"/>
  <Position Value="300" TcType="drop-frame">00:00:10;00</Position>
  <Start Value="-1" TcType="drop-frame">99:59:59;29</Start>
  <End Value="396" TcType="drop-frame">00:00:13;06</End>
  <ClipID>dt-prev</ClipID>
  <FileName>
M:/TestVid/T2V013_Europe3060/Video_1080i30_422_10b_YUV/
T2V013233_Follow_that_ship_1920x1080i30_10b_P422.yuv
  </FileName>
</MediaCmd>
```

Often, to maximize user responsiveness, or to allow for long commands to process, commands need to be sent asynchronously. The asynchronous version of the command is essentially the same as the synchronous with processing version, except the send and return are divided into separate functions:

```
function getClipMode()
{
        // Build the URL to connect to
        var url = "VVWXMLMediaCmd?GetValue&position=0&cmdalt=ClipMode&Flags=-1”;
        // Open a connection to the server
        _xmlHttp.open("GET", url, false);   // indicates sync call
        // Setup a function for the server to run when it's done
        _xmlHttp.onreadystatechange = updateClipMode;
        // Send the request
```

```
        _xmlHttp.send(null);
}
// A response to an 'Mode' request has been received
function updateMode()
        if (_xmlHttpMode.readyState == 4)
        {
                // A complete response has been received
                // Get the clip mode out of the XML response
                var xmlobject = _xmlHttpMode.responseXML;
                if(xmlobject == null)         {
                        return;
                }
                // Get MediaCmd return (in XML)
                var mCmd = xmlobject.getElementsByTagName("MediaCmd");
                if(mCmd[0])
                {
                        // Return the current mode
                        return mCmd[0].getElementsByTagName("Position")[0].getAttribute("Value")
                }
        }
        return “errorValue”;
}
```

VVWXMLMediaCmd Main Commands

The first parameter of the VVWXMLMediaCmd? (following the question mark) must be be one of the following commands:

Stop	Full stop/all stop/E to E
Pause	Pause on current frame, seek or load
Play	Play, either at normal speed or shuttle speeds. May also load and seek.
Record	Record to the disk or tape
RecStop	Prepare for a record
Eject	Eject the current tape or media
Transfer	Transfer to/from an internal channel and a VTR
Insert	Insert media into the clip bin or time code space
Blank	Remove media from the clip bin or time code space
Delete	Delete media from the storage and blank it
Trim	Alter a clip or time code space edit
ChanSelect	Change the currently selected channels
GetState	Get the current channel state
SetState	Set the current channel state
GetValue	Get a setup value
ValueSupported	See if a setup value is supported
SetValue	Change a setup value
Error	Report an error
Terminate	Kill the current operation
Abort	Abort the current operation

VVWXMLMediaCmd Modifiers

With these commands a number of modifiers are available. Each modifier must be separated by an ampersand (&) on the command line.

channel=%d	Specify the channel this command should be sent to
position=%s	Set the position element for a command 1:00:00:00 – go to one hour +5 – go forward from the current location 5 frames -5:00 – go backward from the current location 5 seconds 1800 – go to one minutes (specified as 1800 frames, not drop frame time code0
start=%s	Set the start element (see position for format)
end=%s	Set the end element (see position for format)
speed=%d	Set the speed element for a command 65520 – normal forward play (100%) -65520 – reverse play 32760 – half play speed (50%) -655200 – 10 times reverse speed 0 - pause (no play)
timems	Millisecond time for the command
cmdalt	Set the cmdalt element of the mediacmd
videochannels	Which video channels to use (bitwise)
audiochannels	Which audio channels to use (bitwise)
infochannels	Which information channels to use (bitwise)
clipid	8 character clip identifier
filename	filename for the command
string	String to be used in the command

There are a number of flags that may be used, just like the elements above

Deferred	Wait for previous command to complete before new this command
OverrideDeferred	Override a previously deferred command
Loop	Loop whole clip, or a start/end subset

AllIDs	Command should affect all available clip ids
NoClipFiles	Ignore clip space clips
NoTCSpaces	Ignore conform space files
IsShuttle	The command should be interpreted as a shuttle, even for normal play
UsingCurrent	Use the current start/end/position
UseFrameCount	Use the absolute frame count, not the time code values
Fields	Use fields, is not a progressive signal format
Ripple	When removing a file, ripple the following files back
Trigger	Wait for a trigger
Preview	Doing a preview, not a full play
Test	Don't do the command, just see if it exists
NoReturn	Don't return any information from the command

VVWXMLMediaCmd Examples

VVWXMLMediaCmd?play
Normal play
VVWXMLMediaCmd?play&speed=32760
Play at 50% forward speed
VVWXMLMediaCmd?play&speed=-65520
Play at 100% reverse play speed
VVWXMLMediaCmd?play&start=1:00&end=4:00&loop
Play from one second to four seconds in a loop
VVWXMLMediaCmd?pause
Pause the channel
VVWXMLMediaCmd?stop
Stop (e to e passthrough) the channel
VVWXMLMediaCmd?pause&position=1:00:00
Seek to one minute
VVWXMLMediaCmd?record&clipid=newrec&end=5:00
Record a new file named 'newrec' which will be five seconds long

Dealing with Picon Images

Server Mode, clip: Kroatien, file: KroatienMovie.mov

http://localhost/VVWXMLMediaCmd?SetValue&cmdalt=1000000&clipid=Kroatien&position=200
– Make a new picon from frame 200 of the clip Kroatien – result name: KroatienMovie.picon.jpg
http://localhost/VVWXMLMediaCmd?GetValue&cmdalt=1000000&clipid=Kroatien&position=ffffffff
– Return the actual file name of the picon file (char elem 9) – result name: Kroatien.picon.jpg
http://localhost/VVWXMLMediaCmd?GetValue&cmdalt=1000000&clipid=Kroatien&position=4294967295

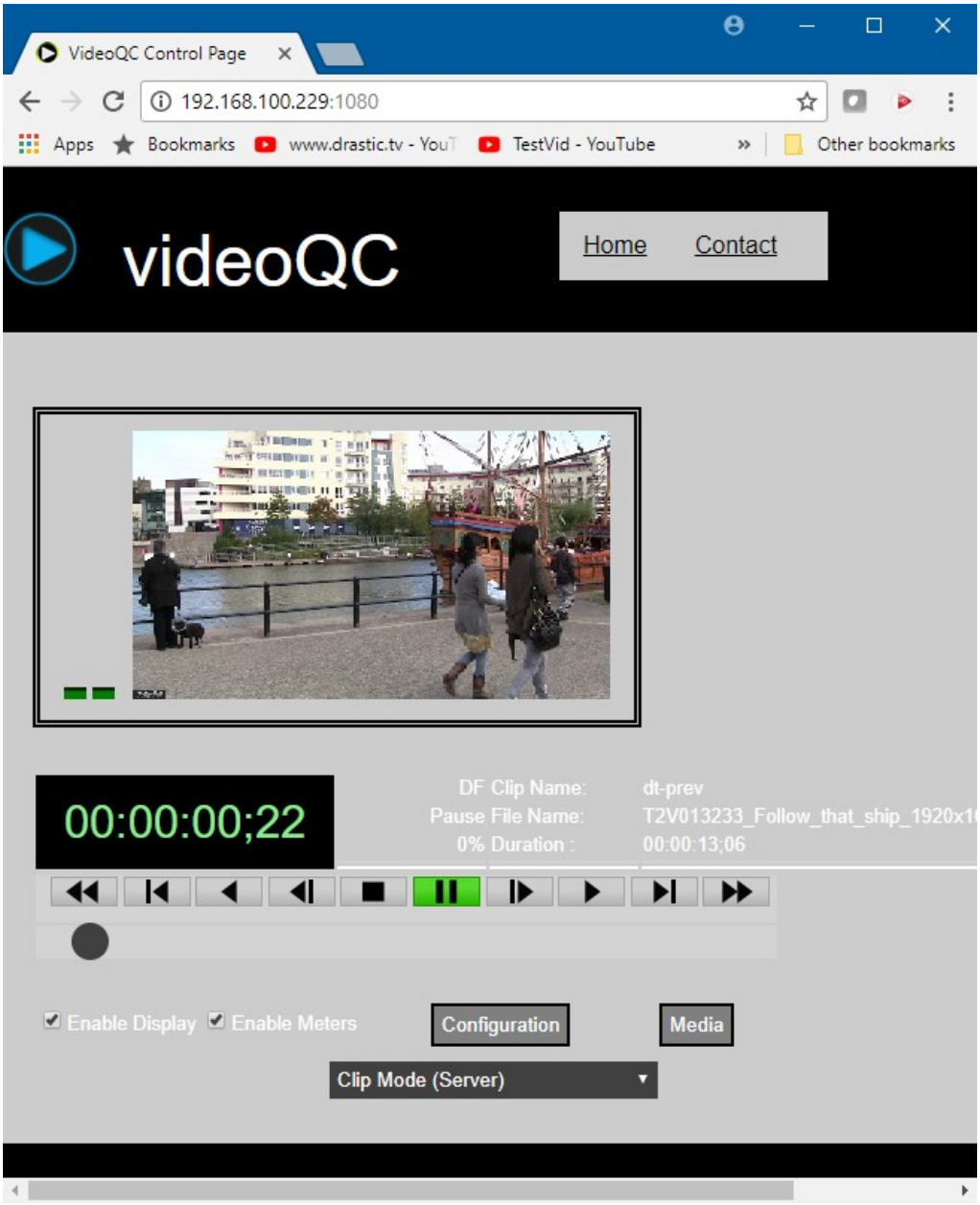
– Return the size of the picon file in the Position elements – result: dwPosition = 7900
http://localhost/VVWXMLMediaCmd? GetValue&cmdalt=1000000&clipid=Kroatien&position=1
– Return the actual bytes of data for the JPEG picon frame in arbID – result: Not available in HTTP, have to use C/C++
http://localhost/VVWXMLMediaCmd? SetValue&cmdalt=1000000&filename=V:\Media\ KroatienMovie.mov&position=100
– Make a new picon from frame 100 without associating it with the clip – result name: KroatienMovie.picon.jpg (not normally used, conflicts with vtr tape mode picon)

VTR Tape Mode, Time line 00:00:01:00 Kroatien.mov?

http://localhost/VVWXMLMediaCmd? SetValue&cmdalt=1000000&filename=V:\Media\ Kroatien.mov&position=1000
– Make a new picon from the frame at position 1000, default for file – result name: Kroatien.picon.jpg
http://localhost/VVWXMLMediaCmd? GetValue&cmdalt=1000000&filename=V:\Media\ Kroatien.mov&position=ffffffff
– Return the actual file name of the picon file (char elem 9) – result name: Nonparticipating
http://localhost/VVWXMLMediaCmd? GetValue&cmdalt=1000000&filename=V:\Media\ Kroatien.mov&position=4294967295
– Return the size of the picon file in the Position elements – result: dwPosition = 7900
http://localhost/VVWXMLMediaCmd? GetValue&cmdalt=1000000&filename=V:\Media\ Kroatien.mov&position=1
– Return the actual bytes of data for the JPEG picon frame in arbID – result: Not available in HTTP, have to use C/C++

Returning Preview Images And Meters

The MediaCMD API includes commands to return a small, RGBA preview image and audio meter level values to provide confidence monitoring. The code to achieve this is used in the videoQC and Net-X-SDI HTML transport examples:



The command to return the preview image is

http://localhost:1080/VVWXMLMediaCmd?getvalue&cmdalt=previewframe&start=320&end=180&position=1&preview

The start is the requested width, the end is the requested height and the preview flag indicates the image should be returned in RGBA order (for web browsers) rather then BGRA order. A "&convert" flag can also be added to cause the frame to be zipped before being sent. The response XML to this will include an <arbID> element with the hex sent as a string.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<MediaCmd>
  <!-- Drastic MEDIACMD xml structure version 1,0 --><CmdID Value="-98237952" />
  <StructSize Value="230476" />
  <Channel Value="0" />
  <Cmd Value="14" UseClipID="1">GetValue</Cmd>
  <CmdAlt Value="1000007" />
  <Position Value="1" TcType="drop-frame">00:00:00;01</Position>
  <Start Value="320" TcType="drop-frame">00:00:10;20</Start>
  <End Value="180" TcType="drop-frame">00:00:06;00</End>
  <arbID Type="hex"   Length="230400">FAFFFDFFF8FDFBFF6....A6267FF7C7579F
  </arbID>
</MediaCmd>
```

From that arbID, the data can be converted back to binary with hexToBytes() and optionally unzipped with Zlib.Inflate(), and then copied into a 2D context for display with imageData.data.set() and then context.putImageData().

The audio meter level can be retrieved as a value between 0 and 65536 for RMS, Peak or Loudness. The command to get the meter value is

http://localhost:1080/VVWXMLMediaCmd?
GetValue&position=0&cmdalt=AudWavePeakRMS&Flags=-1

On return, the left/right RMS values will be in the Start element, value attribute, and the peak will be in the End element, value attribute. To separate the right and left values, remove the upper and lower words of the value:

```
var leftRMS = (Start % 65536);
var rightRMS = (Start / 65536);
```

Special XML Access Commands

XML Returns. These are to be used with Ajax/DOM pages.

http://localhost/VVWXMLGetStatus
– Returns an XML package including state, speed, position start and end points.
<MediaCmd> <!-- Drastic MEDIACMD xml structure version 1,0 --> <CmdID Value="-98237952"/> <StructSize Value="1116"/> <Channel Value="0"/> <Cmd Value="1" UseClipID="1">Pause</Cmd> <Speed Value="0">0</Speed> <VideoChannels Value="-1"/> <AudioChannels Value="0"/> <InfoChannels Value="-1"/> <CmdAlt Value="0"/> <Position Value="627" TcType="drop-frame">00:00:20;27</Position> <Start Value="-1" TcType="drop-frame">99:59:59;29</Start> <End Value="396" TcType="drop-frame">00:00:13;06</End> <ClipID>dt-prev</ClipID> <FileName> M:/TestVid/T2V013_Europe3060/Video_1080i30_422_10b_YUV/ T2V013233_Follow_that_ship_1920x1080i30_10b_P422.yuv </FileName> </MediaCmd>
http://localhost/VVWXMLNextClip
– Returns an XML package with all the clip information. Used to retrieve the clip bin information
http://localhost/VVWXMLClipInfo
– Returns an XML package with all the clip information. Used to retrieve information on a specific clip
http://localhost/VVWXMLEDLState
– Used in conjunction with VVWXMLEDLInfo to retrieve the time code space edits. The command will always be VVWXMLEDLState? position=#&videochannels=#&audiochannels=#&infochannels=#.
http://localhost/VVWXMLEDLInfo
– Used in conjunction with VVWXMLEDLState to retrieve the time code space edits.

Here is a basic EDL retrieval session:

Call··· ·Position··· ·Start··· ·End ··· ·V ··· ·A ··· ·I ··· ·File Name··· ·Comment
VVWXMLEDLInfo··· ·0···
0··· ·0··· ·0···
Restart list at 0
return info··· ·0··· ·0··· ·300··· ·1··· ·2··· ·0··· ·file1.mov··· ·10 sec VA2 from file1
VVWXMLEDLState··· ·0···
0··· ·0··· ·0···
First state sent in above
return state··· ·0···
1··· ·2··· ·0···
Used clip channels to pass back into Info
VVWXMLEDLInfo··· ·0···
1··· ·2··· ·0···
Copy of the return of VVWXMLEDLState above
return info··· ·0··· ·0··· ·150··· ·0··· ·1··· ·0··· ·file2.wav··· ·5 sec A1 from file2
VVWXMLEDLState··· ·0···
1··· ·2··· ·0···
Use the return of the last VVWXMLEDLState
return state··· ·0···
1··· ·3··· ·0···
These are the channels used so far
VVWXMLEDLInfo··· ·0···
1··· ·3··· ·0···
Copy of the return of VVWXMLEDLState above
return info··· ·150··· ·150··· ·210··· ·0··· ·1··· ·0··· ·file3.wav··· ·2 sec A1 from file3
VVWXMLEDLState··· ·0···
1··· ·3··· ·0···
Use the return of the last VVWXMLEDLState
return state··· ·150···
0··· ·1··· ·0···
All edits completed before 150

Take the MEDIACMD struct returned from VVWXMLEDLState and find the next active clip. For the first clip in the timeline, send all zeroes. Other than the first call, all calls should include the position/channel bits from the previous VVWXMLEDLState call and (other than first call), VVWXMLEDLState should be called immediately before VVWXMLEDLInfo.

http://localhost/VVWXMLNextDirEntry
Used to retrieve the directory structure.

Takes 2 parameters:
The base directory you are getting the listing for
The last directory entry returned

Assuming you had a directory structure that looked like this:

\Record\
\Record\Test.wav
\Record\Test.avi
\OfflineMedia\
\OfflineMedia\EmptyDir\
\OfflineMedia\retry.doc
\OfflineMedia\big.tga
\LocalMedia\AnotherDir\
\LocalMedia\test.aiff

The first call would only include the parameter '\'

http://localhost/VVWXMLNextDirEntry?
Returns: <locator>\Record</locator>
This will return the first FileDir XML structure that will include the first locator. To get the next item, return the same base path plus the new locator.

To descend into a sub directory, use the sub directory as the base path. To see what is in \Record

http://localhost/VVWXMLNextDirEntry?\?\Record
Returns: <locator>\Record\..</locator>
http://localhost/VVWXMLNextDirEntry?\?\OfflineMedia
Returns: <locator>\OfflineMedia</locator>
http://localhost/VVWXMLNextDirEntry?\?\LocalMedia
Returns: <locator>\LocalMedia</locator>
http://localhost/VVWXMLNextDirEntry?\Record\&\Record\..
Returns: <locator>END OF LIST</locator>
http://localhost/VVWXMLNextDirEntry?\Record\&\Record\ Test.wav

Returns: <locator>\Record\Test.wav</locator>
http://localhost/VVWXMLNextDirEntry?\Record\&\Record\ Test.avi
Returns: <locator>\Record\Test.avi</locator>
http://localhost/VVWXMLFileInfo
– Used to retrieve information on a specific file.
http://localhost/VVWXMLGetErrorMsg&#
– Used to return one error message from the current list. The first call will not include an error number (just VVWXMLGetErrorMsg). This will return an ErrorNumber to use to get the next message (VVWXMLGetErrorMsg&202 for instance), as will each subsequent call. When all the error messages have been returned, it will return an ErrorNumber of -1.

VVWLibrary.js Support Functions

Installed with videoQC, Net-X-SDI and VVW/DrasticDDR is a series of support/helper functions in a file called vvwlibrary.js under the js folder in the web server folder. These mostly encapsulate MediaCMD calls, but also include many helper functions that make using the API easier.

getHTTPObject()

Returns an AJAX HTTP object to be used to send and receive commands.

Standard HTTP Objects

var g_xmlHttp = getHTTPObject(); // Create the HTTP Object

var g_xmlHttpMs = getHTTPObject(); // Create the Last Change Ms HTTP Object

var g_xmlHttpVvwCmd = getHTTPObject(); // Create the VvwCmdHTTP Object

rightTrim(sString)

Trim any blank spaces from the end of the string

rightTrimSlash(sString)

Trim slashes from the end of a string

hasStdExt(sString)

Verify if we have a standard Extension

hasAnyExt(sString)

Verify if contains a dot for an extension

verifyChars(sString)

Convert # to %23

trimEnd(sString)

Trim the end of the string

toForwardSlashes(sString)

Convert backslashes to forward slashes

areCharsValid(sString)

Check is characters are all valid

getInnerText (node)

Get text value from a node for IE or Firefox/Mozilla

getBase(sString)

Get the base of a file path

d2h(d)

Decimal number to hex string

h2d(h)

Hex number ot decimal string

tc2Sec(szTC)

Timecode to seconds

bytes2String(MBytes)

Byte size to string

trimFrames(szTC)

Trim the frames from a timecode string

parseByte16(str, offset)

Convert two hex characters (e.g. “AD”) to a byte (173)

toString16 (val, uppercase)

Convert a byte to a stream hex value

hexToBytes(hex)

Convert a hex string to a byte array

bytesToHex(bytes)

Convert a byte array to a hex string

getLastChangeMs()

Last change as a millisecond to the current clip log or timecode/edit space.

sendVVWCmd(szCmd)

Send Transport Commands

vvwGetCurMs(lChannel)

Get current ms

vvwPlay(lChannel)

Send a play

vvwPlayOffsetAt(lChannel, dwPosition, lOffset, dwMs)

Play from position or offset at time

vvwPlayAtSpeed(lChannel, lSpeed, dwEnd)

Play at speed

vvwPlayFromTo(lChannel, dwFrom, dwTo, fDeferred, fLoop)

Play from to

vvwLoadClipFromFile(lChannel, sz8CharName, szFileName)

Load a clip from a file to an 8 char clip log name

vvwLoadClip(lChannel, sz8CharName, dwStart)

Load clip by its 8 char clip name

vvwBlankAllClips(lChannel)

Blank (remove) all clips

vvwBlankClip(lChannel, sz8CharName)

Blank (remove) clip

vvwSwitchClip(lChannel, sz8CharName, dwPosition, fUseFrameCount)

Switch clip

vvwPlayClip(lChannel, sz8CharName, fDeferred)

Play clip

vvwPlayClipFromTo(lChannel, sz8CharName, dwFrom, dwTo, fDeferred)

Play clip from to

vvwPlayAtMs(lChannel, dwMs)

Play at ms

vvwPlayFromToAtMs(lChannel, dwMs, dwFrom, dwTo)

Play from to at ms

vvwRecordAtMs(lChannel, dwMs, dwFrom, dwTo)

Record at ms

vvwFastForward(lChannel)

Fast forward

vvwFastReverse(lChannel)

Fast reverse

vvwPause(lChannel)

Pause

vvwSeek(lChannel, szPos)

Send Seek

vvwSeekRelative(lChannel, lFrameOffset)

Seek to a position relative to the current position

vvwStop(lChannel)

Stop

vvwRecord(lChannel)

Record

vvwRecordFromTo(lChannel, dwFrom, dwTo)

Record from to

vvwRecordStop(lChannel, sz8CharClipName, dwDuration)

Record Stop

vvwRecordStopFileName(lChannel, sz8CharClipName, szFileName, dwDuration)

Record Stop with filename

vvwRecordPresets(lChannel, lVidEdit, lAudEdit, lInfEdit)

Set record presets

vvwEject(lChannel)

Eject

vvwUpdateStatus(lChannel)

Get the current status

vvwGetState(lChannel)

Return the last known state (see vvwUpdateStatus)

vvwGetFlags(lChannel)

Return the last known flags (see vvwUpdateStatus)

vvwGetSpeed(lChannel)

Return the last known speed (see vvwUpdateStatus)

vvwGetPosition(lChannel)

Return the last known position (see vvwUpdateStatus)

vvwGetLastMs(lChannel)

Return the last know millisecond time (see vvwUpdateStatus)

vvwGetStart(lChannel)

Return the last known start (see vvwUpdateStatus)

vvwGetEnd(lChannel)

Return the last known end (see vvwUpdateStatus)

vvwGetClipName(lChannel)

Return the last known clip name (see vvwUpdateStatus)

vvwGetFileName(lChannel)

Return the last known file name (see vvwUpdateStatus)

vvwGetCurTC(lChannel)

Return the last known time code (see vvwUpdateStatus)

vvwGetCurState(lChannel)

Return the last known state (see vvwUpdateStatus)

vvwSetMetaData(vvwChannel, sz8CharClipName, vvwiInfoRequest, nValue, szValue)

Set a metadata value

vvwGetMetaData(vvwChannel, sz8CharClipName, vvwiInfoRequest)

Get a metadata value

vvwSaveMetaData(vvwChannel, saveType)

Save the current metadata to:

Type 0 = file, 3 = user, 4 = system

Basic Metadata Elements

```
const vvwiFileName = 0;
const vvwiNativeLocator = 1;
const vvwiUniversalName = 2;
const vvwiIP = 3;
const vvwiSourceLocator = 4;

const vvwiChannel = 5;
const vvwiChannelName = 6;
const vvwiChannelDescription = 7;
const vvwiTitle = 8;
const vvwiSubject = 9;
const vvwiCategory = 10;                    // <-- 10
const vvwiKeywords = 11;
const vvwiRatings = 12;
const vvwiComments = 13;
const vvwiOwner = 14;
const vvwiEditor = 15;
const vvwiSupplier = 16;
const vvwiSource = 17;
const vvwiProject = 18;
const vvwiStatus = 19;
const vvwiAuthor = 20;                              // <-- 20
const vvwiRevisionNumber = 21;
const vvwiProduced = 22;
const vvwiAlbum = 23;
const vvwiArtist = 24;
const vvwiComposer = 25;
const vvwiCopyright = 26;
const vvwiCreationData = 27;
const vvwiDescription = 28;
```

```
const vvwiDirector = 29;
const vvwiDisclaimer = 30;                  // <-- 30
const vvwiEncodedBy = 31;
const vvwiFullName = 32;
const vvwiGenre = 33;
const vvwiHostComputer = 34;
const vvwiInformation = 35;
const vvwiMake = 36;
const vvwiModel = 37;
const vvwiOriginalArtist = 38;
const vvwiOriginalFormat = 39;
const vvwiPerformers = 40;                  // <-- 40
const vvwiProducer = 41;
const vvwiProduct = 42;
const vvwiSoftware = 43;
const vvwiSpecialPlaybackRequirements = 44;
const vvwiTrack = 45;
const vvwiWarning = 46;
const vvwiURLLink = 47;
const vvwiEditData1 = 48;
const vvwiEditData2 = 49;
const vvwiEditData3 = 50;                   // <-- 50
const vvwiEditData4 = 51;
const vvwiEditData5 = 52;
const vvwiEditData6 = 53;
const vvwiEditData7 = 54;
const vvwiEditData8 = 55;
const vvwiEditData9 = 56;
const vvwiVersionString = 57;
const vvwiManufacturer = 58;
const vvwiLanguage = 59;
```

```
const vvwiFormat = 60;                          // <-- 60
const vvwiInputDevice = 61;
const vvwiDeviceModelNum = 62;
const vvwiDeviceSerialNum = 63;
const vvwiReel = 64;
const vvwiShot = 65;
const vvwiTake = 66;
const vvwiSlateInfo = 67;
const vvwiFrameAttribute = 68;
const vvwiEpisode = 69;
const vvwiScene = 70;                       // <-- 70
const vvwiDailyRoll = 71;
const vvwiCamRoll = 72;
const vvwiSoundRoll = 73;
const vvwiLabRoll = 74;
const vvwiKeyNumberPrefix = 75;
const vvwiInkNumberPrefix = 76;
const vvwiPictureIcon = 77;
const vvwiProxyFile = 78;
const vvwiCustomMetadataBlockPointer = 79;
const vvwiImageInfo = 80;
const vvwiUMID = 81;
const vvwiEND_OF_STRINGS = 82;

const vvwiNumericStart = 4096;//0x1000,
const vvwiTimeCode = 4097;
const vvwiUserBits = 4098;
const vvwiVITCTimeCode = 4099;
const vvwiVITCUserBits = 4100;
const vvwiVITCLine3 = 4101;
const vvwiPosterFrame = 4102;
```

```
const vvwiAFrame = 4103;
const vvwiAspectRatio = 4104;
const vvwiOriginalRate = 4105;
const vvwiOriginalScale = 4106;   //
const vvwiConversions = 4107;
const vvwiVersionNumber = 4108;
const vvwiFileSize = 4109;
const vvwiFileDate = 4110;
const vvwiFileTime = 4111;
const vvwiSequenceNumber = 4112;
const vvwiTotalStreams = 4113;
const vvwiTotalLength = 4114;
const vvwiFilmManufacturerCode = 4115;
const vvwiFilmTypeCode = 4116;      //
const vvwiWhitePoint = 4117;
const vvwiBlackPoint = 4118;
const vvwiBlackGain = 4119;
const vvwiBreakPoint = 4120;
const vvwiGamma1000 = 4121;
const vvwiTagNumber = 4122;
const vvwiFlags = 4123;
const vvwiTimeCodeType = 4124;
const vvwiLTCTimeCodeType = 4125;
const vvwiVITCTimeCodeType = 4126;   //
const vvwiProdDate = 4127;
const vvwiUniqueID = 4128;
const vvwiCustomMetadataBlockType = 4129;
const vvwiCustomMetadataBlockSize = 4130;
const vvwiNorthSouthEastWest = 4131;
const vvwiLatitude = 4132;
const vvwiLongitude = 4133;
```

```
const vvwiExposure = 4134;
const vvwiRedGain = 4135;
const vvwiBlueGain = 4136;          //
const vvwiWhiteBalance = 4137;
const vvwiMatrix = 4138;
const vvwiGreenGain = 4139;
/*
const vvwiVideoWidth = 65536;//0x10000,
const vvwiVideoHeight,
const vvwiVideoPlanes,
const vvwiVideoBitCount,
const vvwiVideoCompression,
const vvwiVideoSizeImage,
const vvwiVideoXPelsPerMeter,
const vvwiVideoYPelsPerMeter,
const vvwiVideoClrUsed,
const vvwiVideoClrImportant,
const vvwiVideoReserved,
const vvwiVideoFccType,
const vvwiVideoFccHandler,
const vvwiVideoFlags,
const vvwiVideoCaps,
const vvwiVideoPriority,
const vvwiVideoLanguage,
const vvwiVideoScale,
const vvwiVideoRate,
const vvwiVideoStart,
const vvwiVideoLength,
const vvwiVideoInitialFrames,
const vvwiVideoSuggestedBufferSize,
const vvwiVideoQuality,
```

```
const vvwiVideoSampleSize,
const vvwiVideoEditCount,
const vvwiVideoFormatChangeCount,
const vvwiVideoPitch,
const vvwiVideoDrFlags,
const vvwiVideoFileType,
const vvwiVideoResDrastic,
const vvwiAudioType,
const vvwiAudioChannels,
const vvwiAudioFrequency,
const vvwiAudioBits,
const vvwiLastElementPlus1
        // DO NOT ADD ANYTHING BELOW vvwiLastElementPlus1
*/
```